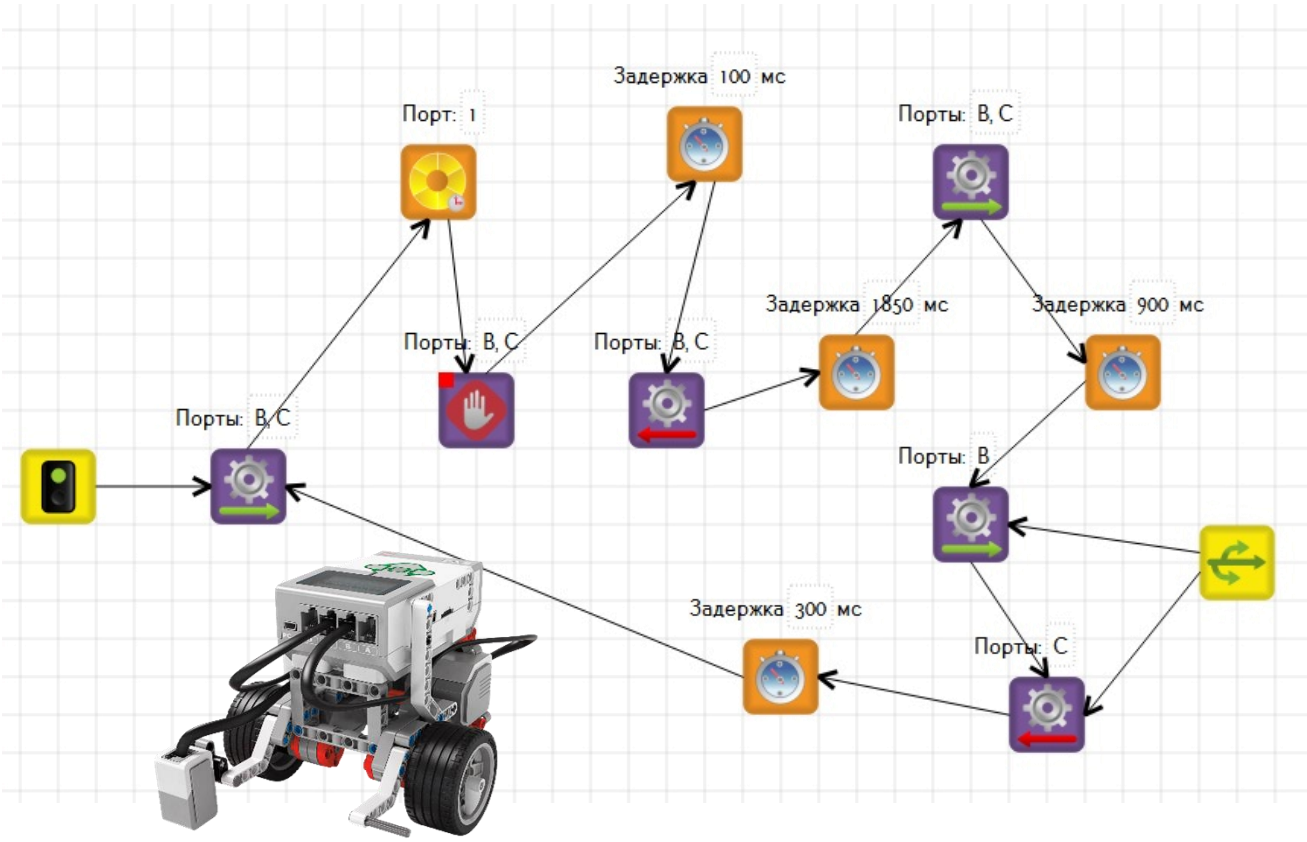
Н. Г. Павлова

Робототехника

Основы программирования робота Lego Mindstorms EV3 в TRIK Studio



Н. Г. Павлова

Робототехника

Основы программирования робота Lego Mindstorms EV3 в TRIK Studio

Практическое пособие

Тюмень
2019

УДК 621
ББК 32.816.2+32.973.2
П 12

Рецензенты: Игнеева А.Д.

Павлова Н. Г. Робототехника. Основы программирования
П 12 робота Lego Mindstorms EV3 в TRIK Studio: практическое пособие / Н. Г. Павлова.- Тюмень: ГАПОУ ТО «Колледж цифровых и педагогических технологий», 2019.- 119 с.

ISNN 978-5-6043732-6-2

В пособии представлены описание и основные возможности среды TRIK Studio, рассмотрены примеры решения задач на основе образовательного набора Lego Mindstorms EV3.

Пособие подготовлено на основе опыта преподавания среды программирования TRIK Studio в рамках клуба робототехники и прототипирования «RoboCraft», действующего на базе Государственного автономного профессионального образовательного учреждения Тюменской области «Колледж цифровых и педагогических технологий».

Практическое пособие предназначено для преподавателей и обучающихся основного и дополнительного образования, а также для студентов и преподавателей среднего профессионального образования по специальности «Педагогика дополнительного образования в области технического творчества».

УДК 621
БКК 32.816.2+32.973.2

Оглавление

Введение

Более тридцати лет назад академик Андрей Петрович Ершов сказал: «Программирование – вторая грамотность», и в истинности этих слов не стоит сомневаться в сегодняшних реалиях. Но учить только программированию – мало, так как детям необходимо развивать также и моторику: пользоваться отверткой, дрелью, паяльником и другими важными в жизни инструментами.

Робототехника – это практически идеальное сочетание для реализации целей образования: обучающиеся проектируют, конструируют, проявляя изобретательность, программируют получившиеся устройства, изучая основы кибернетики. Кроме того, изучение программирования и робототехники – шаг в сторону инженерного образования, о важности которого сейчас много говорится.

При программировании роботов достаточно широко начали использовать среду TRIK Studio, причем не только в России, но и за рубежом.

Поэтому целью данного пособия стало знакомство с основами среды программирования TRIK Studio и формирование умения применять изученные алгоритмы и навыки программирования к решению задач соревновательной робототехники.

Язык TRIK Studio — среда визуального и текстового программирования образовательных конструкторов, которая появилась в рамках развития проекта «QReal:Robots» кафедры системного программирования Санкт-Петербургского Государственного Университета.

TRIK Studio – визуальная среда, интерфейс которой понятен даже для обучающихся начальных классов. Для обучающихся старших классов и студентов возможен переход от визуального программирования к текстовому программированию на языках C, C++, JavaScript или PascalABC.NET, поэтому ограничений по возрасту на использование TRIK Studio не существует.

TRIK Studio поддерживает программирование контроллеров Lego Mindstorms NXT, Lego Mindstorms EV3 и ТРИК российской разработки, а также квадрокоптер «Пионер».

Программа на визуальном языке (диаграмма) может быть исполнена в следующих режимах:

- отладка на симуляторе,
- отладка на компьютере с загрузкой на робота по USB, Bluetooth или Wi-Fi каналам,
- генерация кода на текстовый язык с дальнейшим автономным выполнением его на роботе.

Во время отладки на симуляторе диаграмма интерпретируется на двумерной имитационной модели робота. Пользователь имеет возможность создать двумерную модель мира из цветных линий, стенок, различных полос препятствий и других элементов. Кроме того, в библиотеке TRIK Studio имеются готовые модели полей, используемых в большинстве робототехнических соревнованияй. Эта возможность является удобной для того, чтобы произвести первоначальную отладку программы прежде чем загрузить ее на робота. Наличие симулятора открывает возможности обучения программированию в тех образовательных учреждениях, которые не имеют в наличии необходимого оборудования.

Отладка программы на компьютере с одновременной загрузкой на робота удобна тем, что можно отследить поведение робота и программы в режиме реального времени. Здесь можно просмотреть значения переменных в соответствующем окне среды, а также увидеть графики показаний с датчиков, подключенных к роботу.

Последний режим – генерация кода позволяет перейти от визуального программирования к текстовому. Тестовый код программы будет отображаться во встроенном редакторе при загрузке программы на робота. Данный редактор обладает всеми возможностями полноценного редактора кода. В дистрибутив среды входят необходимые инструменты для построения и передачи программ на робота, поэтому процесс компиляции и взаимодействия с контроллерами остается для пользователей незаметным.

В пособии сделан подробный разбор задач для контроллера Lego Mindstorms EV3. Помимо изучения нового материала и примеров решения задач, пособие содержит упражнения, позволяющие закрепить полученные теоретические знания.

Представленные задачи и упражнения сформулированы без привязки к аппаратной платформе и конкретной среде программирования и, поэтому могут быть использованы для программирования различных робототехнических образовательных наборов.

Предложенное пособие будет полезным как для самостоятельного узучения среды TRIK Studio так и в урочной и во внеурочной деятельности.

1. Обзор среды

Для установки среды TRIK Studio на компьютере необходимо скачать пакеты, которые доступны под Windows, Linux и Mac OS X. Среда полностью бесплатна, имеет открытый исходный код и распространяется под лицензией Apache License 2.0. При этом, даже несмотря на то, что официальные драйверы для Lego NXT не доступны для Linux и Mac OS X x64, в TRIK Studio существует собственная их реализация.

После установки языка программирования в окне приветствия нужно будет выбрать **«Создать Проект»** или **«Открыть проект»** (рисунок 1.1).

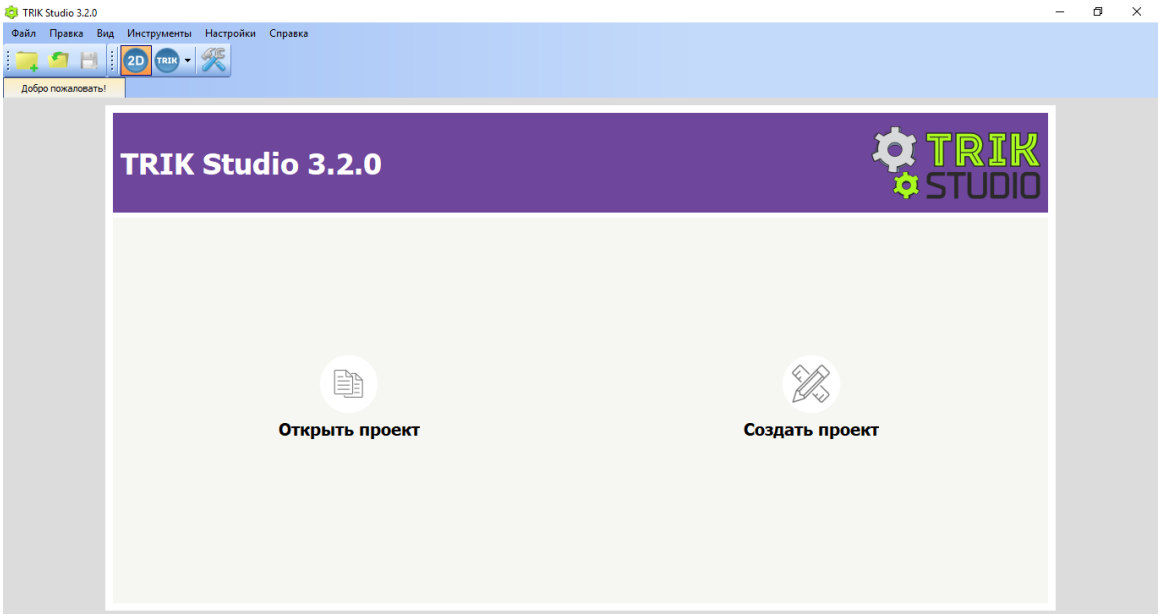


Рисунок 1.1 – Окно приветствия языка TRIK Studio.

После нажатия на кнопку **«Открыть проект»** необходимо выбрать сохранённую ранее программу и открыть ее.

После нажатия на кнопку **«Создать проект»** открывается «Интерфейс» (режим редактирования). Это основной режим, в котором создается тело программы.

Общий вид интерфейса приведен на рисунке 1.2.

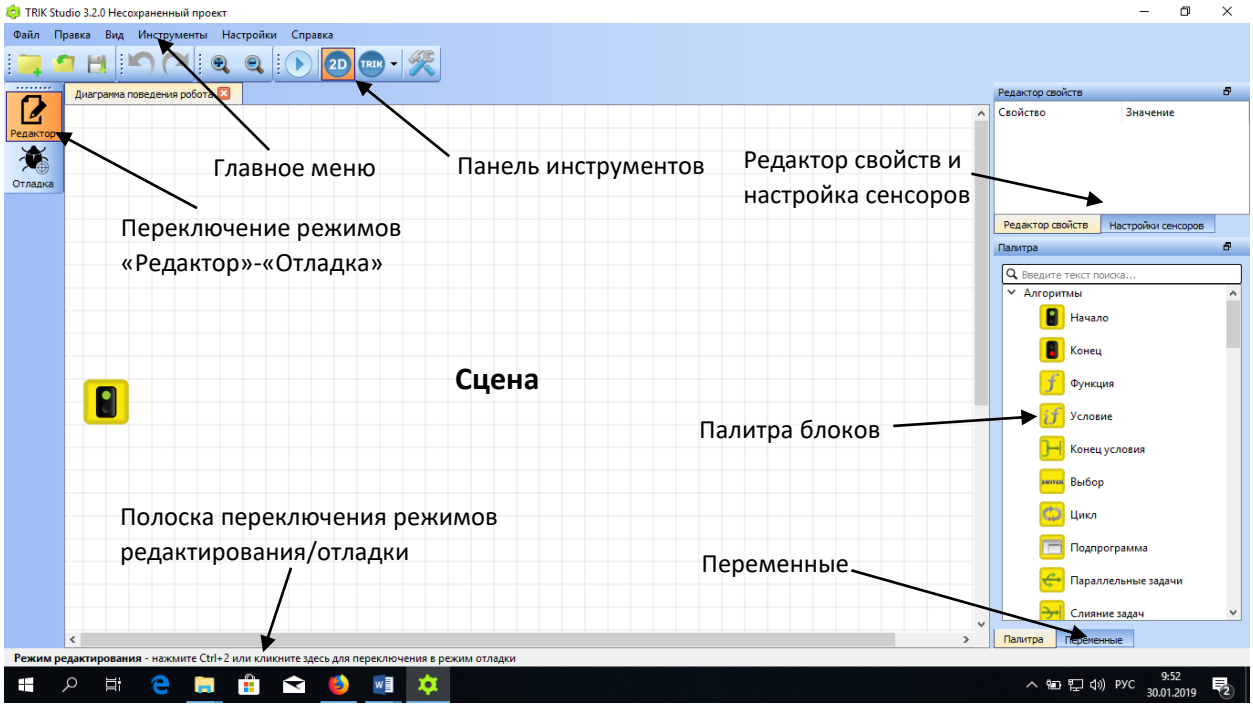


Рисунок 1.2 – Окно редактирования программы.

Главное меню содержит набор базовых операций и настроек среды.

Пункты меню "Файл":

- *Новый проект* – создать новый проект.
- *Недавние проекты...* – открыть один из недавно использовавшихся проектов.
- *Новая диаграмма* – создать новую диаграмму в текущем проекте.
- *Открыть...* – открыть сохраненный проект. Проекты хранятся на диске в файлах с расширением *.qrs.
- *Сохранить* – сохранить текущий проект.
- *Сохранить как...* – сохранить текущий проект в выбранное местоположение. Последующие сохранения будут производиться в выбранный файл.
- *Сохранить диаграмму как картинку...* – сохранить на диск текущее изображение на сцене.
- *Печать* – распечатать диаграмму или текст, используя стандартный диалог печати.
- *Выход* – выйти из программы.

Пункты меню "Правка":

- *Отменить* – отменить выполненное действие.
- *Повторить* – сделать снова отменённое действие.
- *Найти* – открыть диалог поиска по проекту.
- *Копировать* – копировать выделенный фрагмент в буфер обмена.
- *Вырезать* – вырезать выделенный фрагмент из диаграммы и поместить в буфер обмена.
- *Вставить* – вставить скопированный фрагмент из буфера обмена указанное место на сцене.

Пункты меню "Вид":

- *Приблизить* – увеличить масштаб сцены.
- *Отдалить* – уменьшить масштаб сцены.
- *Показывать сплешскрин* – включить/выключить отображение стартового экрана с логотипом TRIK Studio.
- *Панели* – включить/выключить отображение различных плавающих окон.

Пункты меню "Инструменты". Пункты в этом меню могут появляться и исчезать в зависимости от конфигурации программы.

- *Жесты мышью* – открыть вкладку со списком доступных жестов мышью.
- *2D модель* – переключить режим работы интерпретатора на двумерную модель.
- *Реальный робот* – переключить режим работы интерпретатора на управление реальным роботом по WiFi, Bluetooth или USB, в зависимости от используемого конструктора и настроек соединения с роботом.
- *Настройки* – открыть окно настройки модели робота, с которой осуществляется работа в TRIK Studio.
- *Сохранить как упражнение...* – сохранить текущую диаграмму и двумерную модель как упражнение для учащихся. В упражнении нельзя модифицировать некоторые части решения, например,

расположение объектов в двумерной модели или конфигурацию датчиков. Выбор этого пункта меню открывает диалог, в котором можно выбрать фиксируемые части.

- *Загрузить системное ПО на робот* – загрузить в робота прошивку, позволяющую выполнять загруженные в него программы. Обычно этого делать не требуется или требуется сделать только один раз перед началом работы.
- *Генерировать код* – открыть вкладку со сгенерированным кодом, соответствующим текущей диаграмме и выбранному текстовому языку (этот пункт может выглядеть как "Генерировать код на JavaScript", "Генерировать код на F#", "Генерировать код на C" и т.д.). Код не загружается на робота и не исполняется, но может быть отредактирован и загружен позже.
- *Загрузить программу* – только сгенерировать код и загрузить на роботf, не исполняя его.
- *Загрузить и выполнить программу* – сгенерировать код, соответствующий текущей диаграмме, загрузить его и исполнить на роботе.
- *Остановить робота* – прервать выполнение программы и послать роботу команды отключения моторов.
- *Подключиться* – установить соединение с роботом. Робот должен быть включен. Показывается только в том случае, если выбранный режим работы с роботом предполагает необходимость постоянного подключения (например, интерпретация программы по Bluetooth или WiFi).
- *Выполнить* – запустить выполнение отображаемой в данный момент диаграммы на роботе в режиме интерпретации. Показывается только в случае, если выбран режим интерпретации.

Пункты меню "Настройки" (подробнее см. раздел Диалог настроек среды TRIK Studio):

- *Настройки...* – открыть окно настроек среды TRIK Studio.
- *Показать сетку* – включить/отключить отображение сетки на сцене.
- *Показать направляющие* – включить/отключить режим отображения вертикальных и горизонтальных направляющих линий для позиционирования объектов относительно друг друга.
- *Включить сетку* – включить/отключить режим автоматического выравнивания по сетке.
- *Показать весь текст* – включить/отключить отображение всех свойств блока прямо на диаграмме. Если текст под пиктограммами отключён, свойства можно просматривать и редактировать в редакторе свойств слева вверху.

Пункты меню "Справка":

- *Помощь* – открыть справочную систему TRIK Studio.
- *О программе* – просмотреть краткую информацию о TRIK Studio (версия, сайт).
- *О Qt...* – просмотреть информацию об инструментарии Qt, использованном для создания TRIK Studio.
- *Проверить на наличие обновлений* – получить из интернета информацию о наличии новой версии среды или какого-либо из подключаемых модулей и запустить обновление при необходимости.

Большинство команд, расположенных в меню вынесены на Панель инструментов.

Панель "Файл" повторяет основные операции, доступные из меню "Файл".

Панель "Правка" содержит кнопки отмены и повторения операции.

Панель "Вид" содержит кнопки, позволяющие масштабировать диаграммы. Этого же можно добиться вращением колесика мыши, наведя курсор на произвольное место рабочей области диаграммы.

Панель "Интерпретатор" содержит команды запуска и остановки выполнения программы, кнопки переключения режимов интерпретации между двумерной моделью и реальным роботом, а также кнопку открытия настроек

робота, доступных из меню "Инструменты". При работе с реальным роботом в режиме интерпретации эта панель также содержит кнопку "Подключиться".

На рисунке 1.3 представлен режим отладки. Переключения между режимами осуществляются нажатием по соответствующим иконкам или с помощью сочетаний клавиш:

- Ctrl+1 – Режим редактирования;
- Ctrl+2 – Режим отладки.

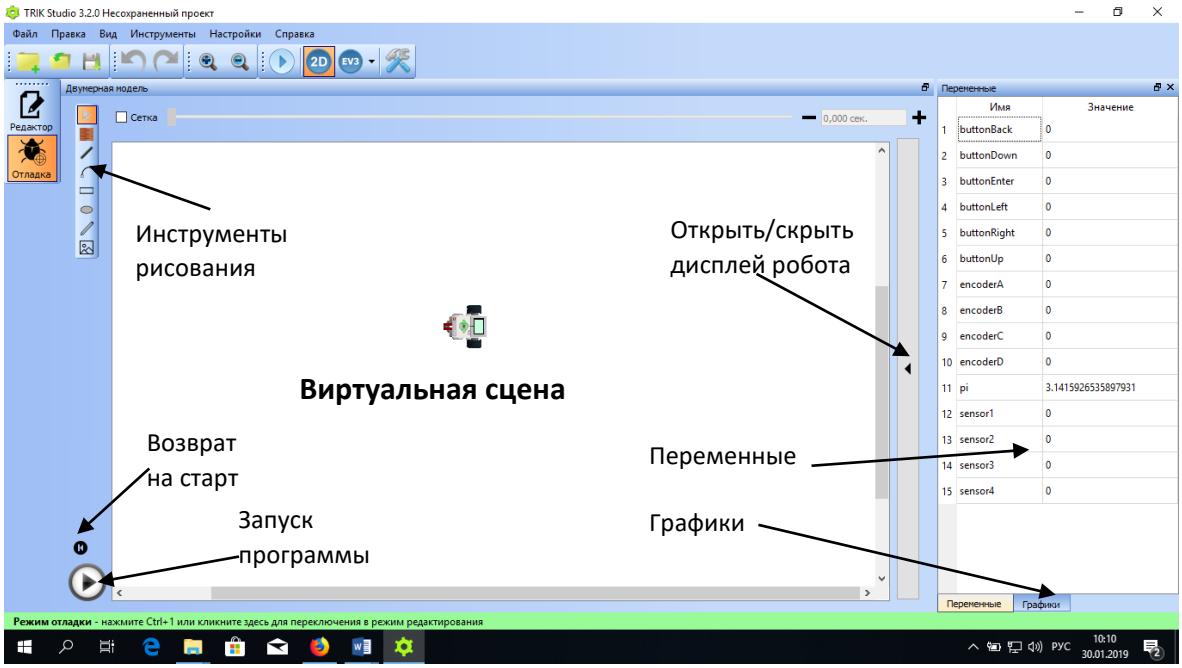


Рисунок 1.3 – Окно отладки программы.

Сцена двумерной модели представляет собой виртуальное пространство, в котором будет двигаться модель робота, обозначенная соответствующей иконкой.

Режим отладки позволяет работать с эмулятором экрана, который можно открыть или закрыть, нажав соответствующую кнопку. Эмулятор экрана и кнопок робота отображает вывод программы на экран робота и позволяет эмулировать нажатие на его кнопки (кликом мыши по соответствующей кнопке на роботе).

Панель «Инструменты рисования» одержит следующие кнопки:

- *Курсор* – переключиться в режим выбора и перестать рисовать стены/линии и т.д.
- *Стена* – добавить на сцену препятствие в виде стены. Препятствие отображается в виде "кирпичной" линии, при касании такой линии сенсором касания или при попадании в зону действия сенсора расстояния

происходит срабатывание сенсора, робот не может проехать сквозь стену. При добавлении с нажатой клавишей Shift стена помещается либо под прямым углом, либо под углом в 45 градусов относительно границ окна.

- *Линия* – добавить на сцену прямую линию. При этом срабатывают находящиеся над линией сенсоры света и соответствующие сенсоры цвета. При добавлении с нажатой клавишей Shift линия помещается либо под прямым углом, либо под углом в 45 градусов относительно границ окна.
- *Эллипс* – добавить на сцену эллипс. При этом срабатывают находящиеся над эллипсом сенсоры света и соответствующие сенсоры цвета. При добавлении с нажатой клавишей Shift добавляется окружность.
- *Карандаш* – добавить на сцену нарисованную мышью линию. При этом срабатывают находящиеся над линией сенсоры света и соответствующие сенсоры цвета.

При выделении линии, карандаша или эллипса появляется всплывающее окно, позволяющее настраивать их цвет и толщину (рисунок 1.4).

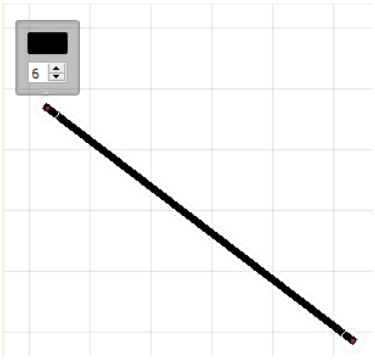


Рисунок 1.4 – Изменение толщины линии.

При выделении робота также появляется всплывающее окно (рисунок 1.5).

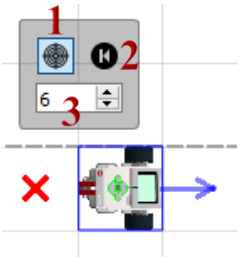


Рисунок 1.5 – Окно при выделении робота.

При нажатии на кнопку 1 включается или выключается центрирование датчиков на роботе. При включённом режиме центрирования сцена автоматически перемещается, выставляя робота в центр, если он выезжает за границы видимого участка.

Кнопка 2 – кнопка возврата на старт. Работает так же, как кнопка в основном окне двумерной модели — возвращает робот на исходную позицию, обозначаемую красным крестиком.

Толщина маркера (кнопка 3) позволяет задать толщину следа, оставляемого роботом при рисовании маркером.

Окно двумерной модели можно отделить от остального интерфейса TRIK Studio, потащив его за заголовок. Вернуть окно на место можно, нажав на кнопку закрытия окна, которое не закроется, а вернётся в главное окно TRIK Studio.

Некоторые важные функции двумерной модели доступны при нажатии на правую кнопку мыши на сцене (рисунок 1.6).

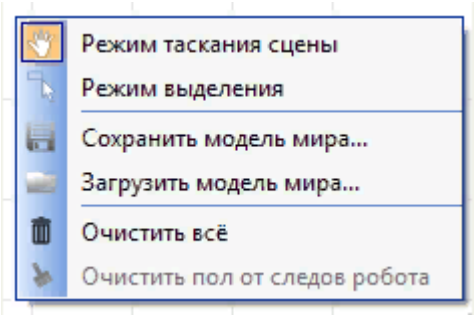


Рисунок 1.6 – Контекстное меню сцены.

- Режим таскания сцены позволяет перемещать сцену с роботом, стенами и линиями при клике и перетаскивании на свободный участок сцены.
- Режим выделения позволяет выделять группу стен, линий и других частей модели при клике и перетаскивании на свободный участок сцены.
- Сохранить модель мира позволяет сохранить нарисованную сцену в виде XML-файла, который потом может быть загружен и использован в другом проекте.
- Загрузить модель мира позволяет загрузить ранее сохранённую модель мира.

- Очистить всё удаляет все стены и цветные линии. Робот при этом остаётся на сцене.
- Очистить пол от следов робота удаляет все цветные линии, нарисованные маркером робота. Остальные элементы остаются на сцене.

Панель «Графики» и «Переменные» будут рассмотрены в разделах 10 и 14 соответственно.

И последний режим – это режим программирования. В TRIK Studio существует возможность визуального моделирования поведения робота. Переключение между режимами программирования, визуального моделирования осуществляется на панели инструментов (рисунок 1.7) либо в меню «Инструменты».

Рисунок 1.7 – Переключение между визуальным режимом и режимом редактирования.

В каждом режиме есть неактивные блоки в палитре, они отображены серым цветом. Это означает, что блок в данном режиме недоступен (рисунок 1.8).

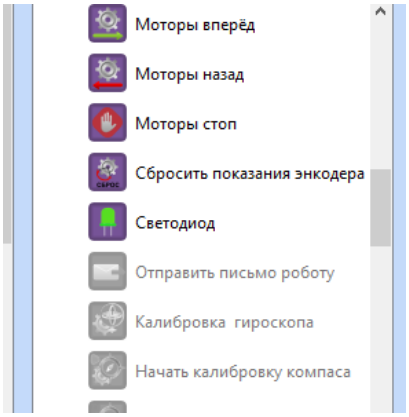


Рисунок 1.8 – Фрагмент палитры блоков.

В TRIK Studio реализовано несколько моделей роботов – Lego EV3, Lego NXT и TRIK. Выбор нужного робота осуществляется при помощи нажатия кнопки «Настройки» на панели инструментов или выбора пункта меню «Инструменты» – «Настройки». В открывшемся окне выбираем робота и

тип подключения. В нашем случае это Лего EV3. Тип модели робота: 2D модель (рисунок 1.9)

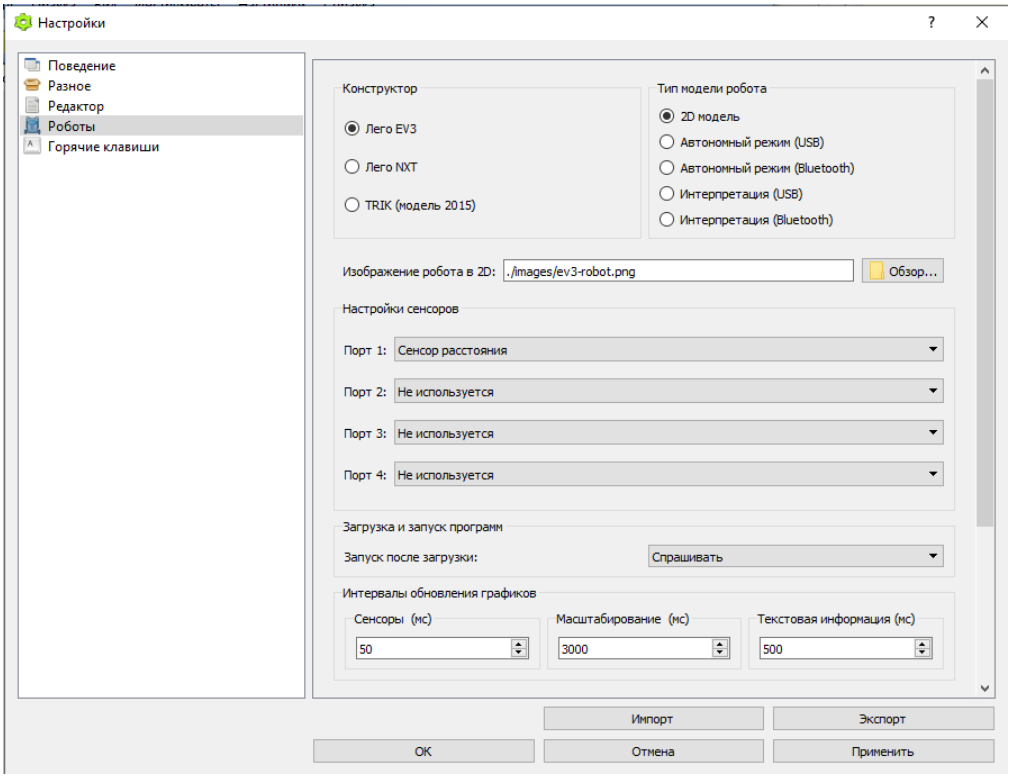


Рисунок 1.9 – Вкладка «Роботы» панели «Настройки».

Кроме выбора конструктора и типа модели здесь можно настроить:

Настройка сенсоров – здесь указывается, какие сенсоры подключены к портам робота.

Запуск после загрузки – запускать программу сразу после загрузки её на робота или нет.

Интервалы обновления графиков/Сенсоры – интервал в миллисекундах, когда добавляется новая точка с показанием сенсора.

Интервалы обновления графиков/Масштабирование – интервал в миллисекундах, когда график автоматически изменяет свой масштаб, чтобы оптимально отобразить значения.

Интервалы обновления графиков/Текстовая информация – интервал в миллисекундах, когда на графике обновляются надписи со значениями.

Рассмотрим другие вкладки панели.

Вкладка «Поведение» (рисунок 1.10).

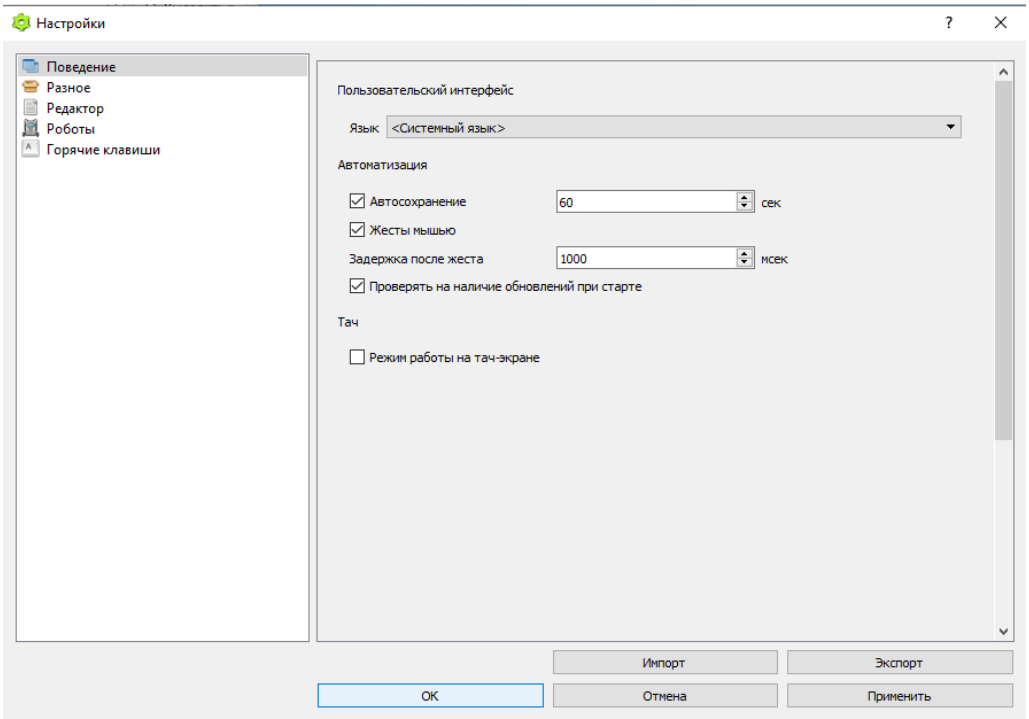


Рисунок 1.10 – Вкладка «Поведение» панели «Настройки».

- *Язык* – язык среды. После изменения требуется перезапуск. Настройка "<Системный язык>" означает, что язык будет определяться по текущему языку операционной системы.
- *Автосохранение* – возможность включить или выключить автосохранение, а также задать интервал автосохранения в секундах.
- *Жесты мышью* – включить или выключить механизм распознавания жестов мышью.
- *Задержка после жеста* – сколько система ждёт после выполнения мышиного жеста, прежде чем выполнит его распознавание. Задержка позволяет рисовать жест в несколько штрихов. Значение указывается в миллисекундах.
- *Проверять на наличие обновлений при старте* – связываться при запуске с сервером обновлений и запускать автообновление при необходимости.
- *Режим работы на тач-экране* – включить оптимизацию пользовательского интерфейса для сенсорных экранов.

Вкладка «Разное» (рисунок 1.11).

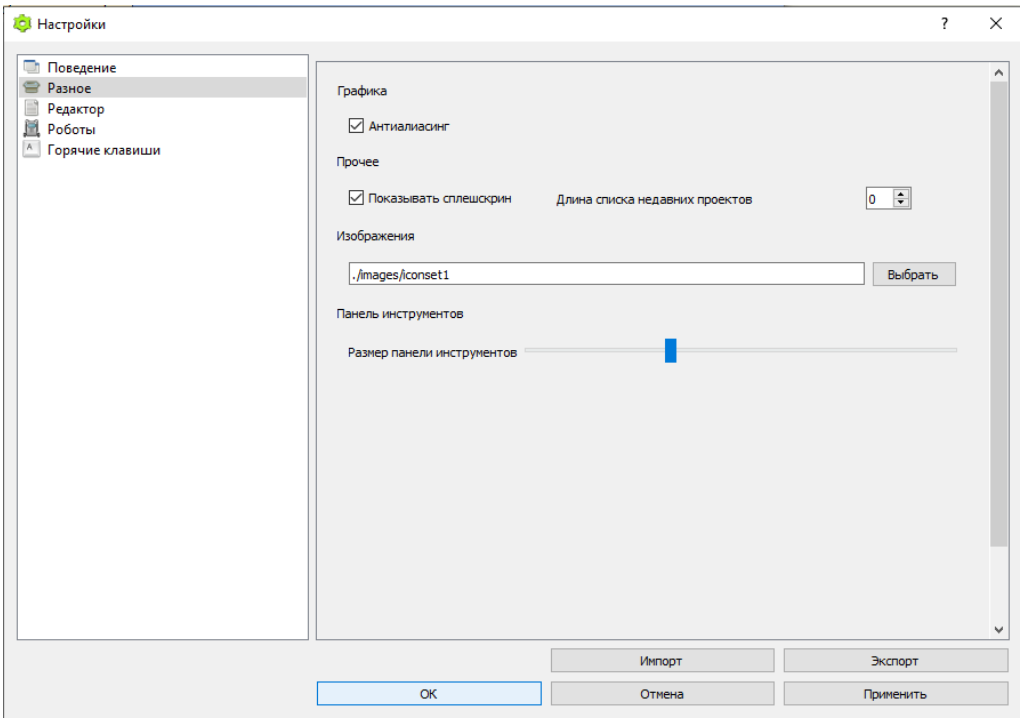


Рисунок 1.11 – Вкладка «Разное» панели «Настройки».

- *Графика/Антиалиасинг* – режим рисования линий со сглаживанием, улучшает внешний вид диаграмм за счёт незначительного снижения скорости работы системы.
- *Прочее/Показывать сплешскрин* – включить/выключить отображение стартового экрана с логотипом TRIK Studio.
- *Прочее/Длина списка недавних проектов* – сколько проектов показывать в пункте "Недавние проекты" меню "Файл".
- *Изображения* – какой набор иконок использовать для отображения диаграммы.
- *Панель инструментов/Размер панели инструментов* – задать размер кнопок на панели инструментов.

Вкладка «Редактор» (рисунок 1.12).

- *Шрифт/Использовать системный шрифт* – заменить шрифт для отображения надписей на диаграмме по умолчанию на один из шрифтов, установленных в системе. Размеры надписей на блоках жёстко заданы, поэтому изменение шрифта может привести к совмещению надписей друг на друга.

- *Сетка/Показывать сетку* – отображать/не отображать выравнивающую сетку на сцене редактора диаграмм.
- *Сетка/Выравнивание по сетке* – осуществлять/не осуществлять выравнивание блоков по сетке.
- *Сетка/Показать направляющие* – отображать/не отображать направляющие линии на сцене редактора диаграмм. Направляющие линии появляются, когда блок находится на одной горизонтали или вертикали с другим блоком, и помогают выравнивать блоки на сцене.
- *Сетка/Выравнивание по направляющим* – осуществлять/не осуществлять выравнивание блоков по направляющим.
- *Сетка/Толщина сетки* – толщина линий сетки. Настраивается в зависимости от яркости монитора или проектора.
- *Сетка/Размер ячейки* – размер одной ячейки сетки. Размер по умолчанию подобран так, чтобы блок накрывал четыре ячейки.
- *Элементы/Размер области масштабирования* – размер области, потянув за которую можно изменить размер блока "Комментарий".
- *Связи/Тип связей* – режим рисования связей на диаграмме.
  - *Ломаные линии* – связи рисуются как ломаные с точками излома, которые добавляются пользователем.
  - *Прямоугольные линии* – связи рисуются как ломаные линии, каждый сегмент которых параллелен осям координат. Точки излома в этом случае добавляются системой автоматически.
  - *Кривые Безье* – связи рисуются как гладкие кривые, кривизна которых может быть задана пользователем.
- *Связи/Отступ связей-петель* – насколько связь, входящая в тот же блок, из которого она исходит, должна отступать от блока.
- *Встроенные линкеры/Размер* – размер встроенного линкера, т.е. кружка рядом с блоком на диаграмме, осуществляющего создание связей между элементами.

- *Встроенные линкеры/Отступ* – отступ встроенного линкера от пиктограммы блока.
- *Надписи на элементах/Разрешить перемещать* – разрешить/запретить произвольно двигать по диаграмме надписи у блоков.
- *Надписи на элементах/Разрешить изменять размер* – разрешить/запретить менять размер области отображения для надписи.
- *Палитра/Представление* – выбор между режимом отображения иконок и названий или только иконок в палитре.
- *Количество иконок в строке* – количество иконок на строку палитры при выбранном режиме "Иконки".

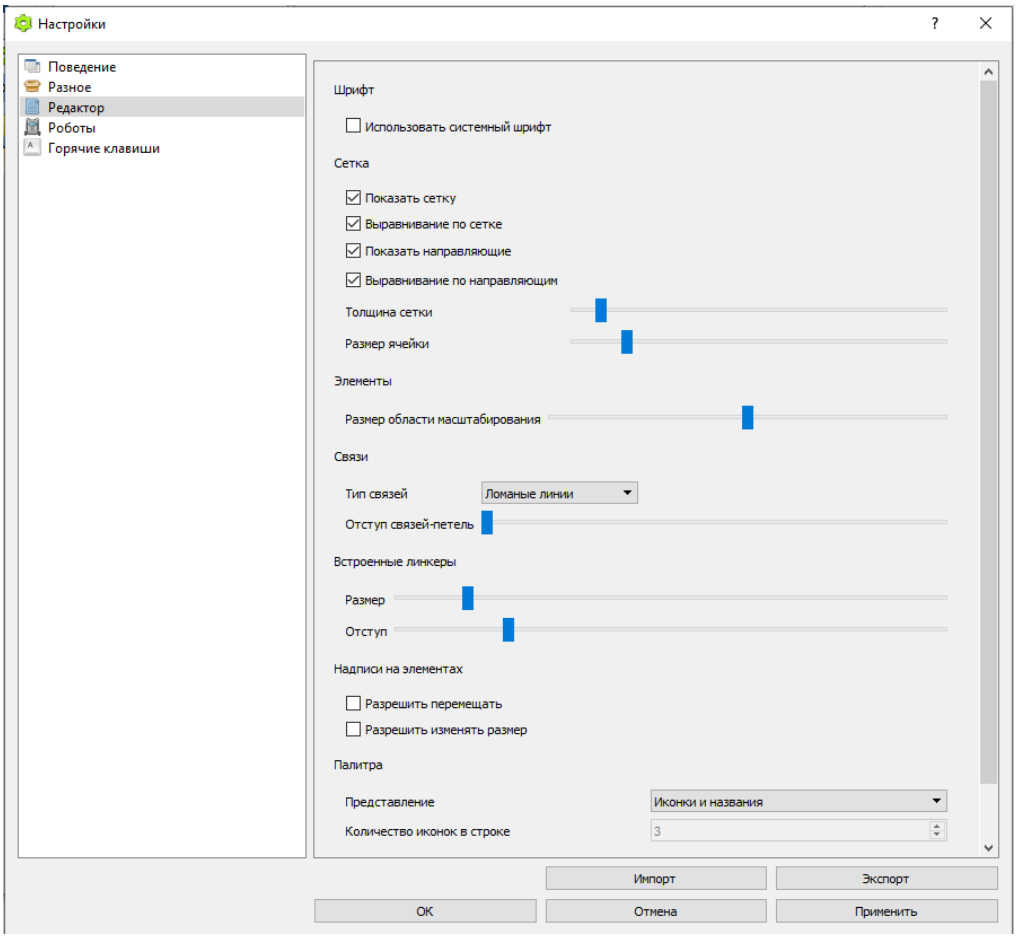


Рисунок 1.12 – Вкладка «Редактор» панели «Настройки».

На вкладке «Горячие клавиши» (рисунок 1.13) можно задать или изменить назначение клавиш для наиболее часто используемых действий. Для этого надо выбрать ячейку, соответствующую действию, и в строке "Сочетание" внизу окна нажать нужное сочетание клавиш. Кнопка "Очистить" удаляет сочетание клавиш из ячейки.

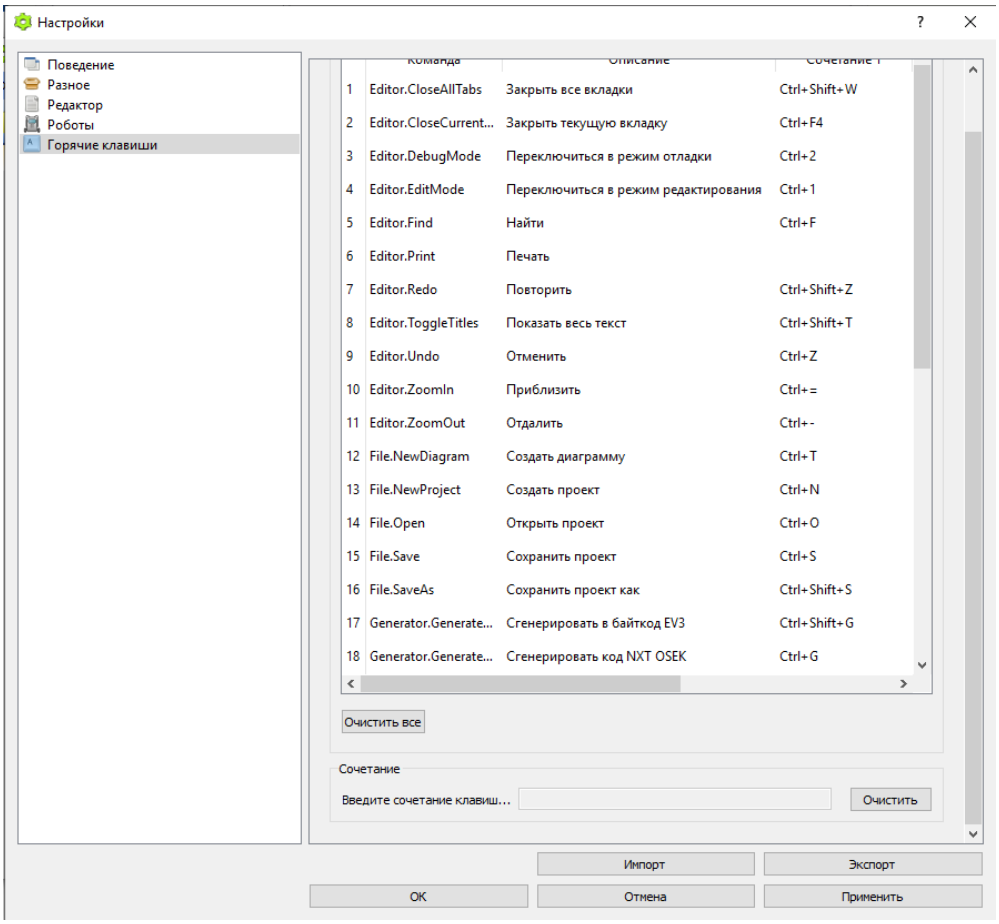


Рисунок 1.13 – Вкладка «Горячие клавиши» панели «Настройки».

Кнопки "Импорт" и "Экспорт" внизу окна позволяют сохранить текущие настройки в файл и загрузить его на другом компьютере.

2. Блоки LEGO EV3

Все блоки в языке TRIK Studio разделены на четыре группы.

Первая группа **«Алгоритмы»**, включающая блоки поддержки основных алгоритмических конструкций, такие как блоки начала и конца исполнения программы или подпрограммы, ветвления, организации арифметического цикла, выбора (switch), распараллеливания и работы с параллельными задачами (их слияния и принудительного завершения), вызова подпрограммы.

Вторая группа **«Действия»** – блоки работы с периферийными устройствами робота. Сюда включаются элементарные действия, не требующие ожидания. К примеру, это блоки подачи импульсов на моторы, проигрывания звука и т.д.

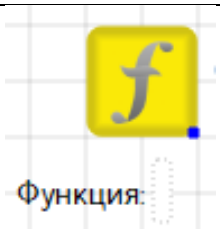
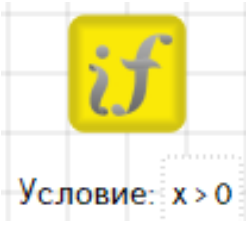
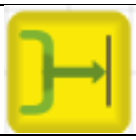
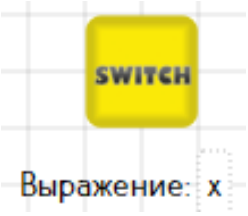
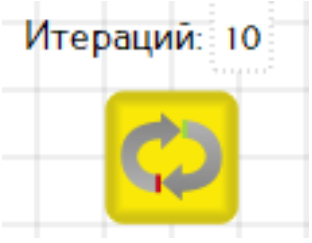
Следующая группа **«Ожидание»** включает в себя блоки, «замораживающие» исполнение текущего действия. Сюда входит блок ожидания заданного количества миллисекунд, блоки ожидания желаемого значения с какого-либо датчика.

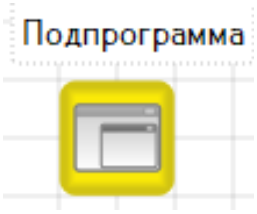
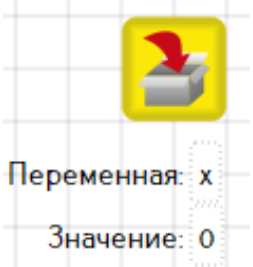
Наконец, четвертая группа **«Рисование»** включает блоки рисования графических примитивов на дисплее робота. К таким графическим примитивам относятся линии, прямоугольники, эллипсы, дуги, текст, картинки. В эту группу включены также специальные блоки управления маркером для рисования в двумерном симуляторе траектории перемещения робота, что позволяет решать в среде класс задач, предлагаемых различными популярными двумерными исполнителями, такими, как «Черепашка Logo» или «Чертежник».

Каждый блок имеет ряд свойств, которые могут быть отредактированы как на самой сцене редактора, так и на отдельной панели редактора свойств. Во всех свойствах блоков, где это уместно, имеется возможность задать вычислимое значение на встроенном в TRIK Studio текстовом языке.

В таблицах 1-4 представлено описание блоков каждой группы, доступных для робота Lego EV3.

Таблица 1 – Группа **«Алгоритмы».**

Название элемента	Пиктограмма	Описание
Начало		Начальная точка выполнения программы. На каждой диаграмме такой блок должен быть только один, в него не должно быть входящих связей, исходящая связь из него должна быть только одна. Любая программа начинается с данного блока.
Конец		Конец программы. Если программы состоит из нескольких параллельных участков выполнения, достижение этого блока завершит соответствующий участок выполнения. У данного блока не может быть исходящих связей.
Функция		Подсчитать значение заданного выражения. Также в этом блоке допускается задание переменных. Подробнее про синтаксис допустимых выражений смотрите в разделе «Синтаксис выражений», вызываемый справкой.
Условие		Разделить выполнение программы в соответствии с заданным условием. Значением параметра «Условие» является логическое выражение, принимающее либо истинное, либо ложное значение. При истинном значении условия выполнение программы передается на исходящую связь «истина», в противном случае – на связь «ложь».
Конец условия		Соединяет ветви блока «Условие».
Выбор		Выбирает ветку, по которой будет продолжено выполнение программы. Значение выражения, указанного в свойстве «Выражение», сравнивается со значениями на исходящих связях. Если среди них найдено равное, то исполнение будет продолжено по этой ветке. В противном случае будет выбрана ветка без маркера.
Цикл		Блок, организующий выполнение последовательности действий несколько раз. Количество повторений задается значением параметра «Итераций». Блок имеет две исходящие связи, одна из которых помечается «тело цикла» (значение параметра «Условие» должно быть «тело цикла»). Другая связь остается непомеченной: по ней осуществляется переход после выполнения тела цикла заданное количество раз.

Подпрограмма		Вызов подпрограммы. Подпрограммы используются для того, чтобы вынести повторяющиеся фрагменты программы на отдельную диаграмму, а затем вызвать данный фрагмент с этой диаграммы в нескольких местах основной программы или других подпрограмм. При добавлении данного блока на диаграмму будет предложено ввести имя подпрограммы, после чего двойным кликом на блоке можно перейти на диаграмму, соответствующей данной подпрограмме. Кроме того появится дополнительная палитра со всеми подпрограммами, которые можно перетаскивать на сцену и использовать как обычные блоки.
Параллельные задачи		Разделяет исполнение программы на несколько задач, которые могут исполняться параллельно с точки зрения программиста. Блок имеет как минимум две исходящие связи. Свойство «Условие» всех исходящих связей должно содержать уникальные идентификаторы задач, и один из идентификаторов должен совпадать с идентификатором задачи, из которой вызван этот блок (считается, что первый блок в программе вызывается из задачи с идентификатором «main»).
Слияние задач		Сливает несколько параллельных задач в одну. Свойство «Условие» единственной исходящей связи должно содержать идентификатор одной из сливаемых задач. Выполнение указанной задачи приостановится до того момента, как остальные задачи подойдут к этому блоку (закончат выполнение). Затем выполнение продолжится в обычном порядке.
Завершить задачу		Заканчивает выполнение указанной задачи.
Инициализация переменной		Присвоить значение данной переменной.

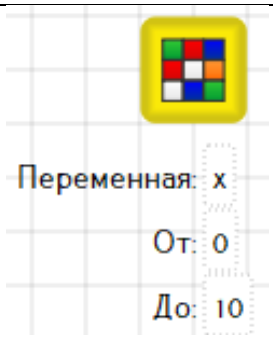
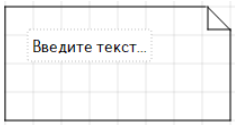
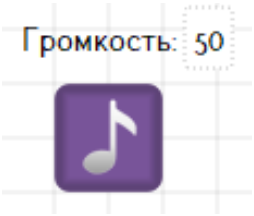
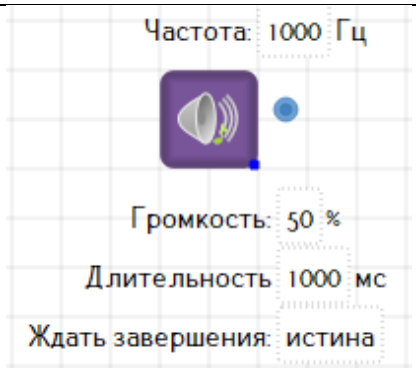
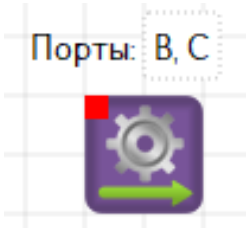
Случайное число		Присваивает переменной случайное число из указанного промежутка.
Комментарий		Блок с текстовыми заметками, которые игнорируются при генерации и интерпретации. Его необходимо использовать для повышения наглядности диаграммы.

Таблица 2 – Группа **«Действия»**

Название элемента	Пиктограмма	Описание
Отправить сообщение в задачу		Отправляет сообщение в указанную задачу.
Гудок		Проиграть на роботе звук с указанной частотой. Имеется два параметра. Первый – это громкость звука, второй означает ждать ли завершения проигрывания или сразу же перейти к следующему блоку. Допустимые значения – истина, ложь
Играть звук		Проиграть на роботе звук с заданной частотой и длительностью. Аналогичен блоку «Гудок», но позволяет задавать параметры звука. Имеется параметр, определяющий, ждать ли завершения проигрывания звука или сразу же перейти к следующему блоку.
Моторы вперед		Включить моторы по заданным портам с заданной мощностью. Порты задаются латинскими буквами A, B, C, D. Мощность задается в процентах от -100 до 100, если задано отрицательной значение, мотор включается в режиме реверса. Кроме того, моторы имеют разные режимы работы: режим торможения и режим скольжения (отображаемые красным или зелёным прямоугольником на блоке соответственно).

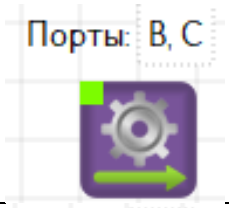
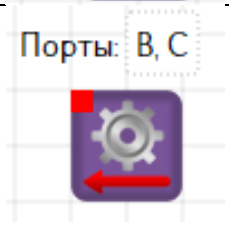
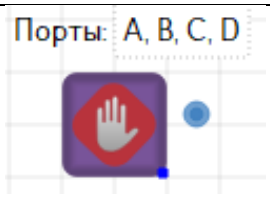
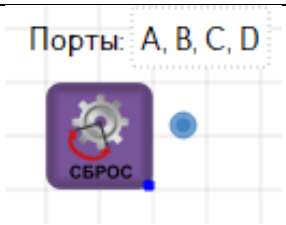
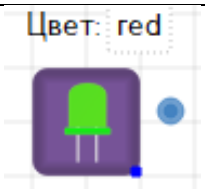
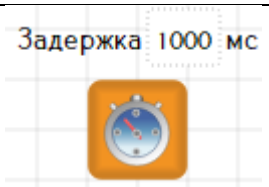
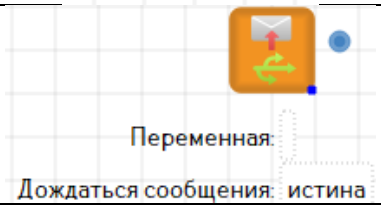
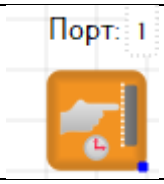
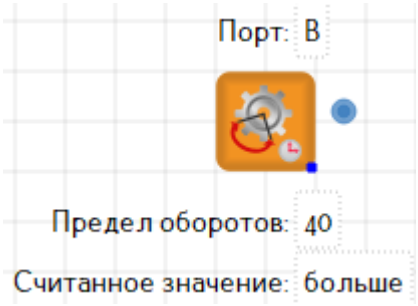
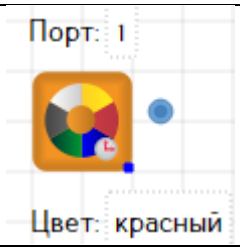
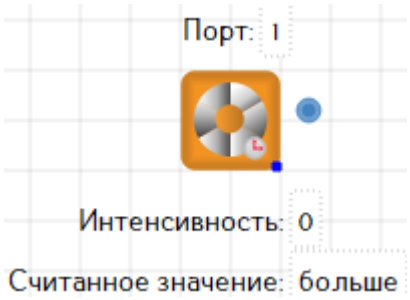
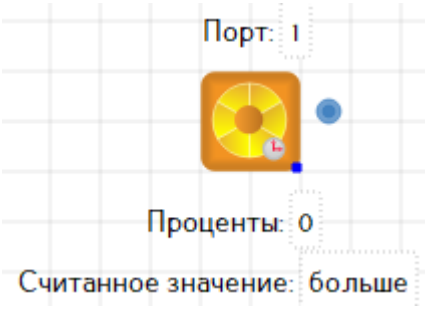
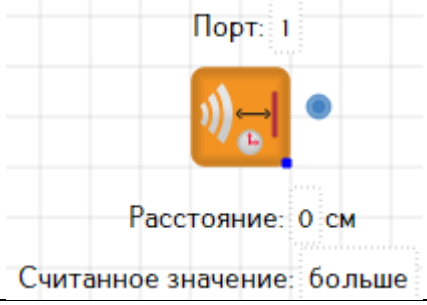
Моторы вперед		Режимы влияют на то, как двигатель отрабатывает команду – режим торможения стопорит двигатель при отключении, режим скольжения позволяет двигателю прокручиваться по инерции.
Моторы назад		Включить моторы в режиме реверса по заданным портам с заданной мощностью. Порты задаются латинскими буквами A, B, C, D. Мощность задается в процентах от -100 до 100, если задано отрицательной значение, мотор включается в обычном режиме.
Моторы стоп		Выключить моторы по заданным портам.
Сбросить показания энкодера		Сбросить показания количества оборотов по заданным портам в 0.
Светодиод		Установить цвет светодиода на передней панели контроллера.

Таблица 3 – Группа **«Ожидание»**

Название элемента	Пиктограмма	Описание
Таймер		Ждать заданное количество времени в миллисекундах. 1000 мс=1 с.
Получить сообщение из другой задачи		Получить сообщение, посланное другой задачей той задаче, из которой вызван этот блок.

Ждать датчик касания		Ждать, пока не сработает датчик касания. Параметром указывается номер порта, к которому подключен датчик. Допустимые значения: 1, 2, 3, 4.
Ждать энкодер		Ждать, пока показания счетчика количества оборотов на заданном моторе не достигнут указанного значения в параметре "Предел оборотов". Также указывается операция, которая будет использоваться для сравнения с введенным пределом оборотов. Так, при исполнении приведенного блока выполнение программы остановится до тех пор, пока значение, возвращаемое энкодером, не будет больше 40.
Ждать цвет		Ждать, пока датчик цвета в режиме распознавания цвета не вернет указанный цвет.
Ждать интенсивность цвета		Ждать, пока значение, возвращаемое датчиком цвета на указанном порту, не будет сравнимо с указанным значением параметра «Интенсивность» (задается в процентах). Еще один параметр – номер порта, к которому подключен датчик цвета. Также параметром указывается операция, которая будет использоваться для сравнения с введенной интенсивностью.
Ждать свет		Ждать, пока значение, возвращаемое датчиком света на указанном порту, не будет сравнимо с указанным значением параметра «Проценты». Еще один параметр – номер порта, к которому подключен датчик цвета. Также параметром указывается операция, которая будет использоваться для сравнения со значением параметра «Проценты».
Ждать сенсор расстояния		Ждать, пока расстояние, возвращаемое ультразвуковым или инфракрасным датчиком расстояния, не будет сравнимо с указанным значением параметра «Расстояние» (задается в сантиметрах, от 0 до 255). Еще один параметр – номер порта, к которому подключен датчик расстояния.

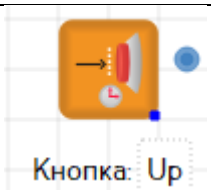
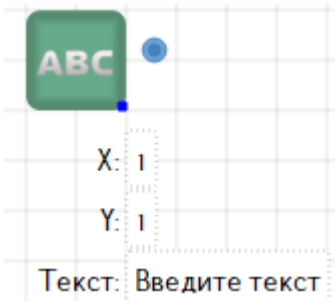
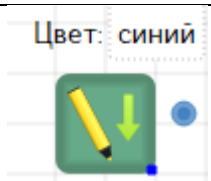
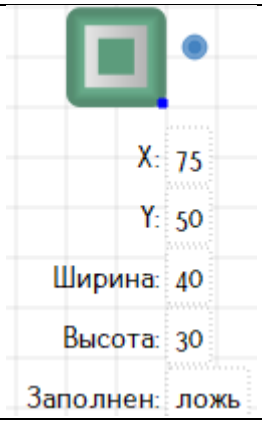
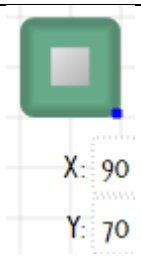
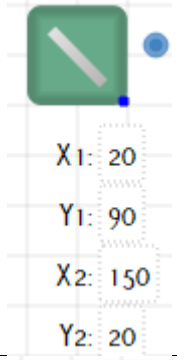
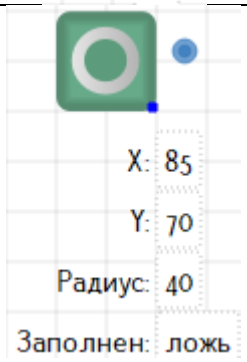
		Также параметром указывается операция, которая будет использоваться для сравнения с введенным расстоянием.
Ждать нажатия кнопки		Ждать, пока не будет нажата указанная в качестве параметра кнопка на корпусе робота.

Таблица 4 – Группа **«Рисование»**

Название элемента	Пиктограмма	Описание
Напечатать текст		Печатает заданную строку в заданном месте на экране робота. Значение свойства "Текст" по умолчанию трактуется как строка в чистом виде, оно так и будет выведено на экран. Чтобы система считала, что это выражение на текстовом языке (это может быть полезно, например, при отладке значения переменных), необходимо поставить галочку "Вычислять" в редакторе свойств.
Очистить экран		Стереть всё, что нарисовано на экране.
Опустить маркер		Опускает маркер робота в 2D модели на пол так, что тот начинает рисовать на полу свою траекторию. Если маркер другого цвета уже опущен, то он будет замещен.
Поднять маркер		Поднимает маркер робота в 2D модели, так что тот перестает рисовать свою траекторию на полу.
Нарисовать прямоугольник		Нарисовать на экране прямоугольник. В качестве параметров указываются координаты левого верхнего угла, ширина и высота прямоугольника, а также заливать его внутреннюю область или нет.

Нарисовать точку		Нарисовать на экране точку в указанных координатах.
Нарисовать линию		Нарисовать на экране отрезок. В качестве параметров блоку указываются концы отрезка.
Нарисовать круг		Нарисовать на экране круг с заданным центром и заданным радиусом, залитый внутри или нет.

Разработчики программы ТРИК Studio большое внимание уделяют usability – удобству использования. Для того чтобы каждый раз не искать нужный блок в палитре, достаточно изобразить в окне «Диаграмма поведения робота» определенный жест мышью, зажав правую кнопку. Ознакомиться со списком поддерживаемых жестов можно открыв меню «Инструменты» – «Жесты мышью».

3. Первая программа

Задача 3.1. Вывести на экран робота в 2D модели текст «Привет, робот!».

Необходимо написать программу и проверить её работоспособность в 2D модели. В ТРИК Studio существует одна виртуальная модель – это робот-тележка. Ниже представлена блок-схема для решения задачи (рисунок 3.1).

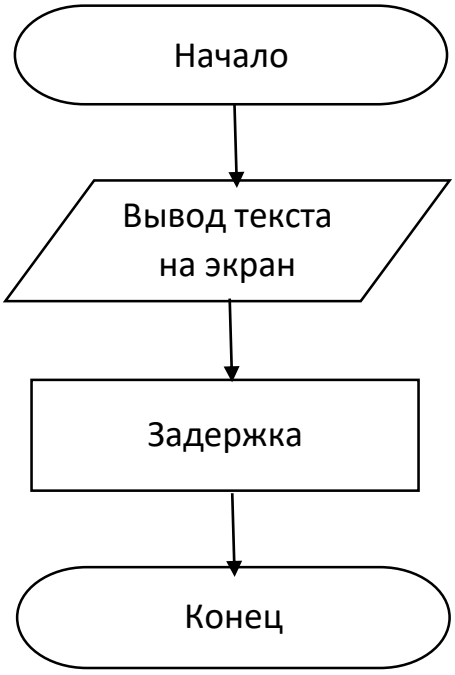


Рисунок 3.1 – Блок-схема алгоритма.

Алгоритм реализации программы

1) Переместите с палитры блоков рядом с блоком «Начало» блоки «Напечатать текст», «Таймер» и «Конец», как показано на рисунке 3.2.

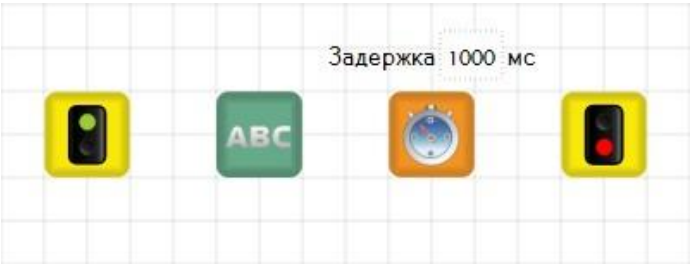


Рисунок 3.2 – Блоки диаграммы.

2) Соедините их последовательно.

Существует два способа соединения:

1. нажать на блок и потащить за маленький кружок к следующему блоку;

2. нажать на блок правой клавишей мыши, держа её, подвести к следующему блоку (рисунок 3.3).

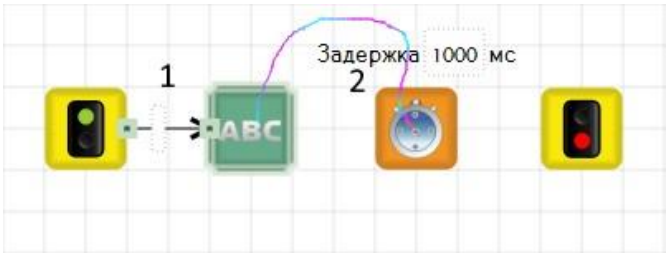


Рисунок 3.3 – Соединение блоков правой кнопкой мыши.

3) Выделите блок «Напечатать текст». У этого блока три свойства: две координаты начала текста и сам текст (рисунок 3.4). Введите текст: «Привет, робот!»

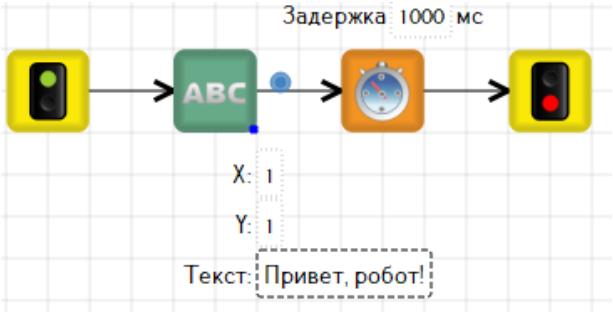


Рисунок 3.4 – Пример ввода текста в блоке «Напечатать текст».

Некоторые свойства отображаются над или под блоком. Редактировать их можно как на самом блоке, так и на панели **«Редактор свойств»** (рисунок 3.5).

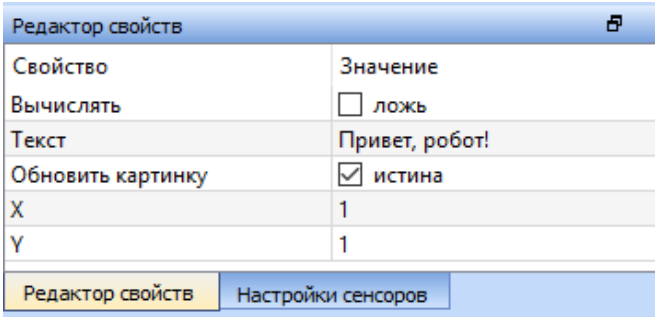


Рисунок 3.5 – Редактор свойств.

При выставленной галочке в окне редактора свойств **«Вычислять»** текст необходимо набирать в кавычках. У блока «Таймер» одно свойство – задержка в миллисекундах. Задайте ему значение 3000 мс, что составляет 3 секунды. Окончательно программа будет выглядеть так, как представлено на рисунке 3.6.

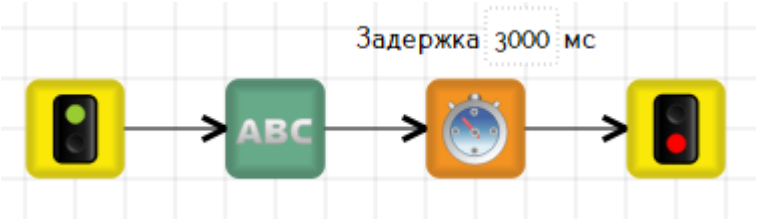


Рисунок 3.6 – Программа вывода на экран текста «Привет, робот!».

4) Проверьте выполнение программы в 2D модели. Прежде необходимо убедится, что вы находитесь в данном режиме, при этом должна быть выделена кнопка «2D» (рисунок 3.7).

Рисунок 3.7 – Выбор 2D режима на панели инструментов.

Перейдите в режим отладки, нажав на иконку, или сочетанием клавиш Ctrl+2 и запустите программу, нажав на кнопку «Play». На дисплее в верхнем левом углу отобразится введенный в программе текст (рисунок 3.8).

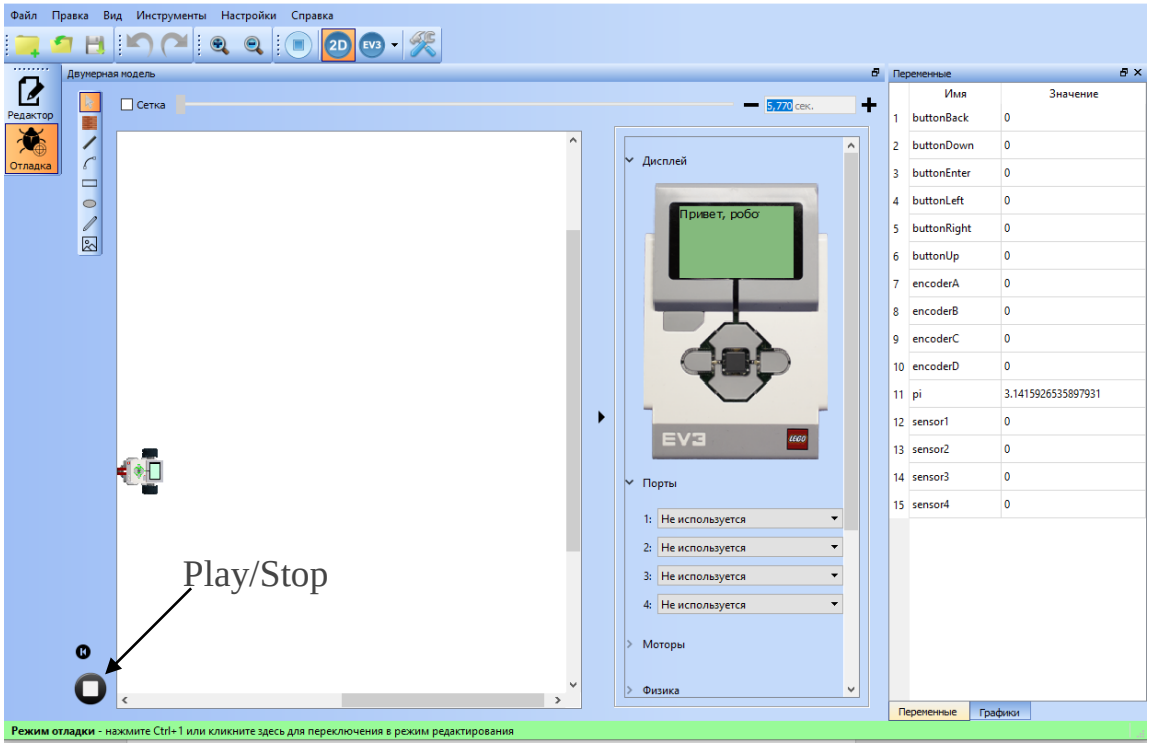


Рисунок 3.8 – Запуск программы в режиме «Отладка».

Если дисплей скрыт, то его надо открыть, нажав на соответствующую стрелку.

Сохраним программу. Для этого нужно выбрать в меню «Файл» вкладку «Сохранить» или «Сохранить как» или воспользоваться соответствующей пиктограммой на панели инструментов. Далее нужно определить папку, где будет храниться проект, в поле «имя файла» ввести название и нажать кнопку «Сохранить». Закроем программу TRIK Studio.

4. Загрузка программы на робота и ее запуск

Задача 4.1. Вывести на экран контроллера EV3: «Привет, робот!» так, чтобы текст располагался в центре экрана.

Для открытия сохраненной ранее программы, нужно воспользоваться командой «Открыть» из меню «Файл» или воспользоваться соответствующей пиктограммой на панели инструментов.

Перед загрузкой программы на контроллер робота необходимо внести в существующую программу некоторые коррективы.

1) Для корректного вывода информации на экране блока текст необходимо ввести с помощью латинской раскладки клавиатуры. Это связано с тем, что на контроллер Lego EV3 установлена операционная система Linux, не поддерживающая русский язык. Например, можно ввести следующий текст «Hello, robot!».

2) Текст должен располагаться в центре экрана, поэтому нужно внести изменения в координаты начало ввода текста. Размеры экрана контроллера Lego EV3 составляют 177 пикселей по горизонтали и 127 - по вертикали, отсчет начинается в левом верхнем углу. Например, можно установить следующие координаты: x – 30, y – 50.

3) Для загрузки робота в контроллер программа должна иметь название, написанное на латинской раскладке клавиатуры.

Загрузим программу на робота. Загрузка программы на робота позволяет сохранить программу на роботе и исполнять её без соединения с компьютером. В отличие от Lego NXT, средства загрузки и исполнения программ TRIK Studio не требуют дополнительных манипуляций с роботом и работают со стандартной прошивкой. Первое, что нужно сделать, чтобы загрузить программу – перейти в режим «Редактор», нажав на соответствующую иконку, или сочетанием клавиш Ctrl+1 и переключиться в режим реального робота EV3 как показано на рисунке 4.1.

Рисунок 4.1 – Панель переключения в режим реального робота.

Далее в зависимости от желаемого способа загрузки необходимо выбрать тип соединения с роботом – по Bluetooth или USB (рисунок 4.2).

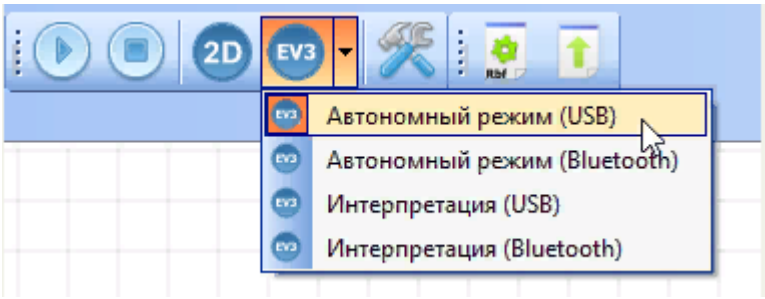


Рисунок 4.2 – Способы загрузки программы на робота.

Осталось нажать кнопку «Play». Ваша программа отобразится на дисплее самого робота. По программе будет сгенерирован код на внутреннем языке EV3, загружен на робота и немедленно запущен на исполнение (рисунок 4.3).

Рисунок 4.3 – Запуск программы на исполнение.

Если требуется только загрузить программу, не запуская её, нужно воспользоваться кнопкой "Загрузить программу" (рисунок 4.4).

Рисунок 4.4 – Загрузка программы на робота.

Эксперты в EV3 могут посмотреть на сгенерированный на внутреннем языке EV3 код с помощью кнопки "Сгенерировать в байткод EV3" (рисунок 4.5). Теоретически, его можно отредактировать и запустить на исполнение, или вообще

использовать для текстового программирования EV3, но делать это неудобно – язык создавался для автоматической генерации, а не написания программ.

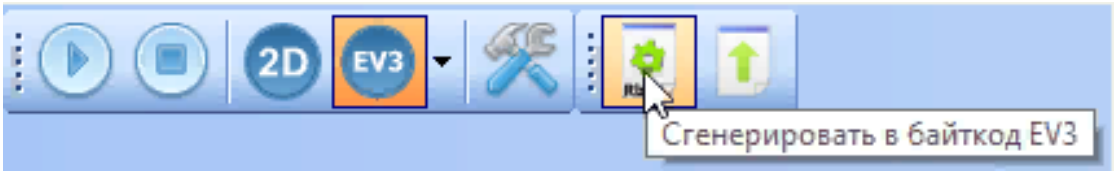
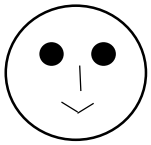
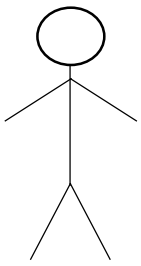


Рисунок 4.5 – Кнопка просмотра байткода EV3

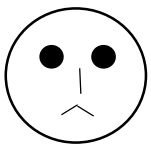
Упражнения для самостоятельного решения

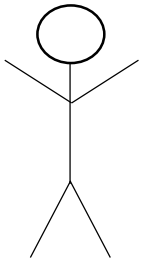
1. Поочерёдно в течение 5-ти секунд вывести на дисплей робота следующие фигуры: круг, квадрат, точку, линию.
2. Изобразите на дисплее следующие рисунки и выведете под ними поясняющий текст:

a. 

b. 

3. Напишите программу, которая меняет рисунки из задачи 1 на следующие:

a. 

b. 

Рекомендации: выполнение программ следует проверить как на модели 2D, так и на реальном роботе.

5. Движение робота по времени

Задача 5.1. Заставить робота двигаться вперед в течении 1 секунды с максимальной мощностью.

Движение осуществляется при помощи моторов. Переместите с палитры блоков рядом с блоком «Начало» блоки «Моторы вперед», «Таймер» и «Конец», соединив их последовательно, как показано на рисунке 5.1.

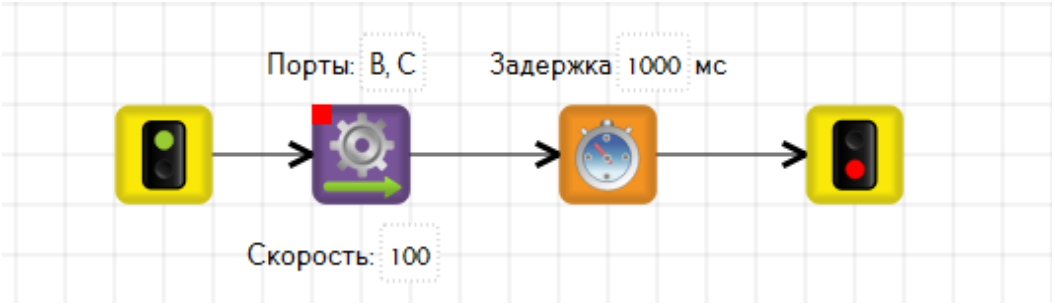


Рисунок 5.1 – Программа движения робота вперед.

Рассмотрим программу подробнее.

Блок «Моторы вперед» в «Редакторе свойств» (рисунок 5.2) содержит следующие записи. У контроллера EV3 имеются четыре разъема для подключения моторов – A, B, C, D. По умолчанию программа использует порты B и C. Это означает, что моторы на роботе также должны быть подключены к данным портам. При необходимости порты можно поменять, щелкнув мышкой по записи «B, C» и вручную прописать нужные. Внести изменения можно как в «Редакторе свойств», так и непосредственно в блоке. Строка «режим» определяет способ торможения, строка «скорость» – мощность моторов (см. раздел 2).

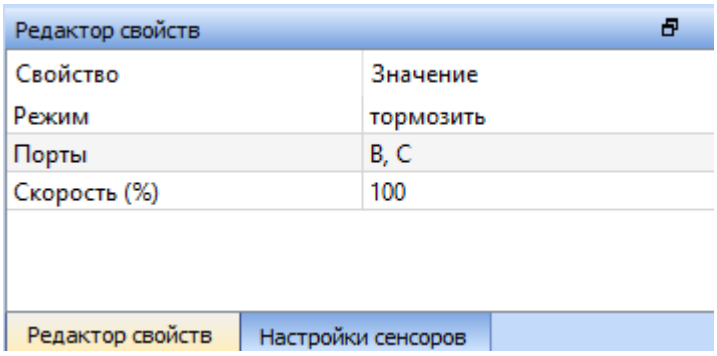


Рисунок 5.2 – Свойства блока «Моторы вперед».

При запуске в 2D модели в режиме «Отладка» необходимо проверить, к каким портам подключены моторы, в нашем случае это порты B и C. В случае необходимости, порты можно поменять, выбрав из списка нужные (рисунок 5.3).

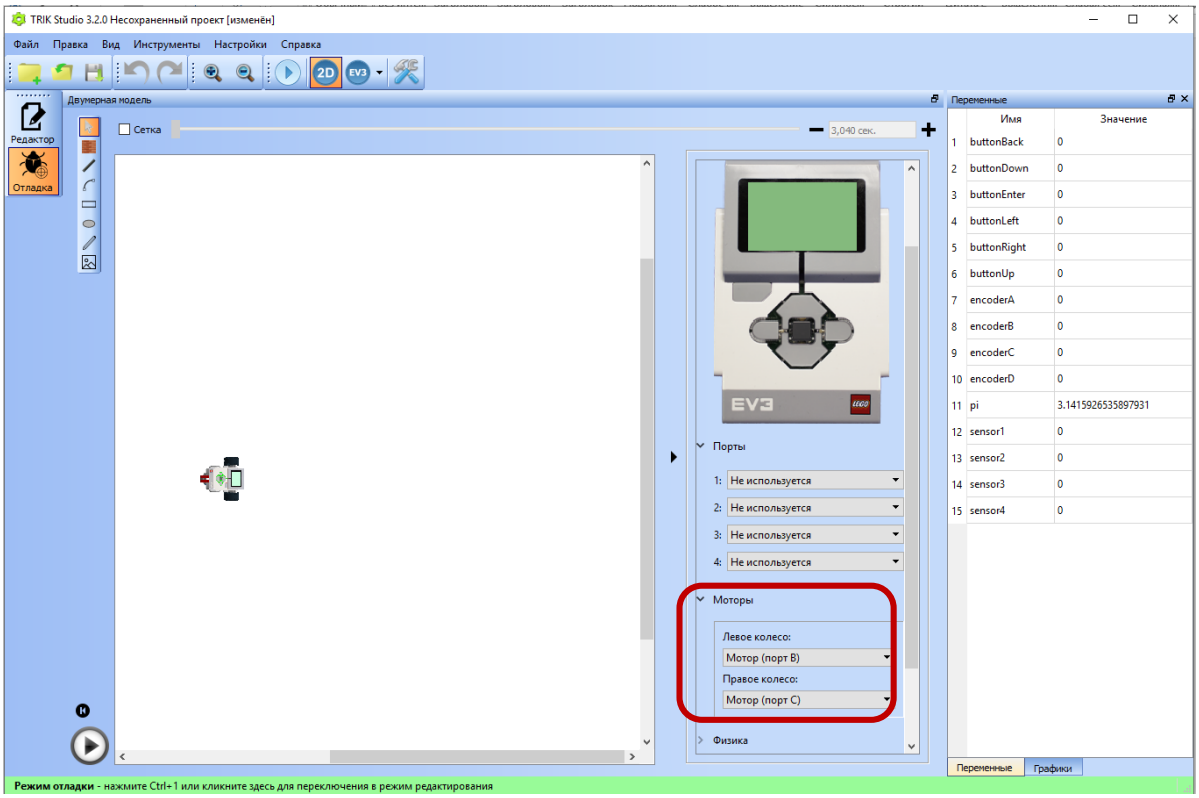


Рисунок 5.3 – 2D модель в режиме «Отладка»: выбор моторов.

Задача 5.2. Заставить робота поворачиваться влево (резкий поворот) в течении 1 секунды с максимальной мощностью.

Чтобы робот повернулся налево, нужно чтобы левое колесо (порт B) не вращалось, а правое (порт C) наоборот – вращалось. Для этого необходимо в блоке «Моторы вперед» убрать порт B. Программа будет выглядеть как показано на рисунке 5.4.

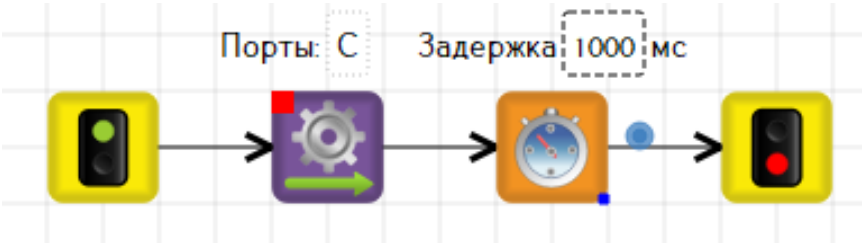


Рисунок 5.4 – Программа поворота робота налево.

Для поворота направо необходимо будет убрать порт C. Для поворота вокруг своей оси (или танковый разворот), мотор B должен крутиться в одну

сторону, а мотор С в противоположную, причем с одинаковыми мощностями (рисунок 5.5).

Рисунок 5.5 – Программа поворота робота налево вокруг своей оси.

Задача 5.3. Задать движение роботу: 2 сек двигается прямо и в 2 раза больше выполняет плавный поворот.

Для движения вперед на сцену добавляем блоки «Моторы вперед», «Таймер», выставляем необходимые параметры. Для остановки моторов добавляем блок «Моторы стоп». Для плавного поворота добавляем два блока «Моторы вперед» и «Таймер». У первого блока «Моторы вперед» оставляем порт С, у второго – порт В и устанавливаем разные мощности, например, 100 и 50. Выставляем необходимое время (рисунок 5.6).

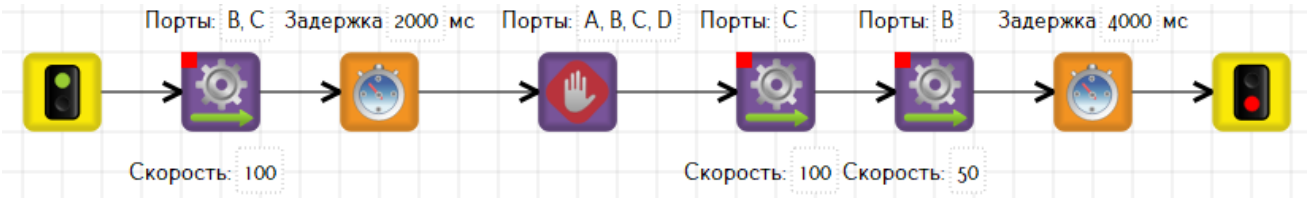


Рисунок 5.6 – Программа к задаче 5.3.

Чтобы увидеть траекторию движения, можно добавить после блока «Начало» блок «Опустить маркер» (только для двумерной модели). Цвет траектории указывается в свойствах блока (рисунок 5.7).

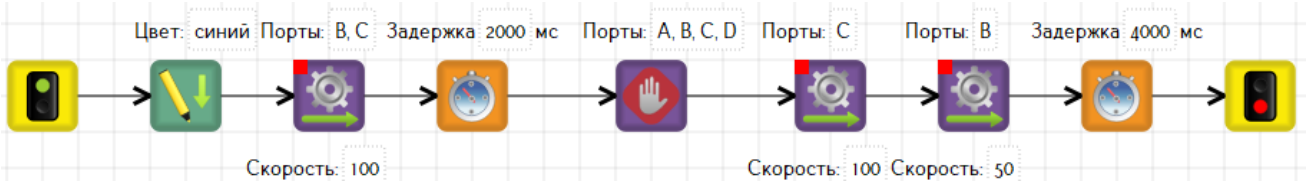
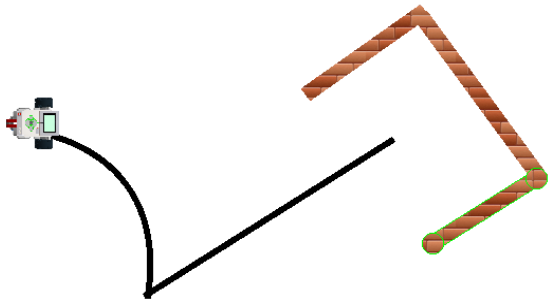


Рисунок 5.7 – Программа к задаче 5.3 с добавлением блока «Опустить маркер».

Упражнения для самостоятельного решения

1. Напишите программу движения робота вперед-назад.
2. Определить, за сколько секунд выполняется поворот на 360 градусов обычный и «танковый».
3. Как будет выглядеть алгоритм «танкового разворота» на 180°?
4. Сколько времени нужно для того, чтобы описать полную окружность при следующих мощностях моторов: B –100%, C – 50%?
5. Измените мощности моторов таким образом, чтобы траектория движения по окружности была большего/меньшего диаметра.
6. Составьте алгоритм движения по восьмерке с опущенным маркером в режиме «2D».
7. Составьте в режиме «2D» алгоритм движения по траектории «цветок» с четырьмя лепестками с опущенным маркером с использованием разных цветов.
8. Робот отправляется на базу по заранее прочерченному маршруту. Траекторию нарисовать в окне «2D», используя кривую Безье и линию, а базу с помощью инструмента «Стена»:

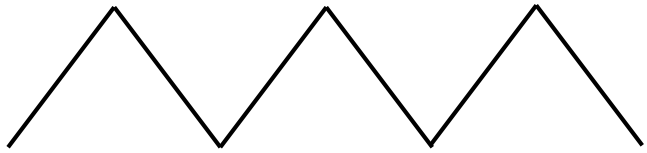


9. Написать программу для движения по заданным траекториям**:**

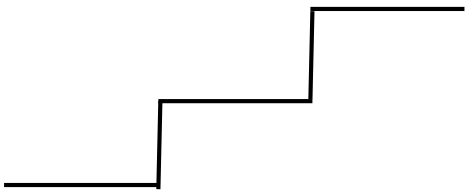
a) 

b) 

c)



d)



10. Напишите на сцене при помощи маркера свое имя.

11. Составьте программу: робот сначала рисует, а затем стирает круг.

Рекомендации: выполнение программ следует проверить как на модели 2D, так и на реальном роботе.

6. Обратная связь

Часто для проверки и отладки программ требуется, чтобы робот сам сообщал о выполнении команд. Такая связь с исполнителем называется обратной. В качестве обратной связи для контроллера EV3 можно использовать блоки «Светодиод», «Гудок», «Играть звук», «Напечатать текст» и управление нажатием кнопок. Рассмотрим все способы связи.

Задача 6.1. Робот ожидает 1 сек, затем меняет цвет свечения кнопок на красный и едет вперед.

При запуске программы на контроллере EV3 по умолчанию загорается зеленая подсветка. Поэтому при запуске программы на определенных участках программы можно использовать кроме зеленой подсветки красную и оранжевую. В свойствах блока «Светодиод» по каждому цвету можно дополнительно выбрать либо непрерывное свечение, либо мигающее.

Итак, к блоку «Начало» на сцену добавим блоки «Таймер», «Светодиод», «Моторы вперед» и «Конец». Выставим необходимы параметры в «Редакторе свойств» и получим программу, изображенную на рисунке 6.1.

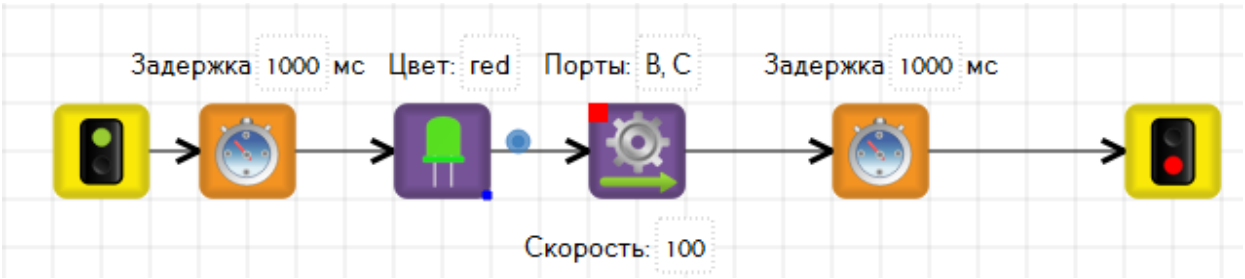


Рисунок 6.1 – Программа к задаче 6.1.

Вместо блока «Светодиод» можно использовать блок «Гудок».

Задача 6.2. Робот ожидает 2 сек, затем издает гудок и поворачивает направо.

Удалим блок «Светодиод» и вместо него добавим блок «Гудок», установив необходимые параметры в редакторе свойств (см. раздел 2). При удалении блока из диаграммы могут появиться связи красного цвета как показано на рисунке 6.2. Это означает, что блоки не соединены. Для соединения блоков достаточно щелкнуть мышью по одному из концов связей.

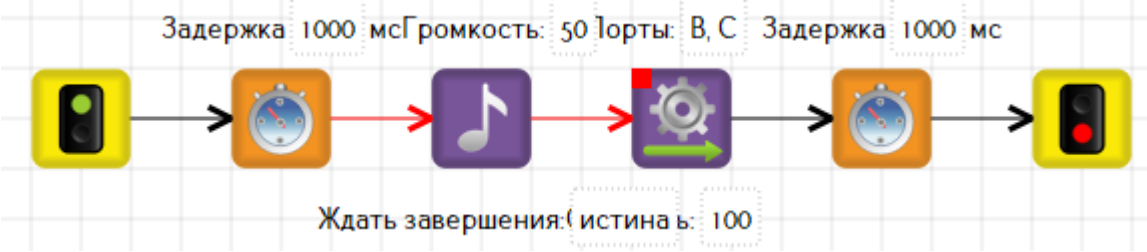


Рисунок 6.2 – Отсутствие связей на диаграмме.

Далее в блоке «Моторы вперед» удалим порт С для поворота направо и внесем изменения в блоке «Таймер». Получим программу как на рисунке 6.3:

Рисунок 6.3 – Программа к задаче 7.2.

Вместо блока «Гудок» можно использовать «Играть звук». Отличие между блоками смотрите в разделе 2.

Вывод текста на экран с помощью блока «Напечатать текст» был описан в разделе 3.

Рассмотрим пример управления кнопками на контроллере.

Задача 6.3. Составить программу вывода на экран текста «влево» при нажатии соответствующей кнопки на контроллере.

Переместим на сцену блоки «Ждать нажатия кнопки», «Очистить экран», «Напечатать текст», «Таймер», «Конец». В «Редакторе свойств» блока «Ждать нажатия кнопки» выбрать значение «Left», а в блоке «Напечатать текст» ввести соответствующий текст (обратите внимание на раскладку клавиатуры). Соединим последовательно блоки. Получившееся программа представлена на рисунке 6.4. Запустим программу на выполнение. Текст на экране не появиться до тех пор, пока не будет нажата кнопка «влево» на контроллере, после нажатия этой кнопки на экране в течении 2-х секунд будет отображено слово «left».

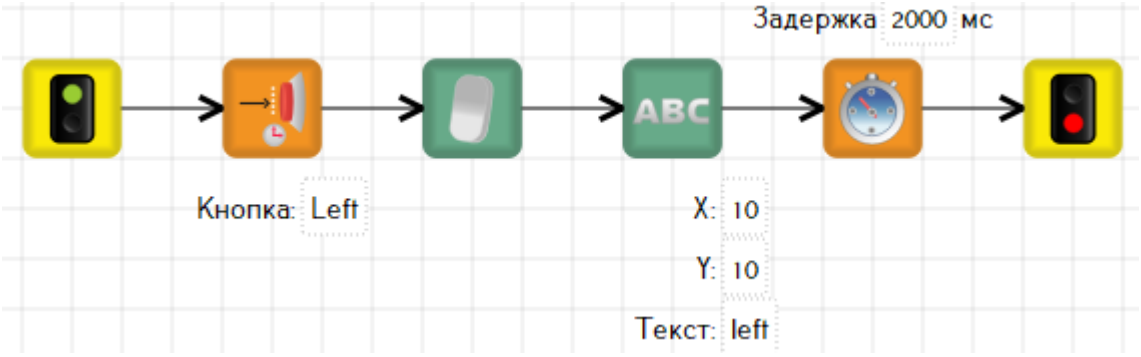
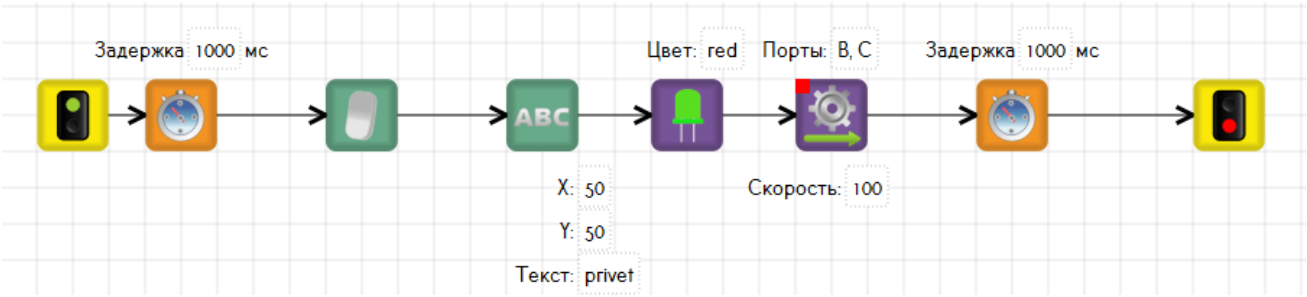


Рисунок 6.4 – Программа к задаче 7.3.

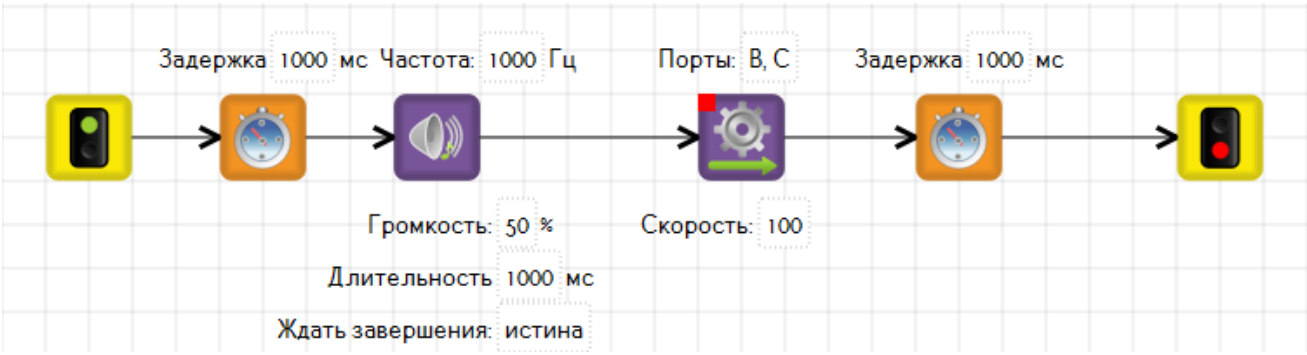
Упражнения для самостоятельного решения:

1. Определить, что выполняют программы:

a)



b)



2. Включать на блоке последовательно каждые 3 секунды подсветку: зеленую, красную мигающую, оранжевую, выключить подсветку.

3. Составить программу, имитирующую работу светофора: изначально на роботе горит красный цвет и отображается рисунок «Грустный смайлик», затем цвет меняется сначала на оранжевый, а затем на зеленый и робот начинает

движение в течении некоторого времени, при этом на экране отображается рисунок «Улыбающийся смайлик».

4. Составить программу вывода на экран информации о нажатых кнопках: вверх, влево, вниз, вправо.

5. Составить программу движения робота по следующей траектории: робот сначала движется прямо пока не будет нажата кнопка «Влево». При ее нажатии робот поворачивает налево, затем снова движется прямо, пока не будет нажата кнопка «Вправо», далее робот поворачивает направо и продолжает движение прямо пока не будет нажата центральная кнопки на контроллере. После этого робот останавливается.

Рекомендации: выполнение программ следует проверить как на модели 2D, так и на реальном роботе.

7. Движение робота по энкодеру

В предыдущих разделах движение робота было организовано по таймеру. Данный подход называется Time-моделью и является несовершенным, так как в этом случае выполняемое действие зависит от заряда аккумулятора на роботе.

Правильно будет использовать ожидание значения энкодеров, то есть энкодерную моделю.

Энкодер (датчик угла поворота) – это устройство, предназначенное для преобразования угла поворота вращающегося объекта (вала) в электрические сигналы, позволяющие определить угол его поворота.

В конструкторах Lego EV3 используются моторы со встроенным датчиком вращения (энкодером), который используется для измерения количества оборотов, совершенных мотором. Датчики в этих моторах определяют количество оборотов в градусах. Полный оборот мотора составляет 360 градусов. Поэтому всегда можно узнать, сколько оборотов совершил мотор, либо самостоятельно задать нужное количество оборотов.

Вращение мотора вперед выражается положительным числом градусов или отрицательным числом при вращении назад. Вращение всегда измеряется как общее число оборотов вперед с момента последнего сброса показаний датчика. Число оборотов назад вычитается из любого накопленного числа оборотов вперед.

Перед началом любого действия показания энкодера необходимо сбрасывать на ноль, чтобы датчик начал измерять общее количество вращений относительно точки сброса. Если показания датчика не сбрасывать, то получается общее количество оборотов, которое мотор совершит с начала программы и выполнять указанные действия будет неверно.

Задача 7.1. Проехать 20 см назад и остановиться с накатом. Составить программу с использованием энкодерной модели.

Выносим на сцену блоки «Сброс показаний энкодеров», блоки моторов, вместо блока «Таймер» перемещаем блок «Ждать энкодер». У блока «Ждать энкодер» есть несколько свойств (рисунок 7.1):

- порт, по которому считать значения,

- считанное значение энкодоров (больше, меньше, не больше, не меньше),
- предел оборотов (то значение, на сколько должен повернутся вал двигателя).

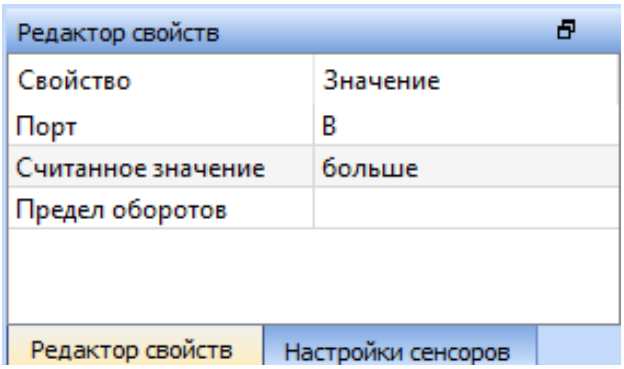


Рисунок 7.1 – Редактор свойств блока «Ждать энкодер»

Движение назад можно организовать двумя способами. Первый – мощность в блоке «Моторы вперед» необходимо установить с отрицательным значением, второй – использовать «Моторы назад» с установкой положительного значения мощности.

Рассчитаем, на сколько градусов должен повернуться мотор, чтобы проехать 20 см.

Один полный оборот колеса составляет 360 градусов. Рассчитаем расстояние, которое проходит робот при повороте оси на один оборот, то есть определим длину окружности колеса, которая вычисляется по формуле:

$$L = 2 \cdot \pi \cdot R = \pi \cdot D,$$

где R – радиус окружности, D – диаметр окружности, π – постоянная величина, равная 3,14.

Определим диаметр колеса. Диаметр колеса можно измерить или посмотреть маркировку на шине. Диаметр шины из базового набора Lego Mindstorms EV3 составляет 56 мм.

Поэтому за один оборот колеса робот проезжает расстояние, равное:

$$L = 3{,}14 \cdot 56 = 175{,}84 \text{ мм} = 17{,}584 \text{ см}.$$

Так как, роботу нужно проехать 20 см, найдем сколько нужно совершить оборотов валом двигателя (N):

$$N = 20 \div 17{,}584 = 1{,}137 \text{ (оборота)}.$$

Итак, определяем на сколько градусов должен повернуться мотор, чтобы проехать 20 см:

$$1{,}137 \cdot 360 = 409^{o}.$$

Поэтому в свойстве «Предел оборотов» выставляем значение 409. Но так как робот должен двигаться назад, потому ставим -409 и считанное значение – «меньше», а в редакторе свойств выберем режим скользить (рисунок 8.2).

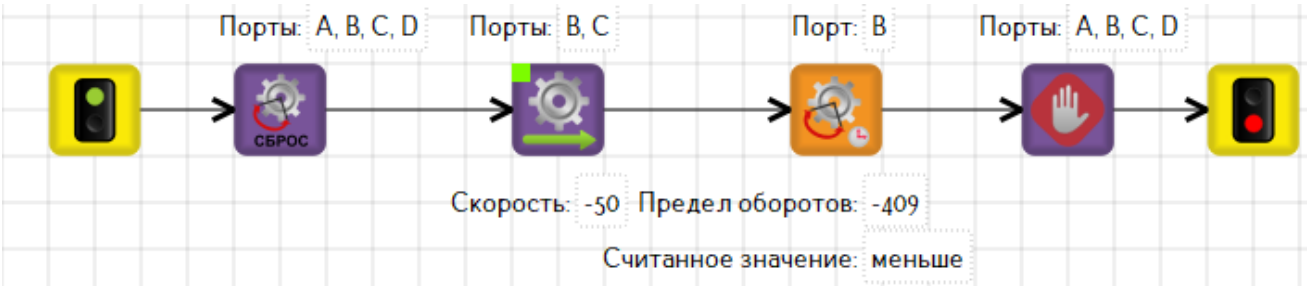


Рисунок 7.2 – Программа к задаче 7.1.

Запустим программу в 2D модели и обратим внимание на панель переменных. В Lego EV3 есть зарезервированные переменные (подробнее см. раздел 14). Для энкодеров – это переменные e*ncoderA, encoderB, encoderC, encoderD* (рисунок 7.3). После исполнения программы в строке *encoderB* видим значение -412. Таким образом, как только значение энкодера *B* стало меньше -409 программа завершилась.

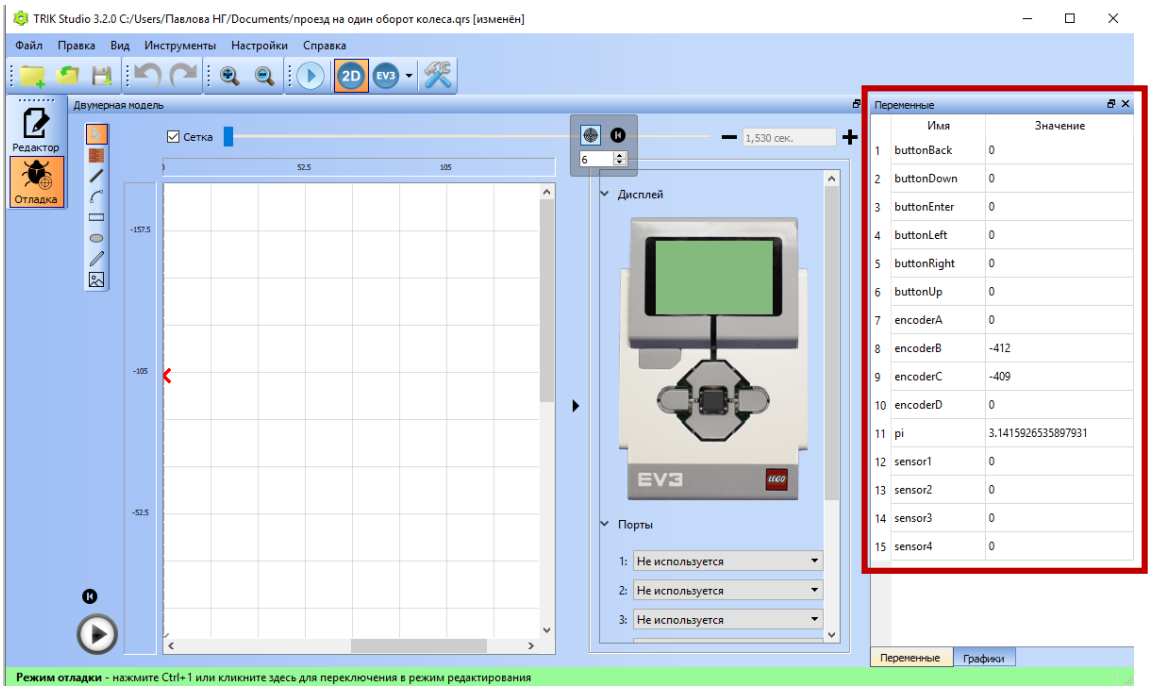


Рисунок 7.3 – Панель «Переменные» в 2D модели.

Загрузим программу в контроллер EV3 и запустим на выполнение. Робот при этом может проехать расстояние больше, чем на 20 см. Таким образом, поведение робота в 2D модели и в реальности различно и это необходимо учитывать при составлении программы для реального робота. Такая же проблема существует и для Time-модели.

Задача 7.2. Вычислить предел оборотов для резкого поворота на угол в 90°. Для этого понадобится расстояние между центрами колес (*R*). В базовой тележке Lego EV3 оно составляет 12 см. Для поворота налево на угол 90°, левое колесо должно оставаться на месте, а правое проехать некоторое расстояние (*P*) (рисунок 8.4), равное:

$$P = \frac{2 \cdot \pi \cdot R}{360 \div X} = \frac{2 \cdot 3{,}14 \cdot 12}{360 \div 90} = \frac{75{,}39}{4} = 18{,}84(см)$$

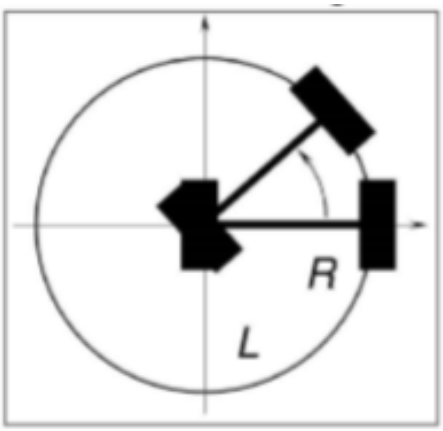


Рисунок 8.4 – Поворот на угол X на левом колесе.

Как была рассчитано выше, за один оборот колесо проезжает 17,58 см, поэтому количество оборотов получаем

$$N = \frac{18{,}84}{17{,}58} = 1{,}07(оборот)$$

Так как за один оборот колеса составляет 360°, получим 385° (1,07·360). Таким образом, значением свойства «предел оборотов» в программе будет являться данное число.

Задача 7.3. Вычислить предел оборотов для поворота вокруг своей оси на угол в 45°.

Для порота по часовой стрелке на угол X левое колесо должно повернуться вперед и проехать расстояние, равное:

$$P = \frac{2 \cdot \pi \cdot \frac{R}{2}}{360 \div X} = \frac{2 \cdot 3{,}14 \cdot \frac{12}{2}}{360 \div 45} = \frac{37{,}68}{8} = 4{,}71(см),$$

При этом правое колесо должно проехать то же расстояние в обратном направлении (рисунок 7.5).

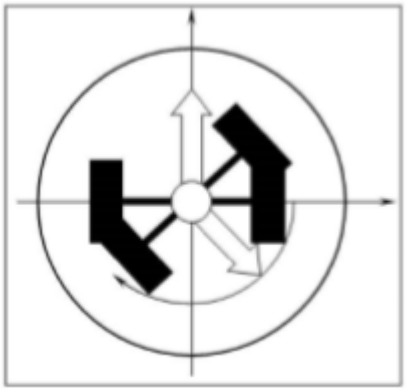


Рисунок 7.5 – Поворот на угол X вокруг своей оси.

За один оборот колесо проезжает 17,58 см, поэтому количество оборотов каждого колеса будет равно:

$$N = \frac{4{,}71}{17{,}58} = 0{,}27(оборотов).$$

Итак, предел оборотов для поворота робота вокруг своей оси на 45° будет составлять 97,2 (рисунок 7.6).

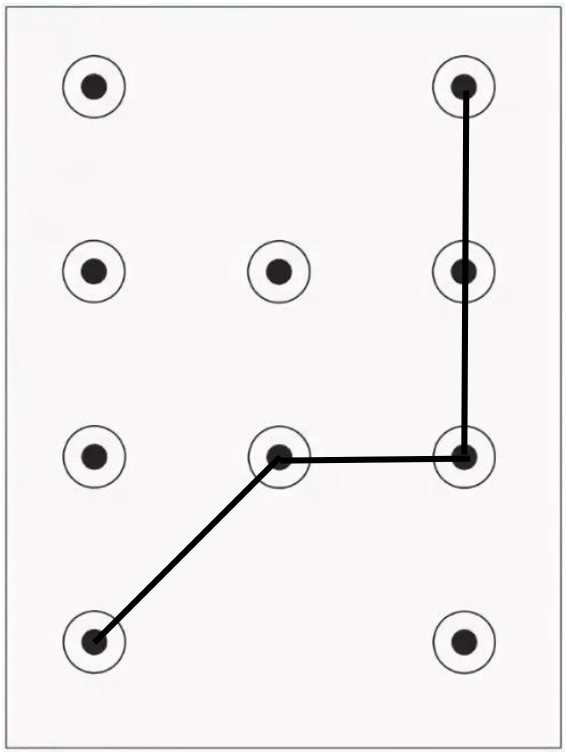
Рисунок 7.6 – Программа к задаче 7.3.

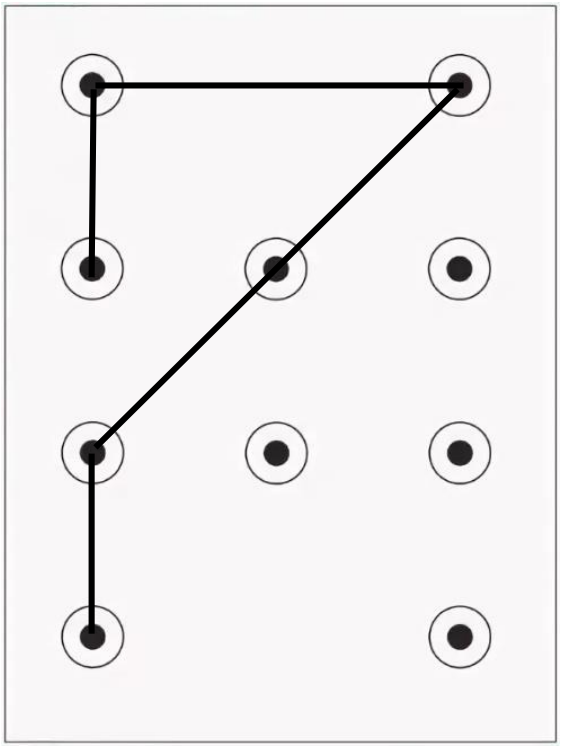
Упражнения для самостоятельного решения:

1. Составить программы с использованием датчика вращения мотора (энкодера):
    a) Повернуться налево на 90º.
    b) Проехать 0,5 м вперед, повернуться на 60º по часовой стрелке вокруг правого колеса и проехать 0,5 м назад.
    c) Проехать по траектории: равносторонний треугольник, квадрат, ромб, равнобедренная трапеция с заданными сторонами и углами.

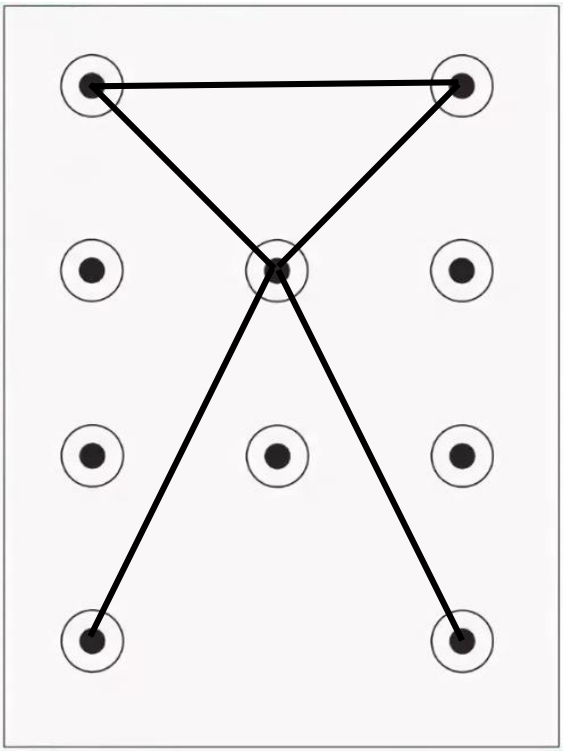
2. Выполнить задание 9, расположенные в разделе 6 с использованием датчика вращения мотора (энкодера).

3. Составить программы проезда указанных траекторий на соревновательном поле «Чертежник», точку старта и финиша выбрать на своё усмотрение:

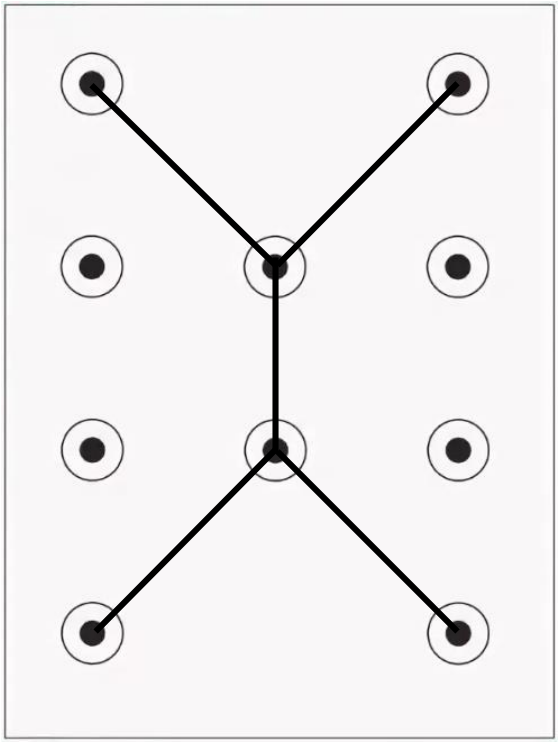
a) 

b) 

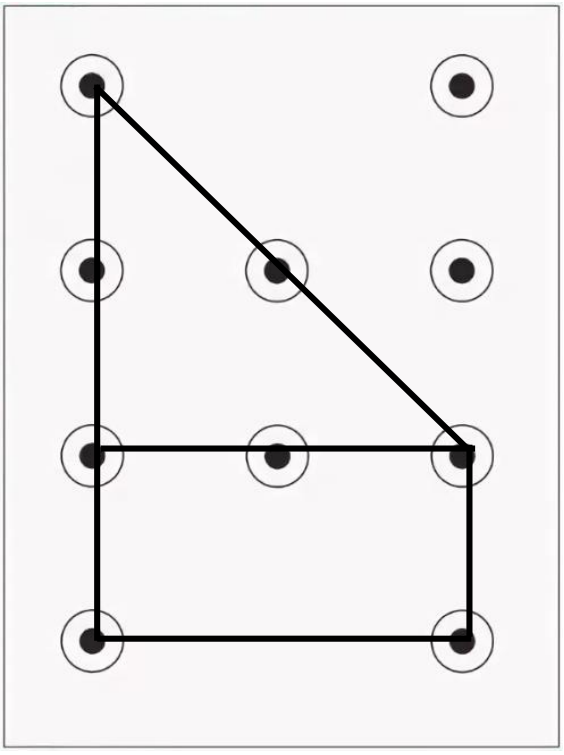
c)



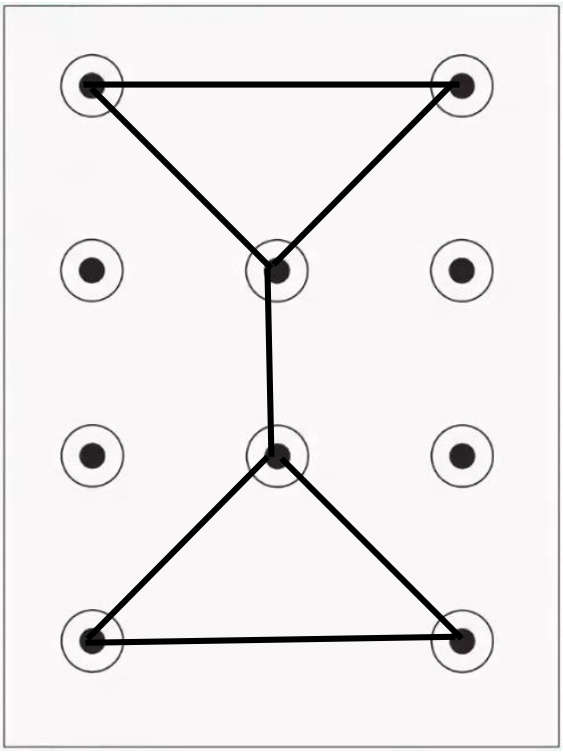
d)



e)



f)



8. Программирование датчиков касания и ультразвука

На контроллере Lego EV3 кроме разъемов для подключения моторов A, B, C, D находятся порты для подключения датчиков и обозначены цифрами 1, 2, 3 и 4.

Задача 8.1. Составить программу движения робота до стены, ударившись о которую, отъехать назад на заданное расстояние.

Для решения поставленной задачи используем датчик касания, которому соответствует в палитре блок «Ждать датчик касания». Данный блок будет являться условием окончания движения вперед, поэтому блоки «Таймер» или «Ждать энкодер» не нужны. В свойствах блока необходимо указать номер порта, к которому датчик касания подключен (рисунок 8.1)

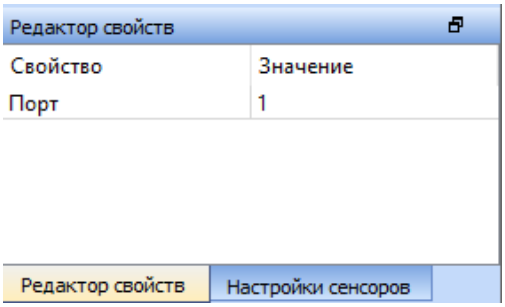


Рисунок 8.1 – Свойства блока «Ждать датчик касания»

Затем на панели «Настройки сенсоров» необходимо из списка в строке «Порт 1» выбрать «Сенсор касания» (рисунок 8.2). В окне «Отладка» на роботе появится изображение датчика, который с помощью мыши можно перетащить к передней или задней панели робота.

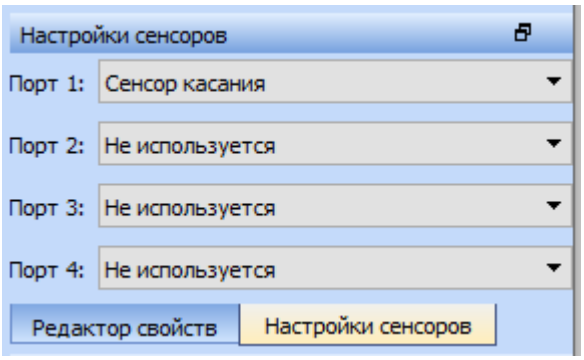


Рисунок 8.2 – Панель «Настройки сенсоров»

Итак, переместим на сцену блоки «Моторы вперед», «Ждать датчик касания», «Стоп моторы», «Сбросить показания энкодеров», «Моторы назад», «Ждать энкодер», «Конец». Соединив блоки последовательно и выставив необходимые свойства, получим программу как на рисунке 8.3:

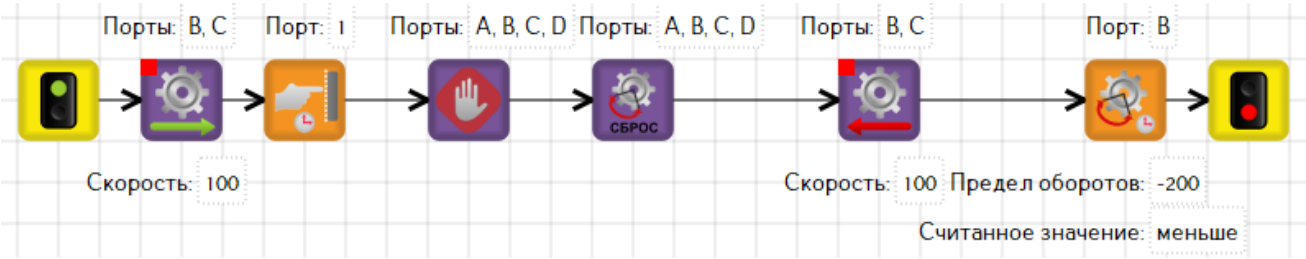


Рисунок 8.3 – Программа к задаче 8.1.

Осталось запустить программу. Для запуска программы в 2D модели сначала необходимо нарисовать стену с помощью инструмента «Стена» (рисунок 8.4). Для проверки программы на реальном роботе, загружаем программу на контроллер и запускаем на выполнение, выставив переднюю панель робота с датчиком касания в направлении стены.

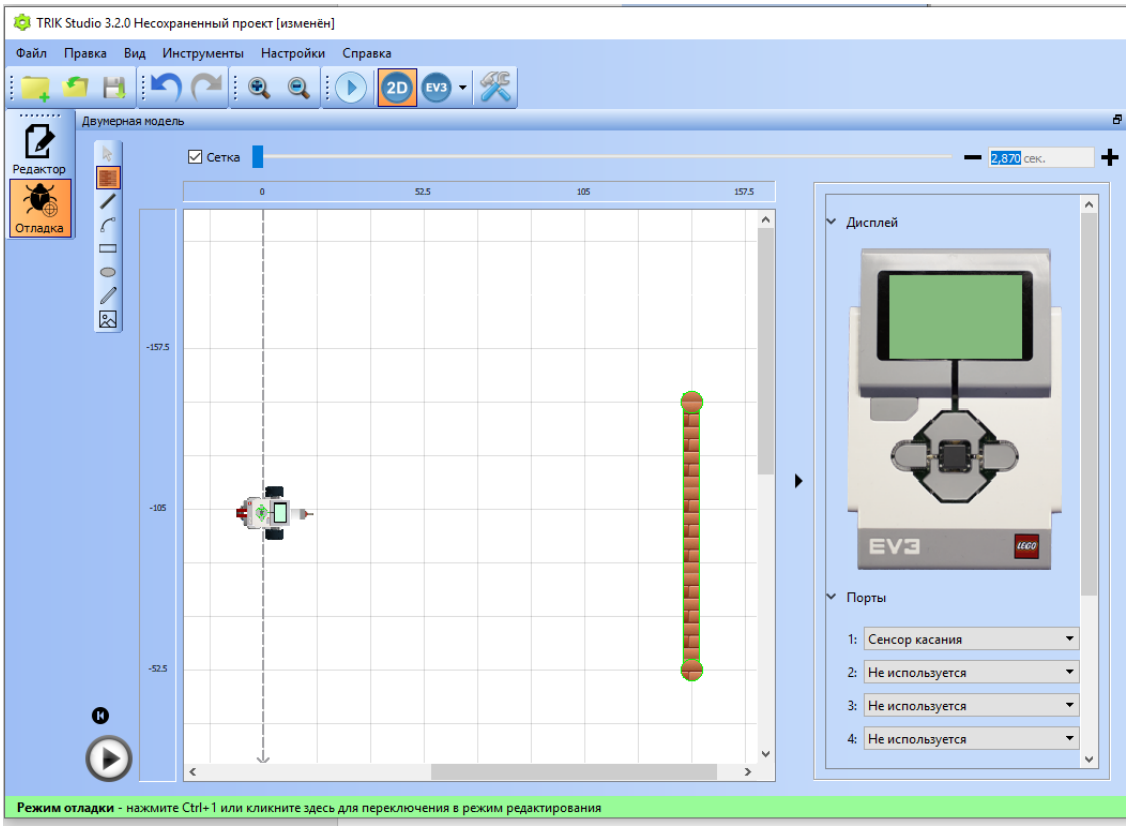


Рисунок 8.4 – Режим отладки в 2D модели с нарисованной стеной.

Задача 8.2. Составить программу движения робота до стены, не касаясь которой, отъехать назад на заданное расстояние.

Отличие от предыдущей задачи заключается в том, что робот не должен ударяться о стену, а остановиться на определенном расстоянии и отъехать назад.

При решении задачи в данном случае необходимо воспользоваться ультразвуковым датчиком.

Цифровой ультразвуковой датчик генерирует ультразвуковые волны и считывает их отражение для обнаружения измерения расстояния до объектов. Минимальное расстояние – 3 см, максимальное – 250 см. Точность определения ±1 см. Точность измерения поддерживается в диапазоне 0-160 см. Если смотреть на датчик спереди, то левая сторона содержит передатчик сигнала, правая – приемник. Угол излучения пучка лучей составляет примерно 20^{o}. Если поместить робота на расстоянии 1 метр, то ширина пучка составит ≈65 см. Соответственно, датчик не сможет обнаружить объекты, расположенные на расстоянии 1 метр от него более чем на ≈ 32 см в сторону от оси (рисунок 8.5).

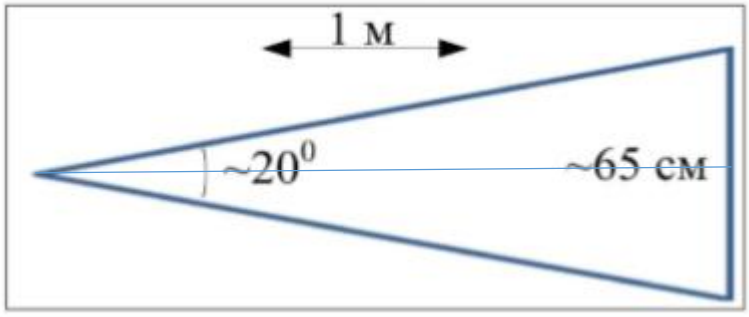


Рисунок 8.5 – Определение угла разброса пучка лучей.

В TRIK Studio ультразвуковому датчику соответствует блок «Ждать сенсор расстояния», имеющий ряд свойств:

- Расстояние (выставляем расстояние до объекта в сантиметрах, на котором должны остановиться).
- Порт (определяем порт, к которому подключен датчик).
- Считанное значение (меньше, больше, не меньше, не больше).

Изменим программу, написанную с использованием датчика касания. Заменим блок датчика касания на блок «Ждать сенсор расстояния». Укажем расстояние, равное 10 см и считанное значение – меньше, что будет означать:

моторы остановятся, как только расстояние до стены станет меньше 10 см (рисунок 8.6).

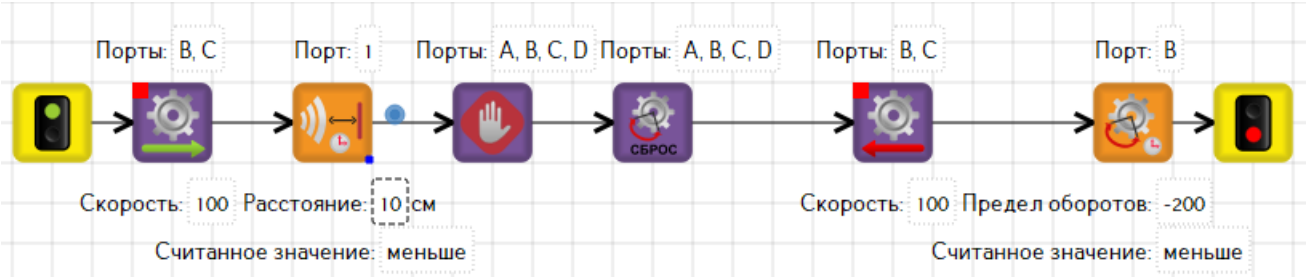


Рисунок 8.6 – Программа к задаче 8.2.

Перед запуском программы необходимо поменять на панели «Настройки сенсоров» у порта 1 «Сенсор касания» на «Сенсор расстояния». В режиме «Отладка» на 2D модели появится изображение ультразвукового датчика (рисунок 8.7).

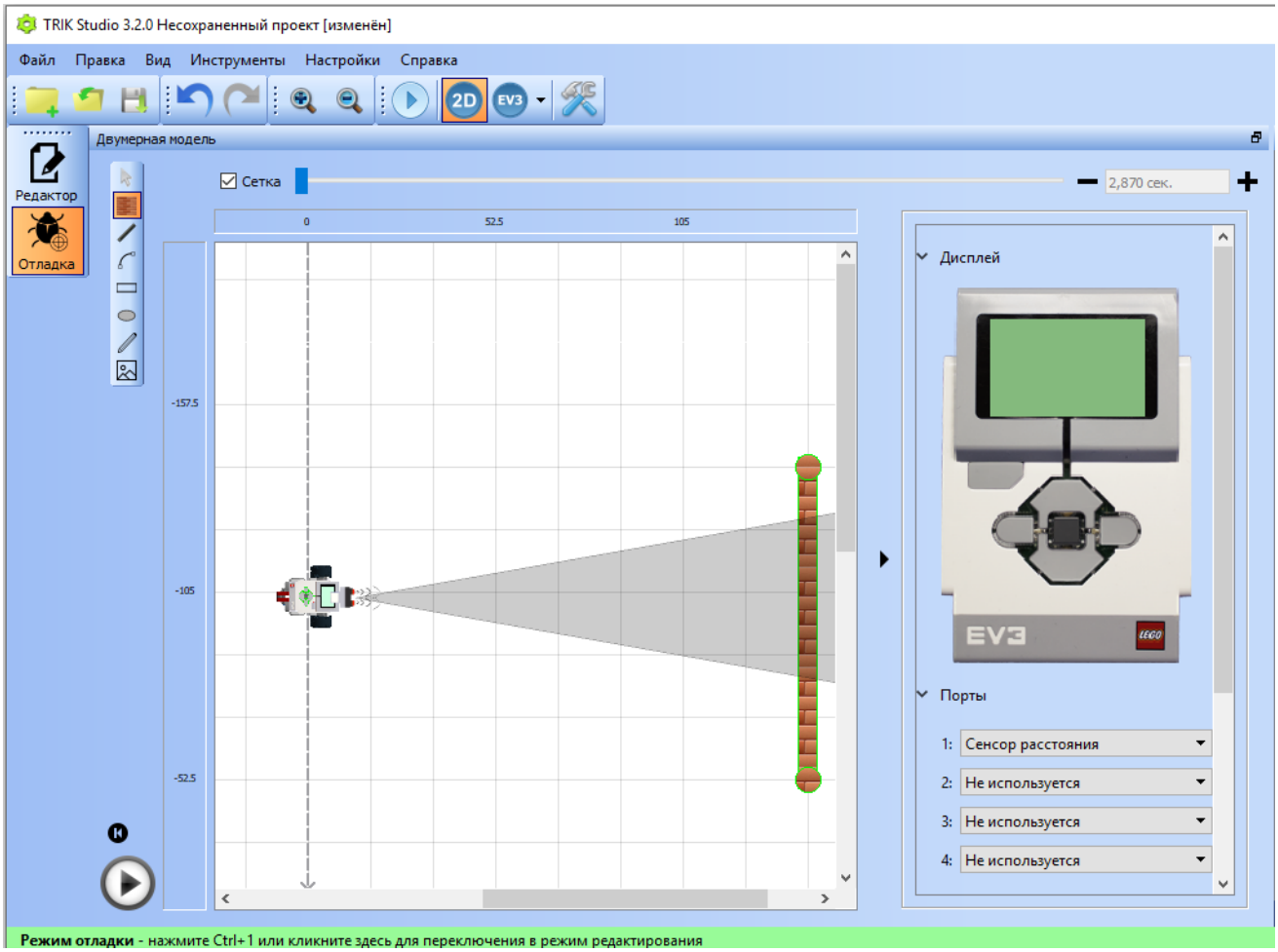
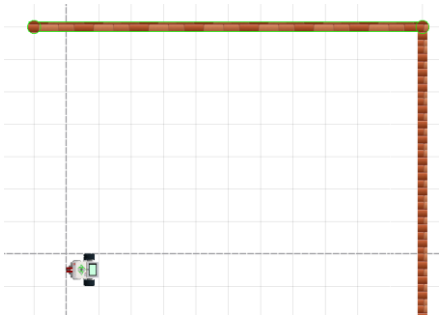


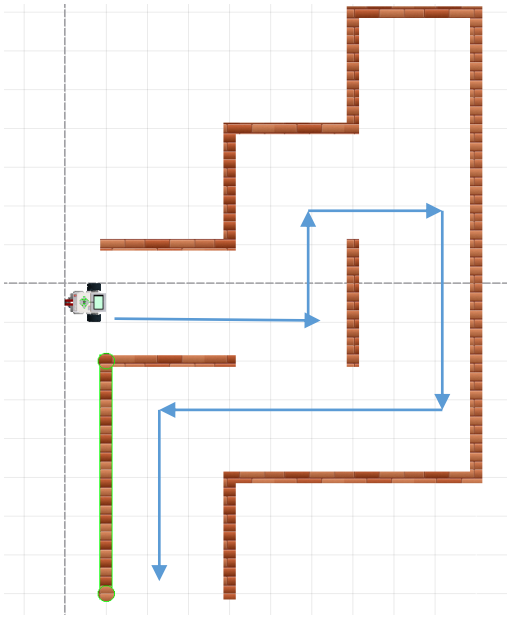
Рисунок 8.7 – Режим отладки в 2D модели с ультразвуковым датчиком.

Упражнения для самостоятельного решения

1. Составить программу: робот перемещается в сторону стены и останавливается на расстоянии 15 см, затем поворачивает налево и продолжает движение вдоль стены, пока не упрется в тупик.



2. Создать программу перемещения робота с помощью датчика касания по следующему маршруту в лабиринте:



3. Используя лабиринт из задания 1, создать программу перемещения робота с помощью ультразвукового датчика.

4. Составить программу: робот движется вдоль забора до тех пор, пока он не закончится, затем разворачивается на 180° и дет обратно.

5. Сигнализация: в случае проникновения в помещение срабатывает звуковая сигнализация. Какие датчики необходимо использовать? Составить программу.

9. Использование подпрограмм

При составлении программ очень часто бывает, что некоторые операции, выполняемые роботом, повторяются в разных частях программы или в разных программах. Такие блоки команд лучше выделить в подпрограммы.

Подпрограмма (функция, процедура) – относительно самостоятельная часть программы, имеющая свое имя и выполняющая определенные действия.

Команды, размещенные в подпрограмме и отделенные от основной программы, выполняются лишь в случае их вызова из основной программы из любого места. Одна и та же подпрограмма может вызываться из основной программы любое количество раз. Чтобы понять, какие подпрограммы потребуются и сколько раз их нужно вызывать, необходимо выполнить декомпозицию задачи.

Декомпозиция задачи – процесс разбиения задачи на элементарные действия, например, движение вперед, поворот, движение назад, движение до стены и т.д.

Для того чтобы не выполнять одну и туже работу, программисты составляют свою собственную библиотеку подпрограмм и используют их при необходимости.

Задача 9.1. Объехать прямоугольное здание по периметру.

Декомпозиция задачи: движение робота состоит из операций движение «Прямо» и «Поворот» (влево или вправо в зависимости от выбора направления).

Для создания подпрограммы, следует перетащить блок «Подпрограмма» из палитры на сцену, ввести название, затем дважды щелкнуть по пиктограмме, чтобы добавить команды в подпрограмму. Начало любой подпрограммы – блок «Начало», а заканчивается блоком «Конец».

Подпрограмма «Go» – это движение по прямой (рисунок 9.1).

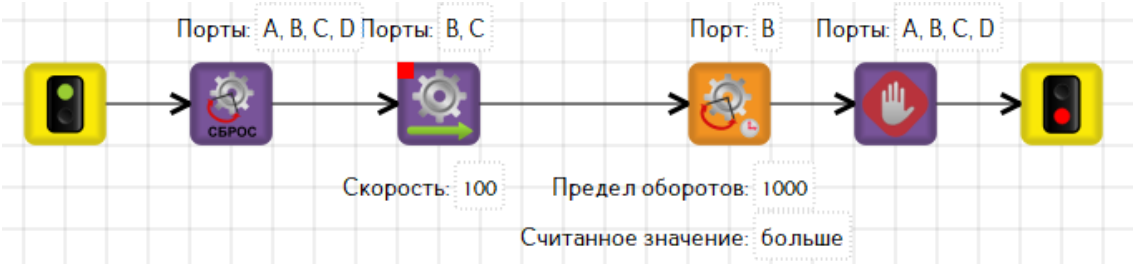


Рисунок 9.1 – Подпрограмма «Go».

Подпрограмма «Turn» осуществляет резкий поворот на 90º (рисунок 9.2).

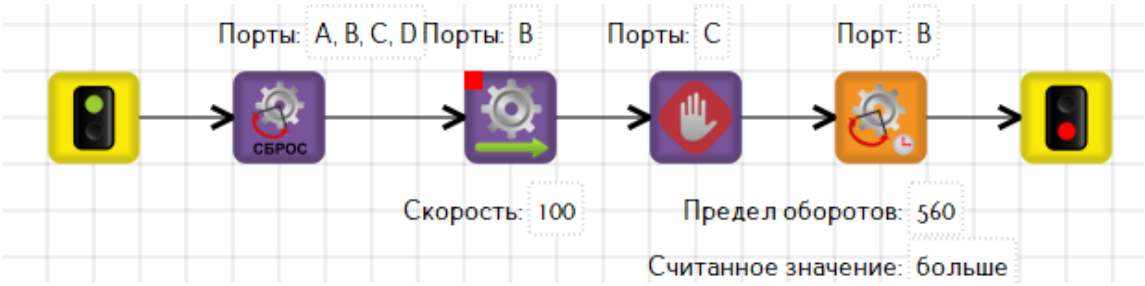


Рисунок 9.2 – Подпрограмма «Turn».

Итоговая программа показана на рисунке 9.3.

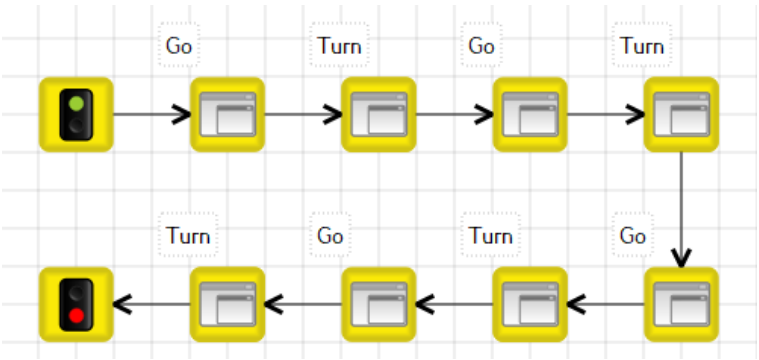


Рисунок 9.3 – Программа к задаче 9.1.

После создания подпрограмм и их открытия рядом с вкладкой «Диаграмма поведения робота» появляются вкладки подпрограмм (рисунок 9.4), а в палитре – панель «Подпрограммы» (рисунок 9.5).

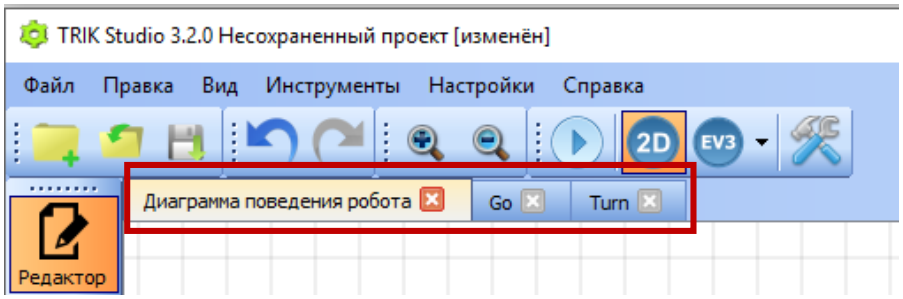


Рисунок 9.4 – Вкладки подпрограмм.

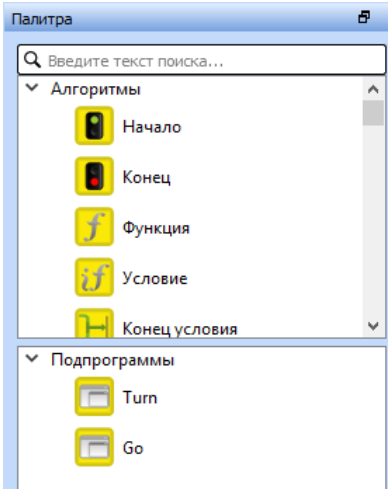


Рисунок 9.5 – Панель «Подпрограммы».

Созданные подпрограммы при необходимости можно корректировать, переходя по нужным вкладкам. Любая подпрограмма имеет ряд свойств, которые можно изменить, если вызвать контекстное меню в палитре «Подпрограммы» и выбрать строку «Изменить свойства» (рисунок 9.6). Данная диалоговая панель позволяет изменить имя, картинку или фон подпрограммы, добавить при необходимости параметры. После внесения изменений обязательно нажать кнопку «Сохранить все».

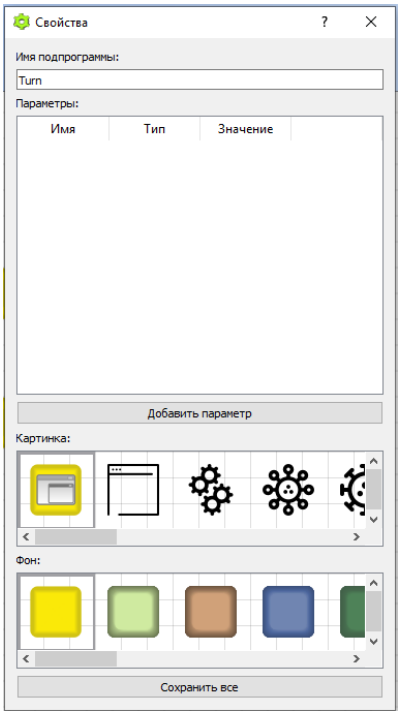


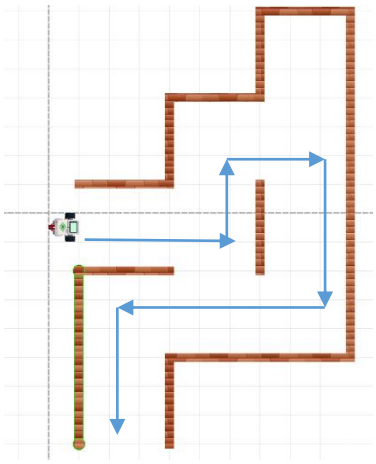
Рисунок 9.6 – Свойства подпрограммы.

Итак, за счет использования подпрограмм, текст основной программы становится намного короче и читабельнее. Такой подход используется

профессиональными программистами во всем мире, в том числе и при коллективной разработке программного обеспечения.

Упражнения для самостоятельного решения

1. Включить в свою библиотеку подпрограмм следующие действия:
   a) Резкий поворот налево на 90°;
   b) Резкий поворот направо на 90°;
   c) Плавный поворот налево на 90°;
   d) Плавный поворот направо на 90°;
   e) «Танковый» разворот налево на 90°;
   f) «Танковый» разворот направо на 90°;
   g) Резкий поворот на 180°;
   h) Плавный поворот на 180°;
   i) «Танковый» разворот на 180°;
   j) Проезд вперед на одну клетку;
   k) Проезд до стены с помощью датчика касания;
   l) Проезд до стены с помощью ультразвукового датчика.
2. Составить программу выхода из лабиринта с использованием соответствующих подпрограмм.



3. Робот рисует восьмерку, в ее центре поворачивается на 90 градусов и рисует еще одну.
4. Робот захватывает манипулятором груз, находящийся перед ним, и переставляет его, поворачиваясь на 180 градусов.

10. Программирование датчика цвета

Кроме датчика касания и ультразвука можно подключить датчик цвета, который работает в трех режимах:

1. определения цвета;
2. яркость отраженного света;
3. яркость внешнего освещения.

В режиме «Цвет» датчик цвета может определить цвет находящегося рядом объекта или цвет поверхности, находящейся рядом с датчиком. При этом для обеспечения более точного определения цвета, объект или поверхность должны находиться очень близко к датчику, но не касаться его (0,1 – 1 см). Датчик может определять семь разных цветов: черный, синий, зеленый, желтый, красный, белый и коричневый, каждый из которых имеет свой код. В TRIK Studio код цвета можно увидеть на панели «Переменные» в режиме отладка 2D модели в строках *sensor1*, *sensor2, sensor3* или *sensor4*, в зависимости от используемого порта.

Объект другого цвета может определяться как «Без цвета» или его цвет может определяться по ближайшему к нему цвету. Например, оранжевый цвет может определяться как красный или желтый в зависимости от содержания красного цвета в оранжевом, или как коричневый или черный, если оранжевый цвет очень темный или находится слишком далеко от датчика. При работе в данном режиме на передней панели датчика загораются красный, зеленый и синий светодиоды.

Важно!

Если поверхность материала блестящая, то определение цвета может быть некорректным из-за бликов от подсветки датчика или яркого солнца. Поэтому, если нет возможности изменить материал, необходимо максимально снижать скорость робота при проезде цветного определяемого участка или закрывать поле от попадания прямых солнечных лучей. Ошибка при определении цвета может возникать и на матовых поверхностях при освещении помещения люминесцентными лампами. Белая поверхность может определяться красной, зеленой, синей, причем в одной и той же точке. Это связано с тем, что принцип работы ламп предполагает мерцание светового потока практически во всей

области видимого спектра. Человек мерцание может не замечать, но датчик, работая с частотой 1 кГц, вместо правильного цвета может фиксировать отраженные цветовые лучи. Единственный способ выхода из данной ситуации – конструировать робота так, чтобы датчик цвета был закрыт сверху и с боков от прямого попадания света люминесцентных лампам. Например, можно надеть шину (рисунок 10.1).

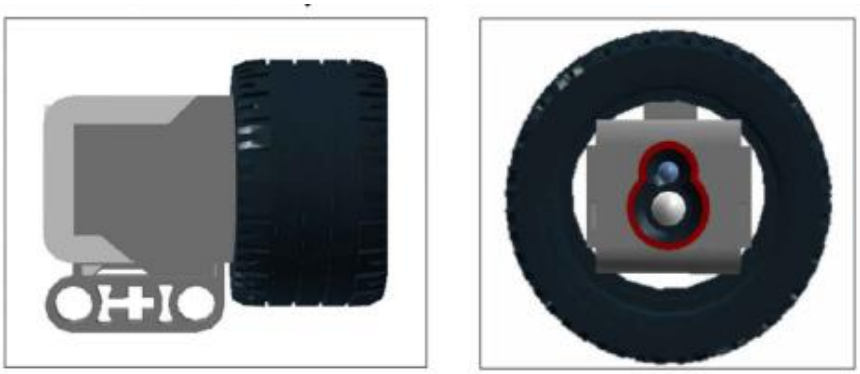


Рисунок 10.1 – Внешний вид шины, надетой на датчик, в двух проекциях.

В режиме «Яркость отраженного света» датчик цвета определяет яркость света, попадающего в датчик и измеряется в процентах от 0 до 100. Если от поверхности ничего не отражается, то значение датчика будет равно 0. Если 100% света, созданного датчиком, отражается, то датчик передаст значение 100.

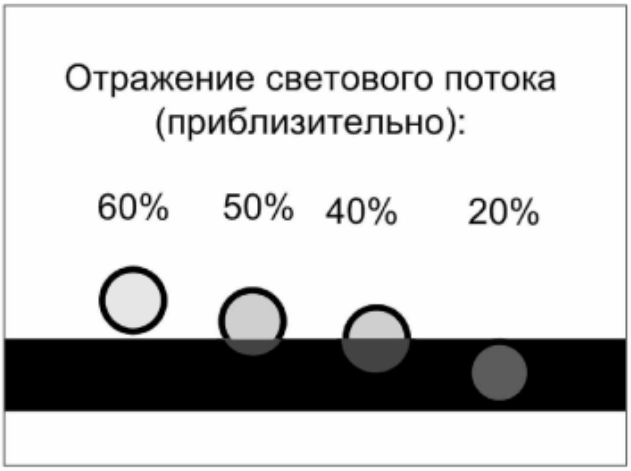


Рисунок 10.2 – Отражение светового потока при разном расположении датчика над поверхностью черной линии.

Так как датчик цвета в указанном режиме фиксирует процент отраженного света, то получаемые значения удобно сравнивать со шкалой градаций серого цвета, то есть можно сказать, что датчик различает 100 градаций серого цвета.

Число на выходе будет зависеть не только от оттенка цвета, но и от внешнего освещения и поверхности.

Когда датчик цвета находится в режиме «Яркость отраженного света», на передней панели датчика загорается красный светодиод.

В режиме «Яркость внешнего освещения» датчик цвета определяет яркость света, попадающего в датчик и измеряется в процентах от 0 до 100, где 0 – очень темный, а 100 – очень яркий. Когда датчик цвета находится в данном режиме, на передней панели датчика загорается тусклый синий светодиод. Его можно использовать для определения яркости освещения комнаты или яркости других источников света, свет от которых попадает в датчик.

В среде TRIK Studio можно увидеть графическое представление датчиков при запуске программы в режиме интерпретации на роботе или в двумерной модели. Для просмотра необходимо открыть режим «Отладка», закладка «Графики» (рисунок 10.3).

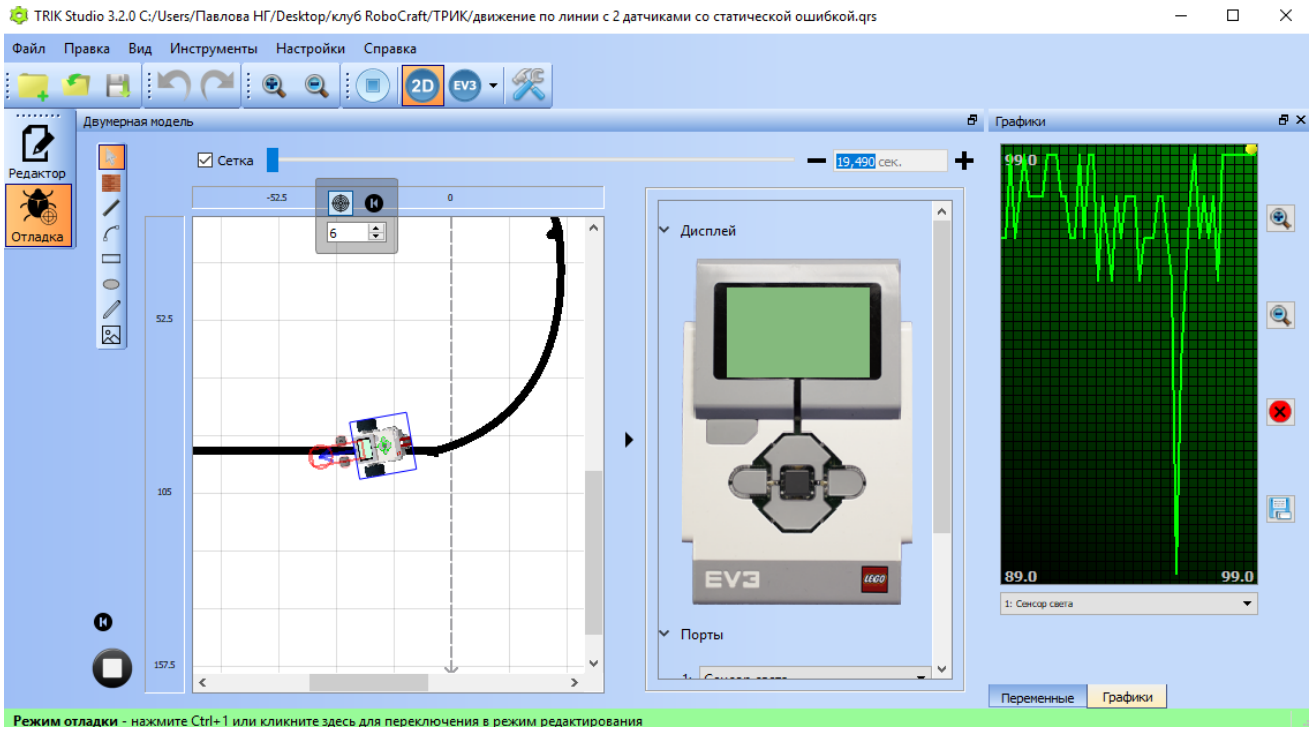


Рисунок 10.3 – Графическое представление сенсора света при движении по линии.

Из выпадающего списка снизу можно выбрать сенсор, значения с которого будут отображаться. Кнопками "Старт" и "Стоп" можно запустить или остановить снятие показаний, не прерывая работы программы. Можно регулировать масштаб графика вручную, кроме того, график автоматически масштабируется, чтобы

кривая показаний помещалась по высоте целиком. При наведении курсора мыши на точку на графике отобразится значение в этой точке. Ниже кнопки "Очистить график" находится кнопка "Выгрузить показания в файл", которая позволяет экспортировать в формат .csv все показания выбранного датчика с начала их записи. Просмотреть показания затем можно, например, в программе MS Excel.

В TRIK Studio для работы с датчиком цвета используются блоки «Ждать цвет», «Ждать интенсивность цвета», «Ждать свет».

Задача 10.1. Робот едет прямо до тех пор, пока датчиком цвета не обнаружит линию зеленого цвета. При обнаружении цвета робот должен остановится и проиграть звук.

Для решения задачи понадобятся блоки «Моторы вперед», «Ждать цвет», «Гудок» или «Играть звук». В редакторе свойств блока «Ждать цвет» необходимо определить порт, к которому подключен датчик цвета и указать цвет.

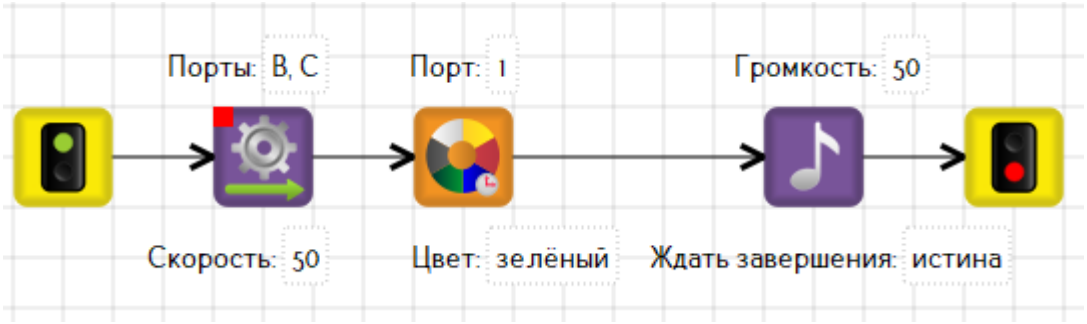


Рисунок 10.4 – Программа к задаче 10.1.

В режиме отладки 2D модели перед запуском программы необходимо нарисовать линию, используя инструмент «Линия», выбрать соответствующий цвет, в разделе «Порты» выбрать «Сенсор цвета (распознавание цветов)» у порта №1. Робот будет двигаться прямо, пока не обнаружит линию зеленого цвета, затем остановиться и произведет гудок.

Задача 10.2. Робот едет прямо до тех пор, пока датчиком цвета не обнаружит линию черного цвета. При обнаружении цвета робот должен остановится и проиграть звук.

Решим задачу с использованием блока «Ждать свет», то есть используя режим «Яркость отраженного света».

Заменим в предыдущей задача блок «Ждать цвет» на «Ждать свет». Блок имеет следующие свойства: «Порт», «Проценты» и «Считанное значение». В нашем случае датчик цвета подключен к порту 1. В окне «Проценты» установим значение, равное 50, так как темные поверхности имеют яркость отраженного света как правило меньше указанного значения. И в свойстве «Считанное значение» необходимо указать «меньше». Получим программу на рисунке 10.5. При запуске программы в разделе «Порты» нужно поменять «Сенсор цвета (распознавание цветов)» на значение «Сенсор света».

Рисунок 10.5 – Программа к задаче 10.2.

Задача 10.3. Подключить к роботу два датчика цвета (рисунок 10.6) и вывести на экран их показания в режиме яркости отраженного света (один датчик расположен на белой поверхностью поля, второй – над черной линией).

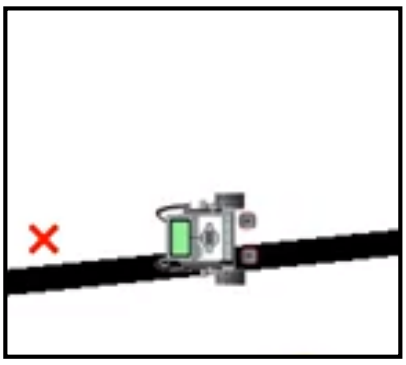


Рисунок 10.6 – Положение датчиков и робота.

При решении задачи воспользуемся блоками «Параллельные задачи» (подробнее см. раздел 13) и «Переменная» (подробнее см. раздел 14). В блоках «Переменная» будут записаны показания датчиков 1 и 2 и в дальнейшем выведены на экран, причем запись и вывод будет выполняться параллельно, с использованием блока «Параллельные задачи» (рисунок 10.7). Для вывода на

экран числовых значений *s1* и *s2*, а не текста, не забываем в редакторе свойств блока «Напечатать текст» ставить флажок «Вычислять».

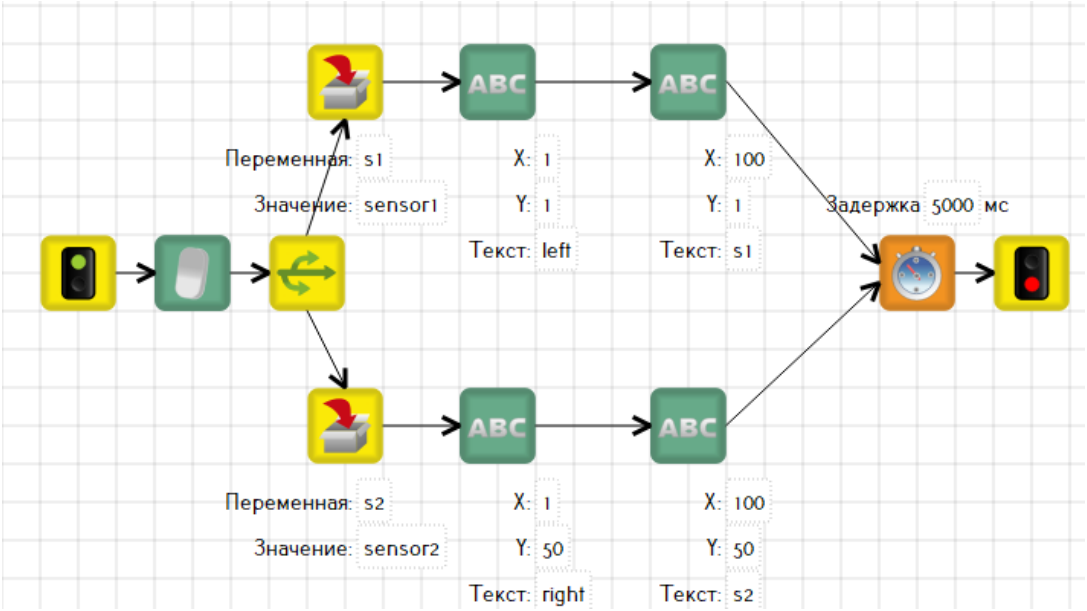


Рисунок 10.7 – Программа к задаче 10.3.

В режиме отладки 2D модели экран будет выглядеть как на рисунке 10.8. Левый датчик находился над белой поверхностью и его показания близки к 100%, а правый датчик – над черной линией и имеет показания, равные 10%. При запуске программы на реальном роботе показания могут отличаться, так как они будут зависеть от внешних факторов, о которых было сказано выше.

Рисунок 10.8 – Вид контроллера во время запуска программы.

Упражнения для самостоятельного решения

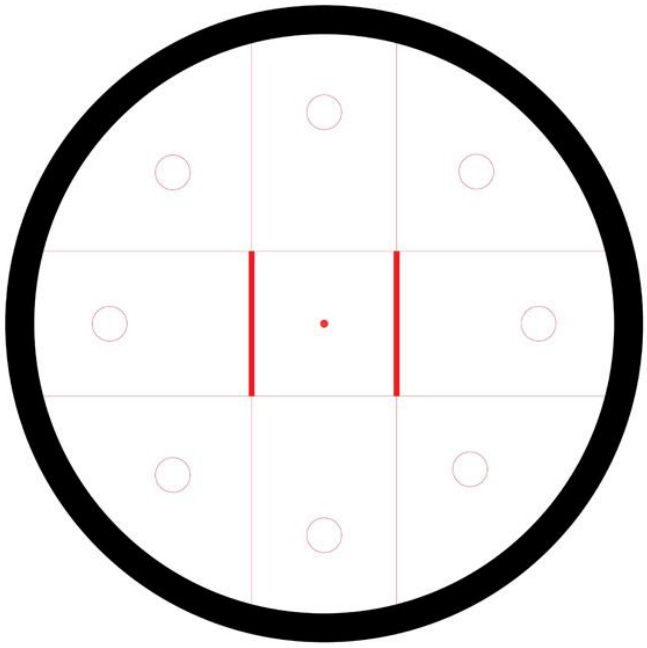
1. Используя панель «Переменные» в режиме отладки 2D модели, определить коды следующих цветов: черный, синий, зеленый, желтый, красный, белый.

2. Составить программу объезда роботом синей лужи.

3. Нарисовано несколько цветных линий. Пересекая желтую линию, робот отпускает маркер красного цвета.

4. Составить программу вывода на экран названия цвета той линии, которую робот проезжает.

5. Задача «Кегельринг»: вытолкнуть все банки за пределы окружности диаметром 1 метр, при этом робот не должен покидать поле.



11. Организация ветвления

Все рассмотренные выше задачи относятся к типу алгоритма «Следование», когда команды выполняются в строгой последовательности друг за другом. Однако часто при решении задач требуется проверять некоторое условие, от которого будет зависеть дальнейшее поведение робота. В этом случае необходимо организовать алгоритм типа «Ветвление».

Ветвление (развилка) - такая форма организации действий, при которой в зависимости от выполнения или невыполнения конкретного условия, совершается либо одна, либо другая последовательность действий.

Условие – логическое выражение, принимающее истинное или ложное значение.

Существует полная форма ветвления (рисунок 11.1), в которой при выполнении условия будет выполняться команда, находящаяся в ветви «да», а если условие не выполняется, то выполняться команда, находящаяся в ветви «нет». Такой тип ветвления называют «Если-то-иначе».

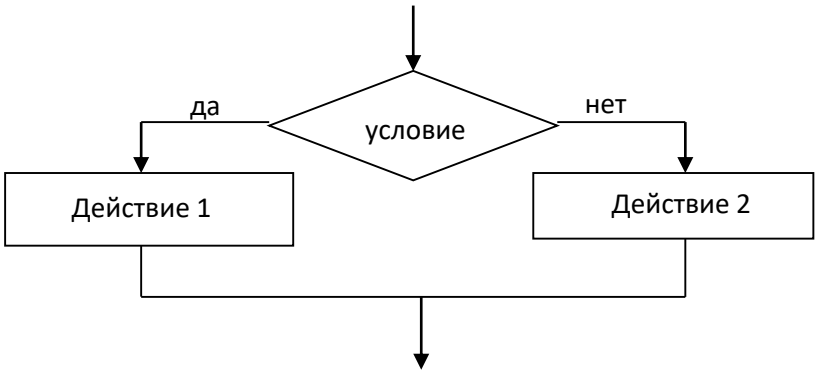


Рисунок 11.1 – Полная форма ветвления.

В неполной форме ветвления («Если-то») при выполнении условия будет выполняться команда, находящаяся в ветви «да», а если условие не выполняется, то выполнение передается команде, находящейся после ветвления (рисунок 11.2).

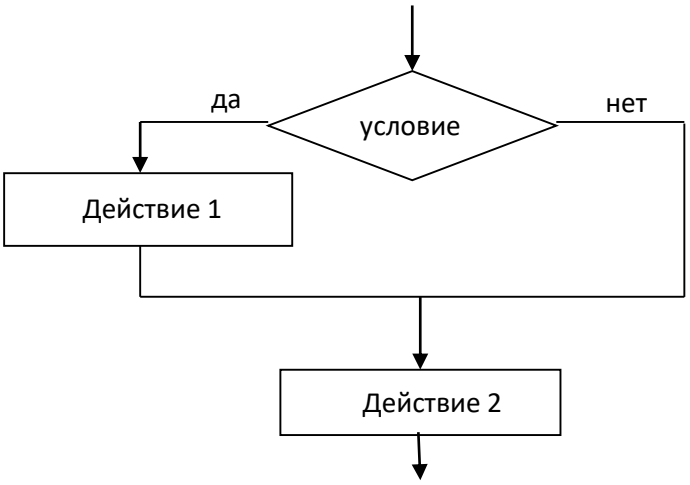


Рисунок 11.2 – Неполная форма ветвления.

В ветвлениях «Если-то» и «Если-то-иначе» предлагается выбор из двух вариантов (да/нет), но есть задачи, где нужно выбрать один из трех, четырех и более вариантов. В данном случае следует использовать ветвление типа «Выбор» или «Выбор-иначе».

В TRIK Studio можно организовать все перечисленные типы ветвления:

1) если-то;
2) если-то-иначе;
3) выбор;
4) выбор-иначе.

Задача 11.1. Робот должен доехать до стены и остановиться на расстоянии 10 см от нее. Если справа от робота стена на расстоянии менее чем 15 см, то он должен повернуть налево. Решим задачу с помощью ветвления «Если-то».

Предварительно создадим обстановку, как на рисунке 11.3. Подключим к портам 1 и 2 сенсоры расстояния, повернув сенсор 1 на 90^{o} вправо.

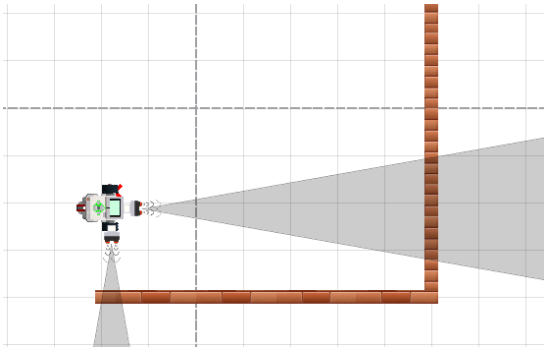


Рисунок 11.3 – Начальное положение робота на виртуальной сцене.

В программе можно использовать ранее созданную подпрограмму «Left-_90» (рисунок 11.4).

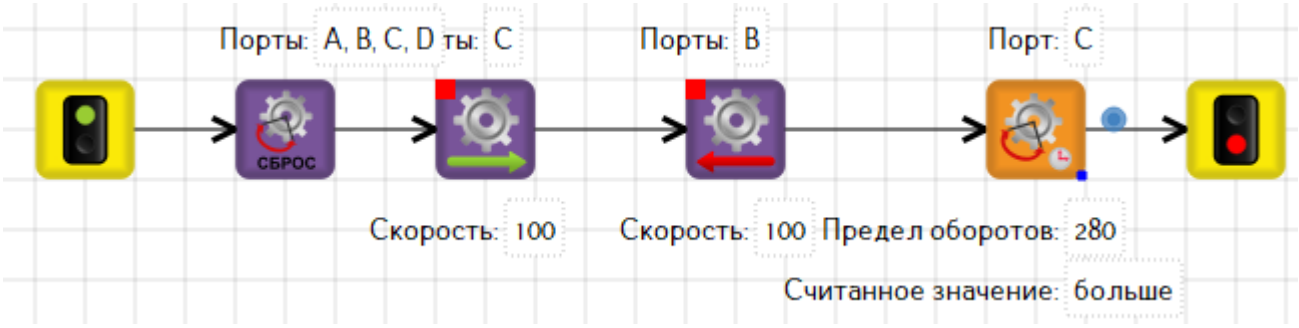


Рисунок 11.4 – Подпрограмма «Left_90».

Для организации ветвления вынесем на диаграмму блок «Условие».

При написании условий используются следующие операции:

- <, <=, >, >= – операции сравнения, применимы к целым и вещественным значениям, результат логический;
- == – операция проверки равенства, применима к значениям любых типов, результат логический;
- ~=, != – операция проверки неравенства, применима к значениям любых типов, результат логический;
- and или && – логическое "и", применим к целым, вещественным и логическим значениям, результат логический;
- or или || – логическое "или", применим к целым, вещественным и логическим значениям, результат логический.

Используя редактор свойств, введем условие *sensor1<15*. В данном случае мы используем зарезервированную переменную *sensor1*. Так как к роботу можно подключить 4 сенсора, то также можно использовать переменные *sensor2*, *sensor3* *sensor4*. Названия переменных можно копировать с панели с одноименным названием.

Из блока «Условие» обязательно должны выходить две связи – «истина» и «ложь». Причем пометить достаточно одну связь. Для внесения метки на связи, достаточно щелкнуть правой кнопкой мыши по ней и в редакторе свойств выбрать «истина» или «ложь». Итак, на ветви «истина» размещаем подпрограмму

поворот, а по ветви «ложь» завершаем программу. Общий вид программы показан на рисунке 11.5.

Рисунок 11.5 – Программа к задаче 11.1.

Задача 11.2. Вывести на экран робота квадрат, если случайное число больше 5, в противном случае окружность. Решим задачу с помощью ветвления «Если-то-иначе».

Вынесем на сцену блок «Случайное число». В его свойствах необходимо указать название переменной и указать диапазон чисел. Выберем диапазон от 1 до 10. В блоке «Условие» укажем $x>5$. В ветви «истина» расположим блок «Нарисовать прямоугольник», а ветви «ложь» – блок «Нарисовать круг». Предварительно необходимо выполнить очистку экрана с помощью соответствующего блока. Получим программу (рисунок 11.6).

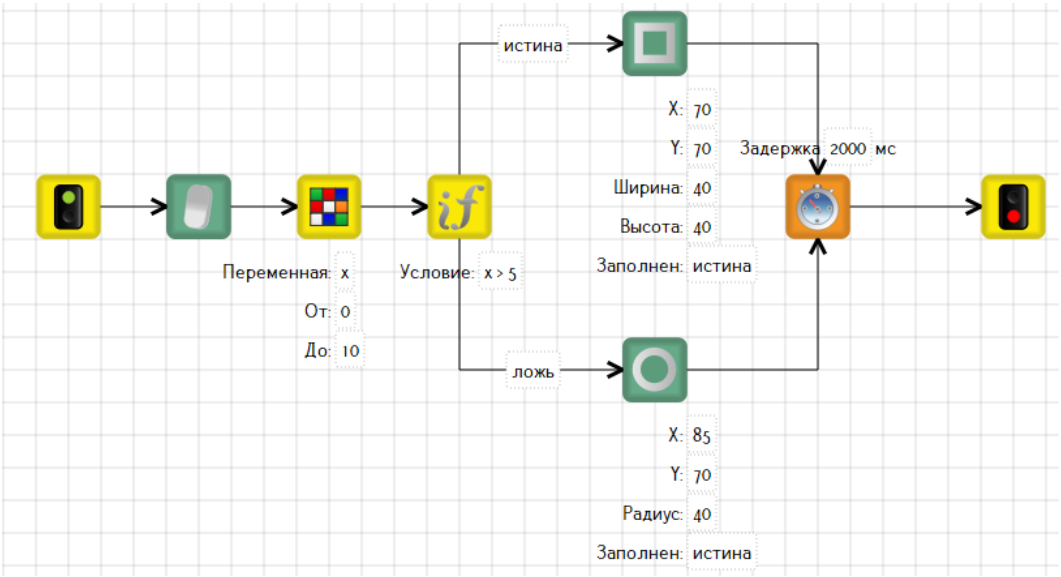


Рисунок 11.6 – Программа к задаче 11.2.

Работает программа следующим образом: случайным образом выбирается число из указанного в блоке диапазона. Например, было выбрано число 4. Проверяется условие: 4>5? Условие принимает ложное значение, поэтому на экране робота будет выведен круг.

Разберем задачу, когда необходимо сделать выбор из нескольких вариантов.

Задача 11.3. Робот при движении вперед должен рисовать цветную линию. Выбор цвета зависит от числа, полученного случайным образом.

При решении задачи понадобиться блок «Выбор» (Switch). Значение выражения, указанного в свойстве «Выражение», сравнивается со значениями на исходящих связях. Если среди них найдено равное, то исполнение будет продолжено по этой ветке. В противном случае будет выбрана ветка без маркера (эта ветка всегда должна присутствовать в блоке). На рисунке 11.7 ветка без маркера идет на блок «Конец».

Таким образом, в задаче 11.3 (рисунок 11.7) сначала случайным образом определяется число *x*, например, *x=2*, затем выполнение программы идет по ветви с маркером *2* и будет нарисована линия желтого цвета при проезде робота вперед в течении 1-й секунды.

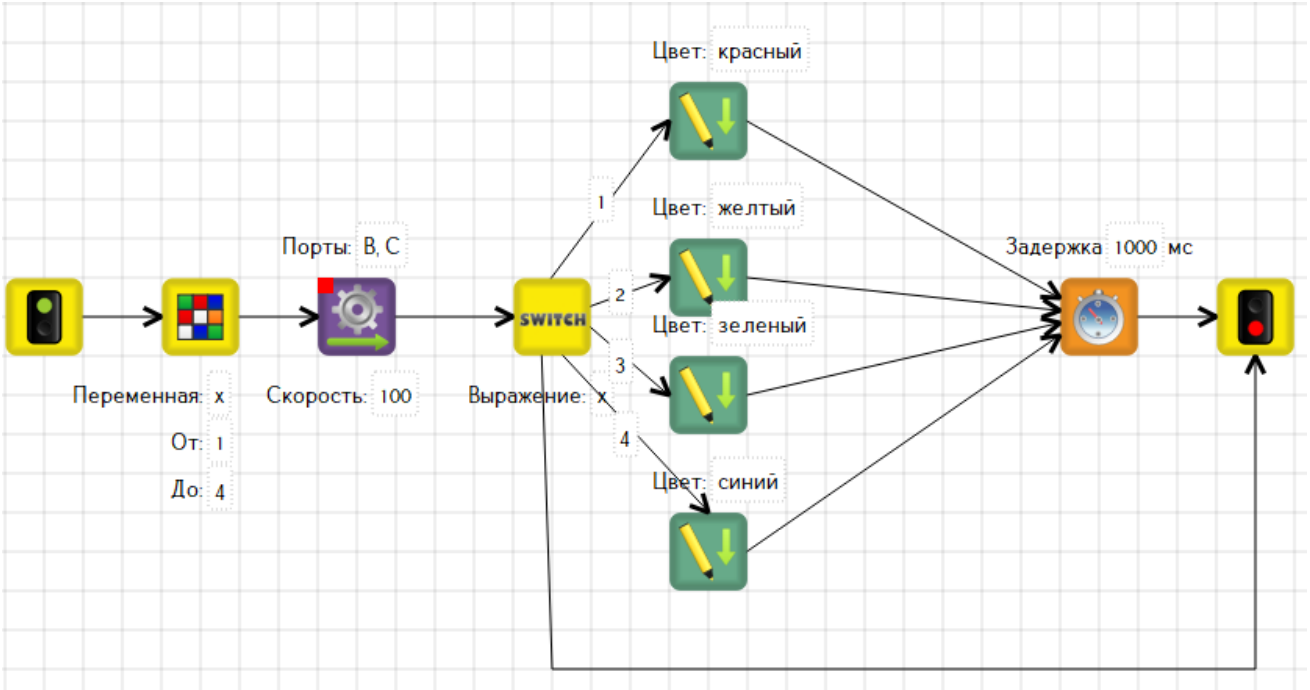
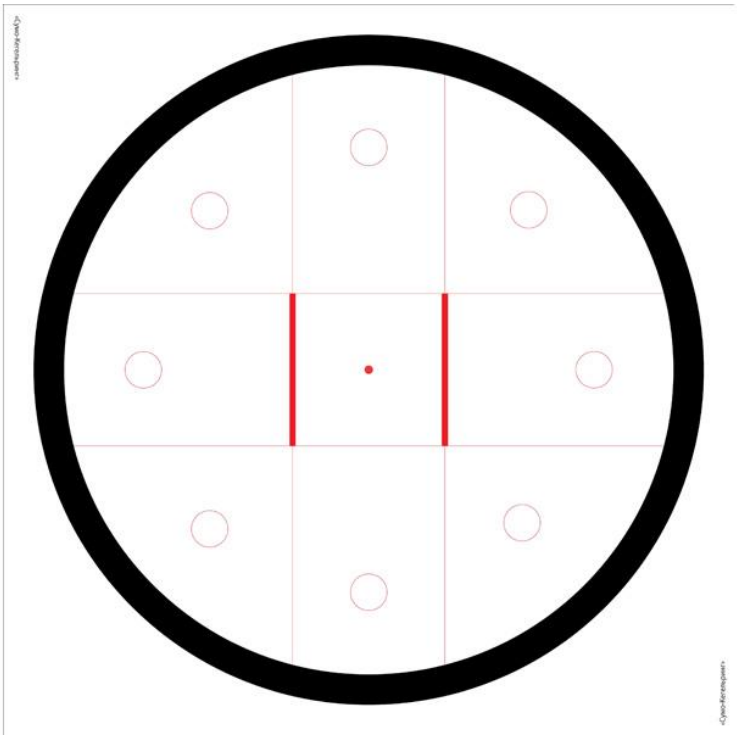


Рисунок 11.7 – Программа к задаче 11.3.

Упражнения для самостоятельного решения

1. Написать программу, имитирующую светофор:
   - если горит красный индикатор, робот должен стоять в течении 2 сек,
   - если горит оранжевый индикатор, робот стоит в течении 2 сек,
   - если горит зеленый индикатор, робот движется в течении 2 сек.
2. На дороге до места назначения 4 светофора. Надо проехать, соблюдая правила дорожного движения.
3. На поле нарисованы разноцветные линии. Составить программу вывода на экран названия цвета той линии, которую робот проезжает, используя блок «Выбор».
4. На поле «Кегельринг» располагаются банки двух цветов (белые и черные), место их расположения заранее неизвестно. Необходимо вытолкнуть банки только черного цвета.



12. Организация циклов

Роботы часто выполняют одну и туже задачу многократно. В данном случае можно говорить об организации циклического алгоритма (цикла).

Циклический алгоритм – алгоритм, в котором группа команд выполняется несколько раз в зависимости от выполнения или невыполнения условия цикла. Группа повторяющихся команд называется телом цикла.

Различают два типа циклов: с известным числом повторений и с неизвестным числом повторений. Существует 3 типа циклических структур:

- цикл с предусловием;
- цикл с постусловием;
- цикл с параметром;

Иначе данные структуры называют циклами типа «Пока», «До», «Для» соответственно.

На рисунке 12.1 представлены блок-схемы циклов с предусловием и с постусловием.

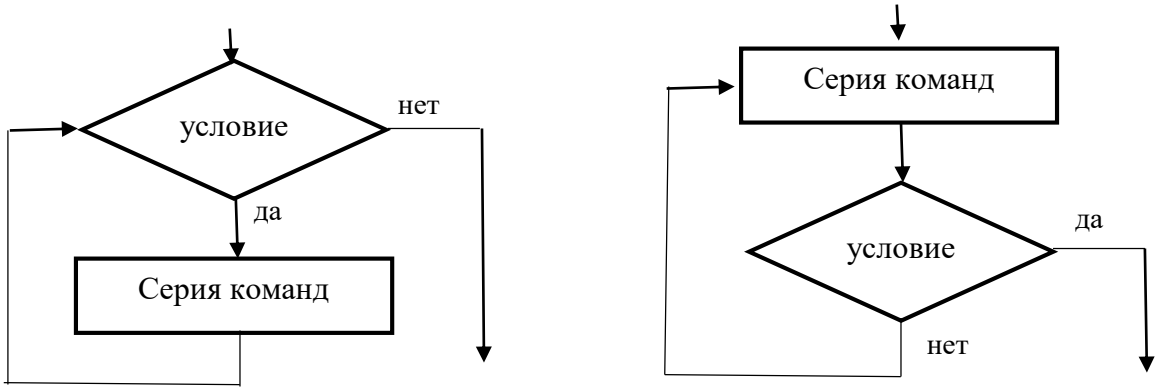


Рисунок 12.1 – Циклы с предусловием и с постусловием.

В цикле с предусловием серия команд выполняется до тех пор, пока выполняется условие. Цикл может не выполнится ни разу, если значение условия сразу же оказывается ложным.

В цикле с постусловием сначала выполняется серия команд, а затем проверяется условие: если оно ложно, то цикл продолжается, если истинно – завершается.

На рисунке 12.2 представлена блок-схема цикла с параметром.

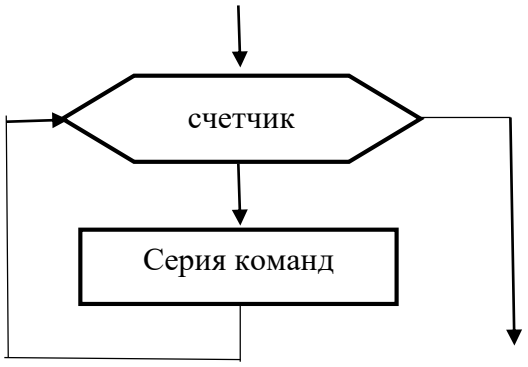


Рисунок 12.2 – Циклы с параметром.

В данном цикле серия команд выполняется определенное количество раз и до тех пор, пока счетчик принимает значения от начального до конечного, например, от 1 до 10.

В TRIK Studio кроме цикла с итерациями (цикл с параметром), цикла с условием (с постусловием, с предусловием) можно организовать и бесконечный (безусловный) цикл.

Рассмотрим пример с использование бесконечного цикла.

Задача 12.1. Нарисовать трехцветную линию, красно-желто-зеленую, бесконечно чередующуюся.

В разделе 11 подобная задача была рассмотрена для случайного числа. В данном примере воспользуемся блоком «Функция» (подробнее о блоке см. в разделе 14), в которую запишем значение величины *x*, равной *1*. Блок «Выбор» будет содержать три ветви, так как необходимо нарисовать трехцветную линию. В каждой ветви будем задавать новое значение для *x* (2, 3 или 1). Для бесконечного выполнения из блока «Таймер» нужно протянуть ветвь к блоку «Выбор», как показано на рисунке 12.3.

Программа будет выполняться, пока пользователь не нажмет кнопку «Стоп».

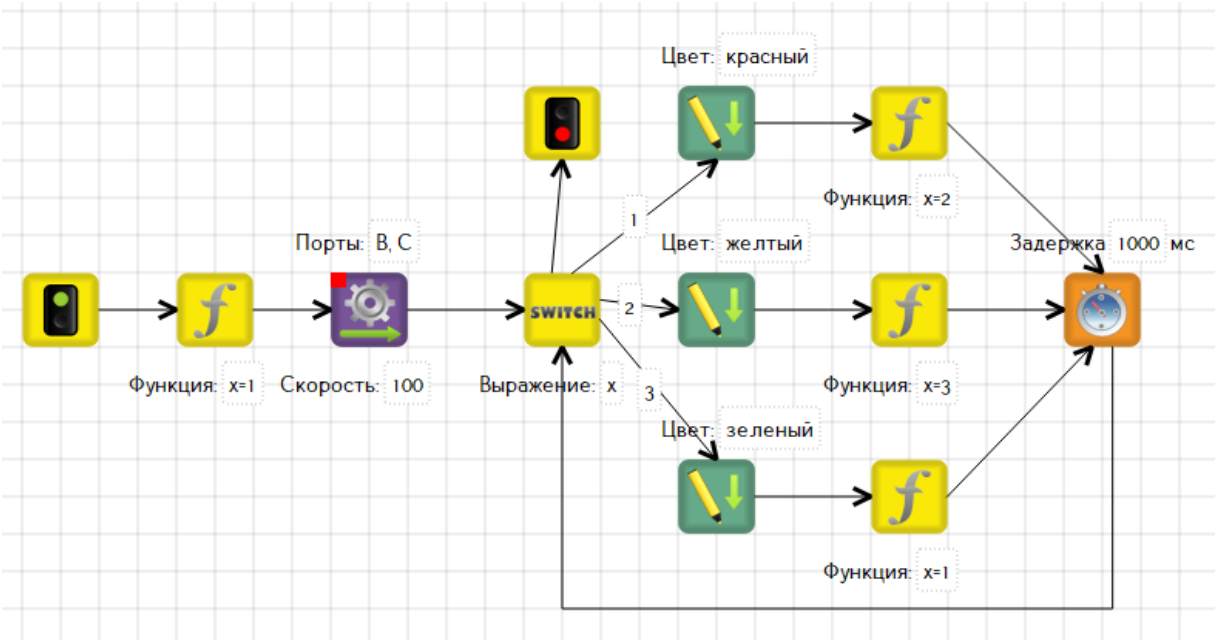


Рисунок 12.3 – Программа к задаче 12.1 использованием бесконечного цикла.

Чтобы программа выполнялась определенное количество раз, необходимо организовать цикл с параметром. Для этого в задаче необходимо использовать блок «Цикл». Данный блок имеет две исходящие линии: одна направлена на начало цикла (в свойствах линии указать «Тело цикла»), другая отправляет на команды, расположенные после цикла. В свойствах блока также нужно указать количество повторений (итераций).

Внесем изменения в предыдущей программе, указав 5 повторений (рисунок 12.4).

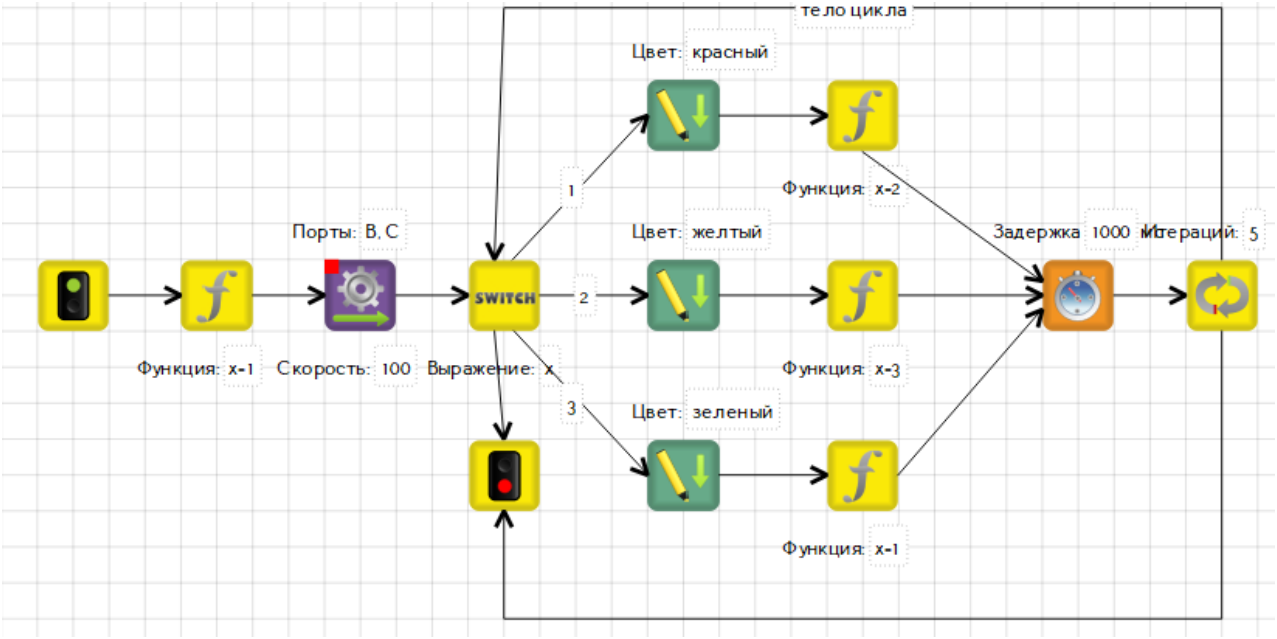


Рисунок 12.4 – Программа к задаче 12.1 с использованием цикла с параметром.

Задача 12.2. Составить программу движения робота до стены, не касаясь которой, отъехать назад на заданное расстояние.

Решим эту задачу с использованием цикла с условием. Добавим блок «Условие» и внесем условие завершения цикла *sensor1<15* (ультразвуковой датчик подключен к порту 1). Ветвь со значением «ложь» направлена на блок «Моторы вперед» (тело цикла), а ветвь «истина» – на выход из цикла. Таким образом, робот будет двигаться до стены тех пор, пока указанное условие принимает ложное значение. Как только условие примет истинной значение, робот начинает движение назад (рисунок 12.5).

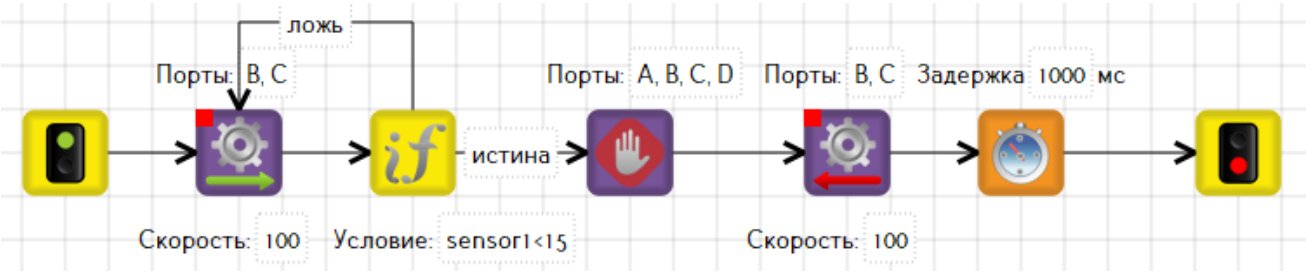


Рисунок 12.5 – Программа к задаче 12.2 с использованием цикла с постусловием.

В данном примере был применен цикл с постусловием, в котором условие расположено после команды. То есть сначала выполняется движение вперед, затем проверяется условие. Если оно не выполняется, то продолжается движение вперед.

Можно поменять в условии знак неравенства (*sensor1>15*), но в этом случае необходимо поменять значения ветвей (рисунок 12.6). Теперь тело цикла будет повторяться, если условие выполняется, то есть робот движется пока расстояние до стены больше 15. Это также пример использования цикла с постусловием.

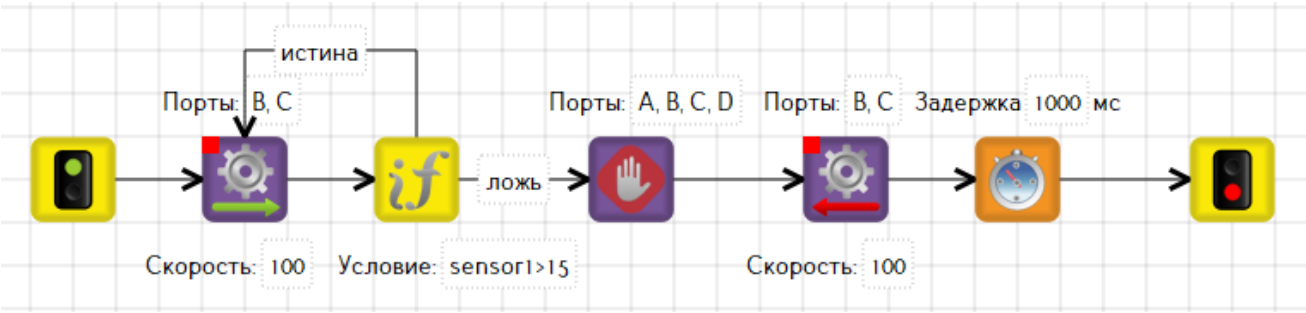


Рисунок 12.6 – Программа к задаче 12.2 с использованием цикла с постусловием.

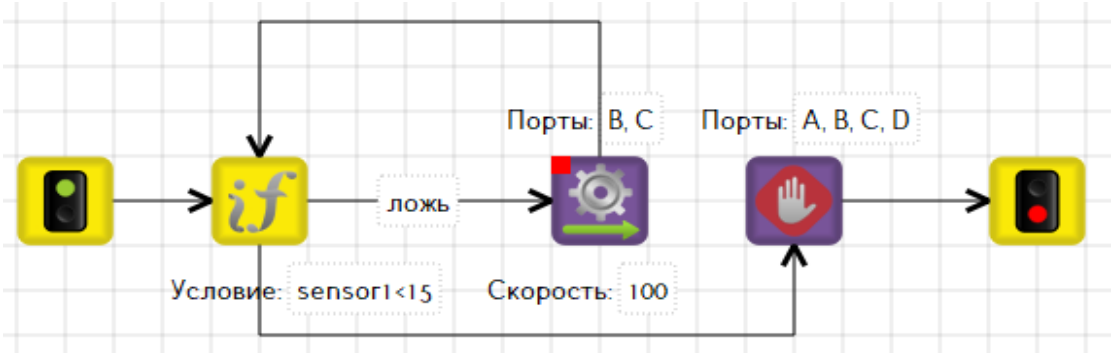
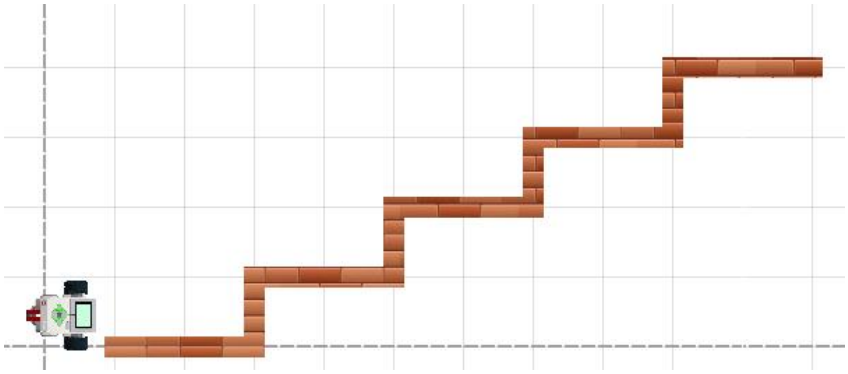


Рисунок 12.7 – Программа к задаче 12.2 с использованием цикла с предусловием.

На рисунке 12.7 представлен пример программы с использованием цикла с предусловием. В данном цикле сначала проверяется условие. Если условие принимает ложное значение, то организуется проезд вперед. Если условие принимает истинное значение, происходит выход из цикла.

Упражнения для самостоятельного решения

1. Используя цикл, организовать объезд вокруг квадратного объекта.
2. Нарисовать 4 ряда по 4 одинаковых квадрата в 2D модели.
3. Подняться по лестнице, у которой 5 ступенек, без датчиков.



4. Используя цикл, составить программу вывода на экран контроллера цвета объекта, подносимого к датчику цвета.
5. Нарисовать окружность с более четко прорисованной нижней половиной, для этого нужно 3 раза проехать туда-обратно по нижней половине окружности, и только после этого дорисовать верхнюю дугу. Повторить это действие 5 раз.
6. Напишите программу, в которой робот рисует четырехцветную линию, цвета выбираются случайным образом.
7. Решить задачу «Кегельринг» с использованием цикла.

13. Параллельные задачи

Многие практические задачи требуется решать в реальном времени, для этого может потребоваться большой объем вычислений. Современные компьютеры могут выполнять несколько операции одновременно. Такой процесс называется параллельным вычислением, основная цель которых – уменьшение времени решения задачи. Контроллеры – не исключение, для организации параллельных процессов в TRIK Studio используются блоки «Параллельные задачи» и «Слияние задач».

Использовать параллельные задачи в цикле нельзя, так как процессы будут множиться до переполнения памяти контроллера. Кроме того, параллельные процессы должны быть независимы друг от друга, то есть решаемые задачи имеют разные цели, например, воспроизведение звука при движении по траектории.

Рассмотрим пример использования параллельных задач.

В задаче 12.2 (см. раздел 12) было организовано движение до стены с отъездом назад при ее обнаружении. Добавим в задаче постоянный вывод на экран информации о расстоянии до объекта. В этом случае нужно добавить блок «Параллельные задачи» и в одной из задач в бесконечном цикле организовать вывод показаний на экран (рисунок 13.1). При этом не забываем в редакторе свойств блока «Напечатать текст» в строке «Вычислять» поставить флажок «истина» для того, чтобы на экране в качестве текста выводилось значение величины *sensor1*.

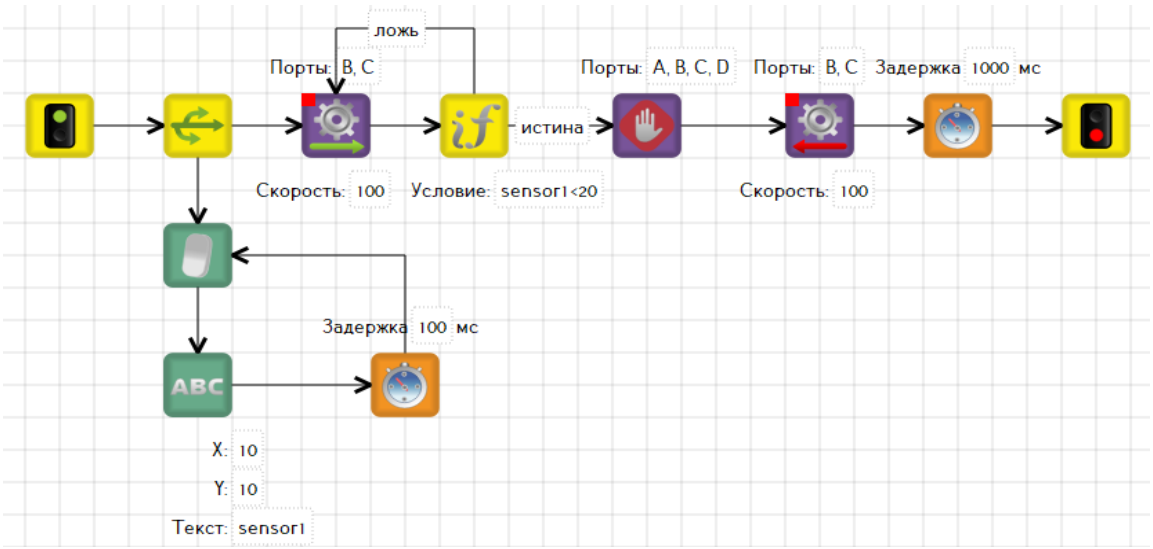


Рисунок 13.1 – Вывод на экран информации о расстоянии до объекта.

Таким образом, в программе параллельно организовано два цикла. Цикл с условием отвечает за движение до стены и отъезд назад. Бесконечный цикл обеспечивает постоянный вывод на экран контроллера расстояния до стены.

Но в этой программе имеется существенный недостаток: задача, отвечающая за вывод информации на экран, не останавливается, соответственно программа также не завершит выполнение. Чтобы вовремя завершить выполнение этой задачи, можно поступить следующим образом.

Первый способ – необходимо добавить счетчик (*i*), который будет отвечать за подсчет итераций ветви, где организован вывод информации о расстоянии (рисунок 13.2).

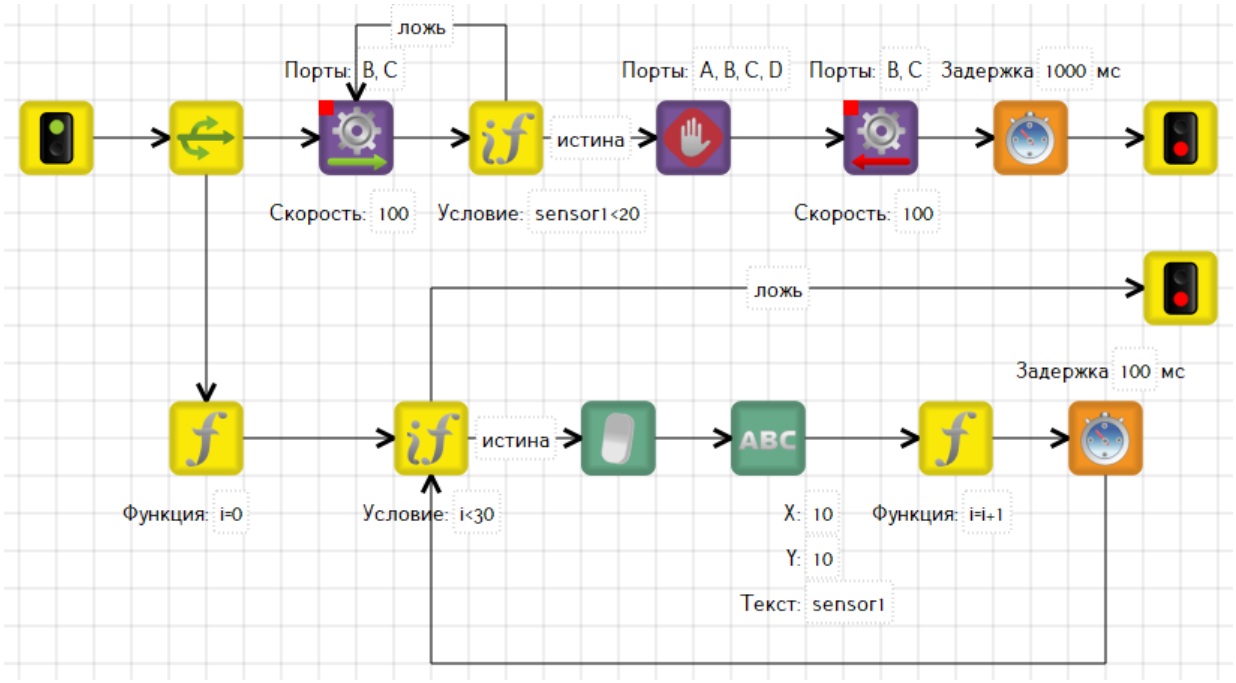


Рисунок 13.2 – Завершение программы с использованием счетчика.

В условии выхода из цикла необходимо подобрать такое значение счетчика, при котором, будет выполнена параллельная ей задача (проезд до стены). Если условие принимает истинное значение, то выводим информацию на экран и увеличиваем значение счетчика на 1. Если условие ложно, то происходит завершение цикла.

Этот способ также имеет недостаток, так как в каждом конкретном случае необходимо подбирать предельное значение счётчика для выполнения главной задачи.

Рассмотрим второй способ. Он заключается в отправке сообщения из главного потока некоторого сообщения-маркера, а в параллельной ему задаче проверяется, пришло ли сообщение. По пришедшему маркеру происходит завершение цикла.

Но сначала параллельным задачам необходимо дать имена. Имя главного потока – *main*, вспомогательную задачу обозначим, например, *a1* (рисунок 13.3).

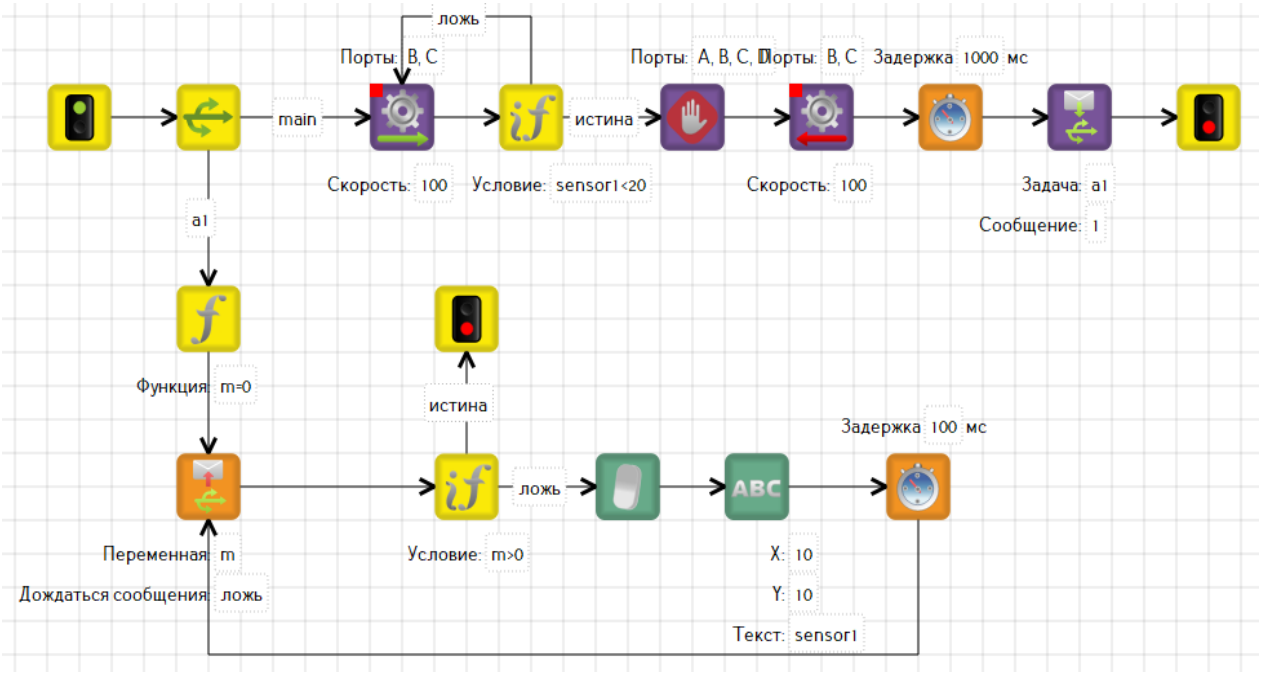


Рисунок 13.3 – Завершение программы с использованием сообщений.

Затем в главном потоке после выполнения всех действий добавляем блок «Отправить сообщение в задачу» и в свойствах указываем название задачи (в нашем случае *a1*) и сообщение (в нашем примере *1*). В задаче *a1* добавляем переменную *m=0*, затем блок «Получить сообщение из другой задачи», в свойствах которого указываем переменную *m* и в строке «Дождаться сообщения» выставляем «ложь», то есть значение переменной будет равно нулю. А далее в условии проверяем, пришло сообщение или нет. Если сообщение не пришло (*m=0*), то продолжаем выводить информацию на экран. Если сообщение пришло (*m=1*), то завершаем задачу и, следовательно, завершаем всю программу.

И наконец, третий способ, наиболее простой, – завершить вторую задачу из главной задачи с помощью блока «Завершить задачу». Для этого достаточно добавить данный блок и указать имя той задачи, которую необходимо завершить (рисунок 13.4).

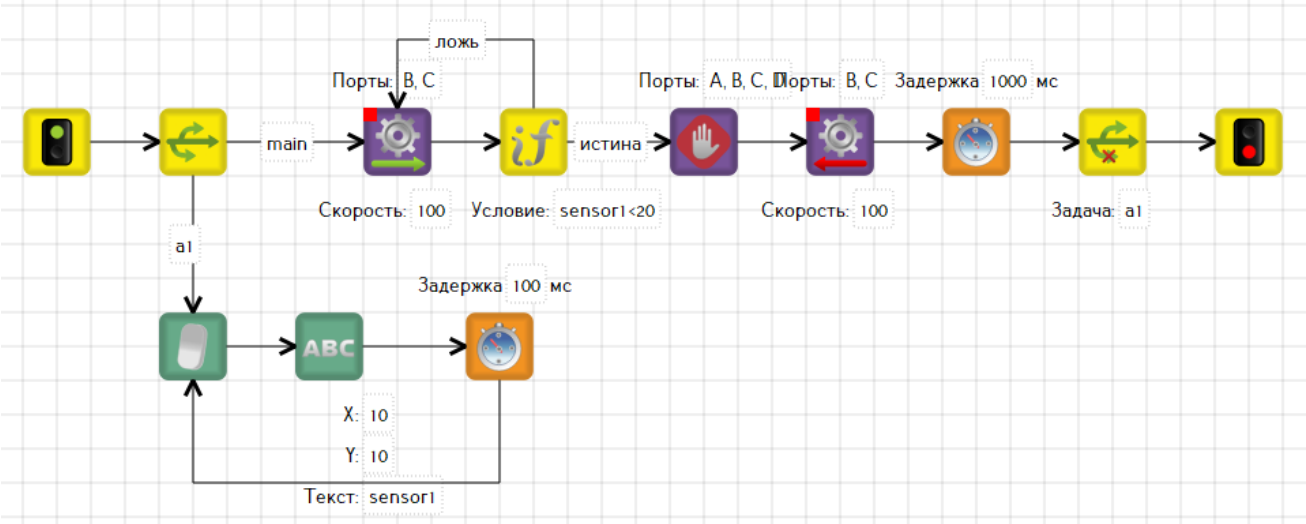


Рисунок 13.4 – Завершение программы с использованием блока «Завершить задачу»

Примечание: Блоки «Слияние задач», «Завершить задачу», «Получить сообщение из другой задачи» недоступны для программирования в режиме EV3.

Упражнения для самостоятельного решения

1. Воспроизводить звук при движении робота по какой-либо траектории.
2. Робот проигрывает музыку и управляет цветовым сопровождением (подсветка контроллера).
3. Робот едет по траектории, с включенной сиреной (звуком) и проблесковым маячком (подсветка контроллера).
4. Придумайте задачу, для решения которой потребуется 4 параллельных потока.
5. Проходя некоторую траекторию, робот выводит на экран контроллера показания своего секундомера.

14. Вычисления

При изучении явлений природы и общества мы постоянно сталкиваемся с различными величинами. Одни из них являются постоянными, неизменяющимися, а другие меняются в зависимости от условий среды. Первые величины называются константами, вторые – переменными. В программировании переменная – это объект, которому дано имя и определенное значение.

Для объявления переменных и задания им определенных значений в TRIK Studio используется блок «Инициализация переменной» (см. раздел 2). В свойствах блока указывается имя переменной и ее значение.

Переменные имеют определенный тип, который должен быть известен во время компиляции. При этом язык TRIK Studio не требует (и даже не позволяет!) явно писать типы переменных, так как используется автоматический вывод типов по использованию переменных. Например, по выражению *a = 1* среда «поймёт», что тип a – целое.

Базовые типы языка таковы:

- Логический (булевый) тип, принимает значения true (истина) и false (ложь).
- Вещественное число, использующее 64-битное представление binary64 стандарта IEEE 754 (позволяет хранить значения до 1.7E+308).
- Целое число, использующее 32-битное знаковое представление (позволяет хранить значения от -2,147,483,648 до 2,147,483,647).
- Строка, позволяет хранить символьные строки произвольной длины в кодировке UTF-8.
- Нулевой тип, имеющий только одно значение nil и означающий отсутствие "настоящего" значения.
- Тип "Таблица" (или массив), позволяющий хранить произвольное количество значений произвольного (но для каждого значения в одной таблице одинакового) типа, в том числе и другие таблицы, и обращаться к значениям по индексу.

Все переменные, используемые в программе, являются глобальными, то есть их значения могут использоваться в любом блоке программы.

Задача 14.1. Составить программу бесконечного движения вперёд-назад.

Воспользуемся блоком «Инициализация переменной» для объявления переменной *x*. В задаче к переменной *x* идет обращение 11 раз, и на каждом шаге значение меняется на противоположное: либо *1*, либо *(-1)*. При этом мощность моторов также является величиной изменяемой: *100* или *(-100)*, так как скорость умножаем на величину *x*. При мощности *(-100)* действие «Моторы вперед» становится действием «Моторы назад» (рисунок 14.1).

Рисунок 14.1 – Программа к задаче 14.1.

При объявлении сразу нескольких переменных или больших вычислений используется блок «Функция». Пример использования блока представлен на рисунке 14.2.

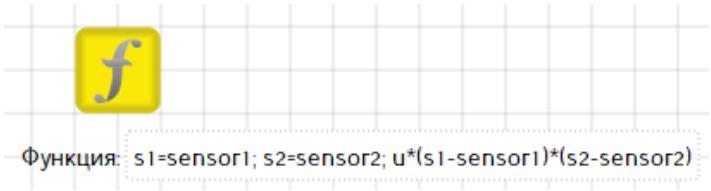


Рисунок 14.2 – Объявление нескольких переменных и вычисления.

В правой части выражения разрешено использовать круглые скобки, числа, базовые арифметические операции, ранее определенные переменные, зарезервированные переменные, список которых можно просмотреть на панели «Переменные» в правой части экрана.

Рассмотрим подробнее о доступных математических выражениях и операциях.

В арифметических выражениях доступны следующие операции:

- + – сложение, применимо к вещественным и целым значениям, результат целый, если оба аргумента целые, иначе вещественный.
- - – вычитание, применимо к вещественным и целым значениям, результат целый, если оба аргумента целые, иначе вещественный.
- * – умножение, применимо к вещественным и целым значениям, результат целый, если оба аргумента целые, иначе вещественный.
- / – деление, применимо к вещественным и целым значениям, результат вещественный.
- // – целочисленное деление, применимо к целым значениям, результат целый.
- ^ – возведение в степень, применимо к вещественным и целым значениям, результат вещественный.
- % – остаток от деления, применим к целым значениям, результат целый.

В математических выражениях можно использовать функции, доступные для любого конструктора:

- time – время в миллисекундах с начала работы программы;
- sin – синус, аргумент передаётся в градусах;
- cos – косинус, аргумент передаётся в градусах;
- ln – натуральный логарифм;
- exp – экспонента (e в степени аргумента);
- asin – арксинус;
- acos – арккосинус;
- atan – арктангенс;
- sgn – знак, возвращает 1, если аргумент положительный, -1, если отрицательный, и 0, если аргумент равен нулю;
- sqrt – квадратный корень аргумента, в случае, если аргумент отрицательный, переменной будет присвоено значение "nan" (Not A Number);
- abs – модуль аргумента;

- ceil – округляет переданный аргумент до целого в большую сторону;
- floor – округляет переданный аргумент до целого в меньшую сторону;
- random – случайное число в интервале от 0 до переданного аргумента.

Приведем пример использования встроенной функции.

Задача 14.2. Вывести на экран контроллера время, прошедшее с начала исполнения.

В данном случае необходимо использовать встроенную функцию *time*. Организуем цикл, а результатом будет являться постоянный вывод на экран времени выполнения программы (рисунок 14.3). Не забываем в редакторе свойств блока «Напечатать текст» ставить флажок «Вычислять».

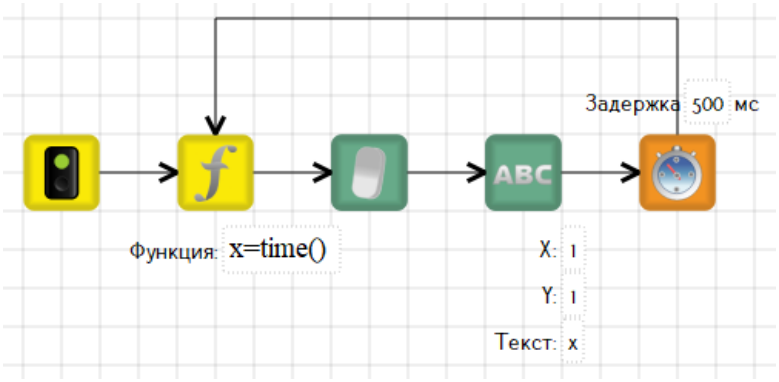


Рисунок 14.3 – Программа к задаче 14.2.

Уточним, что такое зарезервированные переменные. Как мы уже знаем к контроллеру можно подключать четыре датчика и четыре мотора. В каждый момент времени датчики возвращают некоторое числовое значение на контроллер. Это и есть значение переменных *sensorN* и *encoderN*.

Например, к порту 1 подключен датчик ультразвука. Робот находится вплотную около стены. Значение переменной *sensor1* равно 0. После перемещения робота на некоторое расстояние от стены значение изменится в большую сторону. Таким образом, значение переменной *sensor1* – это то, что показывает датчик в данный момент времени.

Кроме переменных *sensorN* и *encoderN* имеются переменные, отвечающие за состояние кнопок на корпусе робота: кнопки "Назад", "Вниз", "Ввод", "Влево", "Вправо", "Вверх" соответствующие переменным *buttonBack*, *buttonDown*,

buttonEnter, *buttonLeft*, *buttonRight*, *buttonUp*. Значение *0* будет соответствовать не нажатой кнопке, *1* – нажатой.

Список переменных можно просмотреть на панели «Переменные» как в режиме «Отладка» (рисунок 14.4), так и в режиме «Редактор» (рисунок 14.5).

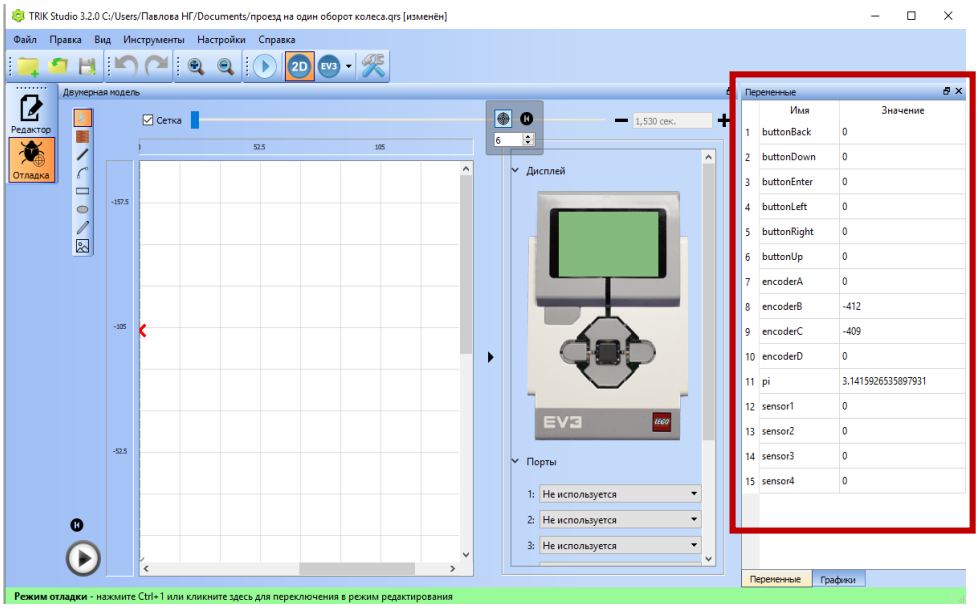


Рисунок 14.4 – Панель «Переменные» в режиме «Отладка».

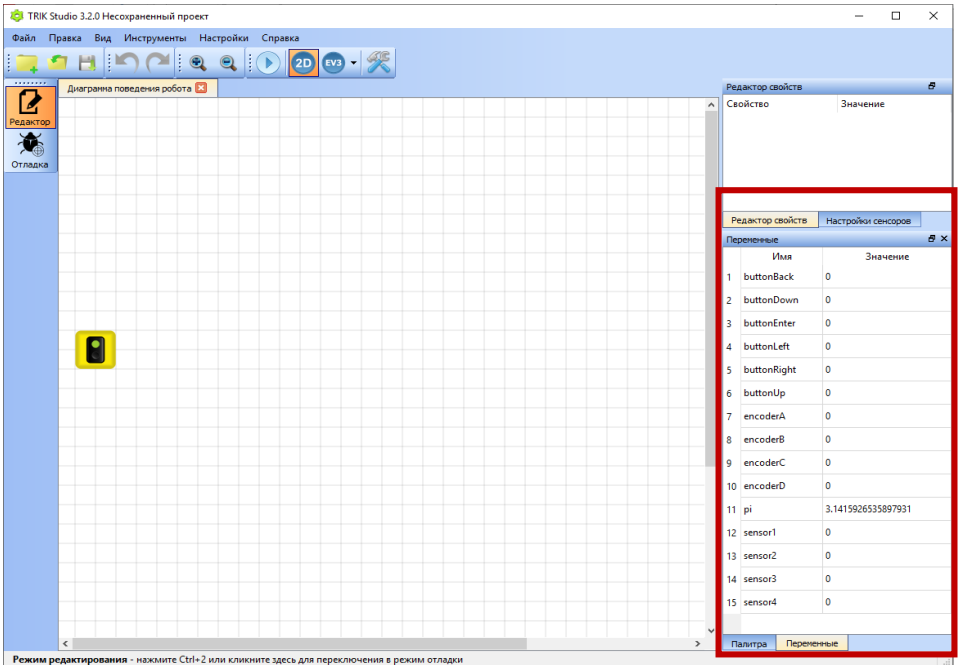


Рисунок 14.5 – Панель «Переменные» в режиме «Редактор»

Задача 14.3. Найти среднее значение серого, используя датчик цвета в режиме яркости отраженного света (калибровка датчика цвета).

Найти значение серого можно вручную и автоматически, то есть выполнить вычисления в программе.

Ручная калибровка заключается в следующем. Помещаем робота так, чтобы датчик цвета располагался над белой поверхностью, фиксируем показания на экране контроллера. Затем помещаем робота над черной поверхностью и снова фиксируем показания. Допустим, показания над белой поверхностью составило 75%, а над черной – 15%. Осталось вычислить среднее значение: (75+15)/2=45.

А теперь реализуем калибровку автоматически.

Для этого подключим датчик цвета, выбрав режим света. При старте робот находится над белой поверхностью, снимает показания с датчика и записывает в переменную *white*, используя блок «Функция». Затем движется до черной линии, останавливается, снимает показания с датчика и записывает в переменную *black*. В блоке «Функция» организуем вычисление *gray=(white+black)/2* и выводим значения переменных *white, black* и *gray* на экран (рисунок 14.6).

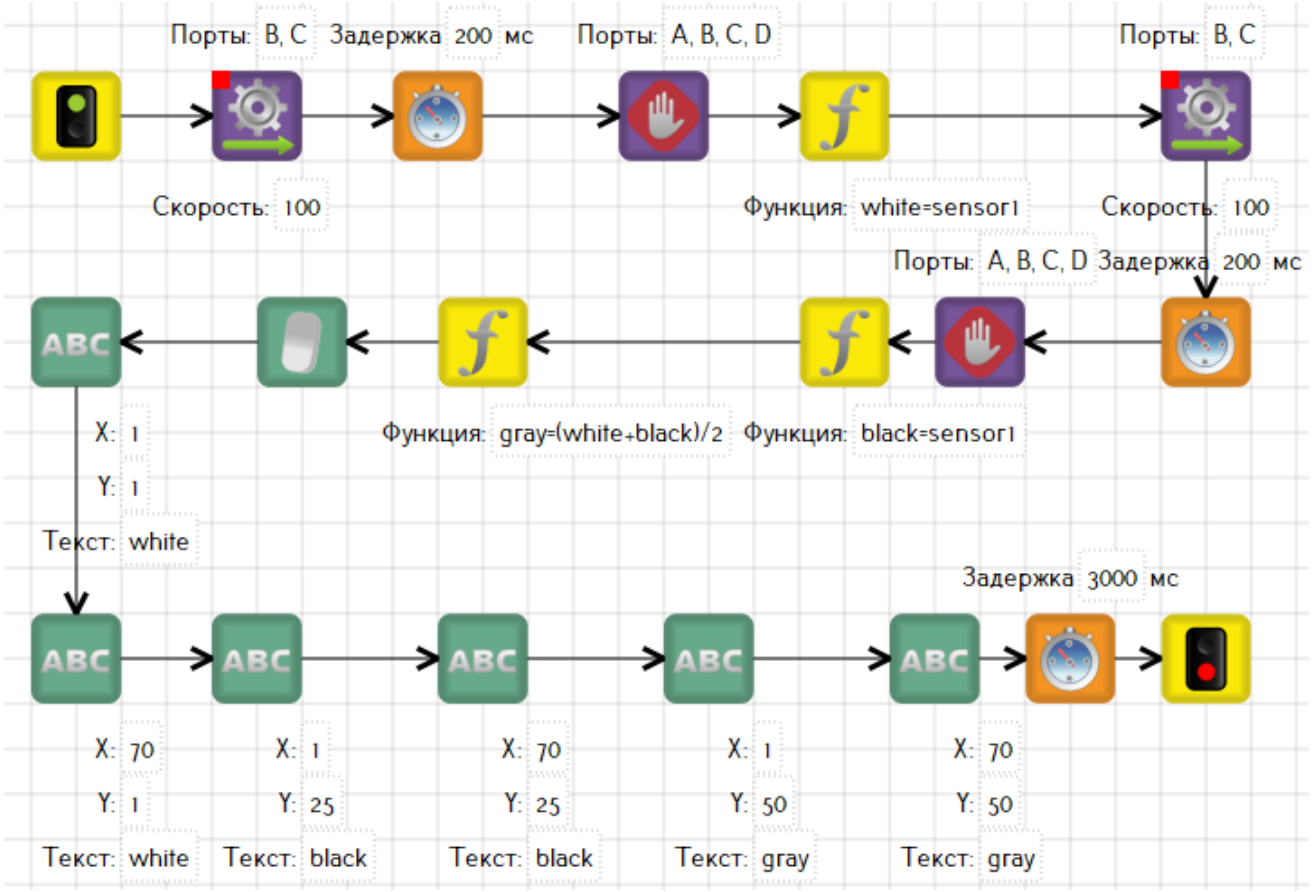


Рисунок 14.6 – Программа к задаче 14.3.

В программе используется по два одинаковых блока «Напечатать текст» для того, чтобы в первом блоке выводился текст – название цвета, а во втором – его значение. При запуске программы, на экране получим изображение как на рисунке 14.7.

Рисунок 14.7 – Результат выполнения программы.

Задача 14.4. На пол наклеены цветные линии. Робот проезжает над ними. Подсчитать количество красных линий и вывести на экран контроллера (рисунок 14.8).

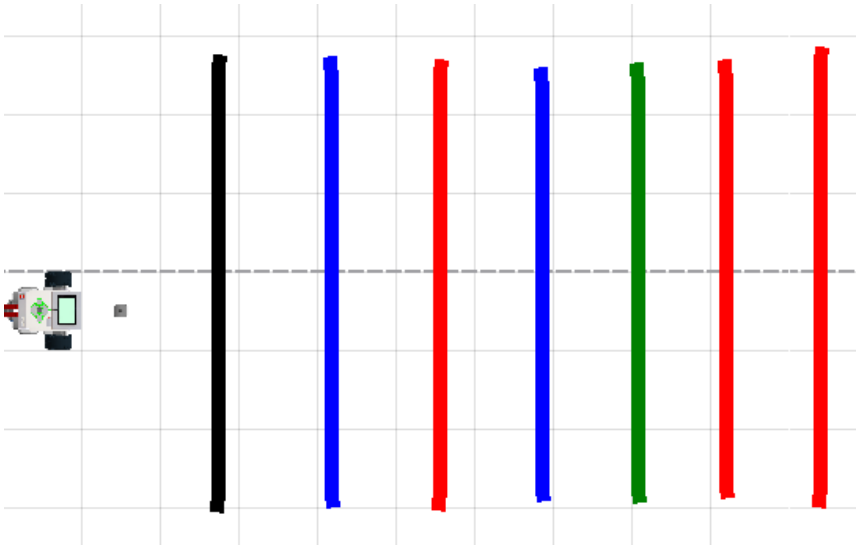


Рисунок 14.8 – Внешний вид поля.

Подсчет организуем в бесконечном цикле. В переменной x будем хранить количество линий. Ее начальное значение равно нулю. При обнаружении линии нужного цвета значение переменной увеличивается на 1 и это значение выводим на экран (рис.14.9). Значение таймера нужно подбирать опытным путем, чтобы датчик цвета не подсчитывал одну линию несколько раз.

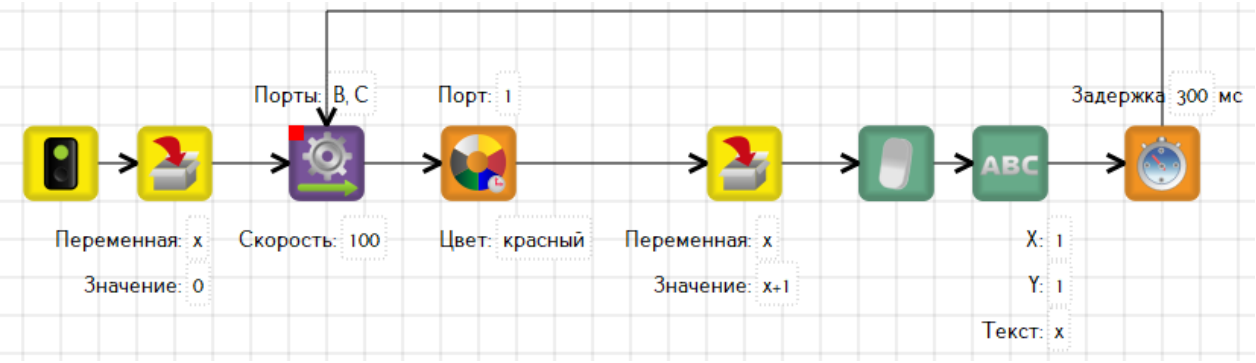
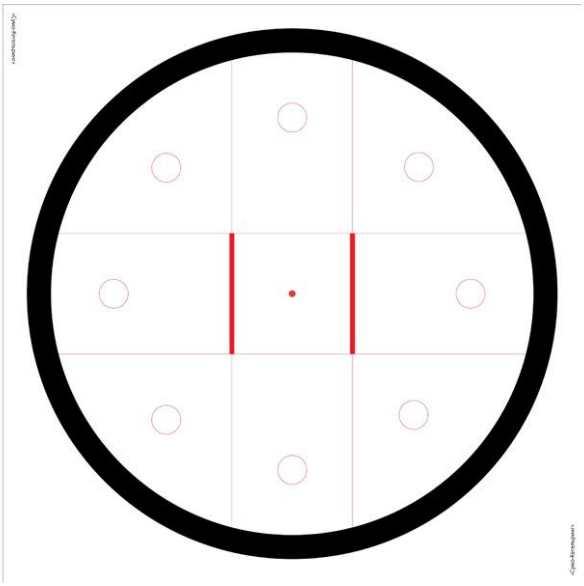


Рисунок 14.9 – Программа к задаче 14.4.

Упражнения для самостоятельного решения

1. Составить программу, в результате которой при нажатии на кнопки контроллера «Вправо» и «Влево» должна включится красная подсветка, а на кнопки «Вверх» и «Вниз» – оранжевая. Примечание: использовать зарезервированные переменные *buttonUp, buttonDown, buttonLeft, buttonRight*.
2. Робот движется в направлении к стене. Составить программу вывода на экран расстояния до стены.
3. Робот проезжает над цветными линиями. Составить программу вывода на экран контроллера количества красных и черных линий.
4. Составить программу проезда роботом вперед на расстояние 30 см. Все вычисления выполнить в программе (примеры расчетов см. в разделе 8).
5. Составить программу поворота робота на угол 30°. Все вычисления выполнить в программе (примеры расчетов см. в разделе 8).
6. Сконструировать прибор для подсчета всех входящих в помещение. Результат вывести на экран.
7. Хаотичное движение: на каждый мотор робота подается случайная мощность.
8. Подсчитать все кегли, расположенные на метках поля «Кегельринг».



15. Простейшие регуляторы для управления роботом

На практике неоднократно встречается такая ситуация, что показания одинаковых приборов могут различаться. Это связано с рядом факторов, таких как условия окружающей среды, несовершенство изделий и т.п. В наборе EV3 датчики и моторы, используемые для конструирования, также не являются идеальными.

Чтобы учесть разницу в показаниях, применяется автоматический регулятор – устройство, которое следит за состоянием объекта управления как системы и вырабатывает для нее управляющие сигналы в соответствии с заданным законом управления. Согласно определению, робот с помощью датчиков измеряет регулируемую величину и вырабатывает требуемое воздействие на моторы.

Регулятор могут быть релейным, пропорциональным, дифференциальным, интегральным и др. Они применяются в следующих задачах:

- синхронизация моторов;
- управление положение мотора;
- следование вдоль стены;
- следование по заданной траектории;
- удержание груза;
- движение по линии;

и другие.

Рассмотрим пример синхронизации моторов.

Задача 15.1. Синхронизировать моторы при прямолинейном движении.

Значения, возвращаемые двумя датчиками вращения моторов, могут не совпадать, в результате чего движение робота непрямолинейно. В программе используем метод релейного регулирования, суть которого заключается в выработке управляющего воздействия, в данном случае на моторы. Для этого в каждый момент времени вычисляется ошибка (u) – среднее значение показаний датчиков вращения моторов (энкодеров), а результат используется для воздействия на моторы, за счет чего движение робота становится прямолинейным (рисунок 15.1).

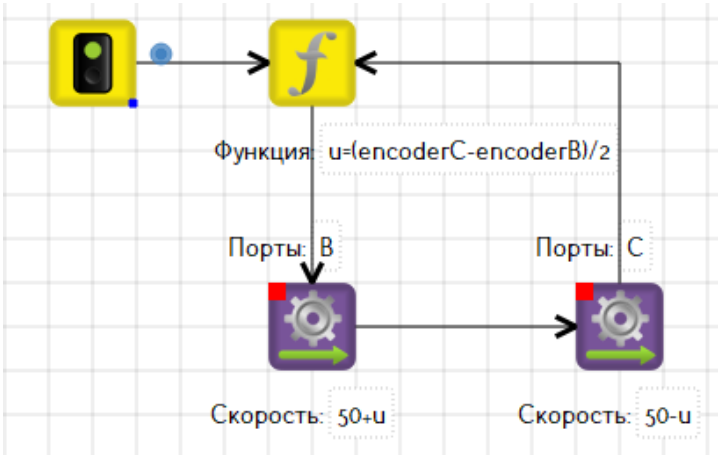


Рисунок 15.1 – Пример синхронизации моторов.

Задача 15.2. Стабилизировать мотор в положении 45 градусов.

При решении задачи для исправления ошибки также воспользуемся релейным регулятором. В данном случае, ошибка – разность показаний энкодера и того положения, в котором необходимо удерживать мотор. Таким образом, мотор будет колебаться около положения 45°.

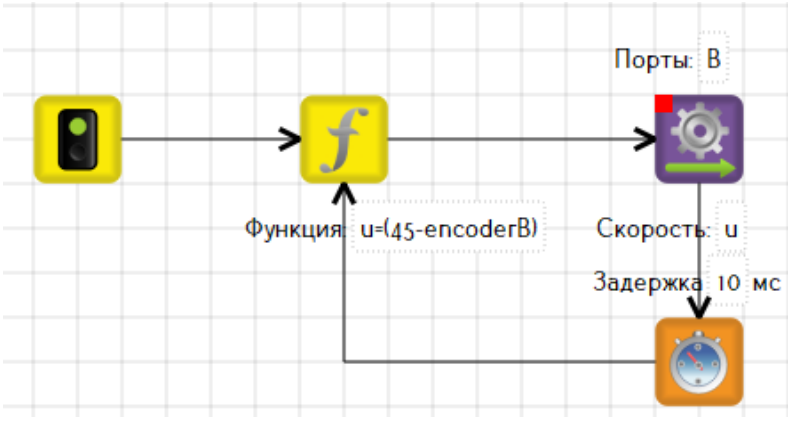


Рисунок 15.2 – Программа к задаче 15.2 на основе релейного регулятора

Более точное исправление ошибки достигается использованием пропорционального регулятора, суть которого заключается в управлении по управлению системы от заданного состояния.

Пропорциональный регулятор (П-регулятор) – это устройство, оказывающее управляющее воздействие на объект пропорционально его отклонению от заданного состояния и выражается формулой:

$$u = k \cdot e,$$

где *k* – коэффициент усиления регулятора, *e* – динамическая ошибка (отклонение регулируемой величины от ее заданного значения).

Исправим алгоритм (рисунок 15.2), используя коэффициент усиления 2.5 (его можно подобрать опытным путем). Поправка *u =2.5·(45-encoderB)*, где *(45-encoderB)* – отклонение энкодера от положения 45 градусов. Соответственно стабилизация мотора на П-регуляторе будет выглядеть как на рисунке 15.3.

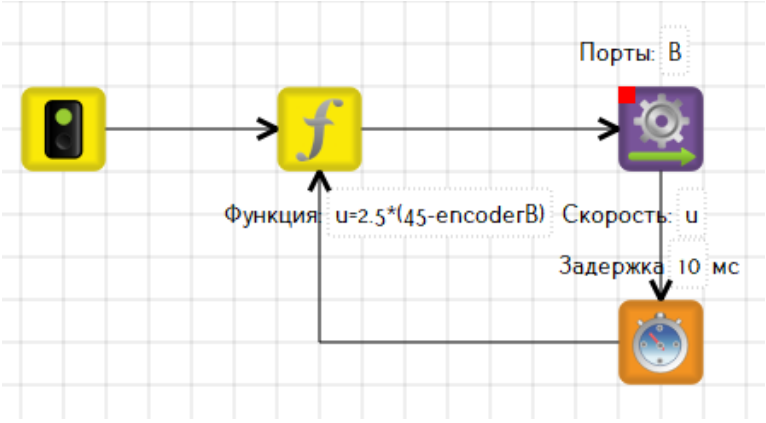


Рисунок 15.3 – Программа к задаче 15.2 на основе П-регулятора

Задача 15.3. Составить программу движения робота вдоль стены на определенном расстоянии L (рисунок 15.4). S1 – текущее показание датчика.

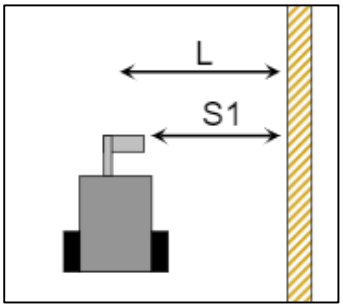


Рисунок 15.4 – Положение робота у стены.

Для движения вдоль стены необходимо использовать ультразвуковой датчик (УЗ), который должен быть вынесен вперед и расположен перпендикулярно движению робота и направлен в сторону стены.

Моторы двигаются со средней скоростью 50% от максимума, но при отклонении от заданного курса на них осуществляется управляющее воздействие *u* (на мотор B *50+u*, на мотор C *50-u*): *u=k*(S1-L)*, где *k* - некий усиливающий коэффициент, определяющий воздействие регулятора на систему.

Таким образом, при *S1=L* робот не меняет курса и едет прямо. В случае отклонения его курс корректируется (рисунок 15.5).

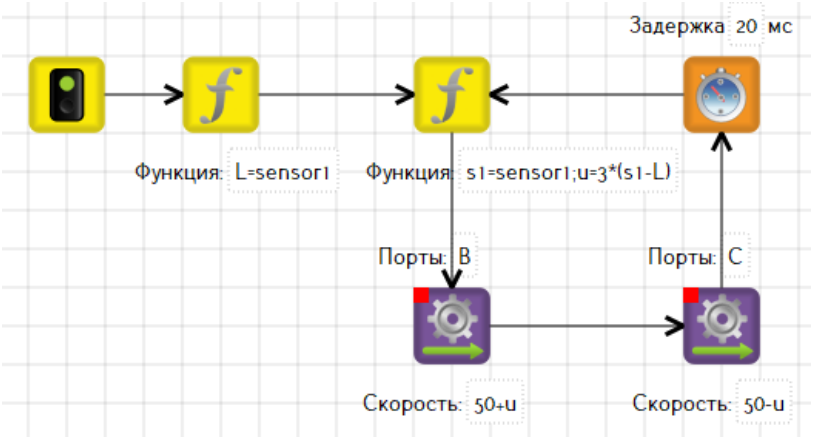


Рисунок 15.5 – Движение вдоль стены на основе П-регулятора.

В данном случае П-регулятор будет эффектно работать только при малых углах отклонения. Например, если робот направлен от стенки, но находится по отношения к ней ближе заданного расстояния, на моторы поступит команда еще сильнее повернуть от стенки, в результате чего с ней может быть утерян контакт (датчик расстояния получает отраженный сигнал практически только от перпендикулярной поверхности).

Для защиты от подобных ситуаций необходимо добавить в регулятор дифференциальную составляющую (ПД-регулятор), которая будет следить за направлением движения робота.

Поправка *u=k1*(sensor1-L)+k2*(sensor1-S)*, где *S* - расстояние на предыдущем шаге.

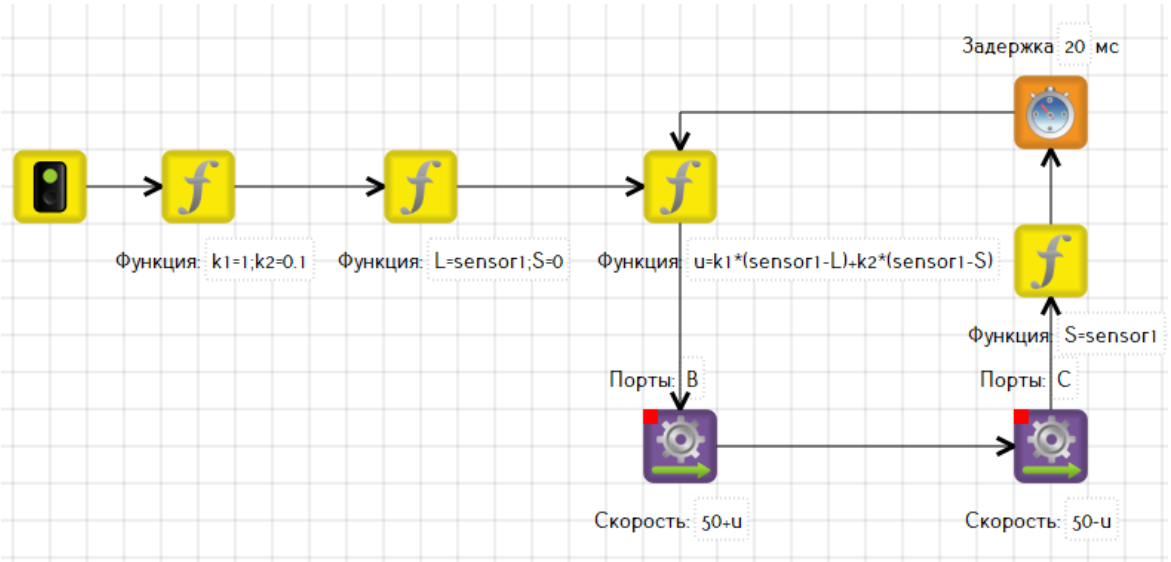


Рисунок 15.6 – Движение вдоль стены на основе ПД-регулятора.

В данном примере необходимо подобрать подходящие значения коэффициентов *k1* и *k2*. Обычно подбор начинается с пропорционального коэффициента (*k1*) при нулевом дифференциальном (*k2=0*). Когда достигнута некоторая стабильность на небольших отклонениях, добавляется дифференциальная составляющая.

Описанный выше робот может объезжать стены только при малых отклонениях от прямой линии. Рассмотрим вариант, при котором на пути движения будут возникать серьезные повороты, вплоть до прямых углов, например, на поле Лабиринт. Для этого потребуется внести модификацию в конструкцию робота (рисунок 15.7).

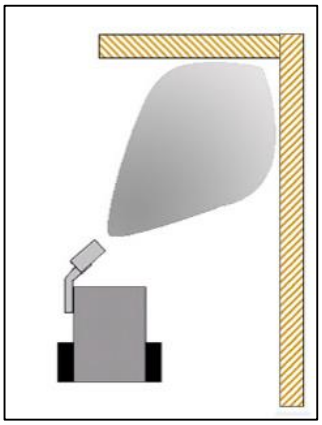


Рисунок 15.7 – Расположение ультразвукового датчика под углом.

Во-первых, робот должен будет смотреть не только направо, но и вперед. Для этого нужно разместить датчик УЗ не перпендикулярно курсу движения, а под острым углом. Таким образом, робот будет видеть препятствия спереди, и более стабильно будет придерживаться курса вдоль стены, постоянно находясь на грани видимости.

Важное замечание. При старте робота надо будет направлять датчиком строго на стену, чтобы процесс считывания начального значения прошел без помех.

Задача 15.4. Собрать манипулятор и составить программу для удержания груза.

Использование манипуляторов стало обыденным явлением. На сегодняшний день не одно промышленное предприятие не обходится без них.

Кроме того, манипуляторы устанавливаются и на мобильных роботах, чтобы расширить возможности управления в труднодоступных для человека местах.

Итак, манипулятор – управляемое устройство, предназначенное для выполнения сложных действий, аналогичных движениям руки человека, в том числе, для управления положением предметов.

На рисунке 15.8 представлен манипулятор с использованием трех моторов. Первый мотор отвечает за горизонтальное перемещение захвата. Второй мотор отвечает за вертикальные перемещения. Третий – за захват объектов. Инструкцию по сборке данного манипулятора можно найти в среде программирования LEGO MINDSTORMS EV3 в разделе «Инструкции по сборке» – «Рука робота H25»

Рисунок 15.8 – Манипулятор «Рука робота H25».

Приведем пример программы удержания груза в заданном положении с использованием П-регулятора (рисунок 15.9).

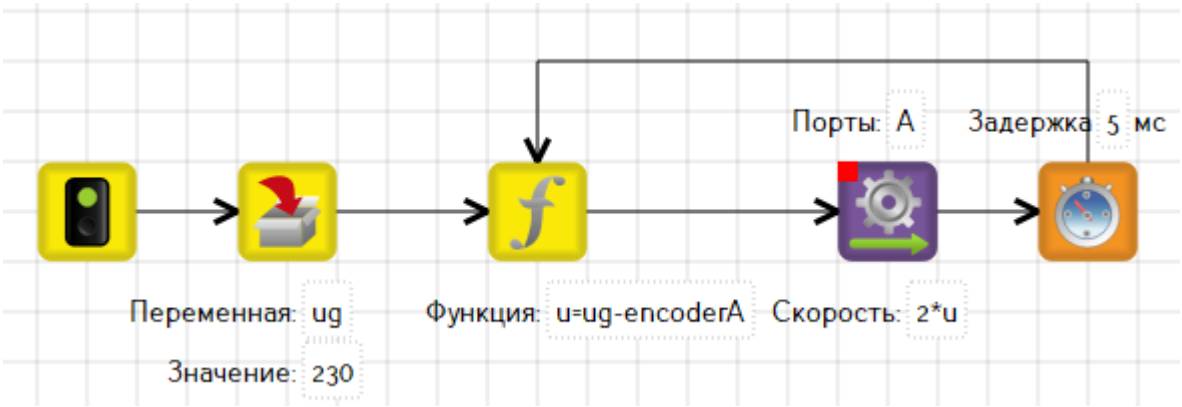


Рисунок 15.9 – Программа удержания груза.

В переменной *ug* задаем значение того положения манипулятора, в котором его необходимо удерживать. Ошибка определяется как разность переменной *ug* и показаниями энкодера, а затем подается на управление мотора. В качестве коэффициента используется число 2.

Данная программа может быть использована в качестве подпрограммы. Но тогда необходимо внести некоторые изменения. В представленной выше программе используется бесконечный цикл. А подпрограмма не может быть цикличной, поэтому необходимо добавить условие выхода из цикла (рисунок 15.10), а переменную *ug* вынести из подпрограммы и расположить в основной программе.

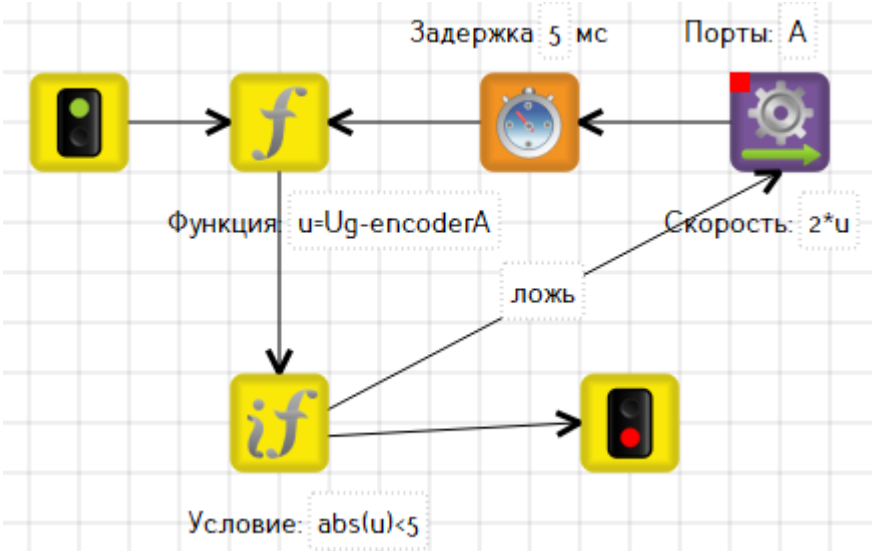


Рисунок 15.10 – Программа удержания груза для использования в подпрограмме.

На рисунке 15.11 представлена программа с фиксированием мотора в двух положениях – 0 и 230 градусов. Подпрограмма *Ungle* представлена на рисунке 15.10.

Рисунок 15.11 – Программа с фиксированием мотора в двух положениях

Упражнения для самостоятельного решения

1. Сделать 5 замеров расстояния до препятствия, вывести значения датчиков на экран контроллера. Сравнить значения и вычислить среднее арифметическое показаний датчика.

2. Роботу требуется повернуть за угол. Составить программу, учитывая, что угол прямой, острый.

3. Препятствие представляет собой прямоугольную коробку. Требуется обойти ее по периметру.

4. Коридор ограничен двумя стенами. Робот должен держаться середины этого коридора. Используется два датчика расстояния.

5. Внутри коридора расположены три стоп-линии, на которых робот должен остановиться и сделать паузу в движении 5 секунд. Задача – пройти коридор.

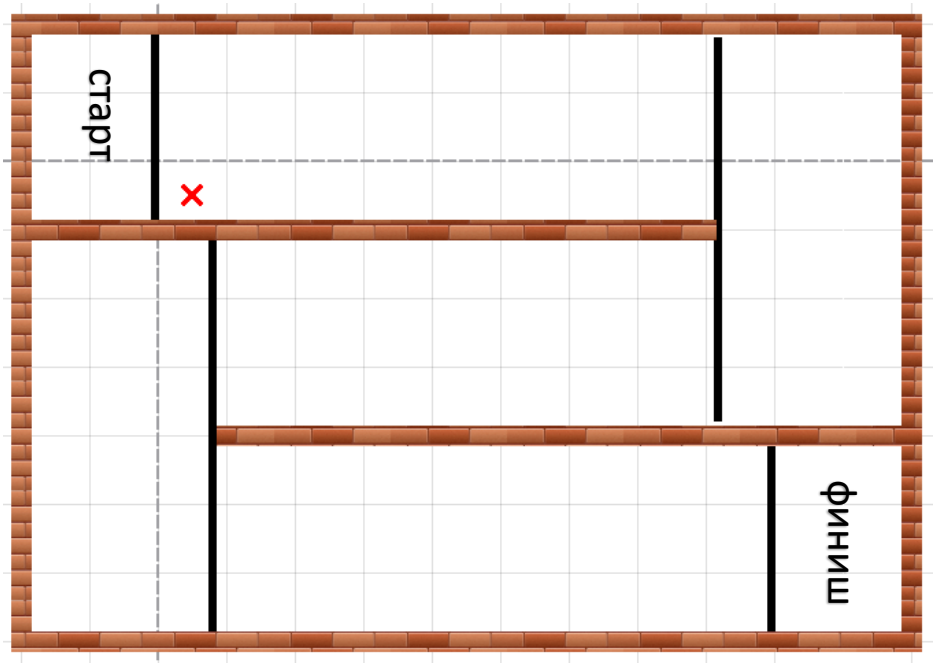
6. По курсу робота находится препятствие. Ему надо его объехать так, чтобы выдержать заданный курс. Сколько и каких датчиков потребуется?

7. Объект находится перед роботом. Его нужно взять и переместить.

8. Необходимо доехать до объекта и транспортировать его.

9. Построить башню: на площадку по очереди доставляют кубики, которые нужно поставить друг на друга.

10. Написать программу для прохождения трассы:



16. Движение по линии

Движение робота по черной линии встречается в разных видах состязаний. На сегодняшний день существует большое количество алгоритмов следования по линии, от простейших в вычислениях и реализации до очень сложных, которые разрабатываются для профессиональных систем управления.

Как правило, в соревнованиях используется белое поле с черной линией, вдоль которой необходимо следовать. Например, поле «Шорт-трек», приведенное на рисунке 16.1.

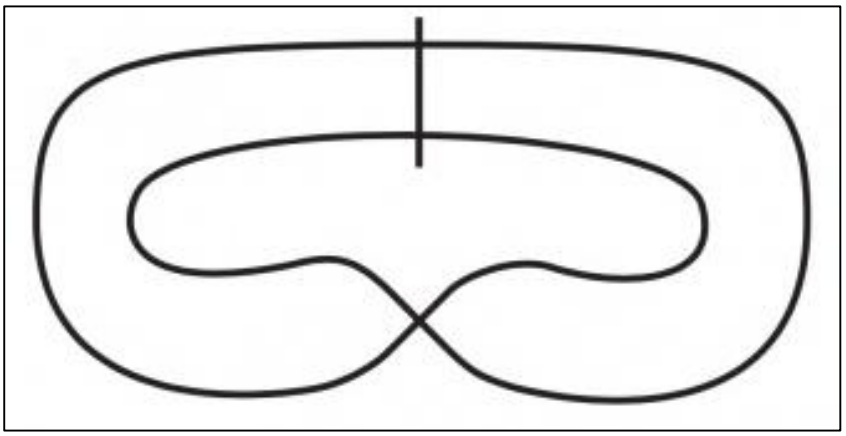


Рисунок 16.1 – Поле «Шорт-трек»

Также могут встретиться и участки черного поля с белой линией (рисунок 16.2), в этом случае говорят об инверсии. Если линии пересекаются, то это перекрестки, которые в задании часто требуется подсчитать.

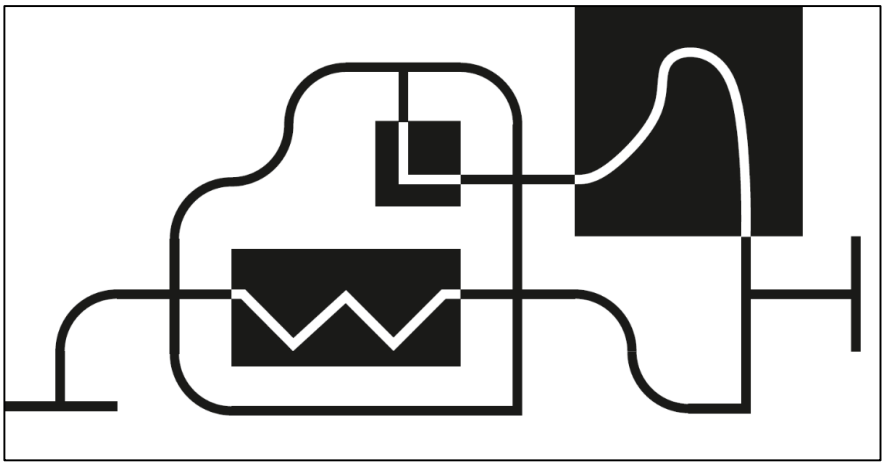


Рисунок 16.2 – Поле с участками инверсии и перекрестками.

Движение по линии может быть организовано с помощью одного датчика цвета (в режиме яркости отраженного света), расположенного с одной стороны линии или с помощью двух датчиков, расположенных по обе стороны линии. Во

втором случае, движения получаются более плавными, и робот практически никогда «не теряет» линию. Использование двух датчиков также позволяет обнаруживать перекрестки. Для скоростного прохождения сложных участков может использовано также три и даже четыре датчика цвета (рисунок 16.3).

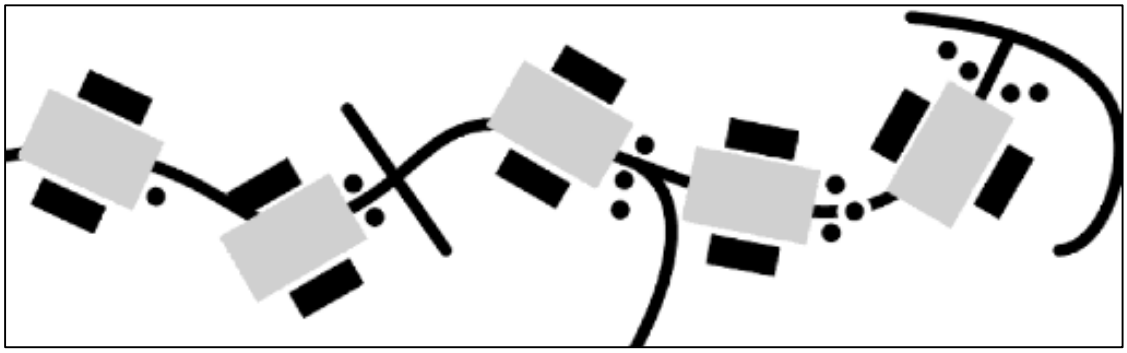


Рисунок 16.3 – Примеры расположения датчиков относительно линии.

При этом датчики должны быть расположены правильно. Учитывая, что робот осуществляет повороты относительно колес и является инерционной системой (между тем, как датчики зафиксируют изменение цвета, процессор подаст управляющий сигнал и моторы реализуют это движение, проходит определенное время), желательно чтобы робот смог заранее сгенерировать и отработать сигнал управления. Поэтому датчики необходимо выносить немного вперед. Но если вынести на значительное расстояние, то это может привести к сходу робота с линии. Также необходимо учитывать расстояние от поверхности поля до датчика, оно должно составлять примерно 0,5-0,7 см.

Рассмотрим основные алгоритмы движения по линии.

Перед началом реализации любого алгоритма следования по линии необходимо проводить калибровку датчиков. Она заключается в том, что нужно найти то состояние, к которому будет стремиться датчик. Так как робот будет двигаться на границе черного и белого цветов, то необходимо рассчитать числовое значение, соответствующее этому положению. Это состояние принято называть «*среднее значение серого*».

Калибровку можно проводить с внесением изменений в программу вручную или в автоматическом режиме (см. задачу 14.3).

Задача 16.1. Алгоритм движения по линии на одном датчике цвета с использованием релейного регулятора.

Этот алгоритм является самым простым: если датчик видит черный цвет, то робот поворачивает в одну сторону, если белый — в другую. Движение робота по линии получается резким, зигзагообразным и скорость робота невелика.

Итак, выполнив калибровку (значение серого равно 45), в цикле выполняем:

- если значение датчика больше среднего (датчик удаляется влево от черной линии), то левый мотор - *100*, правый – *(-100)*.
- если значение датчика меньше среднего (датчик оказался над черной линией), то левый мотор - 0, правый – 100 (рисунок17.4).

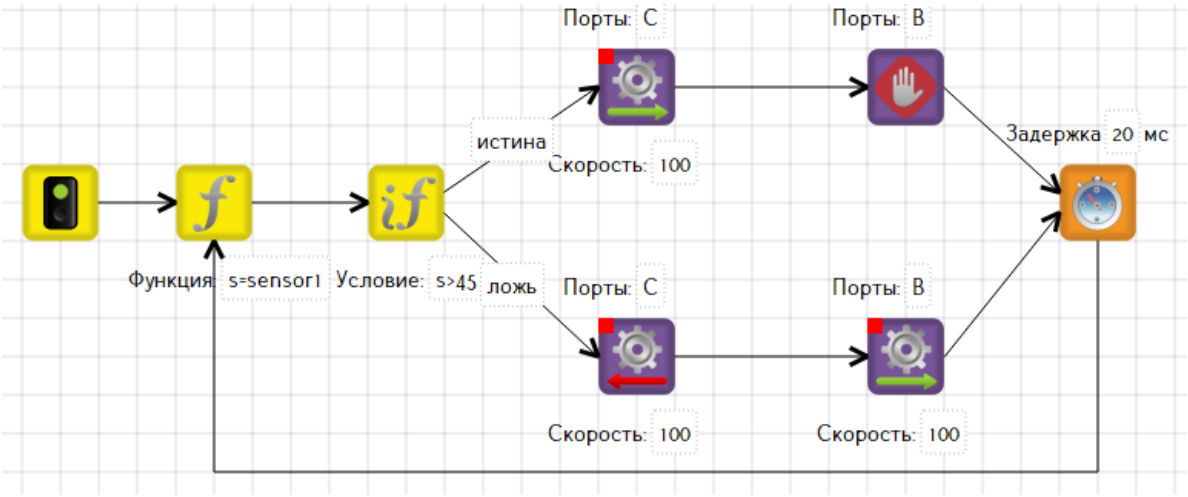


Рисунок 16.4 – Программа к задаче 16.1

В данном примере робота расположить нужно так, чтобы датчик находился слева от линии. Если поставить робота справа, то он «потеряет» её, так как первым движением повернет направо и будет крутиться вокруг правого колеса.

Если необходимо поставить робота справа от линии (сторона движения может быть существенной в том случае, когда для выполнения задания различно – ехать по внешней или внутренней стороне замкнутой линии, проезжать перекрестки, поворачивая в нужную сторону), то в существующей программе нужно изменить следующее:

- если значение датчика больше среднего (датчик удаляется влево от черной линии), то левый мотор - *(-100)*, правый *–100*.
- если значение датчика меньше среднего (датчик оказался над черной линией), то левый мотор - 100, правый – 0.

Задача 16.2. Алгоритм движения по линии на одном датчике цвета с использованием П-регулятора.

Движение робота по линии в данном примере будет более плавным и осмысленным.

В данном алгоритме происходит постоянное сравнение текущего значения датчика цвета со средним значением серого (то есть с состоянием, к которому стремится датчик). Значение, пропорциональное полученной разности, подается на блоки «Моторы вперед». Результатом, в зависимости от знака и величины полученного значения, является торможение или ускорение одного мотора и текущая скорость или торможение, и ускорение другого мотора, возвращающее робота к линии.

На рисунке 16.5 представлена схема П-регулятора на основе одного датчика цвета.

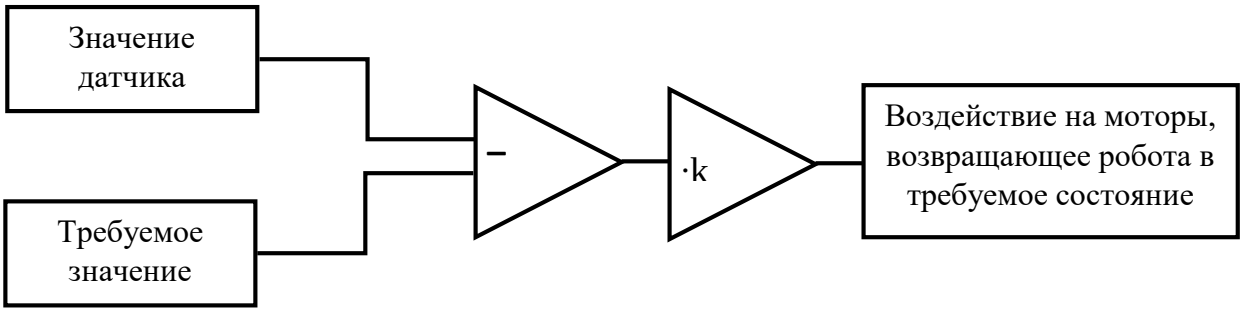


Рисунок 16.5 – Схема П-регулятора на основе одного датчика цвета

Если датчик находиться точно над границей черного и белого, разница ошибки будет мала и управляющее воздействие на моторы будет незначительным и робот едет прямо. При проезде крутых поворотов разница значений становится большой, и робот с высокой скоростью выполняет требуемые маневры.

Как и в предыдущем алгоритме имеет значение, с какой стороны от линии расположен датчик цвета. В случае движения робота с левой стороны от линии вычитаем из полученного значения показаний датчика среднее значение серого. В случае движения робота с правой стороны от линии вычитаем из среднего значения серого полученное значение.

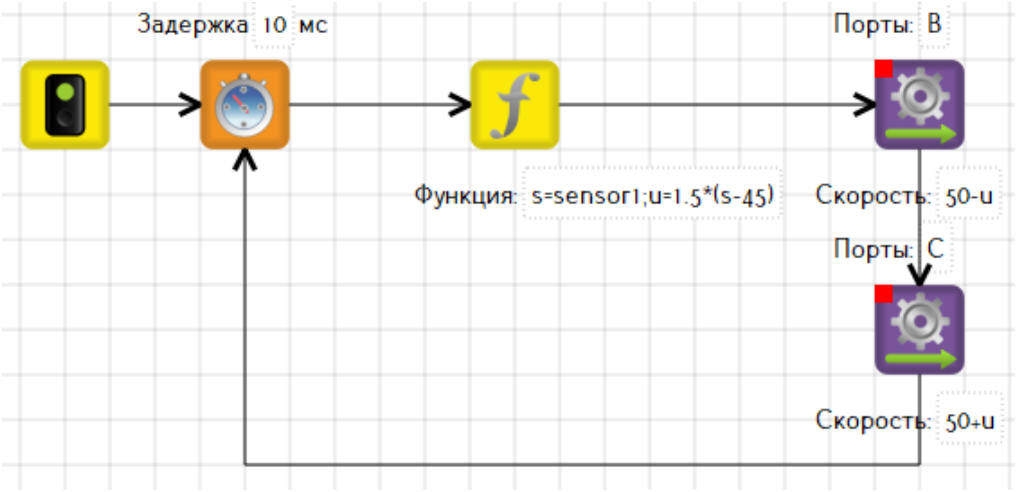


Рисунок 16.6 – Программа к задаче 16.2

Модернизируем конструкцию робота, добавив второй датчик цвета. Датчики нужно установить таким образом, чтобы черная линия проходила между ними.

Задача 16.3. Алгоритм движения по линии с двумя датчиками цвета с использованием релейного регулятора.

Для реализации алгоритма нам потребуется отслеживать показания обоих датчиков, и только после этого задавать движение роботу:

- Если оба датчика над белой поверхностью, то это нормальная ситуация, в которой линия находится между датчиками, поэтому робот должен ехать прямо.
- Если левый датчик еще над светлой поверхностью, а правый датчик уже над темной, значит, робот заехал своей правой частью на линию, а значит ему нужно поворачивать направо, чтобы линия опять оказалась между датчиками.
- Если левый датчик оказался над темной поверхностью, а правый еще над светлой, то для выравнивания роботу нужно поворачивать налево.
- Если оба датчика над темной поверхностью, то в общем случае, робот опять продолжает двигаться прямо.

Подключим левый датчик к порту №1, правый – к порту №2. В цикле используем условный оператор, вложенный в другой условный оператор.

Реализация алгоритма:

1) Проводим калибровку одного датчика, определяем среднее значение серого.

2) Если значение левого датчика (порт 1), больше среднего значения, то анализируем значение правого датчика (порт 2):

- Если значение правого датчика меньше среднего значения, то левый мотор B – 50, правый мотор C – 0 (*робот отклонился влево от линии*);
- Если значение правого датчика больше или равно среднему значению, то левый мотор B – 50, правый мотор C – 50 *(робот находится на линии, датчики расположены по обе стороны от линии, едет вперед)*

3) Если значение левого датчика (порт 1), меньше или равно среднему значению, то анализируем значение правого датчика (порт 2):

- Если значение правого датчика меньше среднего значения, то левый мотор B – 50, правый мотор C – 50 *(оба датчика находятся на линии, значит впереди перекресток, проезжаем прямо);*
- Если значение правого датчика больше или равно среднему значению, то левый мотор B – 0, правый мотор C – 50 *(робот отклонился вправо от линии)*.

На рисунке 16.7 представлен алгоритм движения по линии с двумя датчиками цвета с использованием релейного регулятора.

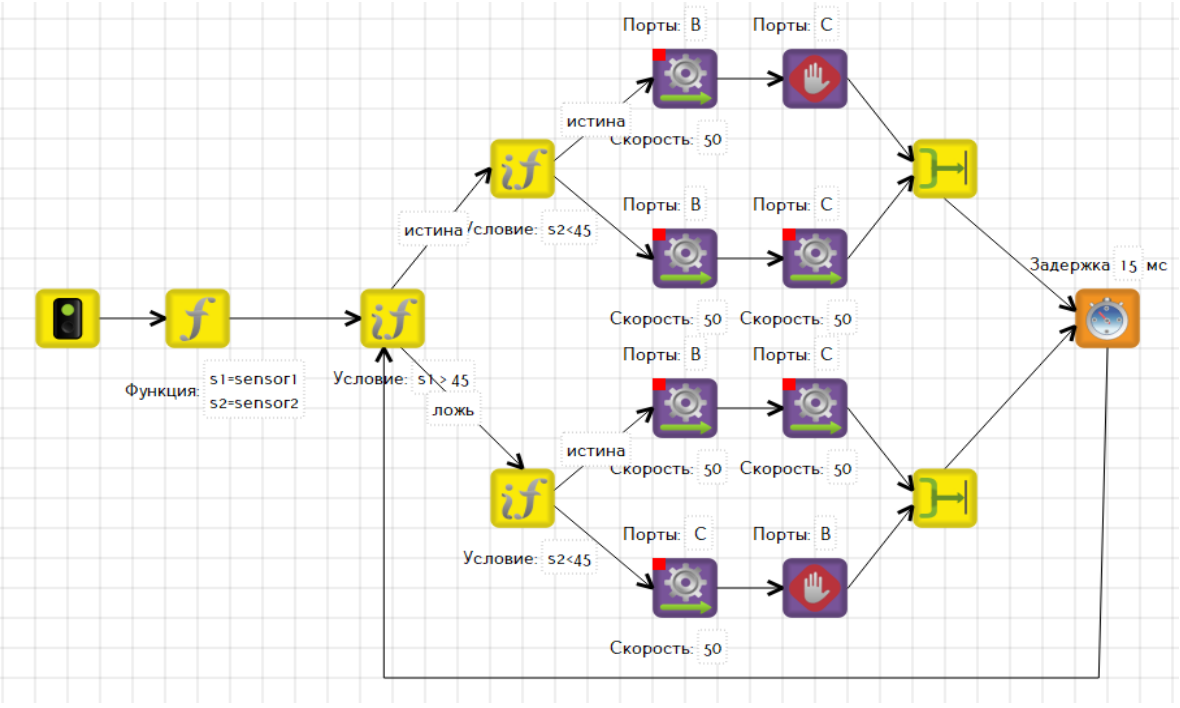


Рисунок 16.7 – Программа к задаче 16.3.

Задача 16.4. Алгоритм движения по линии с двумя датчиками цвета с использованием П-регулятора.

При движении по линии с помощью двух датчиков цвета скорость поворота не зависит от разницы между текущим и средним значением датчика, а от разницы показаний (рассогласования) двух датчиков.

Как правило, при движении по линии на основе двух датчиков цвета, их располагают так, чтобы каждый датчик находился с разных сторон линии. В случае, когда робот находится строго над линией, оба датчика показывают приблизительно одинаковые значения, ошибка рассогласования мала и практически не оказывает влияния на результат. Если робот одним датчиком наезжает на линию, то второй, соответственно, удаляется от нее. Ошибка рассогласования возрастает пропорционально удалению или приближению робота к линии и, будучи умноженной на коэффициент, оказывает воздействие на моторы: чем дальше робот находится от линии, тем скорость возврата становиться больше.

На рисунке 16.8 представлена схема П-регулятора на основе двух датчиков цвета.

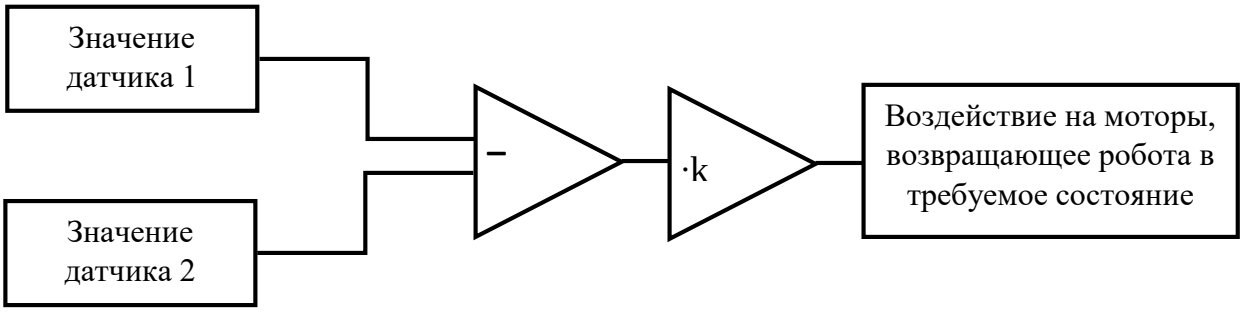


Рисунок 16.8 – Схема П-регулятора на основе двух датчиков цвета.

Важную роль в пропорциональном управлении играет правильный выбор коэффициента (k), используемый при движении. Коэффициент должен быть положительным, его величина подбирается опытным путем или рассчитывается и будет зависеть от конкретной конструкции робота, типа измеряемой величины (свет, ультразвук, угол поворота и т.д.) и внешних условий (освещение, тип покрытия поля, отражающая способность поверхностей и т.п.). Рекомендуется при подборе коэффициента изменять его с дискретностью 0,1, повышая или

уменьшая его. Увеличение коэффициента вызывает резкие толчки, но позволяет проезжать крутые повороты, а уменьшение коэффициента приводит к более плавному движению, но есть риск потерять линию при проезде крутых поворотов. Поэтому рекомендуется использовать алгоритм следования по линии с разными коэффициентами на разных участках траектории. Также опытным путем требуется подбирать значение скорости моторов, причем на разных участках траектории она может быть также разной.

Реализуем алгоритм. Для этого в цикле опрашиваем показания датчиков, подключенных к портам 1 и 2. Затем находим их разность, а результат умножаем на коэффициент и полученную ошибку рассогласования подаем на моторы (рисунок 16.9).

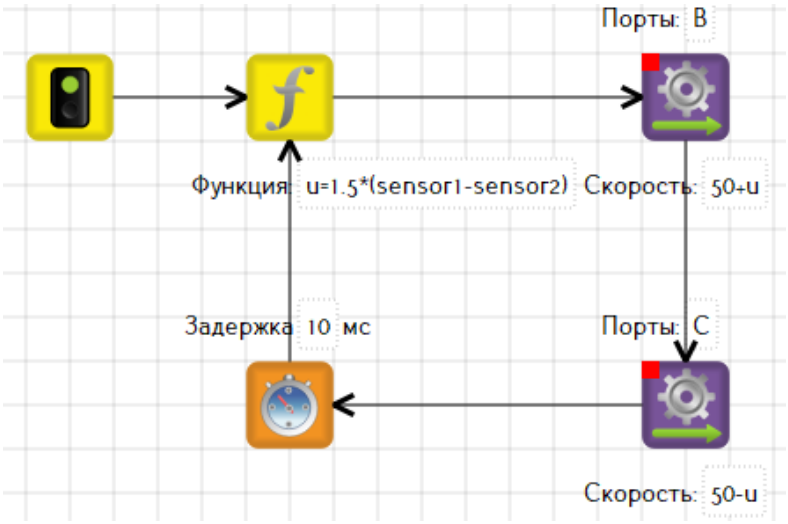


Рисунок 16.9 – Программа к задаче 16.4.

На этой диаграмме представлена программа, позволяющая роботу бесконечно долго следовать по линии. Однако в задачах этого не требуется. Как правило, на перекрестках необходимо роботу выполнять определенные действия. Но как роботу понять, что он наехал на перекресток? Рассмотрим это в следующей задаче.

Эта программа может быть усовершенствована за счет добавления еще одного показателя – статической ошибки. Статическая ошибка – это разность между показаниями датчиков цвета, так как в реальных условиях датчики практически никогда не показывают одинаковые значения на одном объекте. Их показания могут различаться на 3-10 единиц.

Статическая ошибка вычисляется в самом начале, когда робот стоит, причем оба датчика должны располагаться над одним цветом поверхности.

Добавляем блок «Функция», в которой идет вычисление статической ошибки (*st*), а затем вычитаем ее при вычислении основной ошибки (рисунок 16.10).

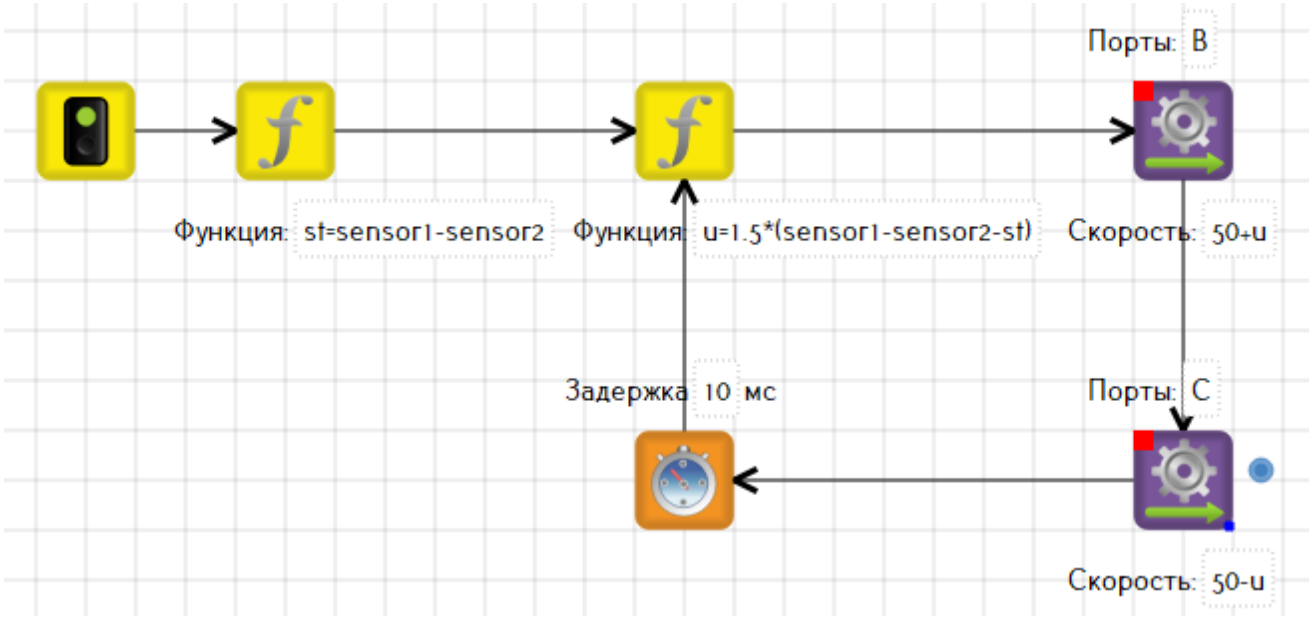


Рисунок 16.10 – Программа к задаче 16.4 с добавлением статической ошибки.

Задача 16.5. Алгоритм движения по линии на основе пропорционального регулятора с поиском и подсчетом перекрестков.

Когда робот окажется на перекрестке, показания обоих датчиков цвета будут меньше, чем среднее значение серого: это означает, что оба датчика обнаружили черную линию.

Алгоритм реализации следующий:

1) В переменной *x* будем хранить количество перекрестков, ее начальное значение равно 0.
2) Определим среднее значение серого, его значение будет хранится в переменной *m*.
3) В цикле опрашиваем датчики цвета и организуем движение по линии.
4) При следовании по линии проверяем, есть перекресток или нет. Наличие перекрестка проверяется условием: *sensor1<m && sensor2<m* (знак && – *логическое и*), что означает показания датчиков одновременно должны быть меньше показаний среднего.

5) Если условие принимает истинное значение (робот попал на перекресток), то увеличиваем значение переменной *x* на 1 и результат выводим на экран.

Чтобы робот не подсчитал перекресток несколько раз, необходимо подобрать значение таймера, расположенного после блока «Напечатать текст», но при этом не съезжал с линии (рисунок 16.11), либо организовать проезд вперед на ширину линии.

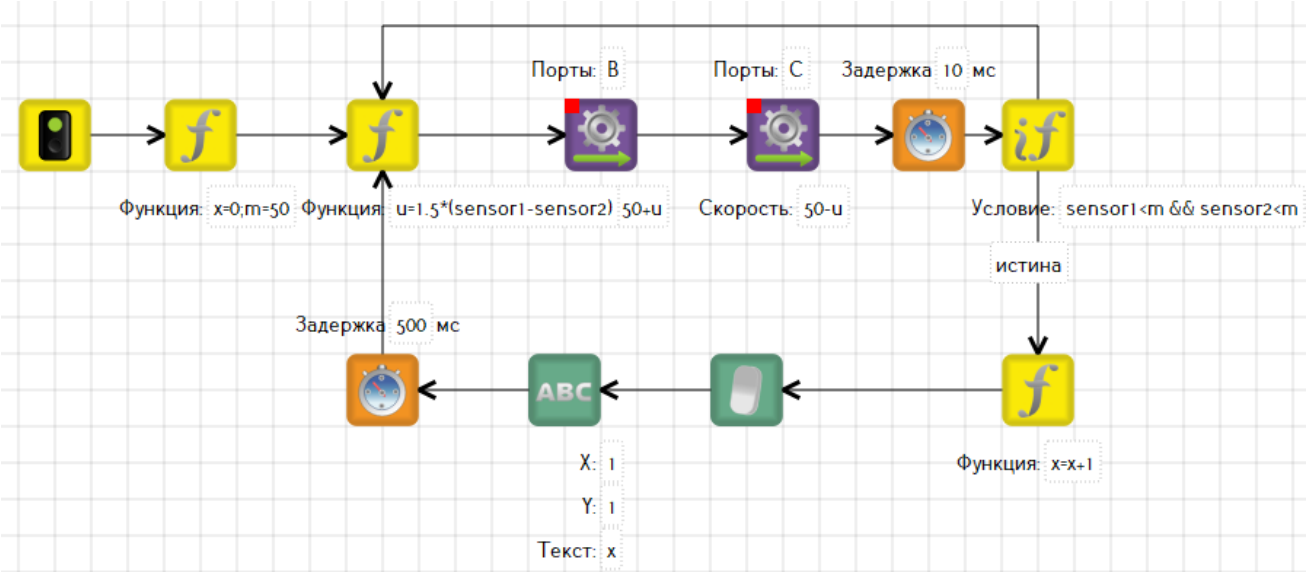


Рисунок 16.11 – Программа к задаче 16.5.

Задача 16.6. Составить программу движения по линии с переходом на инверсную линию.

Инверсия в заданиях, связанных с движением робота по линии, представляет собой резкое инверсное изменение цвета пол: робот ехал по белому полю вдоль черной линии, а продолжить движение нужно по черному полю вдоль белой линии и наоборот. Инверсия, как часть заданий, встречается на соревнованиях всех уровней.

Принцип обнаружения инверсного участка аналогичен принципу обнаружения перекрестка: оба датчика оказались над черным цветом поля и показывают значения *меньше* среднего значения серого.

Принцип обнаружения окончания инверсного участка: оба датчика цвета оказались на белым цветом поля и показывают значения *больше* среднего значения серого.

Принцип проезда инверсного участка следующий: робот едет по алгоритму пропорционального управления на основе двух датчиков цвета. При обнаружении инверсии выполняется тот же алгоритм пропорционального управления движением, но то управление, которое шло на левый мотор, идет на правый, и наоборот. То есть если до этого мы «отталкивались» от черной линии, то сейчас нужно отталкиваться от белой.

На рисунке 16.12 представлена программа проезда инверсной траектории. Приведем описание алгоритма.

1) В цикле организуем стандартный алгоритм пропорционального управления с обнаружением перекрестков (условие *sensor1<m && sensor2<m*).

2) Когда робот обнаружит инверсию (перекресток), первый цикл завершается.

3) Во втором цикле вновь организуем алгоритм пропорционального управления, но с обнаружением окончания инверсного участка. Отличие от первого цикла заключается в изменении знака пропорционального коэффициента (*-1.5*), что позволяет двигаться по белой линии на темном фоне, и изменении условия выхода из цикла: оба датчика оказались на белом поле, то есть показывают значения, больше среднего значения серого.

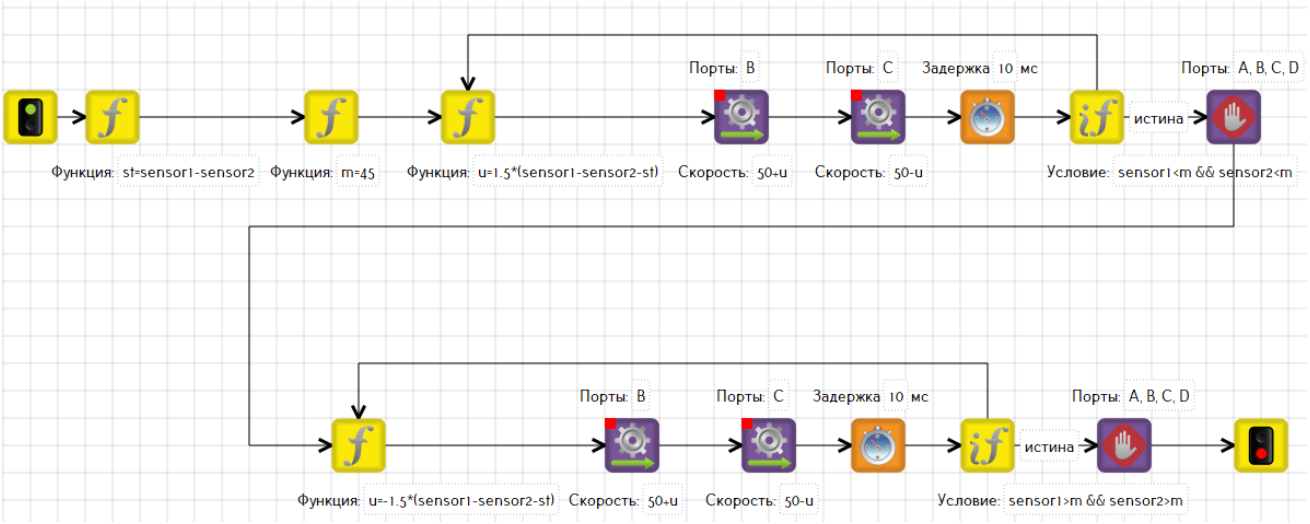
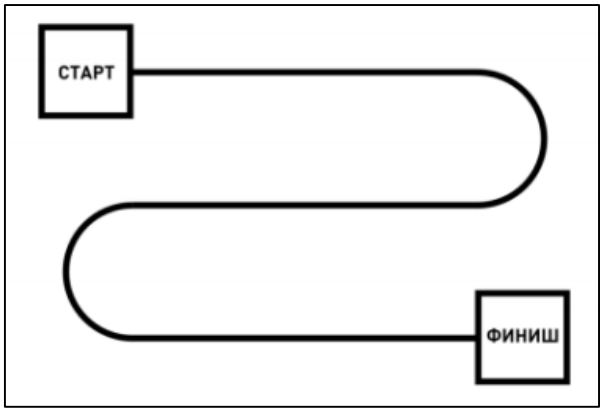


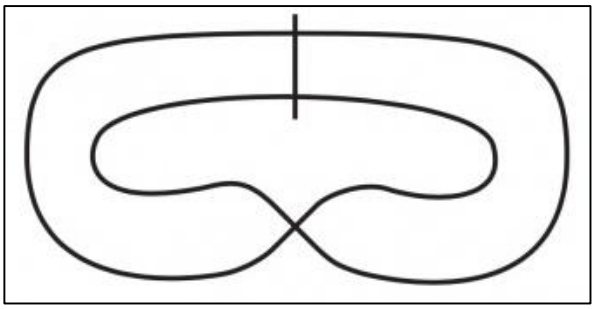
Рисунок 16.12 – Программа к задаче 16.6.

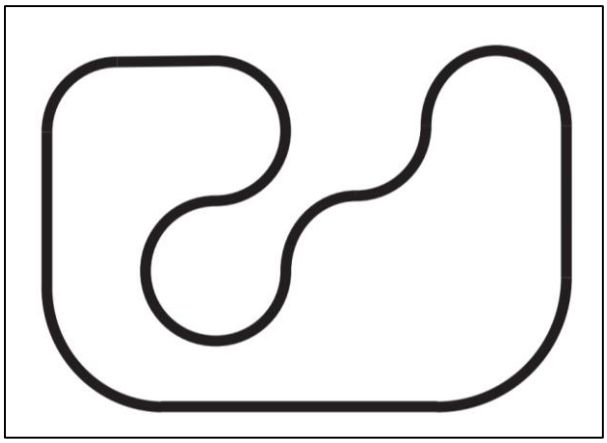
Если роботу необходимо продолжить движение до следующего перекрестка, то вновь организуем цикл, аналогичный первому. Для проезда каждого участка рекомендуется создать подпрограмму.

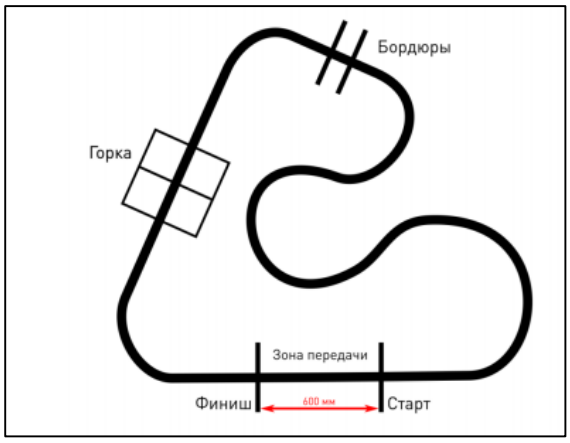
Упражнения для самостоятельного решения

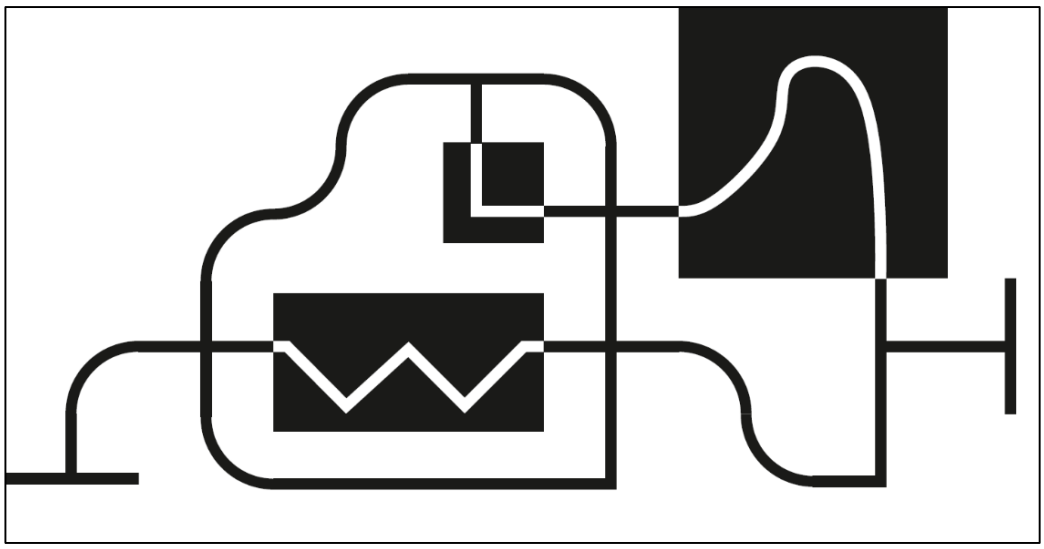
1. Отработать на поле движение робота по линии на основе одного и двух датчиков цвета, подобрать коэффициент и скорость движения.
2. Составить программу остановки робота на ближайшем перекрестке.
3. Составить программу остановки робота на 4-м перекрестке.
4. Сделав остановку на 4-м перекрестке, робот должен развернуться на 180 градусов и вернуться к месту, с которого начал движение.
5. Составить программы следования по линии на следующих полигонах:











17. Пропорциональный интегральный дифференциальный (ПИД) регулятор

Рассматривая, алгоритмы пропорционального регулятора, было определено, что управляющее воздействие на моторы зависит от ошибки рассогласования. Чем больше разница между требуемой величиной и реальным показанием датчика (в случае использования одного датчика) или чем больше разница между показаниями двух датчиков (в случае использования двух датчиков), тем с большей скоростью совершается движение, возвращающее робота в требуемое положение.

Данный алгоритм прост в реализации и достаточно надежен в большинстве случаев. Однако если мы имеем дело с большими скоростями или резкой сменой условий, пропорционального управления не всегда достаточно. Для более точного управления дополнительно вводятся интегральная и дифференциальная составляющие, то есть пропорциональный интегральный дифференциальный регулятор (ПИД-регулятор).

Рассмотрим интегральную составляющую. Уравнение интегральной составляющей имеет вид:

$$u = k_i \cdot Integral,$$

где k_i – интегральный коэффициент, *Integral* – интегральная (суммарная) составляющая ошибок, которую можно вычислить следующим образом:

Integral =0 ,5 · Integral (на предыдущем шаге) + ошибка (на текущем шаге).

Таким образом, интегральная составляющая вычисляется как коэффициент, умноженный на сумму всех предыдущих ошибок.

Физический смысл данной формулы заключается в том, что в каждый момент времени помимо ошибки рассогласования будет вестись учет суммы предыдущих ошибок, которые накапливаются и возрастают. Соответственно, чем дальше и дольше робот будет отходить от линии, тем с большей скоростью регулятор заставит его возвращаться обратно. То есть требуемое управляющее воздействие на моторы будет тем больше, чем дольше робот не может достичь требуемого положения.

Если робот едет прямо, совершая равномерные колебательные движения относительно линии, то ошибка постоянно меняет знак и при суммировании превращается в ноль, не оказывая дополнительного воздействия на моторы.

Рассмотрим дифференциальную составляющую, которая необходима при прохождении крутых поворотов на больших скоростях. Уравнение дифференциальной составляющей регулятора имеет вид:

$$u = k_d \cdot Differ,$$

где k_d – дифференциальный коэффициент, *Differ* – дифференциальная составляющая управления или скорость изменения ошибки, которую можно вычислить по формуле:

Differ = ошибка (на текущем шаге) - ошибка (на предыдущем шаге).

Таким образом, если, сила, возникающая благодаря пропорциональному управлению, толкает робота в сторону линии, то сила, возникающая благодаря дифференциальному управлению, может в определенных условиях действовать в противоположную сторону.

Если значение *Differ* положительно и велико, то есть ошибка быстро увеличивается, и робот отклоняется от линии, то дифференциальная составляющая управления будет складываться с пропорциональной составляющей и быстрее возвращать робота на линию.

Если значение *Differ* отрицательно и велико, то есть ошибка быстро уменьшается, и робот с большой скоростью приближается к линии, то дифференциальная составляющая управления будет вычитаться из пропорциональной составляющей и притормаживать робота до тех пор, пока скорость изменения ошибки не замедлится, тем самым предотвращая проскакивание робота по инерции.

Если значение *Differ* мало, то есть робот едет по прямой или плавно поворачивает, четко следуя изгибам линии, то дифференциальная составляющая управления практически не оказывает влияние на управление робота.

Рассмотрим алгоритм создания ПИД-регулятора.

1) Введем переменные *Integral*, в которой будет храниться сумма ошибок, и *LastError*, в которой будет храниться значение ошибки на предыдущем шаге, и присваиваем им нулевые начальные значения.

2) Далее организуем цикл.

3) В переменную *u1* запишем разность текущих показаний датчиков цвета (при движении по линии на основе двух датчиков цвета) или разность между средним значением серого и текущим значением датчика цвета (при движении по линии на основе одного датчика цвета).

4) Вычисляем интегральную (*Integral=0.5*Integral+u1*) и дифференциальную (*LastError=u1-LastError*) составляющие.

5) Суммируем все составляющие:

   a. пропорциональное управление, равное значению текущей ошибки, умноженной на пропорциональный коэффициент;
   b. интегральное управление, равное сумме ошибок, умноженной на интегральный коэффициент;
   c. дифференциальное управление, равное изменению ошибки, умноженному на дифференциальный коэффициент.

Формула имеет вид: *u=1.5*u1+0.2*Integral+10*LastError* (рисунок 17.1).

В формуле в качестве пропорционального коэффициент указано число 1,5, интегрального – 0,2 и дифференциального – 10. Все коэффициенты подбираются в зависимости от робота и поля, но важно помнить следующее. Самый большой вклад в управление вносит пропорциональная составляющая, поэтому сначала необходимо отработать пропорциональное управление, при этом задав нулевые значения двум другим составляющим. После определения пропорционального коэффициента, отрабатывается интегральный, начав с числа 0,05, и постепенно увеличивая его на 0,01, добиваясь наилучших показателей движения. И в последнюю очередь необходимо получить дифференциальный коэффициент, начав с 1, и постепенно его увеличивая.

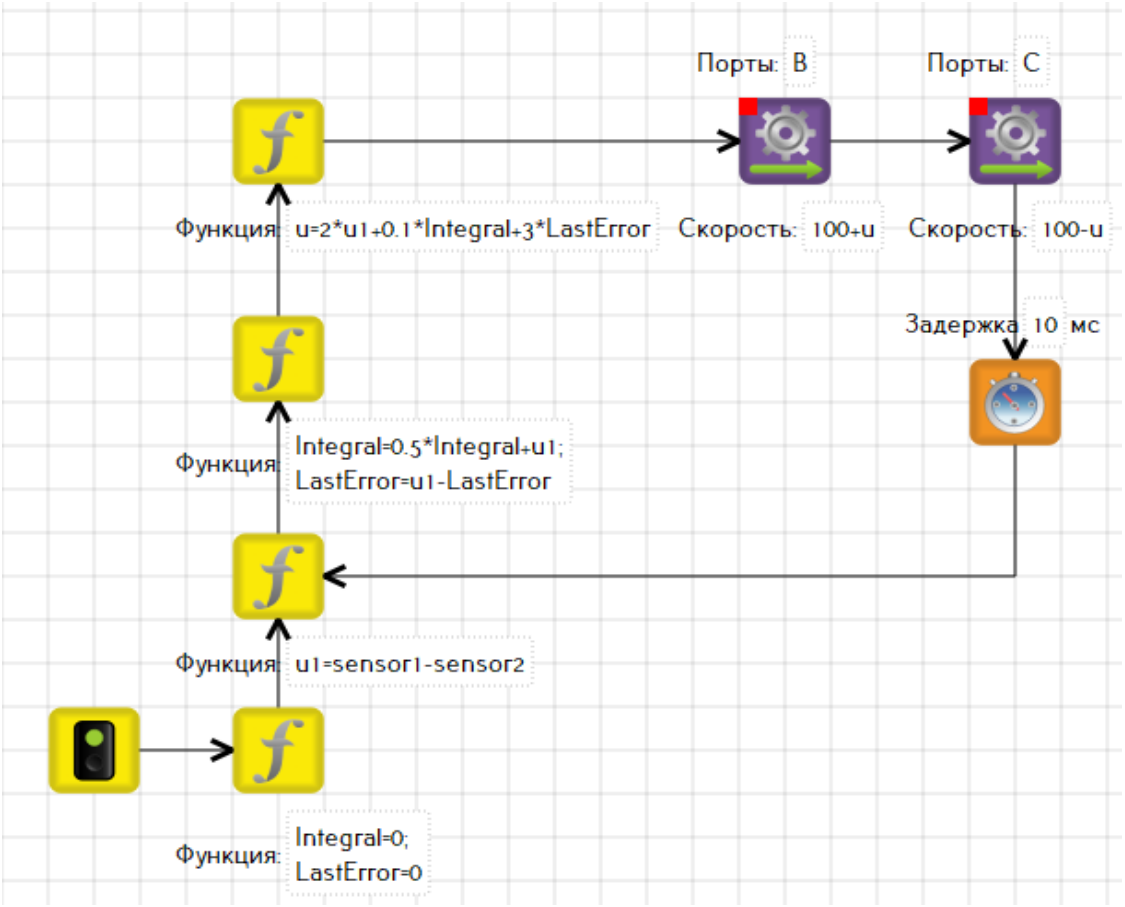
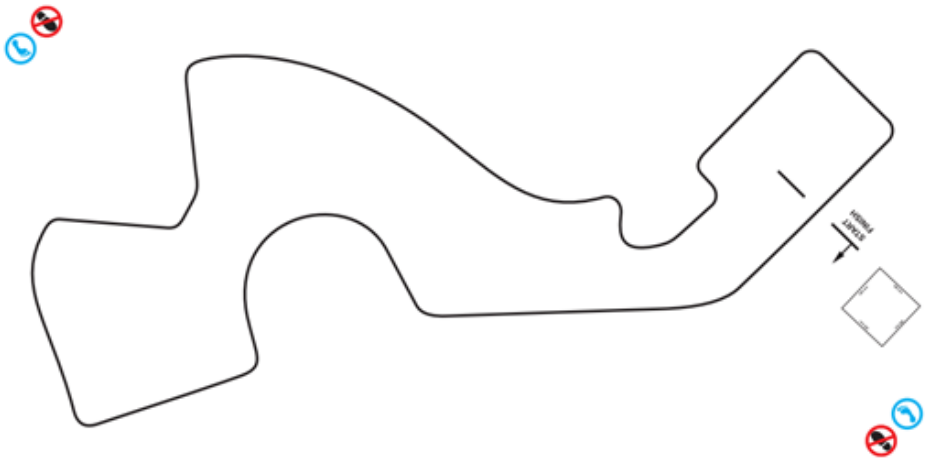
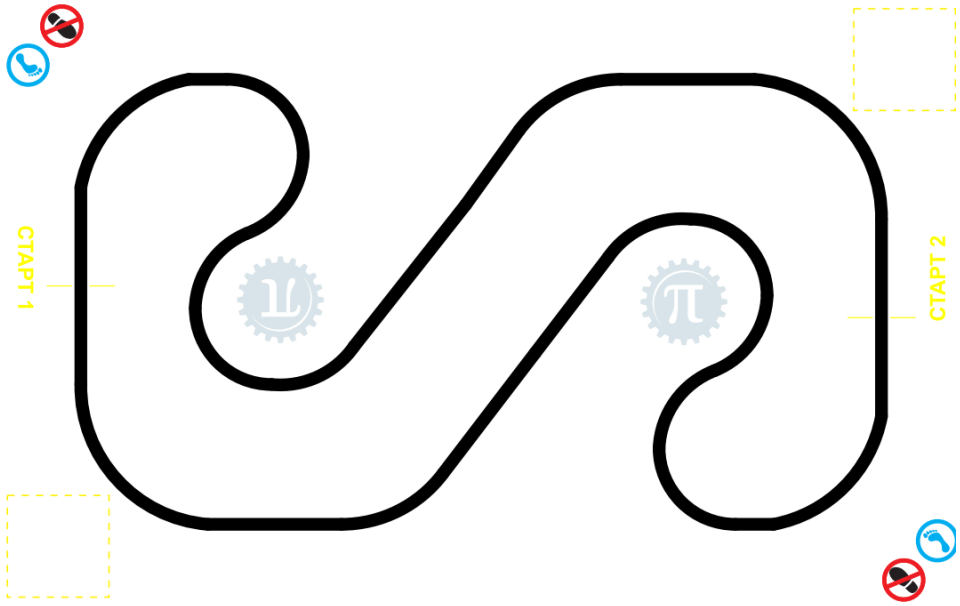


Рисунок 17.1 – Реализация ПИД-регулятора для движения по линии

Упражнения для самостоятельного решения

1. Составить программы следования по линии на следующих полигонах:





Литература

1. Киселев, М. М. Робототехника в примерах и задачах. Курс программирования механизмов и роботов [Текст] / М. М. Киселев. – Москва: СОЛОН-Пресс, 2017. - 136 с.
2. Мордвинов, Д. А. Среда программирования роботов TRIK Studio. [Текст] / Д. А. Мордвинов; Санкт-Петербургский государственный университет. Кафедра системного программирования.- СПб, 2017 – 31 с.
3. Овсяницкая, Л. Ю. Алгоритмы и программы движения по линии робота Lego Mindstorms EV3 [Текст] / Л. Ю. Овсяницкая, Д. Н. Овсяницкий, А. Д. Овсяницкий. – Москва: Перо, 2015. – 168 с.
4. Овсяницкая, Л. Ю. Курс программирования робота Lego Mindstorms EV3 в среде EV3: основные подходы, практические примеры, секреты мастерства [Текст] / Л. Ю. Овсяницкая, Д. Н. Овсяницкий, А. Д. Овсяницкий. - Челябинск: ИП Мякотин И. В., 2014 - 204 с.
5. Овсяницкая, Л. Ю. Пропорциональное управление роботом Lego Mindstorms EV3 [Текст] / Л. Ю. Овсяницкая, Д. Н. Овсяницкий, А. Д. Овсяницкий – Москва: Перо, 2015. – 188 с.
6. Терехов, А. Н. Среда визуального программирования роботов QReal:Robots [Текст] // Терехов, А. Н., Брыксин, Т. А., Литвинов, Ю. В.III Всероссийская конференция «Современное технологическое обучение: от компьютера к роботу» (сборник тезисов). СПб., 2013. С. 2–5.
7. Филиппов, С. А. Робототехника для детей и их родителей [Текст] / С. А. Филипов. - 3 - изд.- СПб.: Наука.- 2013, 319 с.

Интернет-ресурсы

1. LEGO® MINDSTORMS® EV3. Руководство пользователя. [Электронный ресурс]: сайт / The LEGO Group. – Режим доступа: https://www.lego.com/
2. TRIK [Электронный ресурс]: кибернетический конструктор / ООО «КиберТех».- Режим доступа: http://www.trikset.com/

3. TRIK Studio: среда обучения основам программирования и кибернетики [Электронный ресурс]: сайт / ООО «КиберТех». - Режим доступа: http://robots.qreal.ru/
4. Международный фестиваль робототехники «РобоФинист». [Электронный ресурс]: сайт / Благотворительный фонд Темура Аминджанова и Сергея Вильского «Финист». - Режим доступа: https://robofinist.ru/
5. Монахова, О. А. Программирование робота в визуальной среде TRIK Studio [Электронный ресурс]: видео / О. А. Монахова. - Режим доступа: https://www.youtube.com/watch?v=vvnT9bWr7SE
6. Робототехника: инженерно-технические кадры инновационной России. Программа [Электронный ресурс]: сайт / Фонд «Вольное Дело». - Режим доступа: http://www.russianrobotics.ru
7. Солуянов, Е. А. ПО TRIK Studio для EV3: движение робота вперед-назад, энкодеры [Электронный ресурс] / Е. А. Солуянов. - Режим доступа: http://mosmetod.ru/metodicheskoe-prostranstvo/robototekhnika/uchebno-metodicheskie-materialy/lego-konstruirovanie-i-robototekhnika/dvizhenie-robota-vpered-nazad-enkodory-algoritm-na-baze-trik-studio.html
8. Солуянов, Е. А. Учимся программировать EV3 в TRIK Studio. Основы работы со средой TRIK Studio [Электронный ресурс] / Солуянов, Е. А. - Режим доступа: http://mosmetod.ru/metodicheskoe-prostranstvo/robototekhnika/uchebno-metodicheskie-materialy/lego-konstruirovanie-i-robototekhnika/uchimsya-programmirovat-ev3-v-trik-studio.html
9. Справочная система TRIK Studio 3.2.0 [Электронный ресурс] // TRIK Studio 3.2.0: программа.- – Систем. требования: Microsoft ® Windows, macOS 10.9 Mavericks; 4 ГБ или больше ОЗУ; монитор; мышь. – Загл. с экрана.
10. Широколобов, И. Ю. Первый шаг в робототехнику (демо-курс) [Электронный ресурс] / И. Ю. Широколобов. - Режим доступа: https://stepik.org/course/462/

Практическое пособие

Робототехника. Основы программирования робота Lego Mindstorms EV3 в TRIK Studio:

Павлова Наталья Германовна

Редактор-методист Рагозина Т. М.
Технический редактор Стрельбицкая О. О.

Подписано в печать 08.11.2019. Формат 60*84/16. Бумага офисная. Печать цифровая. Тираж 100.

ООО «Издательство Вектор Бук, г. Тюмень, ул. Володарского, 45