ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
Национальный исследовательский университет
«Высшая школа экономики»**

**Московский институт электроники и математики
Национального исследовательского университета
«Высшая школа экономики»**

**Кафедра информационных технологий
и автоматизированных систем**

ПРОГРАММИРОВАНИЕ УПРАВЛЕНИЯ МОБИЛЬНЫМИ РОБОТАМИ LEGO MINDSTORMS RCX 2.0

**Учебно-методическое пособие
по проведению лабораторного практикума
по дисциплине «Управление робототехническими системами»**

Москва 2012

Составители: доц., канд. техн. наук Б.П. Байкальцев
доц., канд. техн. наук А.Б. Беликов
ассистент А.М. Тишкин

УДК 681.3

Программирование управления мобильными роботами LEGO Mindstorms RCX 2.0: Учебно-метод. пособие по проведению лабораторного практикума по дисциплине «Управление робототехническими системами» / Моск. институт электроники и математики Национального исследовательского университета «Высшая школа экономики»; Сост. Байкальцев Б.П., Беликов А.Б., Тишкин А.М. М., 2012. 41 с.

Пособие содержит описание реализации управления роботами путём программирования встроенного микроконтроллера на языке NQC. Данные методические указания предназначены для студентов пятого курса специальности «Автоматизированные системы обработки информации и управления».

ISBN 978-5-94506-311-2

Введение

Целью лабораторного практикума по дисциплине «Управление робототехническими системами» является освоение управления реальными техническими объектами, включающими микроконтроллер, датчики и исполнительные механизмы, а также овладение технологией их программирования на языке высокого уровня с помощью инструментальной среды программирования BricxCC.

В данном пособии рассматривается использование робототехнического конструктора LEGO Mindstorms Robotics Invention System, который был спроектирован и изготовлен при непосредственном участии лаборатории Массачусетского технологического института (Massachusetts Institute of Technology, MIT), активно внедрявшего компьютерные технологии в процесс обучения своих студентов.

Описание программируемого микроконтроллерного блока

Основным компонентом роботов, используемых в данном лабораторном практикуме, является управляющий блок RCX (**R**obotic **C**ommand E**x**plorer) версии 2.0. Его ядро – 8-разрядный микроконтроллер Hitachi H8/3292 с частотой 16 МГц.

Блок RCX имеет 3 входа для сенсоров, 3 выхода для исполнительных устройств, 4 кнопки управления, ЖК-дисплей, встроенный динамик и инфракрасный приёмо-передатчик (рис.1). Порты ввода (1, 2 и 3) нужны для того, чтобы присоединять датчики контакта, освещённости, вращения и температуры (<u>в нашем случае имеются только сенсоры контакта и освещённости</u>).

Рис.1

Порты вывода (A, B и C) используются для того, чтобы присоединять двигатели, источники света и другие устройства вывода. (В нашем случае имеются только двигатели). В микроконтроллерном блоке существуют также 3 внутренних «датчика»: таймер (чтобы отслеживать время), память сообщений (чтобы получать сообщения от других RCX) и определённых пользователем переменных.

Низкоуровневое программное обеспечение (ПО) RCX представлено операционной системой (ОС), которая размещается в ПЗУ (16 Кбайт) на кристалле микроконтроллера и соответственно не теряется при удалении элементов питания. Функция ОС – управление интерфейсом аппаратных средств (Hitachi H8), которое включает загрузку специального ПО через ИК-порт.

Специальная программа (`Firmware`) является виртуальной машиной (интерпретатором команд) и сохраняется в ОЗУ RCX. Она имеет следующие функции:

- интерпретирует байтовый код программы пользователей таким образом, что микроконтроллер H8 может понять его. В нашем случае этот код генерируется компилятором NQC, который используется при создании программы контроллера RCX с помощью ПК;
- отслеживает текущее время;
- управляет выходами на двигатели;
- контролирует входы сенсоров.

Отметим, что ОЗУ имеет объём 32 Кбайт, содержит Firmware размером 26 Кбайт и до 6 Кбайт – программы и данные пользователя. Когда элементы питания удалены более чем на ~60 секунд, ОЗУ будет стёрто. Его необходимо загрузить заново после установки элементов питания. <u>При отсутствии Firmware невозможно загружать в робота собственные программы!</u>

Кнопки и дисплей блока RCX (рис.2) имеют следующее назначение:

Рис.2

1. **On-Off**: включает-выключает блок. Остальные 3 кнопки работают только тогда, когда блок включён.
2. **Prgm**: позволяет переключаться между пятью «слотами» памяти с записанными в них программами. Порядковый номер выбранной программы показывается на дисплее справа.

3. **Run**: запускает или останавливает выбранную программу. Когда программа запущена и выполняется, на дисплее появляется изображение идущего человечка. Иначе – он стоит.
4. **View**: (кнопка работает только после загрузки Firmware) – позволяет получить информацию с сенсоров и о состоянии моторов. В верхней части дисплея появляются стрелочки, указывающие на соответствующий номер датчика. В нижней части показываются данные о состоянии моторов A, B и C.

Когда происходит обмен данными между ПК и RCX, в левой части дисплея появляется индикатор близости в виде символа V, «залитого» до уровня, пропорционального расстоянию между ИК-башней, подключённой к ПК, и ИК-портом блока RCX (рис.3).

Когда программа посылается с ПК в RCX, в верхней части дисплея RCX появляется пунктирная линия.

Индикатор таймера (в формате MM:CC) появляется в центре дисплея (только когда загружена Firmware) и показывает, сколько минут прошло с момента последней установки времени или перезапуска.

Если индикатор таймера отсутствует, значит, RCX находится в «стартовом режиме», требуется загрузить Firmware, чтобы обеспечить обмен данными с ПК. Без такой загрузки можно запускать только пять встроенных стандартных программ:

1. Робот издаёт звуковой сигнал и движется вперёд.
2. Робот движется вперёд и останавливает мотор при срабатывании датчика контакта с этой же стороны.
3. Робот движется вперёд либо останавливается, в зависимости от значения с сенсора освещённости.
4. Пятикратное движение «туда-обратно» с небольшими отклонениями от прямой траектории, длительность движений выбирается случайным образом (от 0 до 3 секунд).
5. Робот движется вперёд, изменяя направление, если срабатывает единственный контакт-сенсор.

После загрузки Firmware на их место можно загружать свои собственные программы.

Следует отметить, что по умолчанию «слот» программы №1 защищён от записи. При необходимости защиту можно снять, например, через настройки фирменного ПО RIS 2.0 (Robotics Invention System).

В данном лабораторном практикуме исследуется программирование мобильных роботов, собранных из конструктора LEGO на базе управляющих блоков RCX 2.0 в виде самоходных трёх-, четырёхколёсных или гусеничных шасси с двумя электроприводами – моторами постоянного тока и датчиками контакта и/или освещённости.

Среда разработки BricxCC и язык программирования NQC

В стандартной поставке робототехнического конструктора LEGO Mindstorms Robotics Invention System 2.0 имеется компакт-диск с программным обеспечением, включающим оболочку графического программирования. По мере сборки пользователем программных графических блоков фирменная среда RIS генерирует файлы программ *.lsc (LEGO Script Code) или *.rcx2, сохраняемые как текст скриптового языка P-Brick.

Но более широкие возможности по управлению роботами даёт свободно распространяемая интегрированная среда разработки прикладных программ Bricx Command Center 3.3 (BricxCC). Она может использоваться для программирования микропроцессорных блоков (bricks) LEGO: RCX (всех версий, а в нашем случае – 2.0), Scout, Cybermaster и Spybotics.

Одним из популярных языков, поддерживаемых средой BricxCC, является NQC (Not Quite C – «не совсем С»). Это упрощённый С-подобный не объектно-ориентированный язык, автором которого является Дэйв Баум (Dave Baum), разрабатывавший NQC специально для программирования блоков LEGO RCX.

Лабораторная работа №1. Знакомство с управлением роботом LEGO Mindstorms RCX 2.0

Задание

- Убедитесь, что ИК-порт (ИК-башня) присоединен к одному из USB-портов ПК и находится в зоне видимости блока RCX (рис.3).

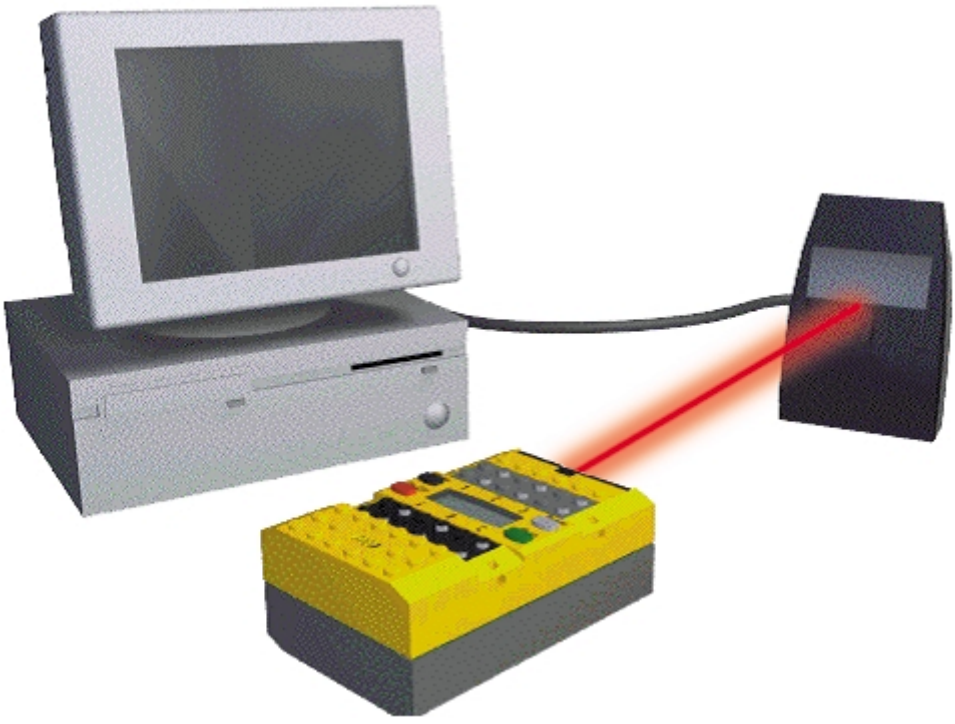


Рис.3

- Запустите Bricx Command Center. При этом появится окно (рис.4) поиска «кирпичика» RCX2.

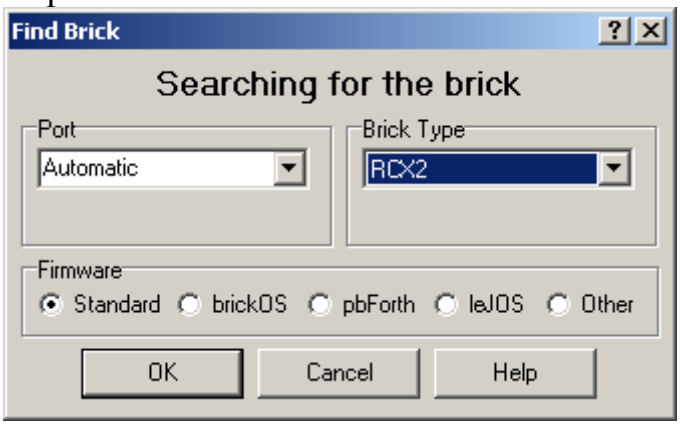


Рис.4

- В списке «Port» выберите *Automatic*, *Search* или *USB*, а в списке «Brick Type» – *RCX2* и нажмите ОК.
- Если программа не находит блок RCX при запуске, можно выбрать пункт `Find Brick` в меню `Tools`, чтобы попробовать ещё раз, например, после включения блока управления робота. Если нет необходимости больше использовать блок управления, можно выбрать `Turn Brick Off` из меню `Tools`. При этом робот выключится, и закроется коммуникация. Если позже нужно снова использовать блок управления, надо включить его и выбрать `Find Brick`.
- Возможно также временно закрыть коммуникацию с RCX, не выключая робота, если выбрать пункт `Close Communication` в меню `Tools`. Коммуникация повторно открывается в следующий раз при выборе любой функции BricxCC, которая пробует общаться с блоком управления.
- Если RCX потерял свою стандартную Firmware (например, при замене источников питания) или нужно загрузить нестандартную ОС (LegOS, leJOS, PBForth и пр.), используйте команду `Download Firmware` в меню `Tools`. Она запросит файл, содержащий Firmware. При загрузке Firmware расстояние между блоком RCX и ИК-башней не должно превышать 10-12 см.
- Используя информацию в Приложении 1 данного пособия, выполните управление роботом и его тестирование по разделам меню `Tools`:
  - ✓ Прямое управление
  - ✓ Диагностика
  - ✓ Наблюдение за блоком управления
  - ✓ «Фортепиано» блока управления
  - ✓ Джойстик блока управления

- ✓ Дистанционное управление
- ✓ Конфигурируемое наблюдение

- Продемонстрируйте преподавателю умение управлять и тестировать робота, а также результаты диагностики и конфигурируемых наблюдений.

Контрольные вопросы

1. Из каких элементов состоит микроконтроллерный блок RCX?
2. Перечислите стандартные датчики и исполнительные устройства робота LEGO Mindstorms RCX 2.0.
3. Опишите ПО, необходимое для управления роботом RCX.

Лабораторная работа №2. Программирование перемещений робота с использованием датчиков контакта и моторов

В данном лабораторном практикуме должен использоваться язык NQC, краткое описание которого приводится в Приложении 2.

Поскольку среда разработки BricxCC поддерживает различные языки программирования, для правильной компиляции файл с программой на NQC должен быть сохранён с соответствующим расширением.

Ожидание сенсора

Начнём с очень простой программы, в которой робот двигается вперёд, пока не сталкивается с чем-нибудь:

```
task main()
{
      SetSensor(SENSOR_1,SENSOR_TOUCH);
      OnFwd(OUT_A+OUT_C);
      until (SENSOR_1 == 1);
      Off(OUT_A+OUT_C);
}
```

Сначала программа сообщает роботу, какой сенсор мы используем. SENSOR_1 – номер входа, с которым соединён датчик. Другие 2 входа называются SENSOR_2 и SENSOR_3 соответственно. SENSOR_TOUCH указывает, что это – сенсор контакта. Для сенсора освещённости используется SENSOR_LIGHT.

После того, как определён тип сенсора, программа включает оба двигателя (A и C), и робот начинает движение вперёд. Следующий оператор – очень полезная конструкция. Он ждёт, пока условие в скобках не станет истинным. Данное условие говорит, что значение с датчика SENSOR_1 должно стать равным 1, что означает нажатие сенсора. Пока он не нажат, значение – 0. Таким образом, оператор `until` в этом случае ждёт, пока сенсор не будет нажат. Тогда мы выключаем двигатели, и задача заканчивается.

Действие по сенсору контакта

Попробуем заставить робота избегать препятствий. Всякий раз, когда его контактный сенсор задевает объект, он должен немного отступить, сделать поворот, а затем продолжить движение вперёд. Ниже представлена соответствующая программа:

```
task main()
{
      SetSensor(SENSOR_1,SENSOR_TOUCH);
      OnFwd(OUT_A+OUT_C);
      while (true)
      {
            if (SENSOR_1 == 1)
            {
                  OnRev(OUT_A+OUT_C); Wait(30);
                  OnFwd(OUT_A); Wait(30);
                  OnFwd(OUT_A+OUT_C);
            }
      }
}
```

Как и в предыдущем примере, сначала указывается тип сенсора. Затем робот начинает движение. В бесконечном цикле постоянно проверяется, сработал ли сенсор, и, если да, робот пятится в течение 0,3 секунды, поворачивает направо в течение 0,3 секунды, а затем снова продолжает движение вперёд.

Режим и тип сенсора

Рассмотренная выше команда `SetSensor()` устанавливает номер входа (1, 2 или 3), к которому подключён сенсор, и его тип. Раздельная установка номера входа и типа сенсора позволяет гибко конфигурировать робота для конкретных задач.

Тип сенсора также можно установить отдельной командой `SetSensorType()`. Имеются четыре различных типа:

- SENSOR_TYPE_TOUCH – для датчика контакта;
- SENSOR_TYPE_LIGHT – для сенсора освещённости;
- SENSOR_TYPE_TEMPERATURE – для температурного сенсора;
- SENSOR_TYPE_ROTATION – для сенсора вращения.

Два последних в нашем случае отсутствуют.

Установка типа сенсора особенно важна для указания, нуждается ли датчик в электропитании (как, например, для источника света в сенсоре освещённости).

Тип данных от сенсора устанавливается командой `SetSensorMode()`. Существует 8 различных типов. Самый важный –

SENSOR_MODE_RAW. В этом типе значение, получаемое от сенсора, представляется числом от 0 до 1023. Это – необработанные (RAW) значения, выдаваемые сенсором. Смысл этих значений зависит от конкретного сенсора. Например, для сенсора контакта, когда он не нажат, выдаваемое значение близко к 1023. Когда сенсор полностью нажат, оно близко к 50. Когда нажат частично – диапазон значений между 50 и 1000. Итак, если сенсор контакта устанавливается в режим RAW, то можно реально определить, нажат ли он частично. Когда датчик является сенсором освещённости, диапазоны значений будут от ~300 (очень светлый) до 800 (очень тёмный). То есть тип RAW даёт более точную информацию, чем тип BOOL, используемый по умолчанию в команде `SetSensor()`.

Второй тип данных от сенсора – SENSOR_MODE_BOOL. В этом типе значение будет 0 или 1. Когда необработанное значение выше, чем примерно 550, булево значение будет 0, иначе – 1. SENSOR_MODE_BOOL является режимом по умолчанию для сенсора контакта. Тип SENSOR_MODE_PERCENT превращает необработанное значение в значение между 0 и 100. Каждое необработанное значение 400 или ниже приводится к 100%. Если необработанное значение становится выше, процентное значение медленно снижается до 0. SENSOR_MODE_PERCENT – режим по умолчанию для сенсора освещённости.

Интересными являются типы данных SENSOR_MODE_EDGE и SENSOR_MODE_PULSE. Они представляют собой число переходов RAW-значения от низкого до высокого уровня или наоборот. Например, когда происходит касание сенсора контакта, RAW-значение изменяется от высокого до низкого уровня. Когда контакт устраняется, получается переход в другом направлении.

Когда устанавливается тип данных от сенсора SENSOR_MODE_PULSE, подсчитываются только переходы от низкого уровня сигнала к высокому. Так, каждый нажим и отпускание сенсора контакта засчитывается как 1. Для типа SENSOR_MODE_EDGE будут подсчитаны оба перехода. Поэтому здесь каждый нажим и отпускание сенсора контакта соответствует 2. Таким образом, можно рассчитать, как часто сенсор контакта был нажат.

Аналогично, в комбинации с сенсором освещённости тип данных PULSE позволяет рассчитать, как часто [яркая внешняя] лампа включается и выключается. Для подсчёта переходов необходимо предварительно обнулить счетчик, используя команду `ClearSensor()`.

Остальные режимы предназначены для датчиков оборотов и температуры и в нашем случае не рассматриваются.

Особенности управления движением

Когда используется команда `Off(),` моторы останавливаются практически мгновенно, используя тормоз. В языке NQC также возможно остановить двигатели плавно, не используя тормоз. Для этого применяется команда `Float()`. Иногда она лучше подходит для конкретной задачи.

Команда `OnFwd()` фактически делает две вещи: включает двигатель и устанавливает направление вперёд. Похожая команда `OnRev()` включает двигатель и меняет направление его вращения на противоположное. В NQC имеются также команды для раздельного включения двигателя и задания направления его вращения. Такие раздельные команды более эффективны, так как при этом в RCX используется меньше памяти, программа выполняется быстрее и позволяет реализовать более плавные движения. Команда `SetDirection()` устанавливает направление (OUT_FWD, OUT_REV или OUT_TOGGLE для переключения текущего направления). Команда `SetOutput()` устанавливает режим (OUT_ON, OUT_OFF или OUT_FLOAT).

Существует много других команд для двигателя, которые комбинируют вышеприведенные команды для их быстрого вызова (см. таблицу функций контроллера RCX в Приложении 2).

Важно отметить, что попытка изменять скорость двигателей с помощью функции `SetPower(outputs, power)` при варьировании значений power от 0 до 7 не даёт большого эффекта. Причина в том, что при этом главным образом меняется крутящий момент мотора, а не его скорость. Такой эффект можно заметить, когда двигатель имеет высокую нагрузку. И даже тогда различие между скоростями движения робота при значениях мощности 2 и 7 весьма незначительно. Если нужно получить лучший эффект, требуется включать и выключать двигатели в быстрой последовательности.

Программирование задач

Обратите внимание, что главный модуль рассмотренных ранее программ определяется ключевым словом **`task`** (задача). Программа NQC может состоять максимум из 10 задач. Каждая задача имеет имя. Одна из задач должна иметь имя `main`, и она всегда будет выполнена. Другие задачи будут выполнены только тогда, когда работающие задачи выдают команды для их исполнения, используя команду `start`. С этого момента обе задачи работают одновременно (т.е. и вызывающая задача продолжает работать). Одна работающая задача может также остановить другую работающую задачу, используя команду `stop`. Позже эта задача может быть снова запущена, но она будет стартовать с начала, а не с того места, где была остановлена.

Например, необходимо сделать программу, в которой робот двигается по периметру квадрата, реагируя на задевание препятствий. Сделать всё это в одной задаче трудно, поскольку робот должен одновременно двигаться по периметру квадрата (то есть включать и выключать двигатели в правильные моменты) и следить за сенсорами. Поэтому лучше использовать 2 задачи: одна будет направлять его по квадрату, а другая – реагировать на сенсоры. Каркас соответствующей программы приведен ниже:

```
task main()
{
      . . .
}
task move_square()
{
      . . .
}
task check_sensors()
{
      . . .
}
```

Главная задача устанавливает тип сенсоров и затем запускает две другие задачи. После этого главная задача заканчивается. Задача `move_square` постоянно перемещает робот по квадрату. Задача `check_sensors` проверяет, нажат ли сенсор контакта. Если да, она предпринимает следующие действия:

- останавливает задачу `move_square`. Это очень важно, так как `check_sensors` теперь берёт контроль над движениями робота;
- перемещает робота немного назад и заставляет его повернуться;
- снова запускает `move_square`, чтобы продолжить движение по квадрату.

Очень важно помнить, что запускаемые задачи работают одновременно. Иногда это может привести к неожиданным результатам, поскольку одна задача может конфликтовать с другой.

Заметим, что аналогичная задача может быть запрограммирована средствами подпрограмм, встроенных функций и макросов (см. Приложение 2).

Работа со звуком

Действия и события робота можно «оживить» звуками встроенного в блок RCX динамика. Имеется 6 встроенных звуков:

0 – Звук клавиши

1 – Бип-бип
2 – Звук убывающей частоты
3 – Звук нарастающей частоты
4 – Звук ошибки «Баxx»
5 – Звук быстро нарастающей частоты

Можно воспроизводить их, используя оператор `PlaySound()`. Вот маленькая программа 2, которая воспроизводит эти звуки:

```
task main()
{
PlaySound(0); Wait(100);
PlaySound(1); Wait(100);
PlaySound(2); Wait(100);
PlaySound(3); Wait(100);
PlaySound(4); Wait(100);
PlaySound(5); Wait(100);
}
```

Возникает вопрос, для чего здесь нужны операторы ожидания. Причина в том, что команда, воспроизводящая звук, не ожидает его окончания, поэтому немедленно начинает выполняться следующий оператор. RCX имеет небольшой буфер, в котором могут храниться некоторые звуки. Однако через некоторое время этот буфер заполняется, и звуки теряются. Это не столь серьезно для звуков, но очень важно для музыки (подробности в Приложении 2).

Отметим, что аргументом `PlaySound()` должна быть константа. Здесь нельзя поместить переменную!

Задание

Используя инструментальную среду BrixCC:

- Скопировать в редактор, скомпилировать, загрузить в робота и исполнить первые 2 программы, код которых приведен выше;
- Разработать и протестировать программу перемещений робота по следующим **вариантам исходных данных:**
  1. Движение по траектории квадрата (по часовой стрелке) размером примерно 0,8 х 0,8 м. Остановка в конце траектории.
  2. Движение по траектории равностороннего треугольника (против часовой стрелки) со стороной примерно 0,8 м. Остановка в конце траектории.
  3. Движение по траектории окружности (как предела многоугольника) с диаметром окружности примерно 0,8 м. Выбор случайного направления «вперёд/назад» после примерно половины траектории.

4. Движение «случайное блуждание» с равномерным распределением случайных величин поворота (в интервале примерно 0…180^{o}) и длины пробега (в интервале примерно 0…0,8 м).
5. Движение по траектории квадрата размером примерно 0,8 х 0,8 м и скоростями: максимальной на первой стороне и вдвое меньших при каждом переходе ко 2-й, 3-й и 4-й. Не использовать малоэффективную команду `SetPower(outputs, power)`, а разработать подзадачу управления скоростью, например, с именем `run_motor`. Остановка в конце траектории

Дополнительные указания:

1. Для всех вариантов использовать контактные сенсоры для останова движения при обнаружении кромки стола и воспроизводить в этом случае один из семи предопределённых звуков. При снятии контакта программа переходит к началу.
2. Для вариантов 1 и 2 вести подсчёт числа срабатываний сенсора контакта и оповещать о них соответствующим количеством звуковых сигналов.

Контрольные вопросы

1. Перечислите основные инструкции языка NQC для работы с сенсорами.
2. Назовите основные инструкции языка NQC для работы с двигателями.
3. Как управлять скоростью вращения двигателей?
4. Назовите возможные типы и режимы сенсоров.
5. Синтаксис определения задач в программе. Как выполняются задачи во времени?

Лабораторная работа №3. Программирование перемещений робота с использованием датчика освещённости и ИК-локатора препятствия

Сенсор освещённости LEGO измеряет количество света в конкретном направлении. Кроме того, он сам излучает свет в этом же направлении. Таким образом, можно сориентировать сенсор освещённости в конкретном направлении и различить интенсивность отражения от объекта в данном направлении. При типе данных RAW диапазоны значений освещённости будут от ~300 (очень светлый) до 800 (очень тёмный). Тип PERCENT превращает необработанное значение в значение между 0 и 100. Каждое необработанное значение 400 или ниже приводится

к 100%. Если необработанное значение становится выше, число процентов медленно снижается до 0. SENSOR_MODE_PERCENT – тип по умолчанию для сенсора освещённости.

Используя приведённые выше свойства сенсора, можно, например, заставить робота следовать по линии на поверхности движения. Для этого нужен достаточно большой лист белой бумаги с замкнутой чёрной линией на ней. Идея управляющей программы: робот во время движения должен постоянно удостоверяться в том, что сенсор освещённости находится над линией. Всякий раз, когда сенсор освещённости уходит от чёрной полоски, интенсивность отражённого света повышается, и программа должна изменить курс робота. Для начала в качестве граничного значения освещённости можно использовать константу 40, но она может быть изменена в зависимости от наличия источников фонового света.

Создание сенсора близости

Используя сенсоры контакта, робот может реагировать на соприкосновение с какими-либо предметами. Но предпочтительнее, если бы он мог реагировать непосредственно перед таким касанием. К сожалению, ультразвуковые сенсоры близости типа сонаров для рассматриваемых роботов не предусмотрены.

Однако существует приём, который можно использовать для цели обнаружения препятствий. Робот LEGO имеет инфракрасный порт, через который может общаться с компьютером или другими роботами. Оказывается, что штатный сенсор освещённости очень чувствителен к инфракрасному свету. Основываясь на этом, можно построить сенсор близости.

Идея состоит в том, что одна задача `send_signal()` постоянно, например, 10 раз в секунду, посылает сообщения через ИК-порт робота. Другая – `check_signal()` измеряет изменения в интенсивности излучения, которое отражается от объектов. Делает она это путём многократного сохранения значения сенсора освещённости, а затем с небольшой задержкой проверяя, стало ли оно выше. Чем больше становится это изменение, тем робот с сенсором ближе к объекту. Если обнаруживается заданное изменение, то задача обеспечивает останов робота и/или некоторое его действие.

Чтобы использовать эту идею, поместите сенсор освещённости с ориентацией вперёд и выше инфракрасного порта на роботе. Таким образом, он будет измерять только отражённый инфракрасный свет. Для сенсора освещённости следует использовать тип данных RAW, чтобы как можно лучше фиксировать изменения.

Неудобство этого метода состоит в том, что он работает при ориентации робота с сенсором только в направлении, близком к нормали

относительно отражающей поверхности препятствия. Другое его неудобство – невозможно связаться с роботом через компьютер, потому что этому препятствуют ИК-сообщения, излучаемые роботом.

Таймеры

Контроллер RCX имеет 4 встроенных таймера. Они пронумерованы от 0 до 3 и работают (тикают) в приращениях по 1/10 секунды. Можно сбросить значение таймера командой `ClearTimer()` и получить текущее значение командой `Timer()`. Рассмотрим пример использования таймера. Нижеследующая программа позволяет роботу двигаться случайным образом в течение 20 секунд.

```
task main()
{
     ClearTimer(0);
     do
          {
          OnFwd(OUT_A+OUT_C);
          Wait(Random(100));
          OnRev(OUT_C);
          Wait(Random(100));
          }
     while (Timer(0)<200);
     Off(OUT_A+OUT_C);
}
```

Инструкции таймеров очень полезны, как замена для команд `Wait()`. Сбрасывая счётчик таймера и затем ожидая, пока он не достигнет конкретного значения, можно реализовать ожидание в течение конкретного периода. В процессе ожидании можно также реагировать на другие события (например, от сенсоров). Следующая простая программа даёт такой пример. Она позволяет роботу двигаться, пока либо истекут 10 секунд, либо сенсор контакта коснётся чего-нибудь.

```
task main()
{
     SetSensor(SENSOR_1,SENSOR_TOUCH);
     ClearTimer(3);
     OnFwd(OUT_A+OUT_C);
     until ((SENSOR_1 == 1) || (Timer(3) >100));
     Off(OUT_A+OUT_C);
}
```

Тестовая площадка

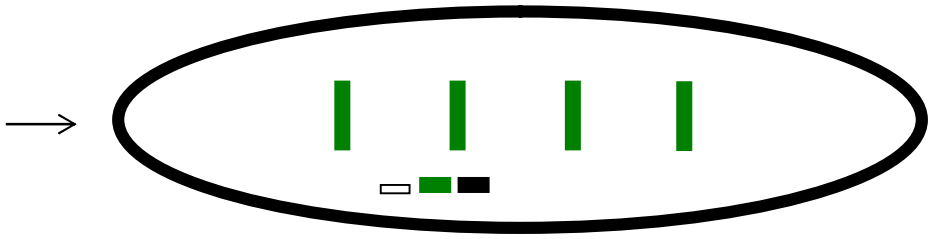


Рис.5

Некоторые варианты заданий предусматривают использование площадки для тестирования программ с сенсорами освещённости, установленными на мобильном роботе вертикально вниз. Площадка представляет собой лист белой бумаги (формата A1) с изображением, как на рис.5. Для сенсора, работающего в режиме по умолчанию SENSOR_MODE_PERCENT, выдаваемые значения будут примерно 34-38%, 41-45%, 47-51% и 53-57% соответственно для чёрного, зелёного, белого цвета и алюминиевой фольги. Эти значения могут слегка изменяться в зависимости от фоновых источников света в используемом помещении.

Задание

Используя инструментальную среду BrixCC, разработать и протестировать программу действий робота по следующим **вариантам исходных данных:**

1. Движение робота с бампером по замкнутому маршруту (жирная чёрная линия овала на белом фоне) с сенсором освещённости, установленным вертикально вниз, по следующему алгоритму:
   a) наезд и «захват» чёрной линии;
   b) движение по маршруту (в первом проходе круговое направление произвольное);
   c) при опускании груза в корзину (срабатывании сенсоров контакта) включается задний ход и осуществляется движение по маршруту в обратном направлении.
2. Движение робота с бампером и с сенсором освещённости, установленным горизонтально в направлении «вперёд», к источнику света по следующему алгоритму:
   a) круговое сканирование сенсором (вместе с роботом) для поиска направления на источник с глобальным максимумом света;
   b) ориентация робота на найденное направление;
   c) движение до объекта – источника (при необходимости – коррекция направления) с остановом по контакту бампера;
   d) стоп.

3. Движение робота (без сенсоров контакта) с программно-аппаратным «сенсором близости» по следующему алгоритму:
   a) прямолинейное движение вперед с работающим сенсором близости;
   b) при обнаружении препятствий - приближение к нему на расстояние 0,1…0.5 м и разворот примерно на 180 градусов;
   c) возврат к п. a).
4. Прямолинейное движение робота (без сенсоров контакта) по направлению стрелки (рис.5) с сенсором освещённости, установленным вертикально вниз, и действия по следующему алгоритму:
   a) подсчёт количества пересечённых полей белого, зелёного и чёрного цвета;
   b) останов на площадке из алюминиевой фольги;
   c) выдача подсчитанных значений числом повторений звуковых сигналов: 1 (бип-бип), 2 (звук нарастающей частоты) и 3 (звук убывающей частоты) соответственно для полей белого, зелёного и чёрного цвета.
5. Движение робота с бампером и с сенсором освещённости, установленным горизонтально в направлении «вперёд», к источнику света по следующему алгоритму:
   a) сканирование сенсором (вместе с роботом) для поиска направления на источник с локальным максимумом света путём поворота по часовой стрелке, пока освещённость не убывает;
   b) поворот против часовой стрелки пока освещённость не убывает;
   c) возврат (поворот) к позиции максимума;
   d) движение до объекта – источника света с отступом при «срабатывании» контакт-сенсора (например, при обнаружении края стола) и остановом.

Дополнительные указания:

1. По возможности программируйте задержки с помощью таймеров.
2. Если ваш робот слишком быстрый (большие ведущие колеса), используйте команду `SetPower` с параметром 2-3 вместо 7 по умолчанию.
3. В задании 1 движение начинать в направлении против часовой стрелки. Учитывать, что направляющим элементом маршрута является граница (внутренняя или внешняя) чёрного и белого. Поэтому примерным порогом срабатывания (подключения поворота к прямолинейному движению) будет значение 40-45%. Его можно уточнить, измерив значение отражения от светового пятна, размещённого на указанной границе.

4. В заданиях 2 и 5 для поворота «на месте» используйте одновременное включение двигателей в противоположных направлениях. В задании 2 «необходимую коррекцию» направления выполнять через последовательные пробеги (примерно) в 1.0, 0.7 и 0.3 м.
5. В задании 3 используйте в качестве препятствий экраны с плохой (тёмные предметы) и хорошей (экран с алюминиевой фольгой) отражательными способностями. Запишите для отчёта (можно в комментариях к программе) примерные расстояния срабатывания вашего программного сенсора близости.
6. В задании 4 уточните диапазоны значений отражённого света (запишите для отчёта), используя переключение дисплея RCX кнопкой «View» на номер сенсора освещённости.

Контрольные вопросы

1. Как измерить текущий уровень освещённости объекта с учётом качества и цвета его поверхности с помощью сенсора освещённости и блока RCX?
2. В чём состоит идея построения искусственного «сенсора близости» для обнаружения препятствия перед роботом?
3. Как в программе используются таймеры для организации временных задержек?

Лабораторная работа №4. Программирование взаимодействия группы роботов с использованием средств коммуникации

Рассматриваемые нами роботы LEGO могут общаться через инфракрасный порт не только с ПК, но и друг с другом. Используя это, можно организовать множество сотрудничающих или конкурирующих роботов. Применяя, например, два «общающихся» блока RCX, можно построить одного «большого» робота, который может иметь до шести двигателей и до шести сенсоров.

Коммуникация, обобщённо говоря, работает следующим образом. Робот может использовать команду `SendMessage()`, чтобы послать значение (0-255) по инфракрасному порту. Все другие роботы получают это сообщение и хранят его. Программа в роботе может прочитать значение последнего полученного сообщения, используя `Message()`. Наконец, программа, основываясь на этом значении, может активировать выполнение роботом определённых действий.

Выдача приказов

Часто, когда действуют два или более роботов, один из них является лидером. Его называют *ведущим (master)*, а других роботов – *ведомыми (slave)*.

Робот «ведущий» посылает приказы «ведомым», и они выполняют их. Иногда «ведомые» могут послать информацию обратно «ведущему», например, в виде измеренного значения от сенсора. Таким образом, для коммуникации необходимо иметь 2 программы: одну – для «ведущего» и другую – для «ведомого»(ых).

Предположим, что существует только один «ведомый». Его программа состоит из бесконечного цикла. В ней устанавливается значение текущего сообщения в 0, при этом используется команда `ClearMessage()`. Затем программа ждёт, пока принятое значение станет не равным 0. Основываясь на полученном в сообщении значении, программа выполнит один из обслуживаемых ею приказов.

«Ведущий» имеет ещё более простую программу. Он посылает сообщения с соответствующим номером приказа, а затем немного ждёт. Например, он приказывает, чтобы «ведомый» двигался вперёд, затем через 2 секунды – назад, а затем, спустя ещё 2 секунды, остановился. Таким образом, «ведущий» дистанционно управляет «ведомым».

После того, как такие две программы написаны, необходимо загрузить их в роботов. Каждая программа должна попасть в один из роботов. Удостоверьтесь, что во время загрузки одного робота другой выключен. После загрузки можно включать обоих роботов и запускать программы: сначала – в «ведомом», а затем – в «ведущем».

Если имеется несколько «ведомых», нужно загрузить программу «ведомого» в каждого из них поочерёдно (не одновременно; см. ниже раздел «Предостережения»). Теперь все ведомые выполнят точно те же самые действия. Такой режим можно назвать *монопольный «ведущий» без адресации «ведомых»*.

Протокол коммуникации

Чтобы позволить роботам общаться друг с другом, им нужно договориться или, иначе говоря, определить так называемый **протокол.** Возможности его создания в нашем случае ограничиваются значением единственной функции `Message()`, возвращающей однобайтовое число 0-255. В пределах этих значений мы должны определить возможные адреса, команды и события нашей системы.

Предположим, что **1 означает команду «двигаться вперёд», 2 – «двигаться назад», 3 – «остановиться», 4 – «поворачивать вправо», 5 – «поворачивать влево», 6 – выдать повторяющийся звук «бип»…** Очевидно, что при необходимости список может быть продолжен.

Очень важно тщательно определять такие протоколы, в особенности, когда организовано множество коммуникаций. Например, ***когда имеется более одного «ведомого», можно определить протокол, в котором посылаются два числа (с маленьким интервалом между ними). Первое число – номер (адрес) «ведомого», а второе – фактический приказ***. «Ведомый», который проверит первое число, выполнит действие только в случае, если это – его адрес.

Установим в нашем протоколе, что адреса «ведомых» будут следующими:

- 241 – для робота 1-го варианта задания;
- 242 – для 2-го;
- 243 – для 3-го;
- 244 – для 4-го;
- 245 – для 5-го;
- 255 – для адресации всех «ведомых» сразу.

В последнем случае мы предусмотрели, что один и тот же «ведущий» в определённый момент времени может обращаться не только к конкретному роботу, но и ко всем роботам сразу. Для этого используется «особый» адрес, например, как у нас 255, означающий «всем ведомым». Обобщенно такой режим называется *широковещательным*.

Режим, при котором один и тот же ведущий может индивидуально общаться с каждым из группы ведомых, назовём *монопольный «ведущий» с адресацией «ведомых»*.

Установим также коды сообщений о событиях на «ведомых» роботах (предполагается, что события фиксируются сенсорами контакта или освещённости при их работе в булевском типе данных):

- 1X1 – срабатывание сенсора 1;
- 1X2 – срабатывание сенсора 2;
- 1X3 – срабатывание сенсора 3;

Здесь X означает № робота (в соответствии с вариантом).

Наконец, установим код подтверждения приёма сообщения: 200.

Определение лидера

Как показано выше, в группе из нескольких роботов каждый из них должен иметь собственную программу. Вместе с тем, было бы удобно загрузить одну одинаковую программу на всех роботов. Но тогда возникает вопрос: кто будет «ведущим»? Ответ прост: нужно разрешить им самим определять лидера, приказам которого другие будут следовать.

Сделать это можно, например, обеспечив при старте в каждом роботе ожидание сообщения в течение случайного времени и, если такового нет – посылку собственного сообщения. Тот робот, который посылает сообщение первым, становится лидером. Такая схема может быть

неудачной, если бы 2 робота задерживали сообщение на строго одинаковое время, что является весьма маловероятным событием.

Такую одинаковую программу необходимо загрузить во всех роботов (поочерёдно, а не одновременно; см. ниже раздел «Предостережения»). Затем нужно запустить роботов примерно одновременно и наблюдать, что происходит. Один из них должен взять командование, а другой(ие) должен следовать приказам. В редких случаях ни один из них не становится лидером. Для нейтрализации такого события требуется применение более сложных протоколов.

Вышеописанный режим можно назвать *случайный «ведущий» с адресацией «ведомых»*.

Предостережения

Необходима осторожность при работе с несколькими роботами, т.к. если 2 робота (или робот и компьютер) посылают информацию одновременно, она может быть потеряна.

Когда в робота загружается программа, он сообщает компьютеру, получена ли программа (или её части) правильно. Компьютер реагирует на это, отправляя новые части или повторно посылая ошибочные. Когда включены 2 робота, оба начнут сообщать компьютеру, правильно ли они получают программу. Компьютер не понимает этого, так как не знает, что есть два робота! В результате искажается контрольная сумма загружаемого кода. В результате – роботы не будут выполнять правильных действий**.**

Всегда удостоверьтесь, чтобы во время загрузки программы был включён только один робот!

Другая проблема состоит в том, что только один робот должен посылать сообщение в данный момент. Если сообщения посылаются одновременно, они могут потеряться. Кроме того, робот не может послать и получить сообщения в один и тот же момент времени. Когда только один робот посылает сообщения (есть только один «ведущий»), такой проблемы не существует, иначе она могла бы быть серьёзной. Например, представим себе, что нужно написать программу, в которой «ведомый» посылает сообщение, когда он сталкивается с чем-либо, для того чтобы «ведущий» мог принять меры по адекватному управлению. Но если одновременно «ведущий» пошлёт свой приказ, то оба сообщения будут потеряны.

Чтобы разрешить такую проблему, важно определить коммуникационный протокол так, чтобы в случае, если сообщения были потеряны, они могли бы быть восстановлены. Например, когда «ведущий» посылает команду, он должен получить ответ (подтверждение) от «ведомого». Если же он не получает ответ достаточно скоро, то должен повторно послать команду.

Иногда при работе с несколькими роботами возникает необходимость, чтобы только самый близкий робот получал бы

коммуникационный сигнал. Этого можно достичь, используя команду `SetTxPower(TX_POWER_LO)` в программе «ведущего». В этом случае посланный ИК-сигнал является очень слабым, и только робот, стоящий рядом и ориентированный на «ведущего», «услышит» этот сигнал. Это особенно полезно при постройке одного большого робота из нескольких блоков RCX. Чтобы снова установить робота в режим дальней передачи, используется команда `SetTxPower(TX_POWER_HI)`.

Задание

Используя инструментальную среду BrixCC, разработать и протестировать программы дистанционного управления роботами в группе *один «ведущий» – два «ведомых»* по следующим **вариантам исходных данных:**

1. Реализовать программу режима связи – ***монопольный «ведущий» с адресацией*** «ведомых» роботов 4 и 5 (соответственно их вариантам задания). Следуя установленному выше протоколу связи, послать команды роботам по следующим адресам:
   - 244 – «двигаться вперёд» и «остановиться» при срабатывании сенсора;
   - 245 – «поворачивать влево» и «остановиться» при срабатывании сенсора.

   При получении сообщения о событии на «ведомом» высылать ему подтверждение приёма и команду на останов. Воспроизвести (на «ведущем») звук «бип» то количество раз, которое соответствует номеру (адресу) ведомого.
2. Реализовать программу режима связи – ***случайный «ведущий» со случайной адресацией*** «ведомых» роботов 4 и 5 соответственно их вариантам задания. Следуя установленному выше протоколу связи, послать команды роботам по следующим адресам:
   - 244 – «двигаться назад» и, при срабатывании сенсора, «двигаться вперёд», включив звуковой сигнал (на «ведомом»);
   - 245 – «поворачивать вправо» и (после поворота примерно на 360 градусов) «остановиться»;

   При получении сообщения о событии на «ведомом» – высылать ему подтверждение приема и команду на останов.
3. Реализовать программу режима связи – ***монопольный и случайный (как вариант) «ведущий» с широковещательной адресацией*** остальных роботов. В соответствии с вышеуказанным протоколом связи послать последовательные команды роботам:
   - «двигаться назад» в течение эмпирически подобранной задержки и «остановиться»;

- «развернуться вправо» примерно на 360 градусов и «остановиться»;
- «двигаться назад» в исходную точку и «остановиться»;
- «развернуться влево» примерно на 180 градусов и «остановиться».

4. Реализовать программу связи – ***«ведомый» от монопольного «ведущего»*** (из вар.1) с адресами: 244 (для «своего» робота) и 242, 243 и 245 для остальных (по вариантам – последней цифре адреса). Посылать «ведущему» сообщение о своём событии до тех пор, пока не получено подтверждение (200) о приёме. Реализовать также исполнение команд «ведущего» на движение, поворот, останов и выдачу повторяющегося звукового сигнала.
5. Реализовать программу связи – ***«ведомый» от случайного и широко вещающего «ведущего».*** В первом случае исходить из адресов: 245 (для «своего» робота), 241, 242, 243 и 244 для остальных (по вариантам – последней цифре адреса). Посылать «ведущему» (в вариантах заданий 1 и 2) сообщение о событиях на собственных сенсорах до тех пор, пока не получено подтверждение (200) о приёме. Реализовать также исполнение команд «ведущего» на движение, поворот, останов и выдачу повторяющегося звукового сигнала.

Дополнительные указания:

1. При выполнении работы приветствуется сотрудничество студентов вариантов 1 и 2 со студентами вариантов 4 и 5. Тестирование проводить в группе из трёх роботов.
2. Желательно сотрудничество студентов вариантов 3 и 5. Объединённая программа тестируется в группе из трёх роботов.
3. Направлением движения робота «вперед» считать главное направление излучения ИК-порта контроллера RCX.
4. В варианте 1 исходное положение для тестирования группы из трёх роботов – на горизонтальной площадке, «ведущий» удален от «ведомых» примерно на 2 м. «Ведомые» располагаются примерно на расстоянии 30 см друг от друга, а их ИК-порты видят ИК-порт «ведущего».
5. В вариантах 2 и 3 исходное положение для тестирования группы из трёх роботов – равномерное размещение на горизонтальной площадке по периферии воображаемого круга диаметром примерно 1 м. Ориентация ИК-портов на центр круга.
6. Срабатывание сенсоров на ведомых роботах осуществлять предпочтительно при ориентации их ИК-портов на «ведущего». В варианте 4 событие генерируется сенсором освещённости, в варианте 5 – сенсором контакта.

Контрольные вопросы

1. Перечислите инструкции языка NQC для работы с ИК-портом блока RCX.
2. Что такое «протокол коммуникации» и что в нём должно быть определено?
3. Понятие «ведущий» и «ведомый» в коммуникации и варианты их назначения.

Отчёт о работе

При защите лабораторных работ №2, 3 и 4 студенты предоставляют отчёты о выполнении, содержащие:

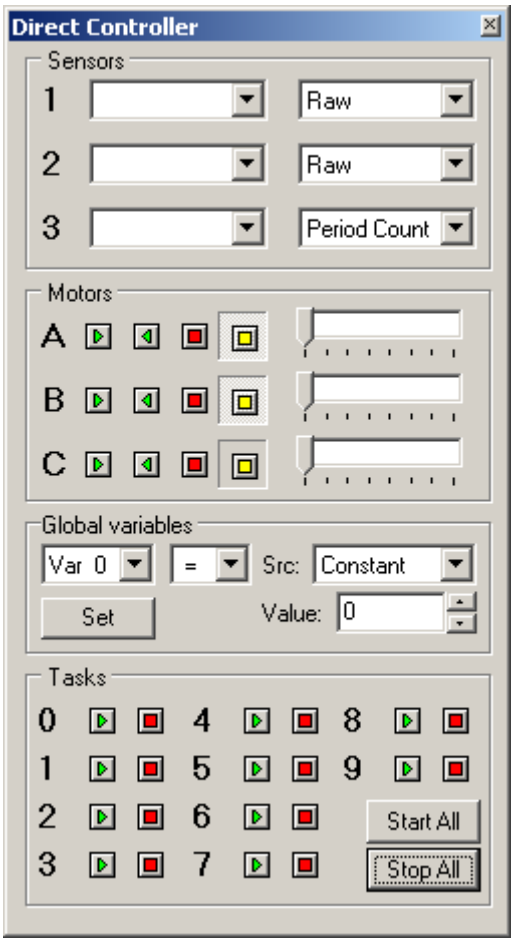
1) задание с указанием варианта;
2) текст разработанной программы;
3) краткое описание использованного в программе алгоритма.

На титульном листе отчёта должны быть указаны: номер и тема лабораторной работы, название дисциплины, ФИО студента, № группы, ФИО преподавателя и год выполнения работы.

Приложение 1. Меню инструментов в среде разработки BricxCC

Прямое управление

Чтобы непосредственно управлять микроконтроллерным блоком робота RCX, выберите команду `Direct Control` из меню `Tools`. Это откроет окно со следующими группами объектов:



Сенсоры (Sensors)

Здесь можно установить типы трёх сенсоров и типы данных от каждого из них.

Двигатели (Motors)

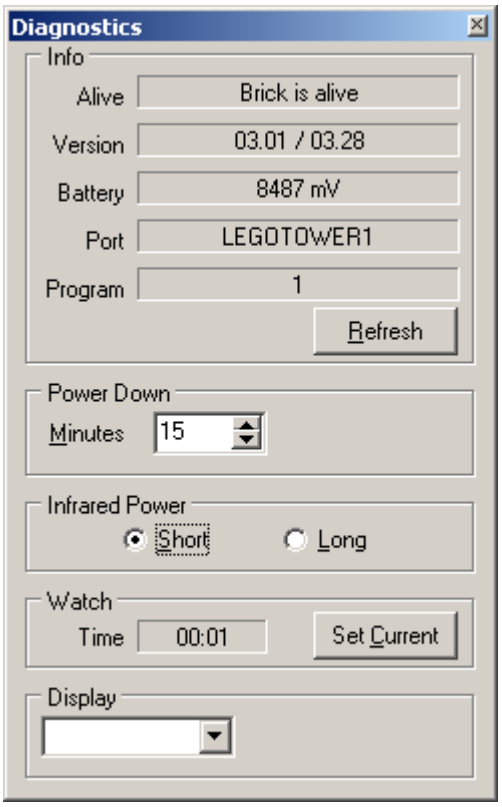
Эта группа позволяет непосредственно управлять тремя двигателями робота. Для каждого двигателя имеется четыре варианта: движение вперёд, назад, отключение `Off` (с торможением) и `float` (плавное). Можно также установить скорость каждого двигателя с помощью ползунков A, B и C.

Переменные (Global variables)

Переменные в блоке управления можно установить здесь. Слева выберите переменную, которую хотелось бы изменить; в середине – операцию (установить, добавить, вычесть, умножить, разделить, логическое И, логическое ИЛИ); справа – задайте значение с помощью стрелок «вверх» и «вниз», или путём ввода с клавиатуры. Нажмите `Set`, чтобы сделать изменение в блоке управления.

Задачи (Tasks)

В данной группе можно запустить и остановить 10 задач, которые находятся в текущей программе блока управления. Если начинают задачу, которая уже запущена, она запустится повторно. «Старт» (кнопка с зелёным символом) и «остановка» (красный символ) задач необходимы для того, чтобы проверить программный код и организовать удалённое управление роботов. Для последнего можно использовать джойстик.



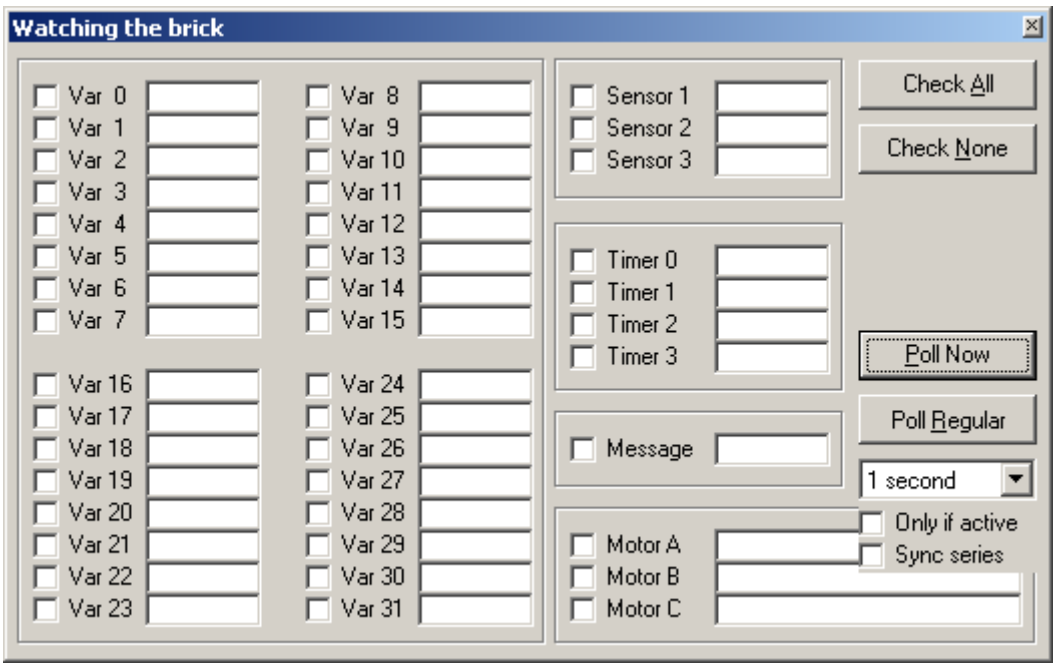
Диагностика

Пункт `Diagnostics` в меню `Tools` открывает окно, показывающее информацию о блоке управления. Например, включён ли блок управления, версия низкоуровневой ОС (`firmware`), состояние элементов питания и т.д. (Если блок управления не включён, включите его и нажмите кнопку `Refresh`, чтобы повторно проверить информацию).

В группе `Infrared Power` данного окна можно установить дальность ИК-связи с блоком управления, а в `Power Down` – период автоматического отключения бездействующего робота. Наконец, то, что пользователь хочет показать на дисплее RCX, можно выбрать из списка `Display`. Это могут быть показания текущего времени, значения входов, выходов и пр.

Наблюдение за блоком управления

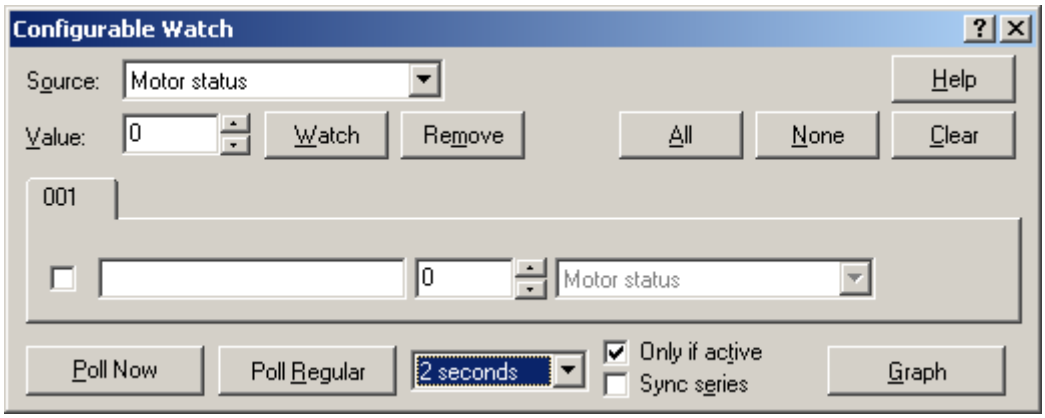
Щелчок пункта `Watching the Brick` в меню `Tools` открывает окно, в котором можно опросить состояние сенсоров, двигателей, переменных и т.д блока управления. Укажите пункты, которые нужно опросить, помечая чекбоксы перед ними. (Можно выбрать или отвергнуть все пункты, нажимая на соответствующие кнопки у основания.) После выбора пунктов нажмите кнопку `Poll Now` (опросить сейчас), чтобы получить текущий статус.



Можно также непрерывно запрашивать информацию. Для этого выберите временной интервал из списка справа и нажмите кнопку `Poll Regular` (опрашивать систематически). Следует учитывать, что опрос блока управления занимает некоторое время. Если требуется запрашивать информацию много и регулярно, то это замедлит как компьютер, так и робота.

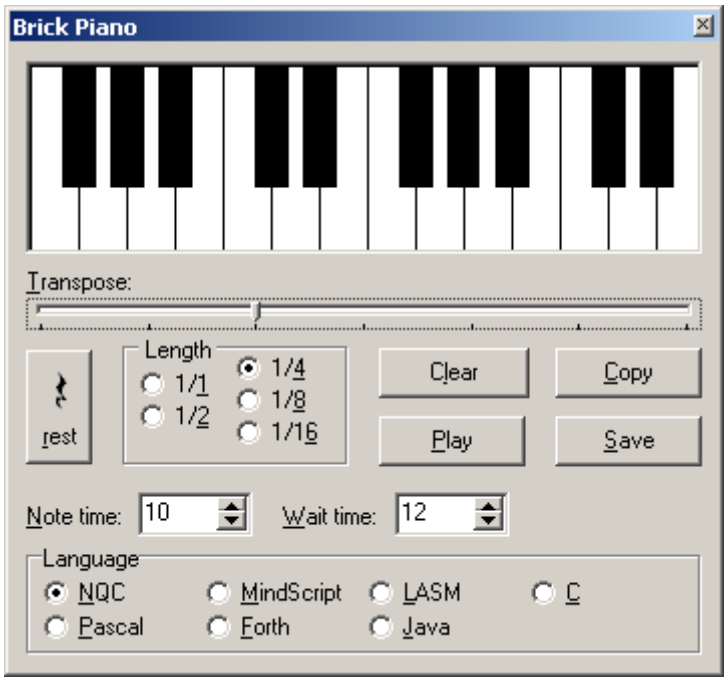
Конфигурируемое наблюдение

Нижеследующее окно предназначено для отслеживания значений от различных источников данных в RCX2, а именно: *переменная, таймер, константа, состояние мотора, случайный генератор, программный слот, значение сенсора, тип сенсора.*



При выборе кнопки `Graph` можно выполнить графический анализ данных во времени.

«Фортепиано» блока управления

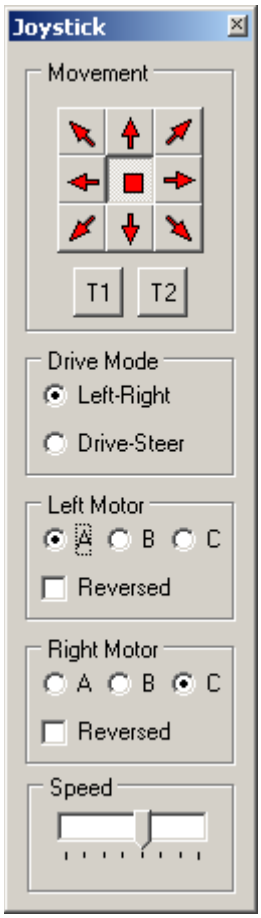


Выбирая пункт `Brick Piano` в меню `Tools`, можно заставить блок управления воспроизводить музыку. Если сыграть с помощью манипулятора «мышь» на изображении клавиш фортепьяно, блок управления издаст соответствующие звуки. В середине окна слева можно выбрать длину нот `(Length)`. Можно добавить к музыке паузы, используя кнопку `rest`.

Последовательность сыгранных нот сохраняется. Нажмите кнопку Play, чтобы сыграть последовательность целиком. Нажмите Clear, чтобы очистить сохраненные ноты.

Нажатие кнопки Save даёт возможность сохранить коллекцию нот как программу, которую затем можно открыть в редакторе и загрузить в робота. Наконец, щелчком на Copy коллекция нот копируется в виде кода в буфер обмена Windows. В редакторе BricxCC можно вставить полученный код в свою программу. Отметим, что «фортепиано» блока управления определяет две константы в коде NQC: __NOTETIME и __WAITTIME. Их можно изменить, чтобы изменить скорость проигрывания мелодии.

Джойстик блока управления

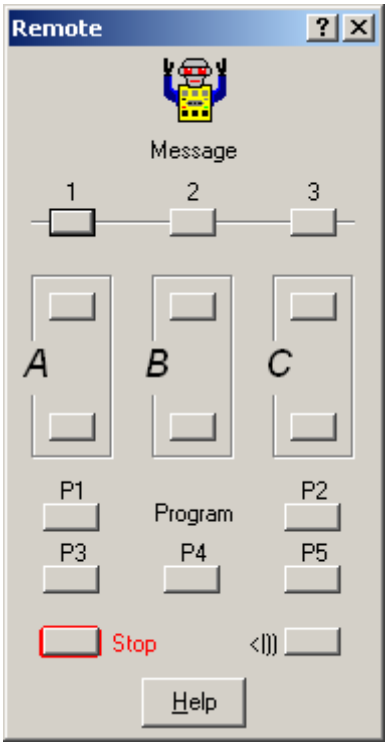


Если необходимо настроить робота для дистанционного управления от компьютера, выберите пункт Brick Joystick в меню Tools. Джойстик блока управления может работать в двух режимах. Первый режим (Left-Right) предполагает, что оба двигателя ведут левое и правое колёса робота. Второй режим (Drive-Steer) предполагает, что один двигатель используется для движения, а другой – для руления. Ниже можно указать, какой двигатель является левым (или ведущим), а какой – правым (или рулящим). Возможно также указать, должны ли они быть реверсивными. Наконец, можно ползунком Speed установить скорость обоих двигателей.

Чтобы управлять роботом с помощью мыши, нажимайте на кнопки с красными стрелками (в верхней части окна). Если используется левая кнопка мыши, робот постоянно движется в текущем направлении, пока кнопка мыши нажата. Если же используется правая, робот продолжает перемещаться, пока не сделан щелчок другой кнопкой. Щелчок на кнопках T1 или T2 запускает задачи №1 или №2, имеющиеся в программе, загруженной в блок управления.

Чтобы управлять роботом с клавиатуры, используйте числовую вспомогательную клавиатуру (удостоверьтесь, что клавиша NumLock нажата). Пока соответствующие клавиши удерживаются нажатыми, робот движется в выбранном направлении.

Дистанционное управление

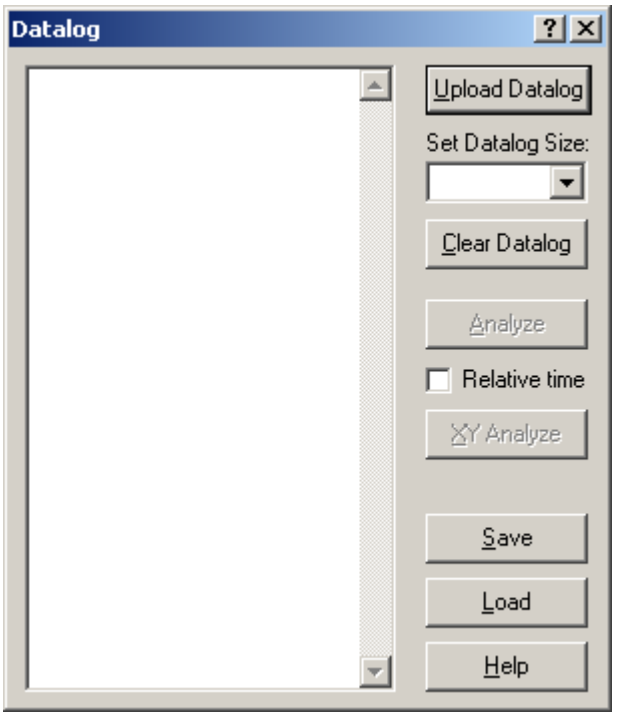


Нажмите пункт Remote в меню Tools, чтобы вызвать соответствующее окно. RCX доступен для управления через специальный пульт дистанционного управления Mindstorms. Данное окно эмулирует удалённое управление от пульта, посылая эквивалентные команды блоку управления.

Datalog

RCX поддерживает возможность регистрации текущих данных (Datalog). RCX может записать данные в собственный архив datalog, который потом может быть выгружен в компьютер. Создание datalog и запись данных в него выполняются в соответствии с программой, работающей в RCX. Выгрузка datalog должна быть сделана компьютером.

Пункт Datalog в меню Tools открывает окно, в котором можно отобразить текущий datalog. Для этого нажмите кнопку, помеченную Upload Datalog.



Можно также изменить размер архива datalog, выбирая или впечатывая числа в окошке Set Data Size. Следует учитывать, что каждая выбранная единица потребует 3 байтов памяти в RCX. Поскольку память в RCX является довольно маленькой, предпочтительно для datalog определять её небольшую область. Изменение размера datalog приведет к стиранию его содержимого. Кнопка Clear Datalog устанавливает текущий нулевой размер памяти.

Как только данные из RCX выгружены, их можно проанализировать (то есть изобразить данные в виде графика) двумя способами. Первый метод, выбираемый через кнопку Analyze, рассматривает каждую точку данных одинакового типа как значение Y от величины X, инкрементируемой для каждой новой точки

данных. Второй метод (XY Analyze) рассматривает данные как множество пар X-Y (то есть каждые два выгруженных элемента данных объединены в одну координату X,Y). Анализ основывается на изображении данных, полученных через опрос блока управления в окне Watching the Brick, в виде графика.

Содержание окна Datalog можно сохранить в файл, нажимая кнопку Save. Также можно и загрузить предварительно сохранённый datalog, используя кнопку Load.

Очистка памяти

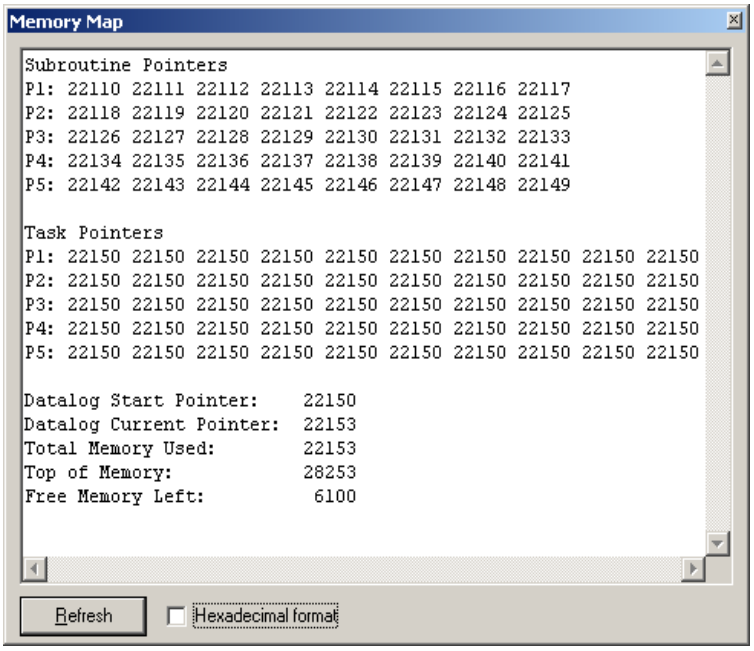
Нажмите пункт Clear Memory в меню Tools, чтобы очистить память в блоке управления. Эта команда удаляет все задачи и подпрограммы во всех программах, а также удаляет RCX datalog (если он существует). Очистка памяти важна, когда загружаются большие программы, потому что блок управления содержит все программы и задачи, даже если Вы выключаете его. После очистки памяти для вашей программы в RCX становятся доступны примерно 6 КБ.

Карта памяти

Выбирая пункт Memory Map в меню Tools, можно получить информацию о памяти блока управления. Это полезно только для отладки, когда Вы сталкиваетесь с проблемами в блоке управления, имеющими отношение к памяти. В частности, этот инструмент полезен, если есть подозрение, что память для собственных программ недостаточна.

Для RCX выдаются следующие данные:

- ✓ стартовые адреса 8 подпрограмм в каждой программе
- ✓ стартовые адреса 10 задач в каждой программе
- ✓ стартовый адрес datalog
- ✓ адрес последней точки datalog
- ✓ общий объём используемой памяти
- ✓ вершина пользовательской памяти
- ✓ количество свободной памяти



Приложение 2. Краткие сведения о языке NQC

Основные операторы

Команда	Описание
while `(cond) body`	Выполнить `body` ноль или более раз, пока условие верно
do `body while(cond)`	Выполнить `body` один или более раз, пока условие верно
until `(cond) body`	Выполнить `body` ноль или более раз, пока условие не верно
break	Выход из тела `while/do/until`
continue	Пропустить до следующего повторения тела `while/do/until`
repeat `(expression) body`	Повторить `body` указанное количество раз
if `(cond) stmt1` **if** `(cond) stmt1` **else** `stmt2`	Выполнить stmt1, если условие верно. Выполнить stmt2 (если присутствует), если условие ложно.
start `task_name`	Начать указанную задачу
stop `task_name`	Остановить указанную задачу
`function(args)`	Вызвать функцию с указанными аргументами
`var = expression`	Оценить выражение и присвоить переменной
`var += expression`	Оценить выражение и добавить к переменной
`var -= expression`	Оценить выражение и вычесть из переменной
`var *= expression`	Оценить выражение и умножить на переменную
`var /= expression`	Оценить выражение и разделить на переменную
`var |= expression`	Оценить выражение и выполнить поразрядное ИЛИ с переменной
`var &= expression`	Оценить выражение и выполнить поразрядное И с переменной
return	Возврат из функции в место вызова
`expression`	Оценить выражение

Условия

Условия используются с операторами управления для принятия решений. В большинстве случаев условие влечёт сравнение выражений.

Условие	Значение
true	всегда истинно
false	всегда ложно
`expr1 == expr2`	проверить, равны ли выражения
`expr1 != expr2`	проверить, не равны ли выражения
`expr1 < expr2`	проверить, что одно выражение меньше, чем другое
`expr1 <= expr2`	проверить, что одно выражение меньше другого или равно ему
`expr1 > expr2`	проверить, что одно выражение больше, чем другое
`expr1 >= expr2`	проверить, что одно выражение больше другого или равно ему
`!condition`	логическое отрицание условия
`cond1 && cond2`	логическое И двух условий (истинно, если и только если оба условия верны),
`cond1 || cond2`	логическое ИЛИ двух условий (истинно, если и только если, по крайней мере, одно из условий верно),

Выражения

Есть множество различных величин, которые могут использоваться в выражениях, включая константы, переменные и значения сенсора. Отметим, что SENSOR_1, SENSOR_2 и SENSOR_3 – макроопределения, которые расширяются соответственно до SensorValue(0), SensorValue(1) и SensorValue(2).

Величина	**Описание**
`number`	Постоянное значение (например, "123")
`variable`	Именованная переменная (например, "x")
`Timer(n)`	Значение таймера n, где n - между 0 и 3
`Random(n)`	Случайное число между 0 и n
`SensorValue(n)`	Текущее значение сенсора n, где n - между 0 и 2
`Watch()`	Значение часов системы
`Message()`	Значение последних полученных ИК сообщений

Значения можно объединить, используя операторы. Некоторые операторы могут использоваться только в оценке постоянных выражений, что означает, что их операнды должны либо быть константами, либо выражениями, включающими только константы. Операторы перечислены ниже в порядке уменьшения их приоритета.

Оператор	**Описание**	**Ассоциативность**	**Ограничение**	**Пример**

`abs()`
 	
`sign()`
	Абсолютное значение
	Значение операнда
		не применяется
		
		не применяется
				`abs(x)`
				
				`sign(x)`

`++` `--`	Инкремент Декремент	слева слева	только переменные только переменные	`x++ or ++x` `x-- or --x`

`-`
 	
`~`
	Унарный минус
	Поразрядное отрицание (унарное)
		справа
		справа
			только константа	`-x`
				`~123`

`*`
`/`
`%`
	Умножение
	Деление
	Модуль
		слева
		слева
		слева
			только константа	`x * y`
				`x / y`
				`123 % 4`

`+` `-`	Сложение Вычитание	слева слева		`x + y` `x - y`
`<<` `>>`	Левый сдвиг Правый сдвиг	слева слева	только константа только константа	`123 << 4` `123 >> 4`
`&`	Поразрядное И	слева		`x & y`
`^`	Поразрядное XOR	слева	только константа	`123 ^ 4`
`|`	Поразрядное ИЛИ	слева x		`x | y`
`&&`	Логическое И	слева	только константа	`123 && 4`
`||`	Логическое ИЛИ	слева	только константа	`123 || 4`

Основные функции контроллера RCX

Большинство функций требует, чтобы все аргументы были постоянными выражениями (числами или операциями, включающими другие постоянные выражения). Исключения – функции, которые используют сенсор как аргумент, а также те, которые могут использовать любое выражение. В случае сенсоров аргументом должно быть имя датчика: SENSOR_1, SENSOR_2 или SENSOR_3. В некоторых случаях имеются предопределённые имена (например, SENSOR_TOUCH) для соответствующих констант.

Функция	**Описание**	**Пример**
`SetSensor(sensor, config)`	Сконфигурировать сенсор	`SetSensor(SENSOR_1, SENSOR_TOUCH)`
`SetSensorMode(sensor, mode)`	Установить тип данных сенсора	`SetSensor(SENSOR_2, SENSOR_MODE_PERCENT)`
`SetSensorType(sensor, type)`	Установить тип сенсора	`SetSensor(SENSOR_2, SENSOR_TYPE_LIGHT)`
`ClearSensor(sensor)`	Очистить показание сенсора	`ClearSensor(SENSOR_3)`
`On(outputs)`	Включить один или более выходов	`On(OUT_A + OUT_B)`
`Off(outputs)`	Выключить один или более выходов	`Off(OUT_C)`
`Float(outputs)`	Выполнить плавный останов	`Float(OUT_B)`
`Fwd(outputs)`	Установить выходы на курс ВПЕРЕД	`Fwd(OUT_A)`
`Rev(outputs)`	Установить выходы на курс НАЗАД	`Rev(OUT_B)`
`Toggle(outputs)`	Переключить направление выходов	`Toggle(OUT_C)`
`OnFwd(outputs)`	Включить в прямом направлении	`OnFwd(OUT_A)`
`OnRev(outputs)`	Включить в обратном направлении	`OnRev(OUT_B)`
`OnFor(outputs, time)`	Включить на указанное число сотых секунды. Время может быть выражением.	`OnFor(OUT_A, 200)`
`SetOutput(outputs, mode)`	Установить режим выхода	`SetOutput(OUT_A, OUT_ON)`
`SetDirection(outputs, dir)`	Установить направление выхода	`SetDirection(OUT_A, OUT_FWD)`
`SetPower(outputs, power)`	Установить уровень выходной мощности (0-7). Мощность может быть выражением.	`SetPower(OUT_A, 6)`
`Wait(time)`	Ожидать указанное время в сотых секунды. Время может быть выражением.	`Wait(x)`

`PlaySound(sound)`	Воспроизвести указанный звук (0-5).	`PlaySound(SOUND_CLICK)`
`PlayTone(freq, duration)`	Воспризвести тон указанной частоты на указанное время (в десятых от секунды)	`PlayTone(440, 5)`
`ClearTimer(timer)`	Сбросить таймер (0-3) на значение 0	`ClearTimer(0)`
`StopAllTasks()`	Остановить все в настоящее время запущенные задачи	`StopAllTasks()`
`SelectDisplay(mode)`	Выбрать один из 7 режимов дисплея: 0: часы системы, 1-3: значение сенсора, 4-6: установка выхода. Режим может быть выражением.	`SelectDisplay(1)`
`SendMessage(message)`	Послать ИК-сообщение (1-255) . Сообщение может быть выражением.	`SendMessage(x)`
`ClearMessage()`	Очистить буфер ИК-сообщения	`ClearMessage()`
`CreateDatalog(size)`	Создать новый datalog данного размера	`CreateDatalog(100)`
`AddToDatalog(value)`	Добавить значение к datalog. Значение может быть выражением.	`AddToDatalog(Timer(0))`
`SetWatch(hours, minutes)`	Установить значение часов системы	`SetWatch(1,30)`
`SetTxPower(hi_lo)`	Установить уровень мощности ИК-передатчика на низкое или высокое значение	`SetTxPower(TX_POWER_LO)`

Константы RCX

Многие из значений для функций RCX сделаны именованными константами, что помогает сделать код удобочитаемым. Чаще используйте именованные константы, а не простое числовое значение.

Для конфигурации сенсора в SetSensor()	`SENSOR_TOUCH, SENSOR_LIGHT, SENSOR_ROTATION, SENSOR_CELSIUS, SENSOR_FAHRENHEIT, SENSOR_PULSE, SENSOR_EDGE`
Для типов данных от сенсоров в SetSensorMode()	`SENSOR_MODE_RAW, SENSOR_MODE_BOOL, SENSOR_MODE_EDGE, SENSOR_MODE_PULSE, SENSOR_MODE_PERCENT, SENSOR_MODE_CELSIUS, SENSOR_MODE_FAHRENHEIT, SENSOR_MODE_ROTATION`
Для типов сенсоров в SetSensorType()	`SENSOR_TYPE_TOUCH, SENSOR_TYPE_TEMPERATURE, SENSOR_TYPE_LIGHT, SENSOR_TYPE_ROTATION`
Для выходов в On(), Off() и т.д..	`OUT_A, OUT_B, OUT_C`

Для режимов в SetOutput()	OUT_ON, OUT_OFF, OUT_FLOAT
Для направления в SetDirection()	OUT_FWD, OUT_REV, OUT_TOGGLE
Для выходной мощности в SetPower()	OUT_LOW, OUT_HALF, OUT_FULL
Для звуков в PlaySound()	SOUND_CLICK, SOUND_DOUBLE_BEEP, SOUND_DOWN, SOUND_UP, SOUND_LOW_BEEP, SOUND_FAST_UP
Для режимов в SelectDisplay()	DISPLAY_WATCH, DISPLAY_SENSOR_1, DISPLAY_SENSOR_2, DISPLAY_SENSOR_3, DISPLAY_OUT_A, DISPLAY_OUT_B, DISPLAY_OUT_C
Уровень мощности передатч. в SetTxPower()	TX_POWER_LO, TX_POWER_HI

Ключевые слова

Ключевые слова – это слова, зарезервированные компилятором NQC для собственно языка программирования. Ошибкой является использование любого из них как имени функции, задачи или переменной. Установлены следующие ключевые слова: `__sensor`, `abs`, `asm`, `break`, `const`, `continue`, `do`, `else`, `false`, `if`, `inline`, `int`, `repeat`, `return`, `sign`, `start`, `stop`, `sub`, `task`, `true`, `void`, `while`.

Подпрограммы

Иногда во многих местах программы необходима одинаковая часть кода. Её в этом случае можно поместить в подпрограмме, которая описывается с помощью ключевого слова `sub` и некоторого имени с круглыми скобками на конце:

```
sub turn_around()
{
     ...
}
task main()
{
     ..
     turn_around();
     ...
     turn_around();
     ...
     turn_around();
     ...
}
```

Вызывается подпрограмма путём указания в необходимом месте её имени (также с «пустыми» параметрами). NQC (а фактически – контроллер RCX) учитывает максимум 8 подпрограмм.

Следует сделать некоторые предупреждения. Дело в том, что в NQC подпрограммы являются несколько необычными. Например, их нельзя вызвать из других подпрограмм. Подпрограммы можно вызывать из различных задач, но это не поощряется, т.к. возникает проблема возможного запуска одной и той же подпрограммы одновременно различными задачами. Всё это может приводить к нежелательным эффектам. Кроме того, при вызове подпрограммы из различных задач становится невозможным использовать усложнённые выражения из-за ограничений в программируемом оборудовании RCX.

Итак, ***не вызывайте подпрограмму из различных задач***, если точно не известно, для чего она нужна.

Функции

Как сказано выше, подпрограммы могут создавать определённые проблемы. Их преимуществом является то, что они сохраняются в памяти RCX только один раз, позволяя её экономить.

Но когда подпрограммы короткие, лучше использовать вместо них функции. Они не сохраняются отдельно, а копируются в каждом месте, где используются. Это занимает больше памяти, но проблемы, подобные использованию сложных выражений, больше не возникают. Здесь также нет никакого ограничения на количество встроенных функций.

Определение и вызов функций происходит точно так же, как и в подпрограммах. Только используется ключевое слово **void**, а не **sub**:

```
void turn_around()
{
      ...
}
task main()
{
      ...
      turn_around();
      ...
      turn_around();
      ...
      turn_around();
      ...
}
```

Функции в отличие от подпрограмм могут иметь аргументы, использующиеся для передачи значений определённых переменных. Они

размещаются в круглых скобках позади имени функции и отделяются друг от друга запятыми.

Определение макросов

Существует ещё одна возможность задать имя маленьким частям кода. Можно определить макрос в NQC (не путать с макросом в RCX Command Center). Мы можем определить константы, используя оператор `#define` и задавая им имя. Таким же образом можно определить любую часть кода.

Пример программы с использованием макроса для разворотов:

```
#define turn_around OnRev(OUT_C);Wait(340);OnFwd(OUT_A+OUT_C);
task main()
{
     ...
     turn_around;
     ...
     turn_around;
     ...
     turn_around;
     ...
}
```

После оператора `#define` и слова `turn_around` в одной строке следует текст. Далее везде, где в программе написано слово `turn_around`, оно заменяется кодом этого текста. Хотя и существует режим оператора `#define` для многих строк, однако он не рекомендуется для использования.

Операторы `#define` могут иметь аргументы. Например, для движения вперёд имеется два аргумента: скорость и время.

```
#define
forwards(s,t)SetPower(OUT_A+OUT_C,s);OnFwd(OUT_A+OUT_C);Wait(t);
task main()
{
     forwards(3,200);
}
```

Очень полезно определить такой макрос. Он делает код более компактным и удобочитаемым. Кроме того, можно проще изменить код, когда, например, изменяются связи с двигателями.

Программирование музыки

Для более сложного воспроизведения звуков (по сравнению с изложенным ранее) язык NQC имеет команду `PlayTone()`.Она имеет два аргумента. Первый – частота тона, второй – его продолжительность (в тиках по 1/100 секунды, как в команде ожидания).

Вот таблица полезных частот:

Тон	1	2	3	4	5	6	7	8
G#	52	104	208	415	831	1661	3322	
G	49	98	196	392	784	1568	3136	
F#	46	92	185	370	740	1480	2960	
F	44	87	175	349	698	1397	2794	
E	41	82	165	330	659	1319	2637	
D#	39	78	156	311	622	1245	2489	
D	37	73	147	294	587	1175	2349	
C#	35	69	139	277	554	1109	2217	
C	33	65	131	262	523	1047	2093	4186
B	31	62	123	247	494	988	1976	3951
A#	29	58	117	233	466	932	1865	3729
A	28	55	110	220	440	880	1760	3520

Как и для простых звуков, здесь RCX также не ждет завершения тона. Так, если используется много тонов подряд, лучше добавлять команды ожидания между ними. Вот пример:

```
task main()
{
PlayTone(262,40); Wait(50); PlayTone(294,40); Wait(50);
PlayTone(330,40); Wait(50); PlayTone(294,40); Wait(50);
PlayTone(262,160); Wait(200);
}
```

Если необходимо получить воспроизведение музыки RCX при движении робота, используйте лучше отдельную задачу. Ниже дан пример программы довольно примитивной задачи, в которой робот двигается назад и вперед, постоянно выдавая музыку.

```
task music()
{
while (true)
      {
      PlayTone(262,40); Wait(50);
      PlayTone(294,40); Wait(50);
      PlayTone(330,40); Wait(50);
      PlayTone(294,40); Wait(50);
      }
}
task main()
{
start music;
while (true)
      {
      OnFwd(OUT_A+OUT_C); Wait(300);
      OnRev(OUT_A+OUT_C); Wait(300);
      }
}
```

Оглавление

Учебное издание

ПРОГРАММИРОВАНИЕ УПРАВЛЕНИЯ МОБИЛЬНЫМИ РОБОТАМИ LEGO MINDSTORMS RCX 2.0

Составители:
БАЙКАЛЬЦЕВ Борис Петрович
БЕЛИКОВ Александр Борисович
ТИШКИН Алексей Михайлович

Редактор Е.С. Резникова
Тех. редактор О.Г. Завьялова

Подписано в печать 19.12.12 Формат 60x84/16.
Бумага офсетная. Печать – ризография.
Усл. печ. л. 2,5. Уч.-изд. л.2,2. Тираж 30 экз.
Заказ . Бесплатно. Изд. № 81.

Московский институт электроники и математики Национального исследовательского университета «Высшая школа экономики».
109028, Москва, Б. Трехсвятительский пер., 3.
Отдел оперативной полиграфии Московского института электроники и математики Национального исследовательского университета «Высшая школа экономики».
113054, Москва, ул. М. Пионерская, 12.