Учебное пособие по командам языка Not eXactly C (NXC)

Версия 0.1

24.04.2010

Автор: Колотов Александр

Оглавление

Введение

Данное пособие является дополнением к образовательному курсу «Программирование Lego-роботов». Цель данного пособия – облегчить понимание основных конструкций языка программирования Not eXactly C, привести примеры использования функций по управлению моторами Lego, работы с датчикам и т.д.

Данное пособие не рассматривает детали языка программирования NXC. Оно не является справочным руководством по данному языку.

Официальное справочное руководство по языку NXC на английском языке доступно в сети Internet по адресу: http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_Guide.pdf

Учебное пособие рекомендуется использовать как набор примеров для упрощения адаптации к новому языку программирования. Примеры призваны ускорить приобретение навыков по программированию Lego-роботов.

Язык программирования NXC – программное обеспечение с открытым исходным кодом. Последние версии данного программного обеспечения можно найти в сети Internet - http://bricxcc.sourceforge.net/nbc/.

Примеры в данном пособии были скомпилированы и проверены на работоспособность с использованием BricxCC (версия 3.3.8.6), NBC/NXC (версия 1.2.1.1) на роботе, собранном на базе LEGO Mindstorms NXT 1.0.

Основы управления моторами

В NXC существует около 40 команд для управления моторами. Их условно можно разделить на команды управления движением, команды остановки, команды опроса датчиков и состояния моторов.

В данной главе рассматриваются команды управления движением.

Пример 1. OnFwd(), OnRev().

Данный пример предназначен для рассмотрения семейств команд OnFwd(), OnRev(), OnFwdSync(), OnRevSync() и его цель - проверить на то, что после OnFwd() управление сразу передается на следующую команду.

```
task main() {
    /*
    Даже если команда "движение вперед" задана до команды "движение назад",
    робот едет сразу назад, поскольку OnFwd() сразу передает управление
    следующей команде.
    */

    OnFwd(OUT_AB, 50);
    OnRev(OUT_AB, 50);
}
```

Пример 2. OnFwdSync(), OnRevSync().

Этот пример подобен предыдущему, только использует другие команды того же семейства. Он выполняет проверку на то, что после OnFwdSync() управление сразу передается на следующую команду.

```
task main() {
    /*
    Даже если команда "движение вперед" задана до команды "движение назад",
    робот едет сразу назад, поскольку OnFwdSync() сразу передает управление
    следующей команде.
    */

    //         Двигатели
    //             |    мощность
    //             |    |   распределение нагрузки между моторами
    //             |    |   |
    OnFwdSync(OUT_AB, 50, 0);
    OnRevSync(OUT_AB, 50, 0);
}
```

Пример 3. RotateMotor().

Здесь рассматривается команда управления моторами, принадлежащая другому семейству: RotateMotor(), RotateMotorEx(). Цель пример - показать отличие от предыдущего семейства команд - RotateMotor() передает управление к следующей команде только после того, как мотор передвинется на заданное количество градусов.

```
task main() {
    /*
    RotateMotor() работает как функция в "обычном" ее понимании - управление
    к следующей команде происходит только после полного выполнения текущей.
    */

    //        Двигатели
    //            |      мощность
    //            |      |   угол поворота оси мотора в градусах
    //            |      |   |
    RotateMotor(OUT_AB, 50, 360);
    RotateMotor(OUT_AB, 50, -360);
}
```

Пример 4. RotateMotorEx().

Функция RotateMotorEx() является одной из самых мощных функций управления моторами из-за больщого количества параметров, которые она может контролировать. Эта функция, как и предыдущая, также передает управление к следующей команде только после того, как мотор передвинется на заданное количество градусов.

```
task main() {
    /*
    RotateMotorEx() работает как функция в "обычном" ее понимании -
    управление к следующей команде происходит только после полного
    выполнения текущей.
    */

    //          Двигатели
    //              |      мощность
    //              |      |   угол поворота оси мотора в градусах
    //              |      |   |    распределение движения между моторами
    //              |      |   |    |  синхронизация моторов
    //              |      |   |    |  |     включать торможение после
    //              |      |   |    |  |     |  окончания движения или
    //              |      |   |    |  |     |  останавливаться своим ходом
    RotateMotorEx(OUT_AB, 50, 360, 0, true, true);
    RotateMotorEx(OUT_AB, 50, -360, 0, true, true);
}
```

Пример 5. RotateMotor() и циклы.

Вызов RotateMotor() (а также и RotateMotorEx) в цикле приводит к серии отрывистых движений.

```
task main() {
    /*
    Переход к следующей итерации цикла происходит только после выполнения
    "движение тележки", поэтому в итоге тележка двигается прерывисто,
    неоднородно.
    */

    int i = 0;
    while (i < 3) {
        RotateMotor(OUT_AB, 50, 360);
        i++;
    }
}
```

Пример 6. OnFwd() и циклы.

В отличие от RotateMotor(),вызов OnFwd() (так же как и OnRev, OnFwdSync, OnRevSync) в цикле позволяет почти одновременно с движением проверять состояние системы (значения переменных, датчиков, таймеров), т.е. движение не будет прерываться на время проверки.

```
task main() {
    /*
    Приведенную ниже конструкцию можно условно описать словами следующим
    образом:
    - начинаем двигаться
    - проверяем i
    - все еще двигаемся
    - проверяем i
    - все еще двигаемся
    . . .
    - цикл закончился.

    Поскольку команды серии (OnFwd) передают управление сразу следующей
    команде, переход к следующей итерации осуществляется сразу после начала
    движения. Останова двигателей при этом не происходит, т.е. общее ощущение
    от движения – плавность, беспрерывность.
    */

    int i = 0;
    while (i < 10000) {
        OnFwd(OUT_AB, 50);
        i++;
    }
}
```

Циклы

Как и в обычном C, в NXC поддерживается несколько видов циклов: for, while, do … while, repeat. Есть даже не совсем обычная команда until, которая является макросом к использованию while().

В этой главе основной акцент сделан не стандартное поведение программы при использовании циклов, а на особенности, существующие в NXC.

Пример 1. while() как функция

В NXC while может быть использован и как стандартный цикл «выполнять пока», внутри которого выполняются различные действия, так и как функция «выполнять пока», которая возвращает управление программе, только тогда когда не выполняется условие, переданное в данную функцию.

```
task main() {
    /*
    Программа ниже может быть описана следующим образом:
    - начать движение прямо
    - двигаться до тех пор пока до препятствия больше 10 см.
    - остановится как только до препятствия осталось меньше (либо равно)
      10 см.

    Следует отметить, что команда "движение вперед" выполняется не цикле, но
    в цикле уже опрашивается сенсор растояния.
    */

    SetSensorLowspeed(S1);
    Wait(100);

    OnFwd(OUT_AB, 50);
    while (SensorUS(S1) > 10);
    Off(OUT_AB);
}
```

Пример 2. while() в одну строчку

Реализация while() в NXC также поддерживает вызов с блоком повторения, записанным одной командой.

```
task main() {
    /*
    Движение будет происходить до тех пор пока расстояние до препятствия
    Больше 10 см. Остановка двигателей (Off) уже на другой строчке - он не
    Выполняется в цикле.
    Поскольку для движения используется RotateMotor(), который отдает
    Управление следующей итерации цикла только после того как повернет
    двигатели на 180 градусов, проверка на расстояние может произойти уже
    когда до препятствия останется уже гораздо меньше расстояния. Т.е. за
    эти 180 градусов, робот может пройти через рубеж в 10 см.
    */

    SetSensorLowspeed(S1);
    Wait(100);

    while (SensorUS(S1) > 10)
        RotateMotor(OUT_AB, 50, 180);
    Off(OUT_AB);
}
```

Пример 3. until()

Как уже было сказано, until() – макрос к *функции* while(). По сути, он позволяет выполнять повторять предыдущее действие до тех пор, пока условие неверно.

```
task main() {
    /*
    Движение назад будет происходить до тех пор, пока расстояние до
    препятствия не станет больше 30 см.
    */

    SetSensorLowspeed(S1);
    Wait(100);

    OnRev(OUT_AB, 50);
    until (SensorUS(S1) > 30);

    Off(OUT_AB);
}
```

Пример 4. Скорость исполнения цикла.

В данном примере не рассматривается никакая особенность циклов. Зато он предназначен для получения ответа на вопрос «сколько итераций цикла исполняется в течение одной секунды»

```
#define OneSecond 1000

task main() {
    /*
    Перед началом цикла засекаем текущее состояние таймера (Start).
    Затем, проверяем разницу между этим значением и новым состоянием таймера.
    Как только разница - 1000 милисикунд - останавливаем цикл.
    Полученные данные выводим на экран.

    Результат: примерно 7640 итераций цикла
    */

    Wait(100);

    unsigned long Start = CurrentTick();
    unsigned long Diff;
    unsigned long i = 0;

    do {
        Diff = CurrentTick() - Start;
        i++;
    } while (Diff < OneSecond)

    //     Координата X
    //     |  Координата Y (задается константой, обозначающей номер строки)
    //     |  |          Значение для вывода на экран
    //     |  |          |
    NumOut(0, LCD_LINE1, Start);
    NumOut(0, LCD_LINE2, i);
    Wait(3000);
}
```

Развороты как траектории движения

Развороты и повороты – необходимость при сложных траекториях движения. Поэтому в этой главе рассматриваются способы, каким программируются эти элементы движения в NXC.

Пример 1. Поворот одним двигателем.

На самом деле в этом примере рассматриваются два способа поворота одним двигателем.

В качестве входных данных для обоих способов, задается необходимость выполнения разворота вправо.

Итак, первый способ.

Поворот одним двигателем вправо. Ось вращения - правое колесо.

```
task main() {
    /*
    Робот крутится вправо до тех пор, пока расстояние до препятствия не
    станет меньше 20 см. Вращение происходит за счет того, что правый мотор
    (OUT_A) не двигается, а левый мотор (OUT_B) двигается вперед. Поэтому
    робот, как бы крутится вокруг правого колеса..
    */

    SetSensorLowspeed(S1);
    Wait(100);

    //Вместо двух моторов, указываем только один.
    OnFwd(OUT_B, 50);
    until (SensorUS(S1) < 20);

    Off(OUT_AB);
}
```

Второй способ –

Поворот вправо "задним ходом". Вращение вокруг левого колеса.

```
task main() {
    /*
    Робот крутится вправо до тех пор, пока расстояние до препятствия не
    станет меньше 20 см. Вращение происходит за счет движения правого мотора
    (OUT_A) назад, в то время как левый мотор (OUT_B) остается неподвижным.
    Поэтому робот, как бы пятится назад вокруг левого колеса.
    */

    SetSensorLowspeed(S1);
    Wait(100);

    //Вместо двух моторов, указываем только один.
    //Мотор двигается назад.
    OnRev(OUT_A, 50);
    until (SensorUS(S1) < 20);

    Off(OUT_AB);
}
```

Пример 2. Поворот двумя двигателями

Основной недостаток поворота одним двигателем - небольшая скорость: усилия одного двигателя направлены и на перемещение тележки и на преодоление силы сопротивления оказываемой неподвижным колесом. Поэтому, популярностью пользуется быстрый поворот вокруг своей оси (точнее ось вращения проходит между колесами).

В данном примере рассматривается поворот робота двумя двигателями влево.

```
task main() {
    /*
    Робот крутится влево до тех пор, пока расстояние до препятствия не станет
    меньше 20 см. Вращение происходит за счет движения обеими моторами –
    правый мотор двигается вперед, левый мотор двигается назад. За такое
    распределение движения между моторами отвечает третий параметр в
    OnFwdSync().
    */

    SetSensorLowspeed(S1);
    Wait(100);

    //         Двигатели
    //            |    мощность
    //            |    |   распределение нагрузки между моторами
    //            |    |   |
    OnFwdSync(OUT_AB, 50, -100);
    until (SensorUS(S1) < 20);

    Off(OUT_AB);
}
```

Пример 3. Распределение мощности при управлении двумя двигателями

В прошлом примере вместе с функцией OnFwdSync() использовался параметр «распределение нагрузки между моторами». Этот параметр имеет смысл только тогда, когда происходит управление двумя двигателями. Он позволяет в одном числе выразить не только, насколько быстро будет крутиться один мотор относительно другого, но также и направление вращения.

```
task main() {

    SetSensorLowspeed(S1);
    Wait(100);

    /*
    Первым действием робот вращается влево, правое колесо крутится медленней
    левого.
    */

    //          Двигатели
    //              |      мощность
    //              |      |   угол поворота оси мотора в градусах
    //              |      |   |    распределение движения между моторами
    //              |      |   |    |  синхронизация моторов
    //              |      |   |    |  |     включать торможение после
    //              |      |   |    |  |     |  окончания движения или
    //              |      |   |    |  |     |  останавливаться своим ходом
    RotateMotorEx(OUT_AB, 50, 360, -60, true, true);

    Wait(1000);

    /*
    Сейчас, робот вращается влево, правый мотор стоит, левый мотор двигается
    назад.
    */

    RotateMotorEx(OUT_AB, 50, -360, -50, true, true);

    Wait(1000);

    /*
    Робот вращается влево двигаясь, правый мотор двигается быстрее левого.
    */

    RotateMotorEx(OUT_AB, 50, 360, -25, true, true);

    Wait(1000);

    /*
    Робот двигается без поворотов, оба колеса двигаются одинаково назад.
    */

    RotateMotorEx(OUT_AB, 50, -360, 0, true, true);
}
```

Пример 4. Альтернативный поворот двумя двигателями

Предыдущий пример показал, что движение при использовании параметра «распределение нагрузки между моторами» по сути – работа двух двигателей с разной скоростью и разным направлением. Следовательно, его можно запрограммировать и без использования этого параметра.

```
task main() {
    /*
    Робот крутится влево до тех пор, пока расстояние до препятствия не станет
    меньше 20 см.
    Вращение вокруг своей оси происходит без использования команд, которые
    принимают парамерт распределения движения между моторами.
    Заместо этого, используется знание того, что команды OnFwd/OnRev отдают
    управление следующим командам сразу после запуска моторов.
    Таким образом, сначала запускается один двигатель, потом сразу же
    запускается второй двигатель. Направление движения каждого двигателя и
    мощность, подаваемая на каждый из них, задают направление и форму
    поворота.
    */

    SetSensorLowspeed(S1);
    Wait(100);

    //Правый мотор двигается вперед
    OnFwd(OUT_A, 50);
    //Левый мотор двигается назад с такой же мощностью как и правый.
    OnRev(OUT_B, 50);
    until (SensorUS(S1) < 20);

    Off(OUT_AB);
}
```

Датчики и сенсоры

Программирование робота, в большинстве случаев, - изменение поведения робота как реакция на информацию, пришедшую к управляющему устройству с датчиков и сенсоров. Эта глава рассматривает примеры опроса сенсоров, а также некоторые особенности, которые необходимо учитывать при работе с ними.

Пример 1. Датчик расстояния.

Для датчика расстояния (ультразвуковой сенсор) используются специальные функции для его инициализации и опроса.

```
task main() {
    /*
    Робот ждет пока перед ним не окажется предмет (ближе чем 20 см.), после
    чего отодвигается от предмета на 50 см.
    */

    //Говорим, что медленный (ультразвуковой) сенсор установлен в первый
    //порт.
    SetSensorLowspeed(S1);
    Wait(100);

    while (true) {
        //Опрашиваем порт первый, как ультразвуковой датчик
        //Ждем, пока расстояние до предмета не станет меньше 20 см.
        until (SensorUS(S1) < 20);

        //Отодвигаемся назад до тех пор, пока расстояние до предмета не
        //станет больше 50 см.
        OnRev(OUT_AB, 50);
        until (SensorUS(S1) > 50);
        // Явно останавливаем двигатели
        Off(OUT_AB);
    }
}
```

Пример 2. Измерение отраженного света.

Работа датчика в режиме измерения отраженного света (пространство перед датчиком дополнительно освещается светодиодом) позволяет повысить его чувствительность. Без использования этого режима, решение такой задачи как распознавание цвета или градаций серого была бы довольно трудно выполнима – обычно окружающий свет не имеет достаточной интенсивности, поэтому соседние градации цветов сливались бы один цвет при измерении. К тому же, измерения при неравномерном освещении давали бы разный результат для одного и того же предмета. Поэтому использование дополнительной подсветки – наиболее частый вариант при управлении роботом с помощью датчика освещенности.

```
task main() {
    /*
    Робот двигается пока поверхность под световым сенсором не потемнеет.
    */

    // Говорим, что световой сенсор установлен в третий порт.
    SetSensorLight(S3);
    Wait(500);

    //Двигаемся вперед, пока значение сенсора в третьем порту не станет
    //меньше 55 процентов.
    OnFwd(OUT_AB, 50);
    until(Sensor(S3) < 55);

    Off(OUT_AB);
}
```

Пример 3. Измерение окружающего света.

В некоторых задачах, дополнительная подсветка только мешает. Например, определить какое место в комнате самое темное. Для этого нужно измерять именно окружающее освещение.

```
task main() {
    /*
    Робот отображает на экране текущую освещенность.
    */

    //Говорим, что световой сенсор установлен в третий порт
    //лампа подсветки - неактивна
    SetSensorType(S3, SENSOR_TYPE_LIGHT_INACTIVE);
    //Сенсор, устанавливаемый через SetSensorType по-умолчанию показывает
    //ненормализованные данные (не путать с RAW). Переводим, его в режим для
    //считывания процентов.
    SetSensorMode(S3, SENSOR_MODE_PERCENT);
    Wait(500);

    while(true) {
       //Выводим на эркан текущие показатели сенсора, перед выводом - очищаем
       //эркан (4-ый параметр - true)
       NumOut(0, LCD_LINE1, Sensor(S3), true);

       Wait(50);
    }
}
```

Пример 4. Использование необработанных данных

По умолчанию, датчики выдают измерения в процентах. В то время как NXC предоставляет возможность использовать необработанные (RAW) данные. Преимущество – очевидно – поскольку необработанные данные могут быть в диапазоне от 0 до 1023, измерения будут в 10 раз чувствительнее.

```
task main() {
    /*
    Робот отображает на экране показания отраженного света.

    Темная поверхность:
        1-ая строка (проценты): значения маленькие (<50%)
        2-ая строка (raw): значения большие (>600)

    Светлая поверхность:
        1-ая строка (проценты): значения маленькие (>50%)
        2-ая строка (raw): значения большие (<400)
    */

    //Говорим, что световой сенсор установлен в третий порт
    SetSensorLight(S3);
    Wait(500);

    while(true) {
       //Выводим на эркан в первой строке текущие показатели сенсора в
       //процентах, перед выводом - очищаем эркан (4-ый параметр - true)
       NumOut(0, LCD_LINE1, Sensor(S3), true);
       //Выводим на эркан во второй строке текущие необработанные показатели
       //сенсора, экран не очищается (4-ый параметр не указывается - берется
       //значение по-умолчанию false)
       NumOut(0, LCD_LINE2, SensorRaw(S3));

       Чтобы избежать мерцания цифр на экране, делаем небольшую паузу.
       Wait(50);
    }
}
```

Пример 5. Датчик вращения двигателя

Поскольку датчик вращения двигателя скрыт от глаз конструктора, часто о его существовании забывают. Тем не менее, в некоторых задачах, с его помощью можно эмулировать еще один датчик столкновения.

```
task main() {
    /*
    Робот начинает двигаться, затем, в цикле дважды опрашивается датчик
    поворота двигателя с небольшой паузой.
    Поскольку пауза происходит параллельно движению двигателя (использование
    функции OnFwd), оценивается поворот двигателя, произошедший во время этой
    паузы.
    Если поворот менше ожидаемого, то скорее всего робот наткнулся на
    препятствие - препятствие мешает провернуться двигателю. После
    обнаружения такой ситуации - движение прекращается.
    */

    int Start;
    int Diff;
    int Desired;
    // Переменная используется, чтобы при первой итерации цикла определить
    //поворот двигателя во время паузы и взять этот поворот за эталон
    int iter = 0;

    OnFwd(OUT_AB, 40);

    do {
        //Считываем показание датчика поворота перед паузой
        Start = MotorRotationCount(OUT_A);
        Wait(100);
        //Считываем показание датчика после паузы и определяем угол поворота
        Diff = MotorRotationCount(OUT_A) - Start;
        //Если это первая итерация цикла - принять угол поворота за эталон
        if (iter == 0) {
            Desired = Diff * 8 / 10;
            //Указать, что больше в этот 'if' заходить не надо
            iter = 1;
        }
    } while (Diff > Desired) // Если угол поворота двигателя меньше эталона

    Off(OUT_AB);
}
```

Пример 6. Скорость опроса датчика расстояния

При работе с датчиком расстояния необходимо помнить, что он является низкоскоростным устройством, т.е. опрос состояния этого датчика занимает определенное время. Это в первую очередь связано с физической природой процесса измерения расстояния: для того чтобы звуковая волна прошла путь до препятствия и вернулась обратно в сенсор, определенно необходимо какое-то время.

```
#define OneSecond 1000

task main() {
    /*
    В каждой итерации цикла опрашивается датчик расстояния. Количество
    итераций определяется выяснением, сколько миллисекунд прошло с начала
    работы программы. Как только количество миллисекунд становится больше
    1000 - цикл прерывается.
    Поскольку ультразвуковой сенсор является низкоскоростным устройством, его
    опрос вызывает задержку.

    Результат показывает, что ультразвуковой сенсор может быть опрошен 34
    раза за одну секунду.
    Т.е. задержка опроса сенсора составляет - 1000/34 = 30 миллисекунд.
    */

    //Указываем, что низкоскоростной сенсор (ультразвуковой) установлен в
    //первый порт
    SetSensorLowspeed(S1);
    Wait(100);

    //Считываем начальное показание внутреннего таймера
    unsigned long Start = CurrentTick();
    unsigned long Diff;
    unsigned long i = 0, value;

    do {
        //Опрашиваем сенсор, его показания не важны для данной программы,
        //важен факт его вызова
        value = SensorUS(S1);
        //Вычисляем разницу между начальным показанием таймера и текущим
        Diff = CurrentTick() - Start;
        i++;
    } while (Diff < OneSecond) //Повторять, пока разница времени меньше секунды

    NumOut(0, LCD_LINE1, i);
    Wait(3000);
}
```