Ярмолинский Леонид Маркович

УРОК 5. ПОИСК ВЫХОДА ИЗ ЛАБИРИНТА

ВВЕДЕНИЕ

На этом занятии мы разберем задачу поиска выхода из лабиринта, а также познакомимся с реализацией модели программирования «машина состояний» (state machine) в Labview.

Для этого занятия нам понадобится робот на гусеницах с двумя датчиками расстояния (рис. 1) и кабель для записи программы на NXT.

Начнем с написания программы для робота с одним датчиком расстояния, который смотрит на стенку справа, и заставим его ехать вдоль стенки на определенном расстоянии.

ДВИЖЕНИЕ ВДОЛЬ СТЕНКИ

Данная задача решается так же, как задача позиционирования мотора, которую мы разбирали ранее.

Для начала создаем шаблон программы (рис. 2).

В самом начале программы в кадре инициализации размещаем функцию **Specify NXT**  , заводим выходной кластер через туннель в цикл нашей программы и на этом с кадром инициализации заканчиваем.

Также сразу мы добавляем проверку нажатия кнопки «Enter» для завершения основного цикла и команду выключения моторов в кадр завершения работы.

Добавляем функцию чтения датчика **NXT I/O → Read Sensor**. В меню выбираем **Read Ultrasonic** 

. Добавляем числовую константу **NXT Programming → Numeric → Numeric Constant** 

, через которую мы сможем задать требуемое расстояние робота до стенки (изначально предлагается указать значение 20 см, но данный параметр нужно будет отрегулировать в зависимости от конструкции робота). Далее находим ошибку расстояния до стенки, то есть разность значения с датчика и значения, записанного в константе (рис. 3).

Рис. 1

Рис. 2

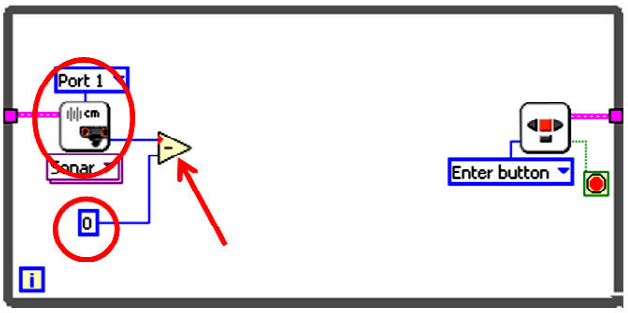


Рис. 3

Умножаем ошибку на пропорциональный коэффициент усиления *Kp*, равный 8, и результат подаем на вход функции **Steering On**  , **NXT Functions → NXT I/O → Complete → Motors → Steering On**. Создаем на входах выбора моторов и задания мощности константы. Также сразу добавим в цикл временную задержку в 30 миллисекунд для облегчения работы программы (рис. 4).

Сохраняем программу.

Записываем ее на NXT и проверяем работу. Для проверки работы программы нам понадобится стенка следующего вида (см. рис. 5).

Внутренний многоугольник изображает стенку, внешняя огибающая линия предполагаемую траекторию движения робота.

Для оптимальной настройки движения робота вдоль стенки необходимо настроить 4 параметра:

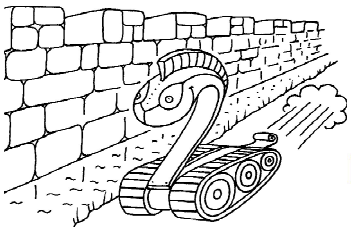
– пропорциональный коэффициент регулятора *Kp* – в нашей программе равный 8;

– мощность моторов при прямолинейном движении робота – в нашей программе 75;

– заданное расстояние, на котором робот должен держаться от стенки, – в нашей программе указано 20 см;

– вынос датчика расстояния от оси вращения робота.

ВНИМАНИЕ! Если вы подобрали вышеуказанные значения для гладкого перемещения вдоль стенки, то нужно понимать, что если вы будете использовать робота с другой конструкцией, то значения ***вышеуказанных параметров****, возможно, придется подбирать заново.*



...робота с одним датчиком ра который смотрит на стенку заставим... ехать вдоль стен на определенном расстоянии.

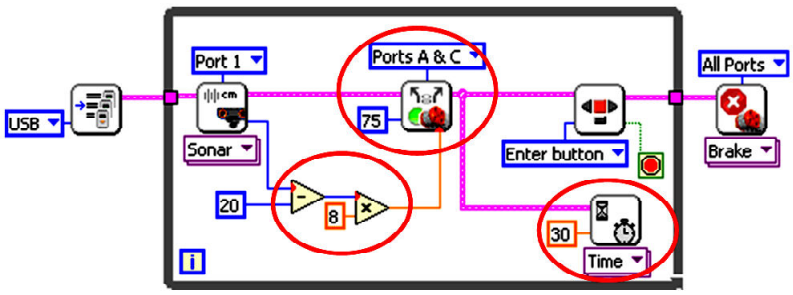


Рис. 4

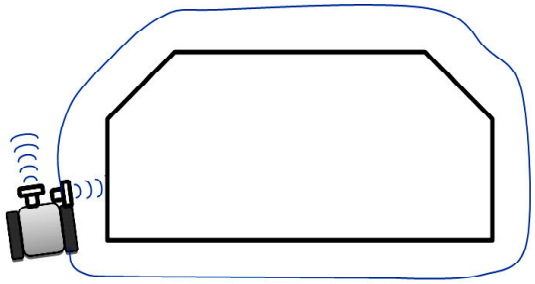


Рис. 5

Рис. 6

ОБЪЕЗД ВЫСТУПА СТЕНКИ С ПОВОРОТОМ НА 180 ГРАДУСОВ

Для проезда лабиринта роботу необходимо адекватно реагировать на перегородки следующего вида (рис. 7).

При пропаже стенки робот, работающий по написанной выше программе, скорее всего, будет вести себя так (рис. 8).

Здесь штриховая линия – желаемая траектория, сплошная линия – фактическая траектория. Кончено, при совпадении хорошей настройки программы движения вдоль стенки и удачной конструкции робота и расположения датчика, робот может правильно преодолеть такое препятствие, но на практике легче дописать еще небольшой алгоритм контроля пропажи стенки.

После опроса датчика расстояния добавляем функцию сравнение **Less?** (Меньше?) **Functions → comparison → Less?**, и сравниваем показания датчика со значением 40 см (рис. 9).

Добавляем структуру **Case**, отводя ей функции вычисления и выдачи задания мощности на моторы. На вход структуры **Case** подадим результат сравнения расстояния до стенки с порогом 40 см (рис. 10).

Выбираем окно структуры **Case** для значения **False**. Добавляем в нем функцию

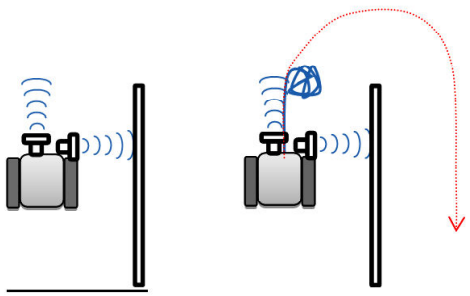


Рис. 7

Рис. 8

Steering on  , **NXT Functions → NXT I/O → Complete → Motors → Steering On** и создаем на всех входах функции константы (рис. 11).

У нас получается, что при показаниях датчика расстояния меньше 40 см робот старается удерживать дистанцию до стенки 20 см при помощи пропорционального регулятора с коэффициентом усиления $Kp = 8$, а при скачке показаний датчика больше 40 см робот будет двигаться по дуге. Таким образом, в случае, когда стенка пропадает из поля

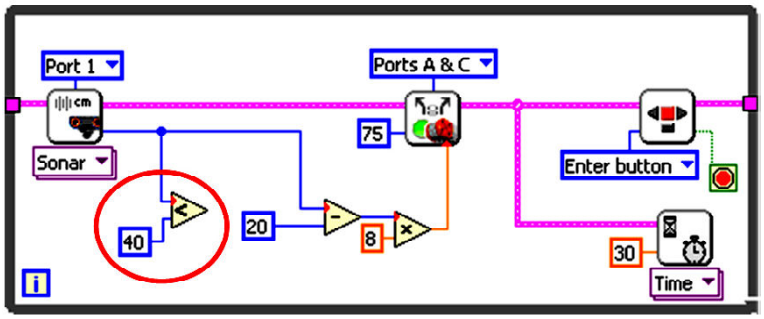


Рис. 9

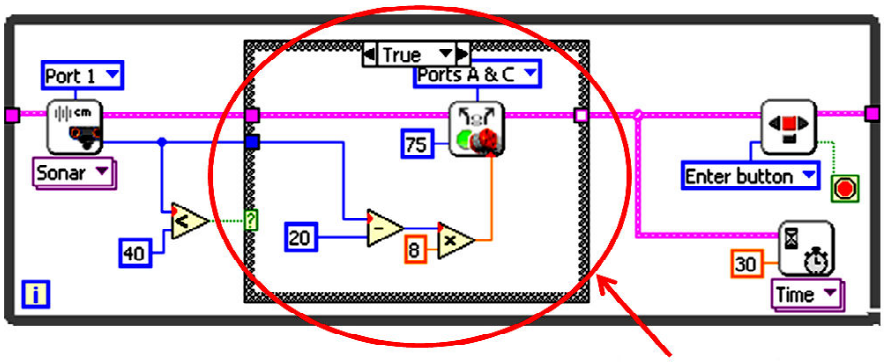


Рис. 10

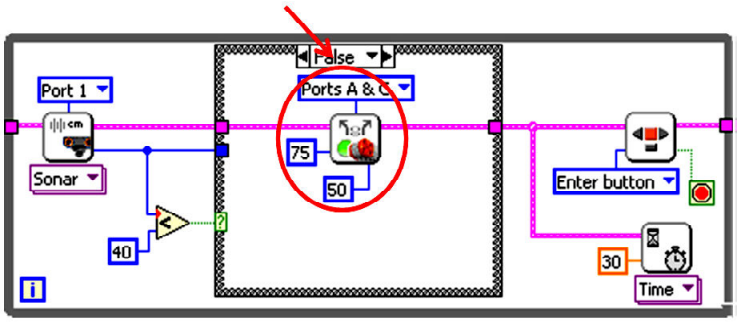


Рис. 11

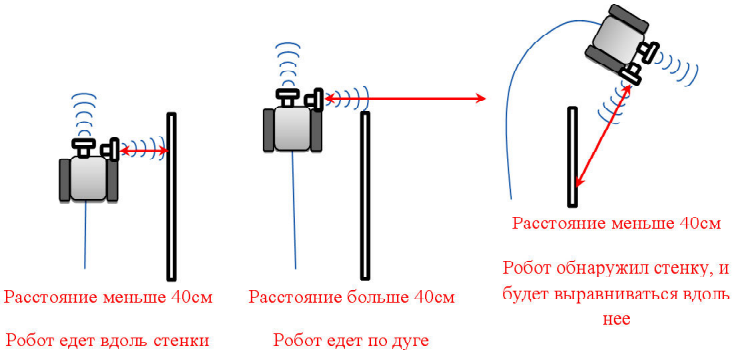


Рис. 12

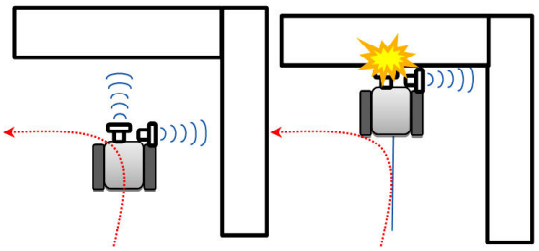


Рис. 13

зрения датчика расстояния при объезде выступа робот не будет крутиться на месте, а плавно объедет обнаруженный выступ стенки (рис. 12).

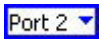
Нам осталось настроить радиус дуги поворота при помощи параметра **Steering**, функции **Steering on**, в данном примере программы он имеет значение 50.

ПОВОРОТ СТЕНКИ ВЛЕВО НА 90 ГРАДУСОВ

Сейчас наш робот ориентируется по стенке, находящейся справа. При повороте стенки влево на 90 градусов робот, работающий по написанной выше программе, просто «уткнется носом» в повернувшую стенку и будет безуспешно пытаться ее переехать (рис. 13).

Для определения стенки спереди робота у нас есть еще один датчик расстояния. На появление стенки спереди наш робот должен реагировать поворотом на 90 градусов влево.

Добавляем в нашу программу после структуры **Case** опрос датчика расстояния **NXT I/O → Read Sensor**, в меню выбираем **Read Ultrasonic** 

. Создаем константу с указанием порта, к которому подключен датчик, и указываем в ней **Port 2** 

(рис. 14).

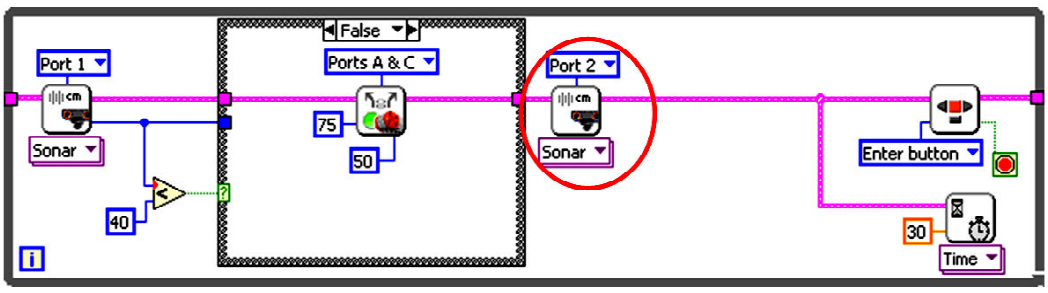


Рис. 14

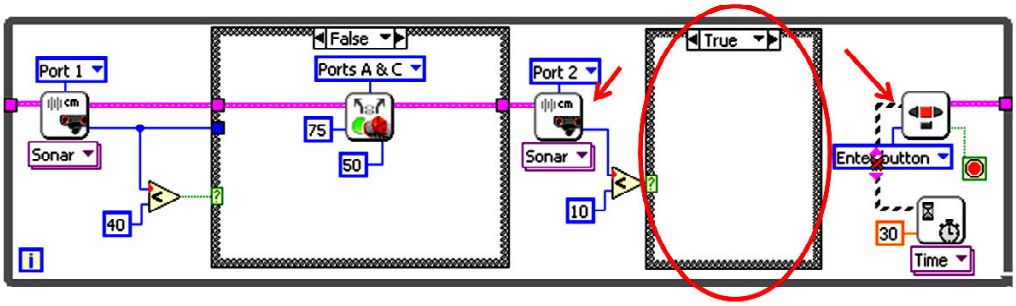


Рис. 15

Так же, как и в случае определения пропажи стенки справа, добавляем алгоритм при помощи функции сравнения **Less?** (Меньше?) **Functions → comparison → Less?** и структуры **Case**. В качестве порога, с которым сравнивается показание датчика, берем 10 см. Так же удаляем связь по основному кластеру между функциями **Read Ultrasonic** и **Read NXT Buttons** (рис. 15).

Вкладка **True** нового ветвления (новой структуры case) соответствует обнаружению стенки. Когда стенка обнаружена, нужно выдать команды поворота налево. Добавляем внутрь функцию **Steering on** с параметрами мощности (power) 100 и поворот (steering) –100. Для указания длительности поворота воспользуемся функцией **Wait for rotation** **NXT I/O → Wait For** в выпадающем меню **Wait for rotation**. Входным параметром этой функции является значение угла поворота выходной оси мотора. В зависимости от конструкции робота необходимо подобрать значение этого параметра, чтобы робот поворачивал на 90 градусов (рис. 16). Соединяем входной и выходной туннели во вкладе **False** (рис. 17).

Отлаживаем работу поворота налево, подбирая значения входного параметра функции **Wait for rotation**.

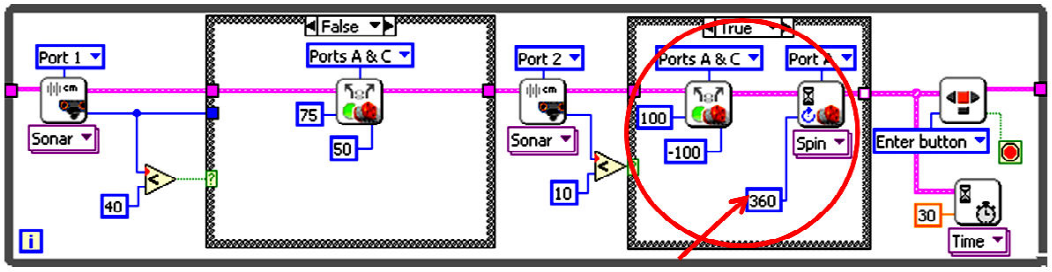


Рис. 16

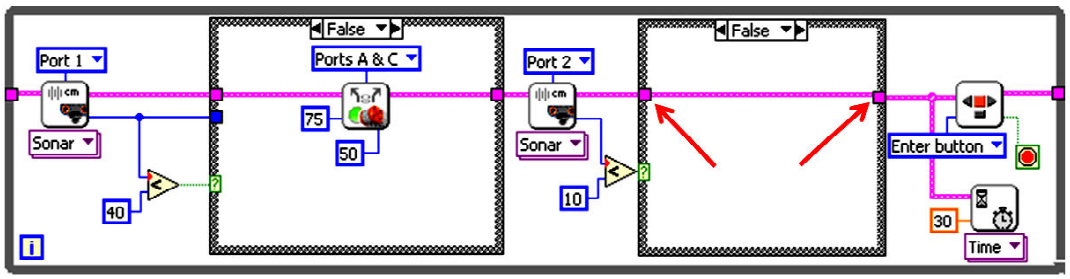


Рис. 17

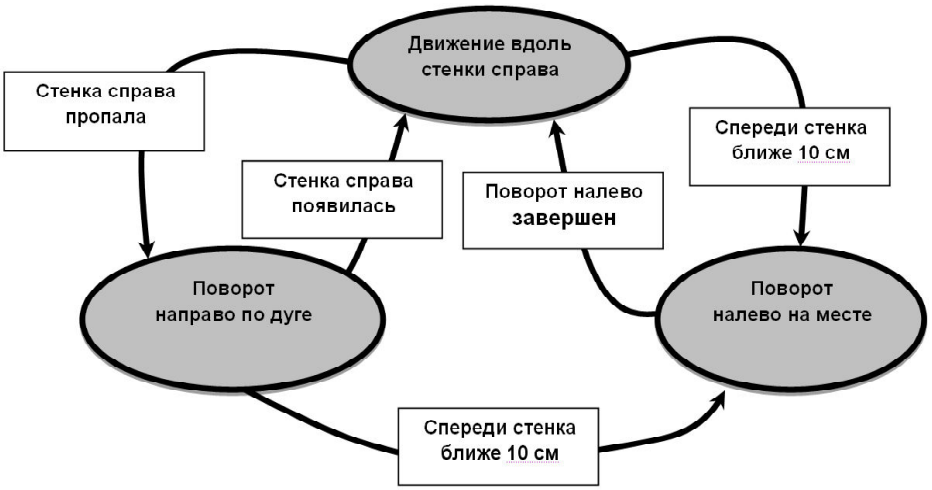


Рис. 18

ВНИМАНИЕ! Функция ***Wait for rotation*** *учитывает направление вращения выходной оси мотора. Если знак входного параметра и направления вращения мотора не совпадают, то в нашем случае поворот налево будет бесконечным!*

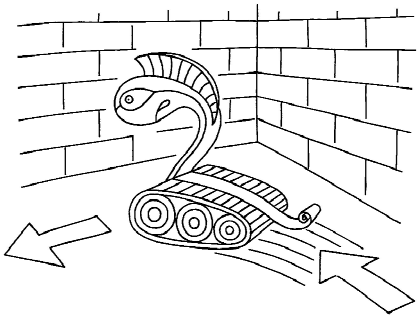
Программа готова! Сохраняем и проверяем работу программы.

МАШИНА СОСТОЯНИЙ

Используем написанную программу для знакомства с моделью программирования в виде машины состояний. В упрощенной форме, согласно этой модели, программа разбивается на состояния и переходы между этими состояниями по каким-либо условиям.

Нашу программу можно разбить на три состояния, и описать диаграммой состояний (рис. 18).

Для реализации этой диаграммы создаем новую программу и новый шаблон (рис. 19).



Когда стенка обнаружена нужно выдать команды поворота налево.

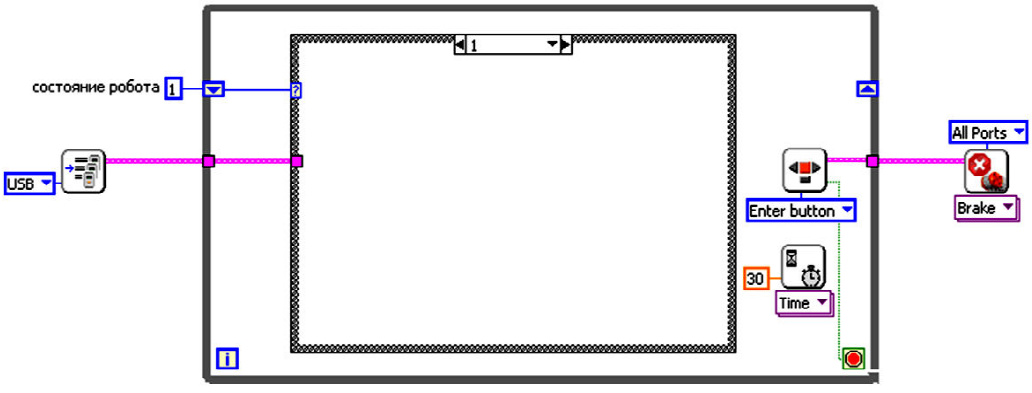


Рис. 19

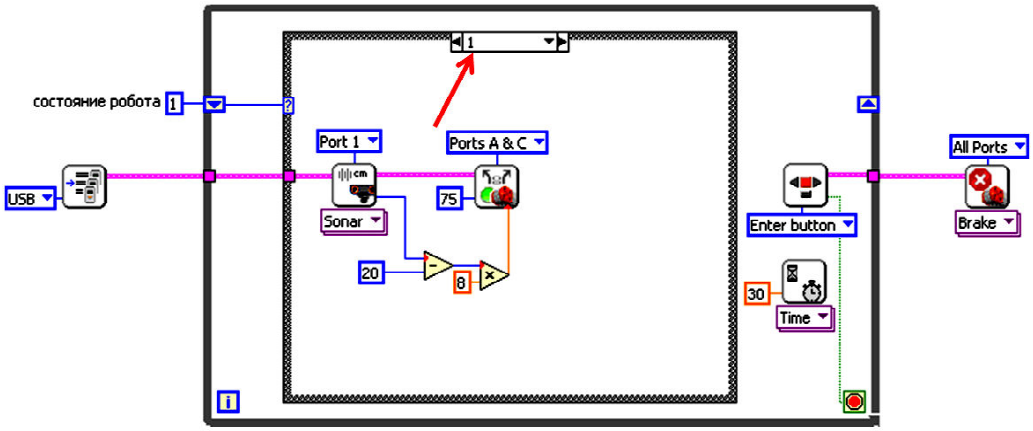


Рис. 20

У цикла появился сдвиговый регистр (знакомство с регистрами было в предыдущей статье), в котором у нас будет храниться номер текущего состояния робота. Внутри цикла создана структура **Case**. Каждая ветка этой структуры является определенным состоянием робота. В нашем случае есть три состояния, присвоим им следующие номера:

1. Движение вдоль стенки.
2. Поворот направо по дуге.
3. Поворот налево на месте.

Проверяем, что открыта ветка для состояния #1.

Из предыдущей программы копируем в ветку #1 часть программы, отвечающую за движение робота вдоль стенки (рис. 20).

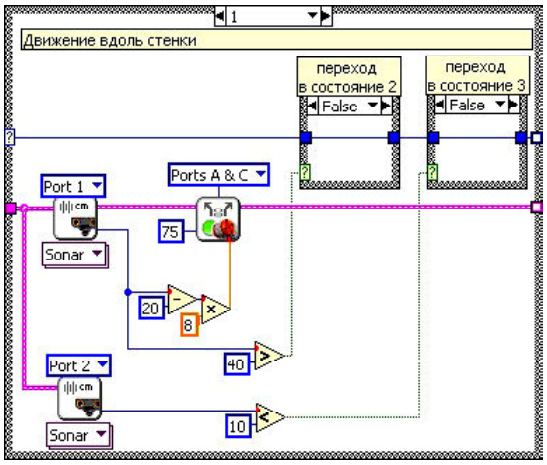


Рис. 21

Само состояние готово, теперь нужно указать условия перехода в другие состояния. Всего их два: условие для перехода в состояние #2 и в состояние #3. Добавляем две структуры ветвления внутри состояния #1. Через ветку **False** в обоих ветвлениях проводим текущее состояние робота (рис. 21).

В ветках **True** создаем по константе с номером состояния, в которое производится переход по соответствующему условию (рис. 22).

Нажимаем правой кнопкой мышки на рамку главного ветвления и выбираем **Add case after**. Появится новая ветка для состояния #2

Последовательность действий, описанную для состояния #1, повторяем для состояний #2 и #3. В результате получаем:

- Состояние №2 – поворот направо по дуге (рис. 23).
- Состояние №3 – поворот налево на месте (рис. 24).

Переходим в ветку для значения «0» и соединяем входы текущего состояния и ос-

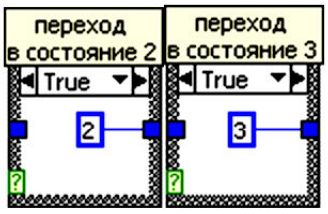


Рис. 22

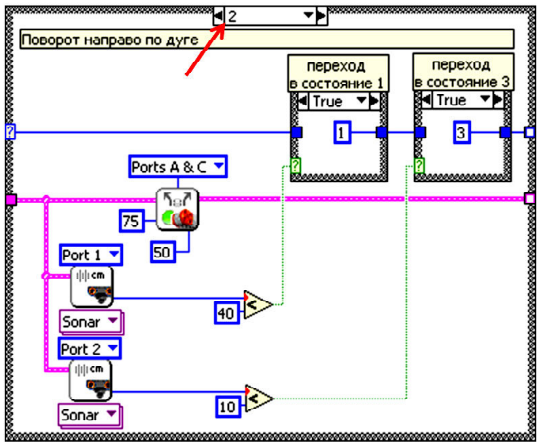


Рис. 23

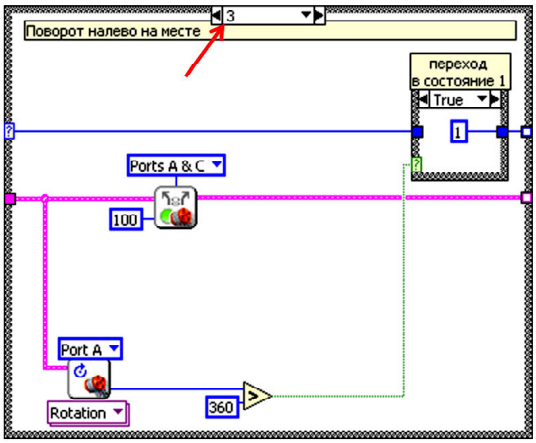


Рис. 24

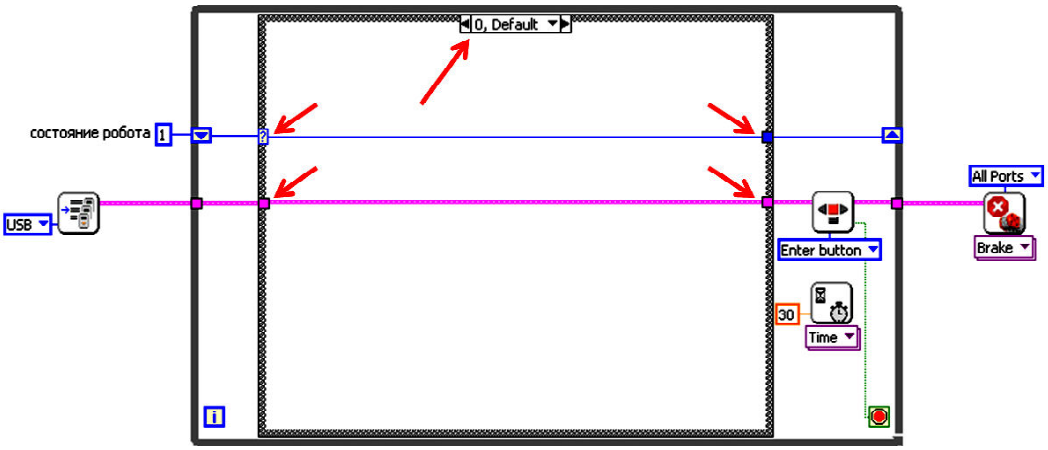


Рис. 25

новного кластера (рис. 25).

Программа готова, сохраняем и проверяем работу.

Одним из преимуществ данного метода программирования является возможность написания и отладки каждого состояния в отдельности с последующим соединением уже заведомо работающих частей алгоритма.

На этом наше занятие заканчивается, успехов!

Ярмолинский Леонид Маркович,
ведущий инженер-программист
ООО «ПромАвтоматика»,
педагог дополнительного образования
(кружок робототехники)
ГБОУ СОШ № 255.