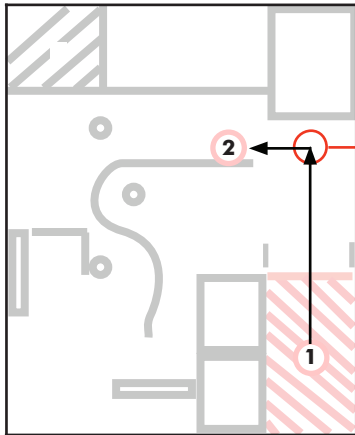


## Sensing

### Wall Detection **Touch vs. Timing**

We've learned a lot about how to make the robot move, including how to make it go forward and backward for specific lengths of time, how to adjust its speed, and how to make it go as straight as possible. But motor control alone won't be enough to let the robot to stay on the obstacle course below, because we don't know exactly where the robot will start.



#### **Turn left**

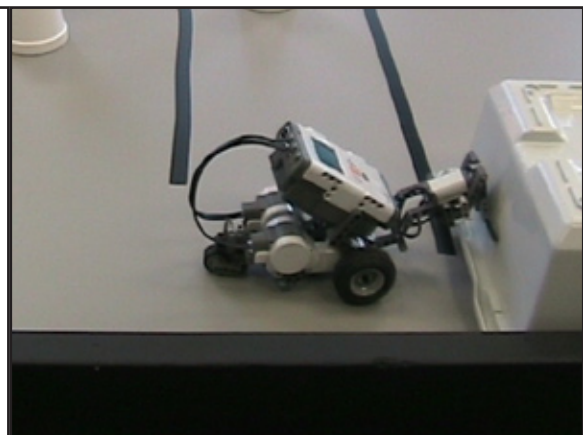
To get from position 1 to position 2, the robot has to turn left just in front of the wall, at the red circle. In this challenge, we don't know exactly where the robot will start in the red hatched area. It is therefore impossible to make the robot turn in the correct place using motor control only.

We know we want the robot to make a left turn just in front of the obstacle course wall. What we need is a way for the robot to find out where that wall is, and adjust its course accordingly. In this lesson, we'll attach a Touch Sensor to the robot and use it to detect the wall. By using feedback from the sensor, we can make the robot turn in the correct place no matter how far away from the wall it started.



#### **Touch Sensor**

The Touch Sensor, above, can enable the robot to detect physical contact with objects like walls.



#### **Touch Sensor detecting a wall**

A robot uses sensors to gather information from the environment and uses the information to plan movement.

## Sensing

### Wall Detection **Touch vs. Timing** (cont.)

*In this lesson, you will learn to use feedback from a Touch Sensor to let the robot detect a solid object and adjust its course accordingly.*

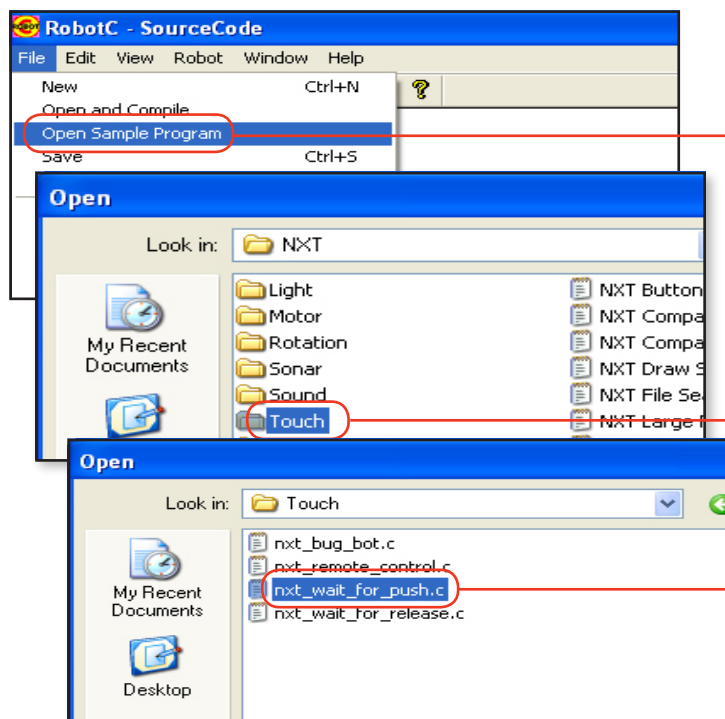
1. Add the Touch Sensor attachment to the robot (if it doesn't have one already). Connect the sensor to Port 1 on the NXT brick. The bumper assembly helps the sensor to detect collisions that are not centered directly on the sensor's orange contact surface.



#### 1. Build the Touch Sensor attachment

Building instructions are available through the main lesson menu. Connect the Touch Sensor to port 1.

2. Load the program "nxt\_wait\_for\_push.c" on the NXT.



#### 2a. Open sample program

Click File > Open Sample Program.

#### 2b. Open Touch folder

Double-click the "Touch" folder to open it.

#### 2c. Open *nxt\_wait\_for\_push*

Double-click "*nxt\_wait\_for\_push.c*" to open the program.

## Sensing

### Wall Detection **Touch vs. Timing** (cont.)

#### Checkpoint

The program should look like the one below.

```

c
//p
//
// *****
//                               Wait for Push
//                               RobotC on NXT
//

```

3. Note that the program has 3 major parts. (Lines 1-35 have been omitted, since they contain only comments that do not affect how the program works.)

Auto `const tSensors touchSensor = (tSensors) S1;`

#### Touch Sensor setup

At the top of the program is a special line that tells ROBOTC to look for a Touch Sensor on Port 1, and to call it "touchSensor".

```

36 task main()
37 {
38     while(SensorValue(touchSensor) == 0)
39     {
40         motor[motorA] = 100;
41         motor[motorB] = 100;
42     }
43     motor[motorA] = -75;
44     motor[motorB] = -75;
45
46     wait1Msec(1000);
47 }
48
49

```

#### While() loop

Next, we have the while() loop. It's called a "while" loop because it will do something *while* certain conditions continue.

#### Movement commands

Finally, we have two sets of movement commands: one *inside* the while() loop, and one right *after* the while() loop. The positioning of these commands inside and outside of the loop is important, but otherwise, these are the same commands you have already used to move the robot in previous programs.

#### Checkpoint

You will learn more about the Sensor Setup and while() loop parts of the program later in this lesson. For now, however, **look carefully at the motor commands**. Which motor ports do they address? What ports are your motors plugged into? Do they match?

The sample program assumes your motors would be on ports A and B, but your robot's design has them on C and B! The program will not work without modifications. Software (programs) and hardware (like the physical robot) are dependent on each other to produce correct behaviors.

## Sensing

### Wall Detection **Touch vs. Timing** (cont.)

4. Modify the motor[] commands to send power to the correct motors by changing all the motorA references to motorC (motorB is the same in both).

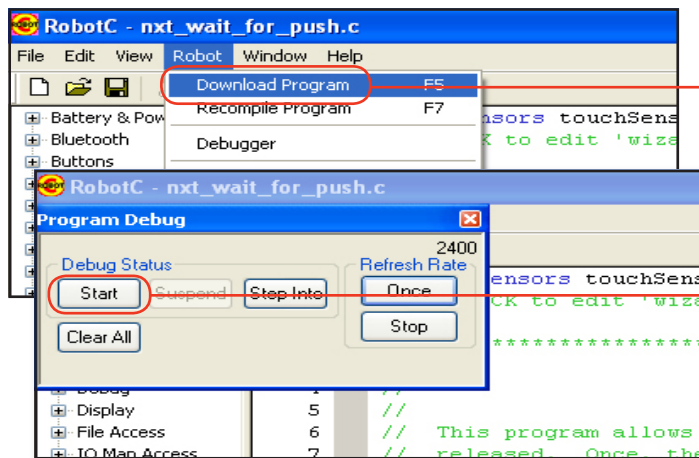
```
Auto const tSensors touchSensor = (tSensors) S1;
```

```
36 task main()
37 {
38     while(SensorValue(touchSensor) == 0)
39     {
40         motor[motorC] = 100;
41         motor[motorB] = 100;
42     }
43
44     motor[motorC] = -75;
45     motor[motorB] = -75;
46
47     wait1Msec(1000);
48 }
49 }
```

**4. Modify this code**

Change the motorA references to instead use motorC, where your left motor is actually attached.

5. Download and run the program.



**5a. Download the program**

Click Robot > Download Program.

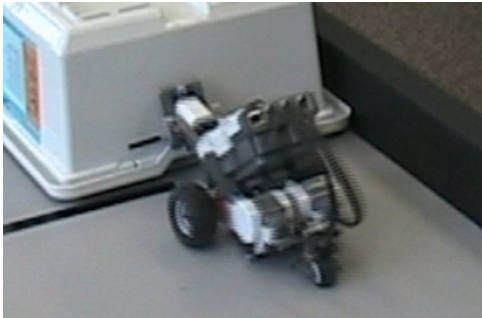
**5b. Run the program**

Click "Start" on the onscreen Program Debug window.

## Sensing

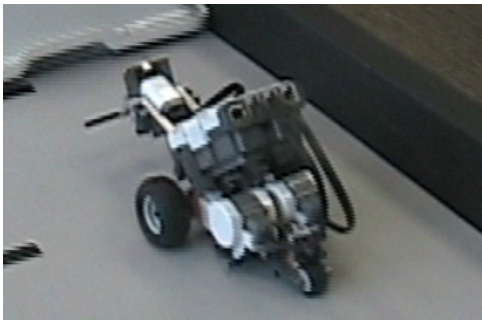
### Wall Detection **Touch vs. Timing** (cont.)

6. Run the program on the Obstacle Course board. Observe the sample program's behaviors.



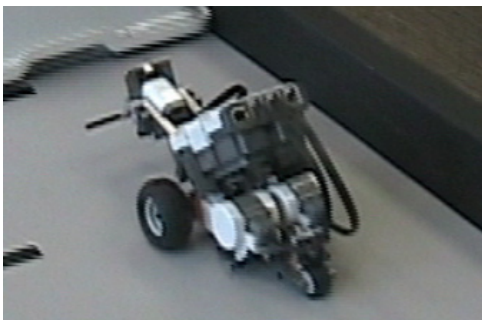
**6a. Forward until touch**

The robot runs forward as long as the touch sensor is not pressed in.



**6b. React to touch**

When the touch sensor is pressed, the robot will back up for one second, then stop.



**6c. End**

The program ends after one touch-and-reverse cycle.

---

#### End of Section

We've taken a crucial step forward in solving the problem of getting the robot to adjust its course when it touches a wall by adding a Touch Sensor attachment, downloading a program, and demonstrating that the robot will reverse its direction when it reaches a solid object. The next step is to understand the program, so that you can write one like it yourself.

## Sensing

# Wall Detection Configuring Sensors

Now that we've seen the wall detection program work, we're going to take it apart piece by piece to understand how it works. In this lesson we'll examine the first section of the program, where we set up the sensors. This configuration process tells the robot which sensors are present, and which ports they're connected to.

In ROBOTC, sensor configuration is done through the Motors and Sensors Setup dialog, which we'll go through in this lesson. You don't have to, and shouldn't, type any code inside the sensor configuration section at all, unless you're an experienced programmer.

Auto `const tSensors touchSensor = (tSensors) S1;`

```

36 task main()
37 {
38     while(SensorValue(touchSensor) == 0)
39     {
40         motor[motorC] = 100;
41         motor[motorB] = 100;
42     }
43
44     motor[motorC] = -75;
45     motor[motorB] = -75;
46
47     wait1Msec(1000);
48 }
49 }
```

### Touch Sensor set up

At the top of the program is a special line that tells ROBOTC to look for a Touch Sensor on Port 1, and to call it "touchSensor". Don't type in this area unless you know what you're doing!

## Sensing

### Wall Detection **Configuring Sensors** (cont.)

*In this lesson, you will learn how to use the Motors and Sensors Setup dialog to configure the Touch Sensor.*

1. Begin by saving the program under a new name. You can't save changes directly to the sample programs, and you want to have a copy of the program for yourself anyway.

The screenshot shows the RobotC IDE interface. The 'File' menu is open, and 'Save As...' is selected. A 'Save As' dialog box is open, showing the 'touch\_sensor' folder selected in the 'Save in:' field. The 'File name:' field contains 'wall\_touch.c'. The 'Save' button is highlighted. The background code shows a comment: '1. The touch sensor should be mounted on the'.

**1a. Save program As...**  
Select File > Save As... to save your program under a new name.

**1b. Browse to an appropriate folder**  
Browse to or create an appropriately named folder within your program folder to save your program.

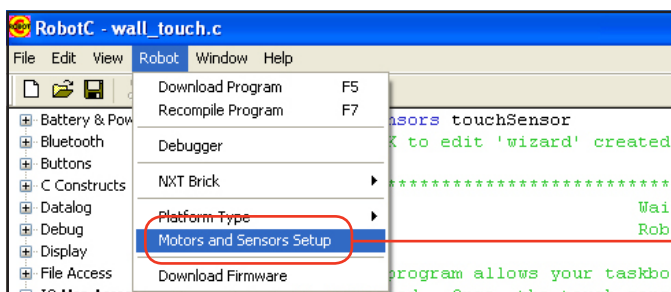
**1c. Rename program**  
Give this program the new name "wall\_touch".

**1d. Save**  
Click Save.

## Sensing

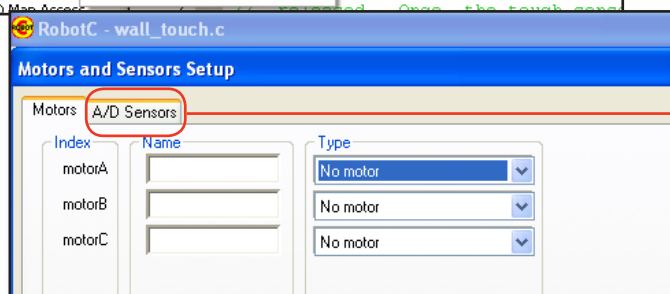
### Wall Detection **Configuring Sensors** (cont.)

2. Open the Motors and Sensors Setup menu, and select the A/D Sensors tab.



**2a. Open “Motors and Sensors Setup”**

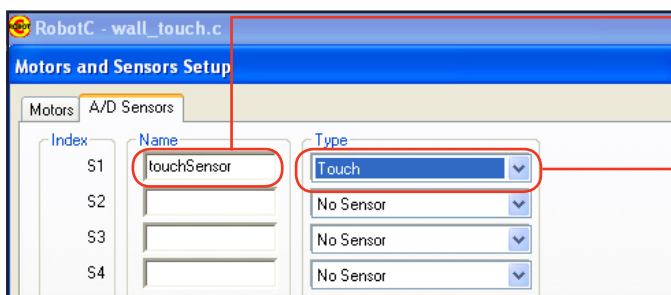
Select Robot > Motors and Sensors Setup to open the Motors and Sensors Setup menu.



**2b. Select the A/D Sensors tab**

Click the “A/D Sensors” tab” on the Motors and Sensors Setup menu.

3. Note the Motors and Sensors Setup menu configuration. To the right of S1 are boxes indicating the name and type of sensor attached to sensor port 1.



**3a. Sensor “Name”**

Assigns the name “touchSensor” to the sensor on port 1. “touchSensor” is a name chosen for convenience, following certain rules (see below).

**3b. Sensor “Type”**

Identifies the sensor attached to sensor port 1 as a Touch Sensor.

### Naming Things in ROBOTC

Here are some basic rules for giving names to things (such as Sensors) in ROBOTC:

- Words that are already part of the ROBOTC language (like “while” or “motor”) cannot be used as names
- Names may not contain spaces
- Names may not contain punctuation
- Names may not START with a number, but may contain them anywhere else
- CaPiTaLiZaTiOn maTTeRs



## Sensing

### Wall Detection **Configuring Sensors** (cont.)

4. Let's try changing some settings to see what happens. Move all the Touch Sensor entries in the menu from S1 to S2.

The first screenshot shows the 'Motors and Sensors Setup' dialog box with the 'A/D Sensors' tab selected. S1 has an empty 'Name' box and 'No Sensor' selected in the 'Type' dropdown. S2 has an empty 'Name' box and 'No Sensor' selected in the 'Type' dropdown. A red circle highlights the 'Name' box for S1.

**4a. Delete "touchSensor" from S1**  
Delete the name "touchSensor" from the S1 Name box. The Type box for S1 will change to read No Sensor after your cursor leaves the Name area.

The second screenshot shows the same dialog box. S1's 'Name' box is empty and S1's 'Type' dropdown now shows 'No Sensor'. S2's 'Name' box contains 'touchSensor' and S2's 'Type' dropdown still shows 'No Sensor'. A red circle highlights the 'Name' box for S2.

**4b. Enter "touchSensor" in S2**  
Type the name "touchSensor" in the S2 Name box.

The third screenshot shows the same dialog box. S1's 'Name' box is empty and S1's 'Type' dropdown shows 'No Sensor'. S2's 'Name' box contains 'touchSensor' and S2's 'Type' dropdown now shows 'Touch'. A red circle highlights the 'Type' dropdown for S2.

**4c. Change S2 Type to Touch**  
Select Touch from the S2 Type dropdown menu.

The bottom of the dialog box shows four buttons: 'OK', 'Cancel', 'Apply', and 'Help'. A red circle highlights the 'OK' button.

**4d. Click OK**  
Click OK to save your sensor configuration changes.

### Checkpoint

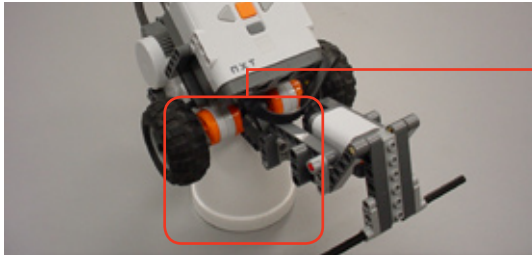
The first line of the program should now look like this. Make sure that the first line of the program contains "S2" and not "S1". The sensor on S2 is named touchSensor and set to work as a Touch Sensor.

```
Auto const tSensors touchSensor = (tSensors) S2;
```

## Sensing

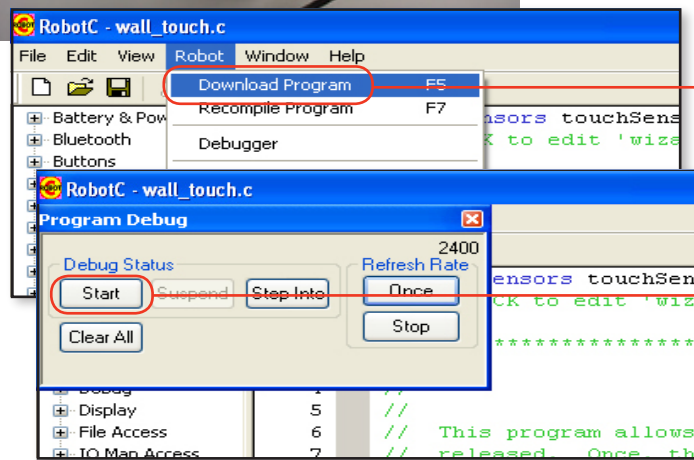
### Wall Detection **Configuring Sensors** (cont.)

5. Download and run the program, but before you run it, pick up or block up the robot so it doesn't run into anything.



**5a. Block up the robot**

Place an object under the robot so that its wheels can't reach the table. This lets you run the robot without having to chase it around.



**5b. Download the program**

Click Robot > Download Program.

**5c. Run the program**

Click "Start" on the onscreen Program Debug window.

6. While the program is running, press the Touch Sensor. The robot continues to move forward, rather than reversing direction.



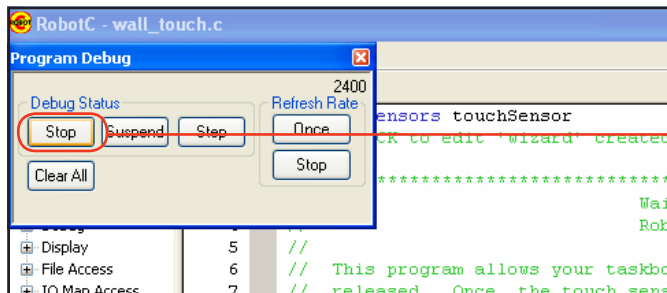
**6. Press the touch sensor**

Press the orange button on the Touch Sensor and observe the robot's reaction (or lack thereof).

## Sensing

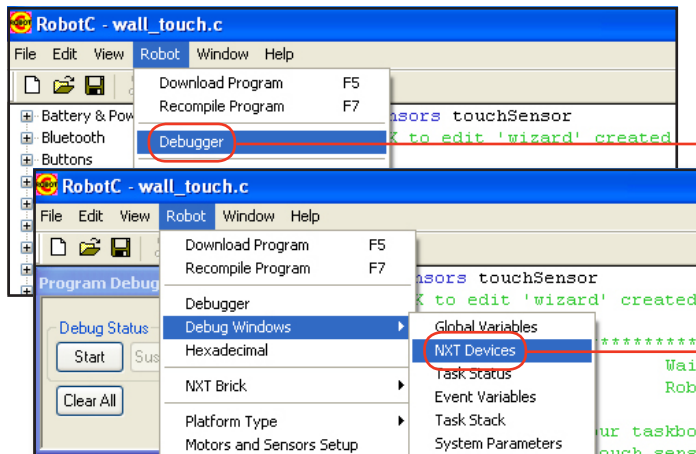
### Wall Detection **Configuring Sensors** (cont.)

7. Stop the program to conserve battery power.



**7. Stop the program**  
Click "Stop" on the Program Debug window.

8. Bring up the NXT Device Control Display window to find out why the Touch Sensor no longer makes the robot reverse direction. If the NXT Device Control Display window is already visible, skip this step.



**8a. Bring up the Debugger**  
Select Robot > Debugger.

**8b. Bring up the Device Window**  
Select Robot > Debug Window > NXT Devices.

## Sensing

### Wall Detection **Configuring Sensors** (cont.)

9. Observe the changes in the NXT Device Control Display window when you run the program, and when you touch the Touch Sensor.

**9a. Observe S2**  
Find the S2 box under Sensor in the NXT Device Control Display. Under Type, it should say "Raw Value".

**9b. Change Refresh Rate to Continuous**  
If you see a button labeled "Continuous", press it. Otherwise, skip this step.

**9c. Start the program**  
Click Start in the Program Debug window.

**9d. Observe "Type" of S2**  
The "Type" of sensor on S2 is now a Touch Sensor, just as we set it to be.

**9e. Observe "Value" of S2**  
A Touch Sensor will show a "Value" of 1 if the sensor is pressed, and a value of 0 otherwise. What does this value indicate?

**9f. Press the Touch Sensor and watch the S2 value**  
Press the Touch Sensor, and watch for a change (or lack of change) in the "Value" box in S2. Should it change?

## Sensing

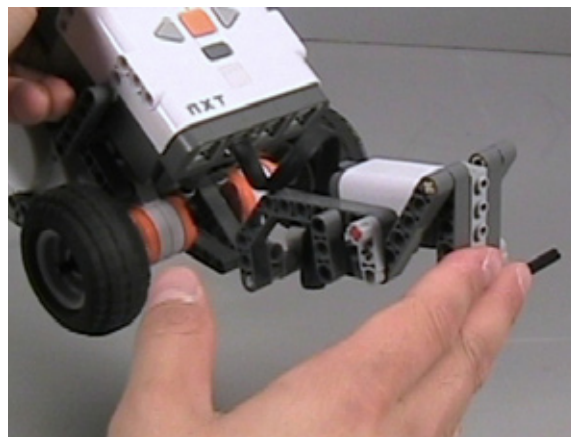
### Wall Detection **Configuring Sensors** (cont.)

- 10.** Even when you press the Touch Sensor, the S2 value remains 0. This makes sense, because the Touch Sensor is attached to Port 1, not Port 2. Try connecting the Touch Sensor to port 2 and see what happens.



**10a. Switch sensor ports**

Disconnect the Touch Sensor from port 1 and reconnect it to port 2 on the NXT.

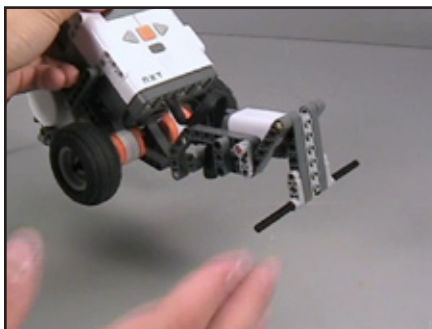


**10b. Press the Touch Sensor and watch the S2 value**

Press the Touch Sensor, and watch for a change (or lack of change) in the S2 Value in the NXT Device Control Display.

### Checkpoint

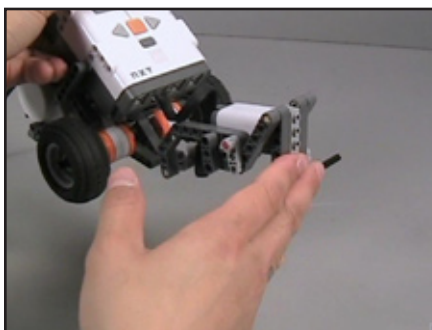
Now when you press the Touch Sensor, the S2 value turns to 1. A value of 1 indicates “pressed” on a Touch Sensor. Also, the program now works as it did before. When you press the Touch Sensor, the motor now reverses for one second and stops.



0	OFF(Float)	0	none	Idle
100	ON(Brake)	3	none	Running
100	ON(Brake)	3	none	Running
	Mode	Value	Raw	Var
e	modeRav	1023	1023	Syr
e	modeBoc	0	1023	Syr
e	modeRav	1023	1023	Bal
e	modeRav	1023	1023	Sle
				Vol

**Not Pressed**

The value for the sensor S2 is 0 while the Touch Sensor remains unpressed



0	OFF(Float)	0	none	Idle
-75	ON(Brake)	3	none	Running
-75	ON(Brake)	3	none	Running
	Mode	Value	Raw	Var
e	modeRav	1023	1023	Syr
e	modeBoc	1	102	Syr
e	modeRav	1023	1023	Bal
e	modeRav	1023	1023	Sle
				Vol

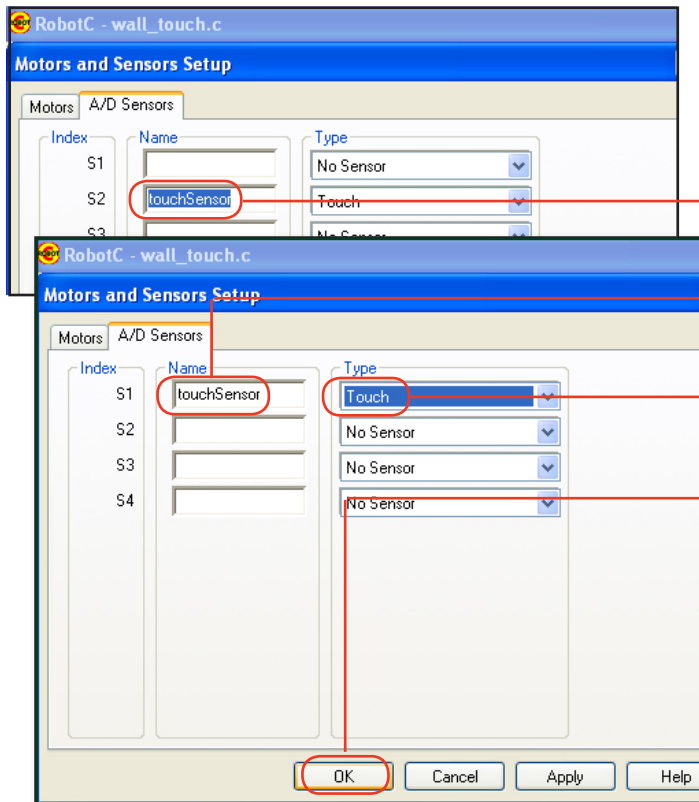
**Pressed**

The value for the sensor S2 is 1 when the Touch Sensor is pressed.

## Sensing

### Wall Detection **Configuring Sensors** (cont.)

**11.** Use the Motors and Sensors Setup menu to change the sensor settings back to the way they were so we can move on with the program.



**11a. Delete "touchSensor" from S2**  
Delete "touchSensor" from the S2 Name box.

**11b. Enter "touchSensor" in S1**  
Type "touchSensor" in the S1 Name box.

**11c. Change S1 Type to "Touch".**  
Select "Touch" from the S1 "Type" dropdown menu.

**11d. Click OK**  
Click OK to confirm the change.



**11e. Switch sensor ports**  
Disconnect the Touch Sensor from port 2 and reconnect it to port 1

### End of Section

You have successfully used the Motors and Sensors Setup menu to configure the Touch Sensor to work on port 2, and now changed it back to port 1. This is the universal process for configuring sensors in ROBOTC. You also learned to use the NXT Device Control Window to view sensor values. Finally, you also saw the two values the Touch Sensor can provide: 0 (unpressed) and 1 (pressed). It's time to move on to the next lesson, where you will examine the part of the program called the while loop.



## Sensing

# Wall Detection The while() Loop

Your robot's ability to sense and respond to touch revolves around a structure in the program called a while() loop. The while() loop in this program uses the Touch Sensor feedback to decide whether the robot should continue on its current course, or back up and turn.

*In this lesson, you will learn what a while() loop is and how it works.*

Below is the code for the sample program's while() loop. Reading this statement out loud tells you pretty much exactly what it does:

*"While the sensor value of the **Touch Sensor** is equal to zero, run motors C and B at 100% power."*

```

38     while(SensorValue(touchSensor) == 0)
39     {
40         motor[motorC] = 100;
41         motor[motorB] = 100;
42     }

```

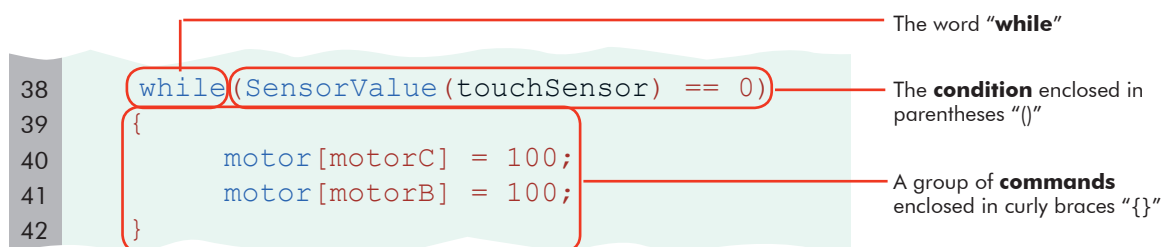
The **decision-making nature of the while() loop** may not be apparent at first, but making decisions that control the flow of the program is actually the while() loop's main purpose. The while() loop above instructs the program to use the Touch Sensor's status to **decide** how long to keep the motors running.

When the program reaches most commands, it runs them, and then moves on. When the program reaches the while() loop, however, it steps "inside" the loop, and stays there as long as the while() loop decides that it should. The loop also specifies a set of commands that the robot will repeat over and over as long as the program remains inside the loop.

The programmer specifies in advance *under what **conditions** the program should remain in the loop*, and *what **commands** the robot should repeat while inside the loop*.

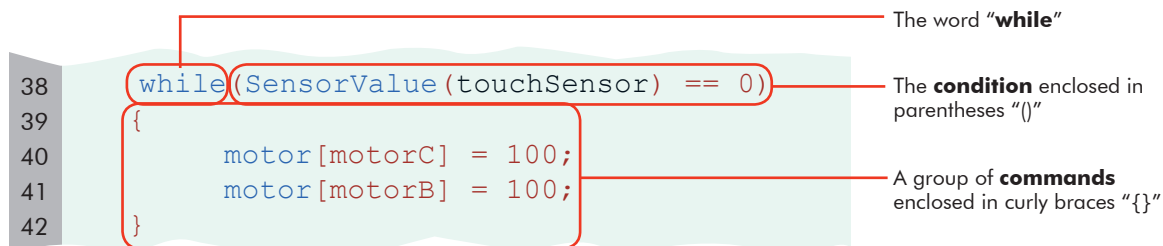
The while() loop therefore has three parts, in order:

- The word "**while**"
- The **condition** enclosed in parentheses "()"
- A group of **commands** enclosed in curly braces "{}"



## Sensing

### Wall Detection **The while() Loop** (cont.)



#### while

A while() loop always starts with the word "while".

<pre> while(condition) {     commands; }           </pre>	<b>General form</b> while() loops always follow the pattern shown here
---	---

#### The (condition)

The statement in parentheses specify the *condition(s)* under which the loop should continue looping. These conditions are specified in the form of a true-or-false statement, like the one in the example above, "The sensor value of the Touch Sensor is equal to zero". The statement is either *true* (the value IS zero) or it is *false* (the value IS NOT zero).

The *true* (or *false*) value of the statement determines whether the loop will continue or end. As long as the condition is true, the while loop will continue to run. If the condition becomes false, the loop will end and the program will move on to the commands that come after it.

#### Example

In the code above, the condition is "The sensor value of the Touch Sensor is equal to zero." This (condition) statement is true as long as the Touch Sensor reads zero. Recall from the previous lesson that the Touch Sensor reads 0 whenever its button is not pressed in, and it reads 1 when the button is pressed in.

So, as long as the Touch Sensor button is *NOT* pressed, the sensor value will be zero, and the condition will be true. As long as the condition remains true, the commands inside the curly braces will run. If the Touch Sensor is ever pressed, its value will become 1, not 0, and the condition will become false. The loop would then end.

#### The {commands}, sometimes called the "body"

These are the commands that are run while the condition is true. The commands inside the braces are run in order. When they have all been run, the program goes back to check the condition again. If the (condition) is still true, the loop continues and the {commands} are run again. In the code shown above, the {commands} are to run both of the robot's motors at full power forward, and the program will do that as long as the touch sensor remains unpressed.

#### End of Section

The while() loop allows the program to make a decision about program flow, based on a true-or-false statement. It works by checking to see if a (condition) is true, then, if it is true, running a group of {commands}, and looping back to recheck the (condition). If the (condition) ever stops being true, the while() loop skips over the {commands}, and moves on to the next section of the program.

In the wall\_touch program, the robot will move forward at full power while the Touch Sensor remains unpressed, then exit the loop and move on to the rest of the program. A well-planned choice of commands to follow the loop tell the robot to back away from the obstacle afterwards.

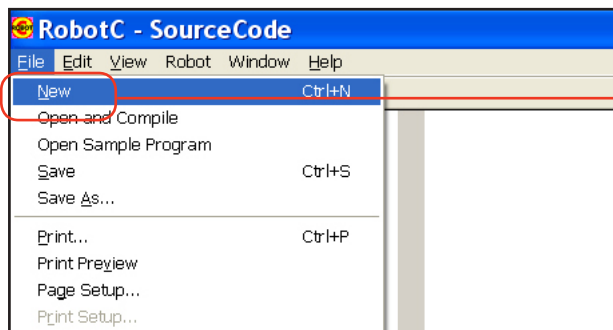


## Sensing

# Wall Detection **Putting it Together**

Now that you have examined and worked with the pre-written “Wait for Touch” program, it’s time to write one on your own.

### 1. Start with a new program.



#### 1. Create new program

Select File > New to create a blank new program.

### 2. Remember the program has to do three things:

- Configure the sensor port to recognize a Touch Sensor on Port 1
- Create a while() loop that runs forward while the touch sensor is unpressed
- Back away from the obstacle afterwards

## Sensing

### Wall Detection **Putting it Together** (cont.)

3. So let's do them in order, starting with the first: configure the sensor port.

**3a. Open "Motors and Sensors Setup"**  
Select Robot > Motors and Sensors Setup to open the Motors and Sensors Setup menu.

**3b. Select the A/D Sensors tab**  
Click the "A/D Sensors" tab" on the Motors and Sensors Setup menu.

**3c. Give S1 the Name "bumper"**  
In the "Name" box next to S1, type "bumper".

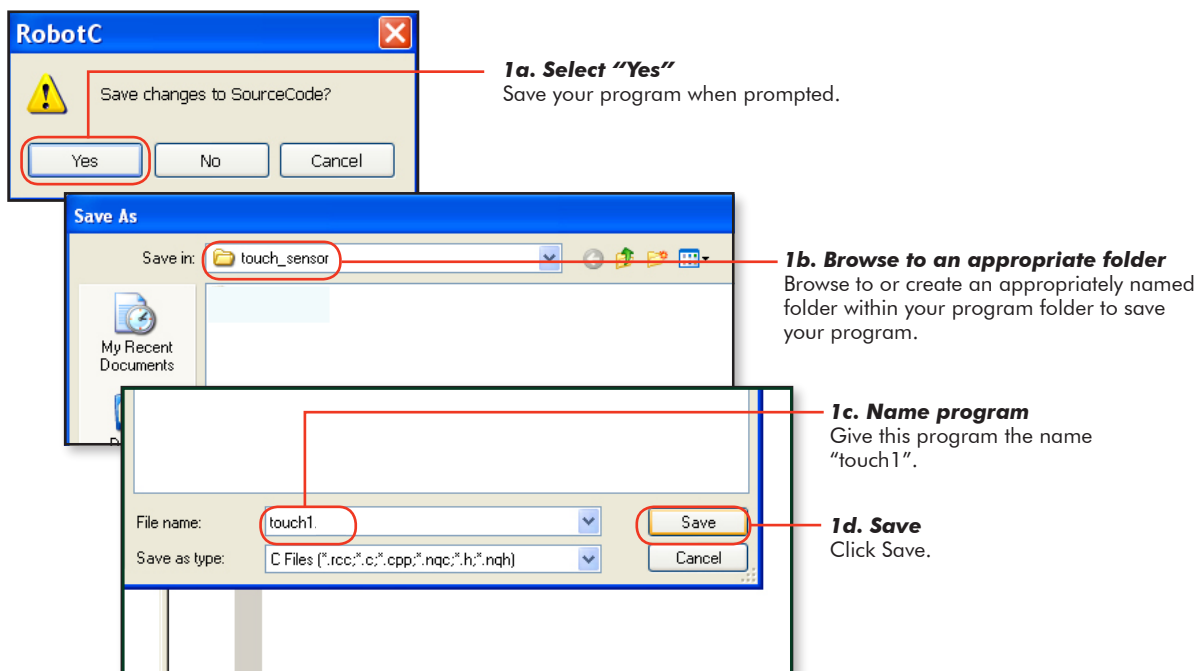
**3d. Designate S1 as a Touch Sensor**  
Select "Touch" from the dropdown box in the "Type" area.

**3e. Click OK**  
Click the "OK" button to save your changes.

## Sensing

### Wall Detection **Putting it Together** (cont.)

4. ROBOTC will want you to save your program at this point. Save your program with your other programs as "touch1".



## Sensing

### Wall Detection **Putting it Together** (cont.)

5. Create task main().

```
Auto const tSensors bumper = (tSensors) S1;
Auto /**!!CLICK to edit 'wizard' created sensor
1
1 task main()
1 {
1
1
1
1
1 }
```

**5. Add this code**

Create the basic task main() {}.

6. Create the while() loop.

```
Auto const tSensors bumper = (tSensors) S1;
Auto /**!!CLICK to edit 'wizard' created sensor
1
1 task main()
1 {
1
1 while()
1 {
1
1
1
1
1
1
1 }
```

**6. Add this code**

Add the while() loop: the word "while", the parentheses to hold the condition, and the curly braces to hold the commands.

### Checkpoint

Your program should now look like this, with a while() loop within task main(). Line numbers will not update until you compile, so the line of 1s is normal.

```
Auto const tSensors bumper = (tSensors) S1;
Auto /**!!CLICK to edit 'wizard' created sensor
1
1 task main()
1 {
1
1 while()
1 {
1
1
1
1
1
1
1 }
```

## Sensing

### Wall Detection **Putting it Together** (cont.)

7. Write the condition.

```
Auto const tSensors bumper = (tSensors) S1;
Auto /*!!CLICK to edit 'wizard' created sensor
1
1 task main()
1 {
1
1     while(SensorValue(bumper) == 0)
1     {
1
1
1
1     }
1
1 }
```

**6. Add this code**

The condition should test whether the Touch Sensor is unpressed. Thus, the condition is that the Touch Sensor value is equal to zero. Recall that == means "is equal to".

7. Tell the robot what to do while the Touch Sensor is unpressed: go forward at a prudent 50% power (since are expecting to run into an object at some point).

```
Auto const tSensors bumper = (tSensors) S1;
Auto /*!!CLICK to edit 'wizard' created sensor
1
1 task main()
1 {
1
1     while(SensorValue(bumper) == 0)
1     {
1
1         motor[motorC] = 50;
1         motor[motorB] = 50;
1
1     }
1
1 }
```

**6. Add this code**

Turn Motors A and B on forward at full speed. Because this code is inside the while loop's {} braces, they will be run repeatedly as long as the condition remains true.

## Sensing

### Wall Detection **Putting it Together** (cont.)

9. Tell the robot what to do after the Touch Sensor is pressed and the while loop ends.

```
Auto const tSensors bumper = (tSensors) S1;
Auto /*!!CLICK to edit 'wizard' created sensor

1
1 task main()
1 {
1
1     while (SensorValue(bumper) == 0)
1     {
1
1         motor[motorC] = 50;
1         motor[motorB] = 50;
1
1     }
1
1     motor[motorC] = -50;
1     motor[motorB] = -50;
1     wait1Msec(1000);
1
1 }
```

**9a. Add this code**

Run motors A and B backward at 50% power. Because this code comes after the } of the while loop, it will be run only after the loop is done.

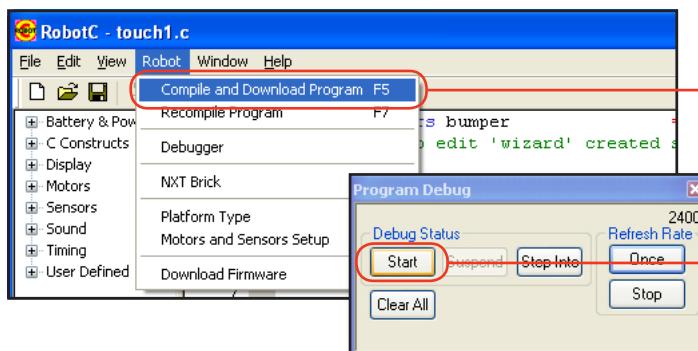
Thus, the robot will back up AFTER the loop ends.

**9b. Add this code**

Leave the motors running for 1 second.

### End of Section

Download and run your program. Congratulations, you have now programmed your robot to use a sensor to detect and respond to its environment! In fact, you've just created your first true robot. The ability to use sensor feedback to govern its own behavior is what sets a robot apart from other machines.



**Download the program**

Click Robot > Download Program.

**Run the program**

Click "Start" on the onscreen Program Debug window.



**Forward until touch**

The robot runs forward as dictated by the while() loop, then, when the touch sensor is pressed and the loop ends, the program continues on to the backing-up commands.