Automatic Threshold Values and Variables

In this lesson, we're going to look a little deeper into the world of "values," and pay special attention to the programming structures that are used to represent and store values, which are called "variables."

In the previous lesson, "Speed Based on Volume", the robot set its motor power levels based on sound sensor readings. To the robot, this was no different than setting the power level to 25, 50, or 100. These numbers – 25, 50, 100, Sound Sensor readings – are all interchangeable **values** that could be used to set the motor power levels.

There are some situations where values need to be stored for later use. For example, a robot sent into a cave to gather Light Sensor values needs to both **record those values** inside the cave and **be able to recall them** afterwards.

Robot enters the cave
The robot enters the cave (dark area on the right) to gather data.

Robot takes sensor readings
The robot must take and **store** sensor readings inside.

Robot returns
The robot backs out of the cave and displays the values from inside.

Without some way to store these values, they will be lost by the time the robot leaves the cave. **Variables are the robot's way of storing values for later use.** They function as containers or storage for values. Values such as the cave robot's sensor reading can be placed in a variable when calculated (inside the cave), and retrieved at a later time (outside the cave) for convenient use. A variable is simply a place to store a value.

There are, however, different types of values. For instance, there are different types of numbers (integers versus decimals, to name just two), and there are values that aren't even numbers, like words. Since there are different types of values, there are **different types of variables** to hold them. In order to create (or "declare") a variable, the programmer must identify two key pieces of information: the **type** of value it will hold, and a **name** for the variable.

Automatic Threshold Values and Variables (cont.)

The **names** of variables can include anything that follows the general ROBOTC naming rules (see the "Wall Detection (Touch)" lesson for a list of rules). For **types**, ROBOTC breaks values down into a few simple categories.

Number values in ROBOTC are broken down into two different kinds of numbers:

Integer, or "`int`" values are numbers with no fractional or decimal component.

Integers

10, 0, -10

Non-Integers

Floating point ("`float`") numbers are so called because the decimal point "floats" around in the value, allowing decimal places to be used. Floating point numbers can be positive, negative, or zero, but they may also represent decimals. Floating point numbers take up more memory on the robot, and are slower to calculate with, so integer values are preferred when decimals aren't necessary.

Floating Point Numbers

3.1456, 31.456, 0.0, -314.56

Other kinds of values also exist, including text like "Hello", and logical values like True.

Strings ("string"): Text in ROBOTC is always a "`string`". In ROBOTC, the word "Hello" is really a collection of letters – 'H', 'e', 'l', 'l', 'o' – "strung" together to form a single value. In fact, while all words are strings in ROBOTC, all strings are not words, and do not even have to be collections of letters. A string may be a series of numbers, or a series of mixed numbers and letters.

Strings

"Hello", "my name is", "a16Z"

Boolean ("`bool`") values represent "truth" or "logic" values, in the form of "true" or "false".

Boolean Values

true, false

Automatic Threshold Values and Variables (cont.)

To declare a variable, simply call out its type, then its name, then end with a semicolon.

`int lightValue;` will create a new integer-type variable named lightValue.

`bool isAwake;` will create a new true-or-false (Boolean) variable named isAwake.

Optionally, you can also **assign a value** to the variable at this point, but it is not necessary.

`int lightValue = 0;` will create a new integer-type variable named lightValue, with a starting value of 0.

`bool isAwake = true;` will create a new true-or-false (Boolean) variable named isAwake, with a starting value of true.

Data Type	Description	Example	Code
Integer	Positive and negative whole numbers, as well as zero.	-35, -1, 0, 33, 100, 345	`int`
Floating Point Decimal	Numeric values with decimal points.	-.123, 0.56, 3.0, 1000.07	`float`
String	A string of characters that can include numbers, letters, or typed symbols.	"Counter reached 4", "STOP", "time to eat!"	`string`
Boolean	True or False. Useful for expressing the outcomes of comparisons.	true, false	`bool`

End of Section

Things like motor powers and sensor readings are **values**. Values can be of different types, like integers or strings. When you need to store them, you can use a **variable** of the appropriate type to hold the value for later use. Variables must be declared by assigning them a suitable **type** and a **name**. Names must follow the usual ROBOTC naming rules, and should be chosen so that you will be able to remember what each variable is supposed to be doing when you read or troubleshoot your code later.

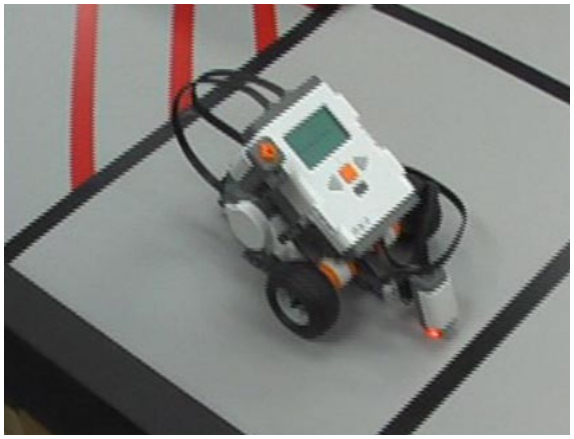
Automatic Threshold Variables and Threshold

Having to reprogram the robot every time the lighting conditions change is not efficient.

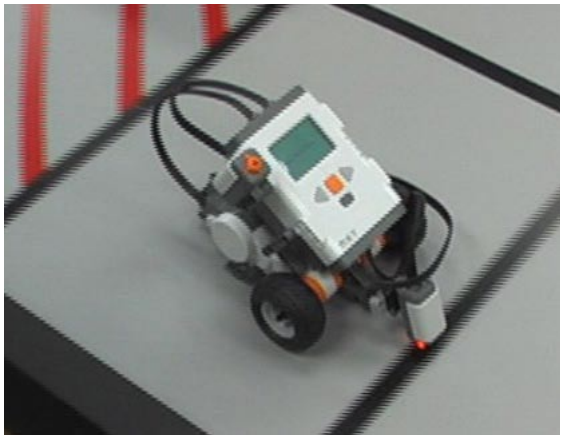
> *In this lesson, we will give the robot the ability to configure itself at the beginning of every run, with only a little human assistance.*

When the program begins, the user will be prompted to "scan" a light surface with the Light Sensor, and then "scan" a dark surface. The robot will then calculate its own Light Sensor threshold, wait a few seconds, and proceed as normal.

We'll begin by going through the threshold calculation process manually, and taking note of the important values that the robot will have to keep track of. Every time a number or value has to be remembered, make a note.



Scanning light
The robot's light sensor is first positioned over a **light** surface and told to read and store its value



Scanning dark
Then, the robot's light sensor is positioned over a **dark** surface and told to read and store its value

Automatic Threshold Variables and Threshold (cont.)

1. Turn on your NXT and navigate to the "View" mode using the gray arrows.

1a. Push the orange button
Turn on the robot by pushing the orange button. The screen should display "My Files" when it is on.

1b. Go to the "View" menu
Navigate to the "View" menu using the arrow buttons. Press the orange button to go into it.

1c. Select "Reflected Light"
Select "Reflected Light", not "Ambient Light". You will get different values otherwise.

1d. Select your port number
Select the port number that your Light sensor is plugged into.

2. Record your Light and Dark readings. Record these values.

2a. Record the light value
Place the robot on the light surface, and record the value that the Light sensor is reading.

Automatic Threshold Variables and Threshold (cont.)

2b. Record the dark value
Place the robot on the dark surface, and record the value that the Light sensor is reading.

5. Find the average of the light and dark readings by adding them together and dividing by two. This thresholdValue will be used for future comparison.

5a.

light value + dark value = sum

66 + 33 = 99

5b.

sum / 2 = average

99 / 2 = 49.5

5c.

Note: Get rid of the decimal number

49.5 = 49

Get rid of the decimal
ROBOTC will get rid of the decimal automatically when using integers.

5d.

average = threshold

49 = thresholdValue

Automatic Threshold Variables and Threshold (cont.)

Checkpoint
Four values were either recorded or calculated: light value, dark value, sum, and threshold.

light value + dark value = sum
66 + 33 = 99

Calculate "sum" value
The sum value is found by adding the light value and dark value.

sum / 2 = threshold
99 / 2 ≈ 49

Calculate average/"threshold" value
The average is found by dividing the sum value by 2. The resulting average is the threshold value.

In order to write a program that will auto-calculate the value of threshold, we will need to create four variables to store the four values that the calculation needs. To declare each variable, a **name** and **type** must be specified. The name should help you to remember what the variable does. For this lesson these values will be named:

- lightValue
- darkValue
- sumValue
- thresholdValue

In addition to a name, the **type** of value (integer, floating point decimal, string, boolean value) that each variable will hold needs to be determined.

Light Sensors yield values that are whole numbers. So lightValue and darkValue will be "declared" as **integers**. Since the sum of two integers is also an integer, sumValue will be declared as an integer as well. Dividing by two might result in a decimal, but since the threshold is an estimate to begin with, rounding won't hurt it, and so thresholdValue will also be declared as an integer.

Declaring Variables

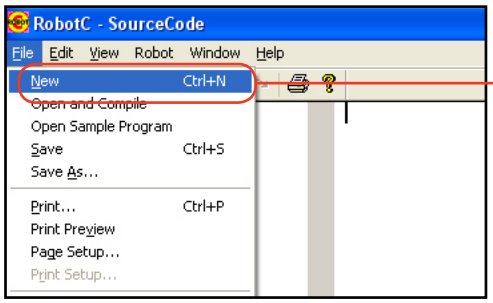
To create a variable, you must "declare" it with two pieces of information:

datatype then ***name***;

Example:
`int lightValue;` will create a new integer-type variable named lightValue.

Automatic Threshold Variables and Threshold (cont.)

7. Place the four variables declared as integers in a new program.



7a. Create new program
Select File > New to create a blank new program.

```
2  task main()
3  {
4
5
6
7
8
9
10 }
```

7b. Add this code
These lines form the main body of the program, as they do in every ROBOTC program. Leave four lines between curly brackets for the variables.

```
2  task main()
3  {
4
5    int lightValue;
6    int darkValue;
7    int sumValue;
8    int thresholdValue;
9
10 }
```

7c. Add these lines
Declare the four variables, lightValue, darkValue, sumValue and thresholdValue as integers. Remember that typographic errors can keep the program from functioning!

End of Section
Four variables have been created to store the four values needed to calculate a Light Sensor threshold. In the next lesson we will write the remainder of the program.

Automatic Threshold Programming with Variables

In this lesson, you will learn how to store Light Sensor values in the variables you created, and how to use a Touch Sensor as a user interface button.

The robot will take the first Light Sensor reading over a light surface when the Touch Sensor is pressed, then take a second reading over a dark surface when the Touch Sensor is pressed a second time.

```
2  task main()
3  {
4
5    int lightValue;
6    int darkValue;
7    int sumValue;
8    int thresholdValue;
9
10 }
```

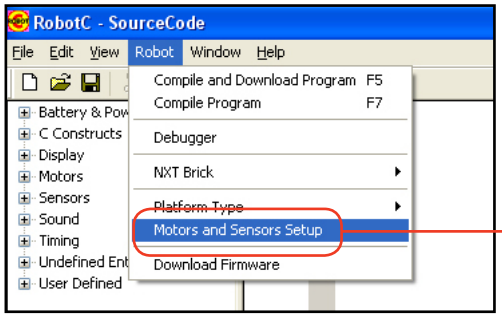
Existing program
Your program should currently look like this.

First, we'll configure the Light and Touch Sensors.

Light Sensor

Touch Sensor

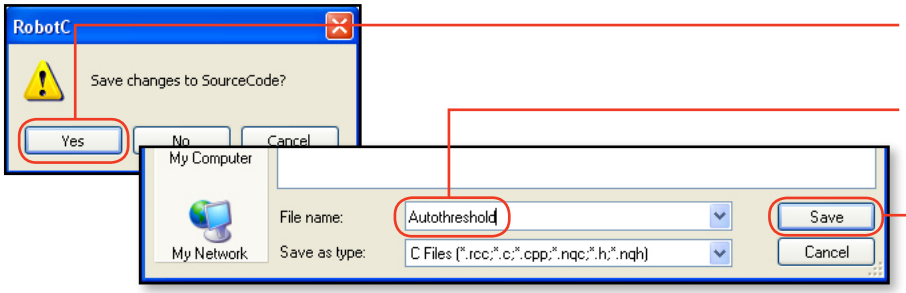
1. Open the Motors and Sensors Setup menu.



1. Open "Motors and Sensors Setup"
Select Robot > Motors and Sensors Setup to open the Motors and Sensors Setup menu and configure the sensors.

Automatic Threshold Programming with Variables (cont.)

2. ROBOTC will ask if you want to save your program. Click Yes, then save the program as "Autothreshold".

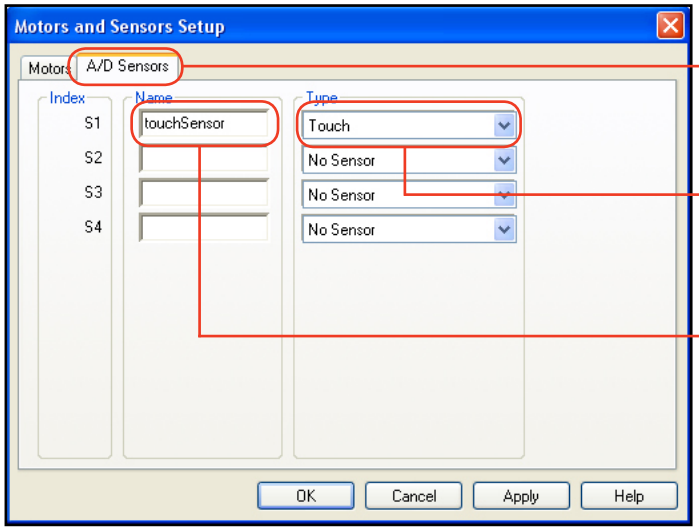


2a. Select "Yes"
Save your program when prompted.

2b. Name the program
Name the program "Autothreshold".

2c. Save the program
Press Save to save the program with the new name.

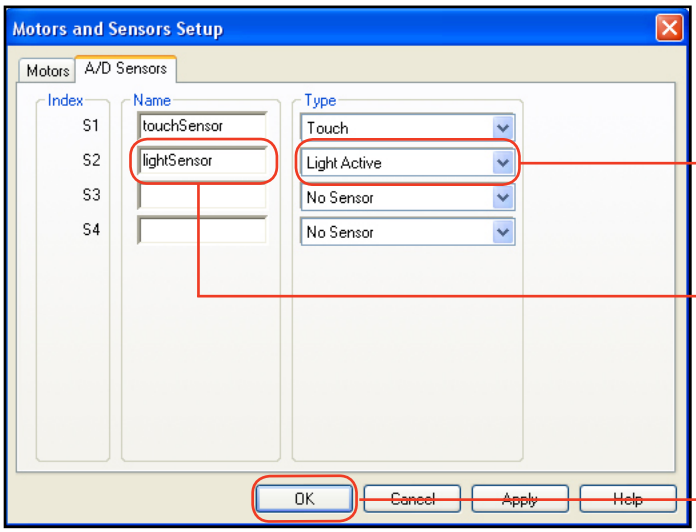
3. Select the A/D Sensors tab, and make Port 1 the Touch Sensor, named touchSensor, and Port 2 the Light Sensor, named lightSensor.



3a. Select "A/D Sensors" tab
Selecting this tab allows you view your sensors set up menu.

3b. Set sensor type
Identify the Sensor Type as a "Touch" sensor.

3c. Name the sensor
Name the Touch Sensor on port S1 "touchSensor".



3d. Set sensor type
Identify the Sensor Type as a "Light Active" sensor.

3e. Name the sensor
Name the Light Sensor on port S2 "lightSensor".

3f. Click OK

Automatic Threshold **Programming with Variables** (cont.)

The next step is for the robot to take the first Light Sensor reading over a "light" surface when the Touch Sensor is pressed. Then, take the dark reading on the next Touch Sensor press.

Automatic Threshold Programming with Variables (cont.)

4. The robot should wait for the Touch Sensor to be pressed. A while() loop is used to check the touchSensor value to watch for a press. As long as the Touch Sensor isn't pressed, `(SensorValue(touchSensor)==0)` remains true, `and` the robot does nothing.

```
2  task main()
3  {
4
5   int lightValue;
6   int darkValue;
7   int sumValue;
8   int thresholdValue;
9
10  while (SensorValue(touchSensor)==0)
11  {
12  }
13
14 }
```

4. Add this code
This `while()` loop **idles** (i.e. runs an empty {} code block) while the Touch Sensor is not pressed.

---

5. After the Touch Sensor is pressed, record the Light Sensor's value to the variable lightValue. Assign the value of the sensor to the variable. `LightSensor = SensorValue(lightSensor)` Note: A single equals sign means, "set to the value of".

```
2  task main()
3  {
4
5   int lightValue;
6   int darkValue;
7   int sumValue;
8   int thresholdValue;
9
10  while (SensorValue(touchSensor)==0)
11  {
12  }
13
14  lightValue=SensorValue(lightSensor);
15
16 }
```

5. Add this code
This line puts the Light Sensor's value into the variable lightValue.

Automatic Threshold Programming with Variables (cont.)

6. Next, the robot records the dark value. Either retype the wait-for-press loop, and the storing of the value manually, or just highlight and copy the code you just wrote, (starting with "while" and ending with the semicolon) and paste another copy of it below. In this second recording, of course, you want to record the value to the dark Value.

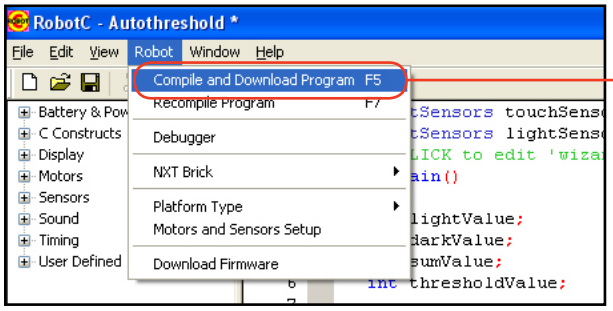
```
2  task main()
3  {
4
5   int lightValue;
6   int darkValue;
7   int sumValue;
8   int thresholdValue;
9
10  while (SensorValue(touchSensor)==0)
11  {
12  }
13
14  lightValue=SensorValue(lightSensor);
15
16  while (SensorValue(touchSensor)==0)
17  {
18  }
19
20  darkValue=SensorValue(lightSensor);
21
22 }
```

6a. Add this code
This `while()` loop idles while the Touch Sensor is not pressed, just like the previous one.

6b. Add this code
This line puts the Light Sensor's value into the variable "darkValue".

Checkpoint
Check to see if the program is working. It is almost always better to write code in small bits and test often, rather than waiting to test a long section of code in which many mistakes could be hiding.

7. Compile, Download and run your program.



7. Compile and Download
Robot > Compile and Download Program

Automatic Threshold Programming with Variables (cont.)

8. Run the program. Put the Light Sensor over a light surface. Press the Touch Sensor. Keep an eye the robot... it may not do what you expect!

---

9. The program seems to end immediately when the Touch Sensor is pressed. That's not what we wanted!

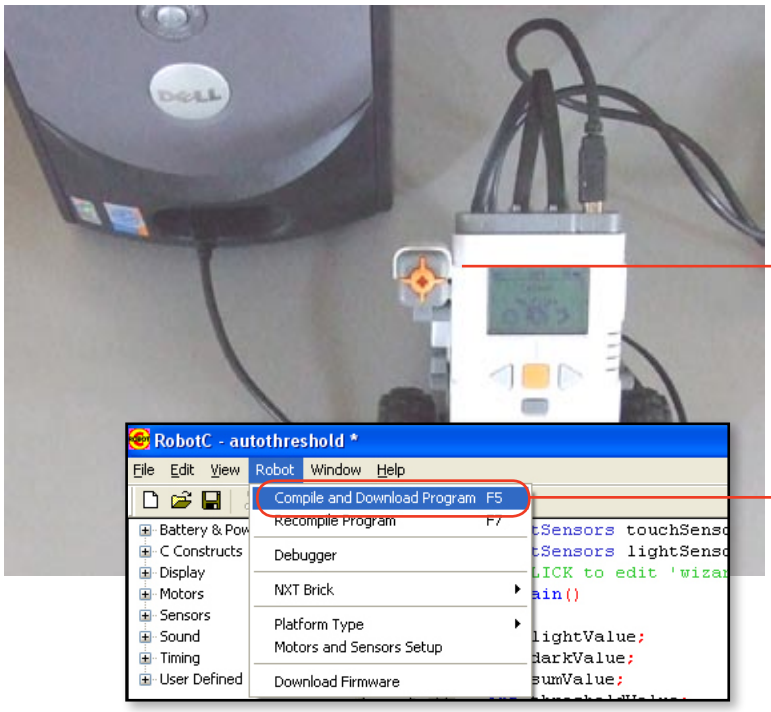
End of Section

Something is wrong with the program. In the next lesson, the debugger will be used to fix the problem.

Automatic Threshold Variables and the Debugger

In this lesson, the Debugger windows will be used to determine why the program is not running properly. The debugger can be used to "freeze time" for the robot and allows you to step through the program at whatever speed you want.

1. Something is obviously wrong with the program. Download the program again, but this time, make sure the robot stays plugged into the computer, and watch the code window.

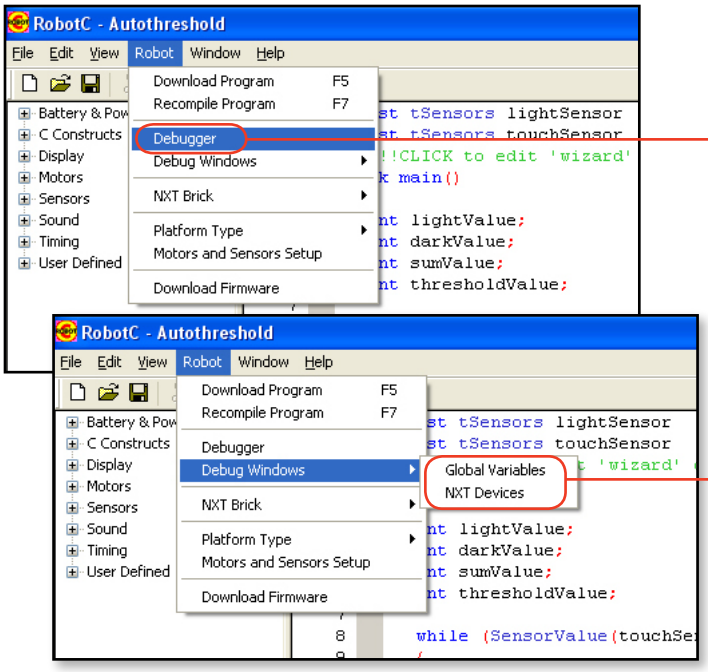


1a. Plug the robot back in
Robot has to be plugged into the computer, via USB, to be able to view the code window.

1b. Compile and download
Select Robot > Compile and Download Program. The option may just read "Download Program", which is fine also.

Automatic Threshold Variables and the Debugger (cont.)

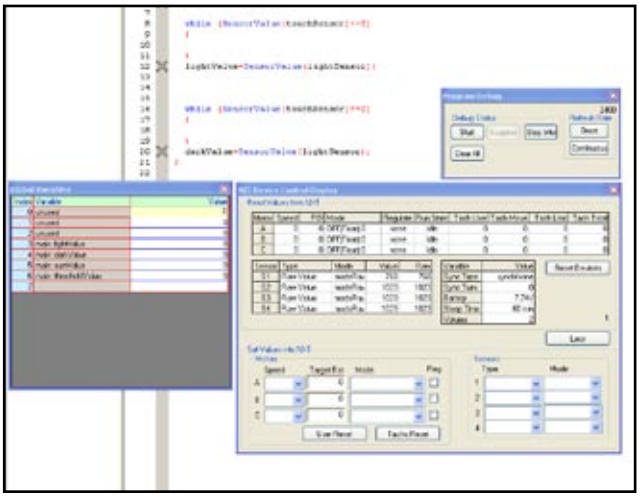
2. After you have downloaded the program to your robot, fix the problem by open up the Debugger, then select both the Global Variables and the NXT Devices options so both these windows are visible.



2a. View Debugger
Select Robot > Debugger to open up the Program Debug window.

2b. View Debugger Windows
Select Robot > Debug Windows and select both Global Variables and NXT devices if they are not already checked.

Checkpoint
The screen should look like the sample below with three windows visible: Program Debug, Global Variable and NXT Device Control Display.



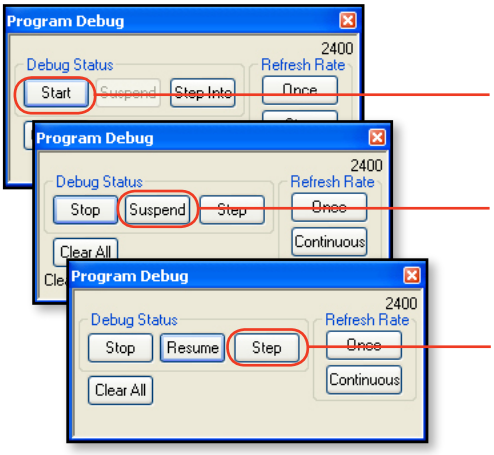
Automatic Threshold Variables and the Debugger (cont.)

3. Run the program. Observe what happens when you push the Touch Sensor.

3. Push Touch Sensor
Pushing the Touch Sensor allows the program to move forward out of the while () loop.

Checkpoint
The button was pressed once, and the program shot straight to the end. You can tell the program is finished because the Start button on the Program Debug window is highlighted. (If the program was still running, the Suspend button would be highlighted.)

4. Run the program again, but this time use the Program Debug window to "freeze" time and step through the program while suspended. To do so, press the Suspend button, then the Step button.



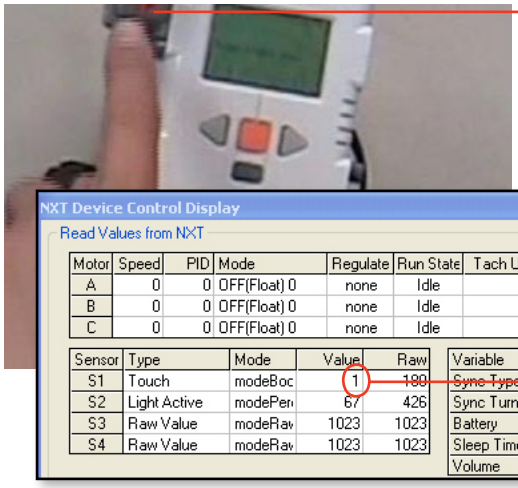
4a. Press Start button
Press the Start button to get the program started.

4b. Press Suspend button
Press the Suspend button on the Program Debug window to "stop" time and leave the program right where it is.

4c. Press Step button
Press the Step button to go to the next line of code.

Automatic Threshold Variables and the Debugger (cont.)

5. Press the Touch Sensor and observe in the NXT Device Control Display that it is pushed and working properly.



5a. Push Touch Sensor
Pushing the Touch Sensor allows the program to move forward out of the while () loop.

5b. Observe the Touch Sensor
The value of the Touch Sensor, 1, means that it is pressed.

Since you have suspended the program, the robot's program remains "frozen" at the first while() loop (where the yellow line appears in the code). The NXT Device Control window on your PC screen, however, remains operational, and will continue to report the value of the sensors.

```
2  task main()
3  {
4
5    int lightValue;
6    int darkValue;
7    int sumValue;
8    int thresholdValue;
9
10   while (SensorValue(touchSensor)==0)
11   {
12   }
13
14   lightValue=SensorValue(lightSensor);
15
16   while (SensorValue(touchSensor)==0)
17   {
18   }
19
20   darkValue=SensorValue(lightSensor);
21
22 }
```

Line about to run
The program will run this step when the Step button is pressed again.

Because the line is a while loop, it will evaluate the (condition) and decide whether to loop, or move on.

Automatic Threshold Variables and the Debugger (cont.)

6. While continuing to hold the Touch Sensor in the pushed position, click the Step button on the Program Debug control panel to allow the program to move past the while() loop.

6a. Push the Touch Sensor
Hold the Touch Sensor in the pushed position while pressing the Step button.

6b. Press Step button
Press the Step button while pushing the Touch Sensor to allow you to go to the next step of the code.

Since the Touch Sensor value is not 0 at the time the while loop checks, the program moves past the loop to the next step. The next line turns yellow now to indicate that this command is *about* to be executed.

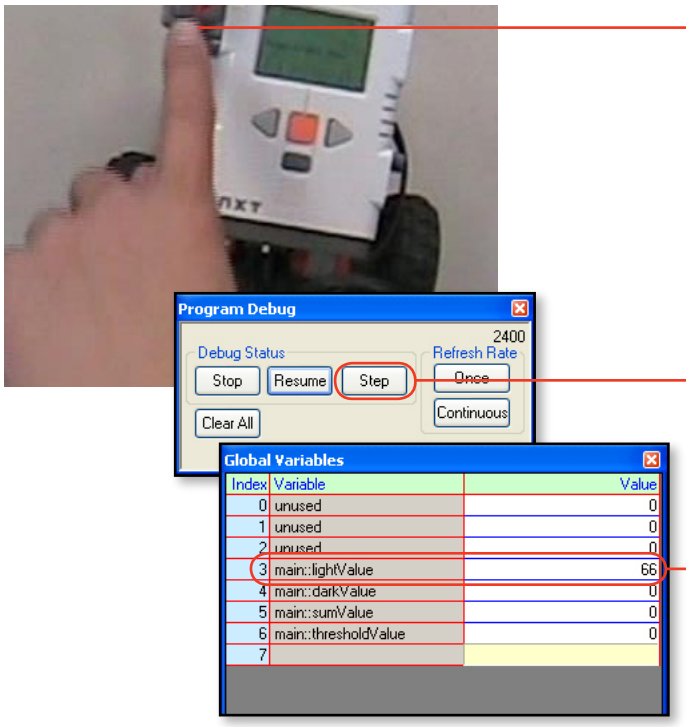
```
2  task main()
3  {
4
5   int lightValue;
6   int darkValue;
7   int sumValue;
8   int thresholdValue;
9
10  while (SensorValue(touchSensor)==0)
11  {
12  }
13
14  lightValue=SensorValue(lightSensor);
15
16  while (SensorValue(touchSensor)==0)
17  {
18  }
19
20  darkValue=SensorValue(lightSensor);
21
22 }
```

Line that was run
When you pressed Step, this line was run. The (condition) was False because the touchSensor value was 1 (and not 0), so the program exited the loop and moved on.

Line about to run
The program will run this line when the Step button is pressed again.

Automatic Threshold Variables and the Debugger (cont.)

7. Find the variable lightValue in the Global Variables window. Push the Touch Sensor. Keep it pushed in while pressing the Step button. The Light Sensor's value when the Step button was first pressed is now stored in the variable lightValue.



7a. Push the Touch Sensor
Hold the Touch Sensor in the pushed position while pressing the Step button.

7b. Press Step button
Press the Step button while pushing the Touch Sensor to enable the program to move to the next line of code.

7c. Stored Variable
The lightValue variable now equals the value of the Light Sensor when the Touch Sensor was first pushed, as shown in the Global Variables window.

```
13
14   lightValue=SensorValue(lightSensor);
15
16   while (SensorValue(touchSensor)==0)
17    {
18    }
19
20    darkValue=SensorValue(lightSensor);
21
22  }
```

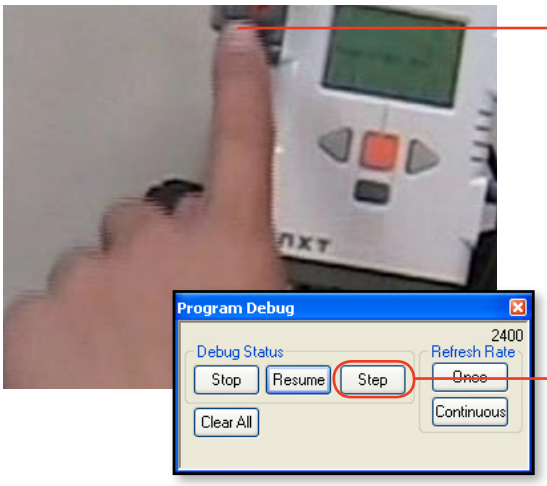
Line that was run
When you pressed Step, this line was run, and stored the value of the Light Sensor in the variable.

Line about to run
The program is now ready to run this next step when Step is pressed again.

Because the line is a while loop, it will evaluate the (condition) and decide whether to loop, or move on.

Automatic Threshold Variables and the Debugger (cont.)

8. While continuing to hold the Touch Sensor in, press the Step button several times to step through the rest of the program.



8a. Keep the Touch Sensor pressed
Hold the Touch Sensor in the pushed position while pressing the Step button.

8b. Press Step button
Press the Step button several times while pushing the Touch Sensor to step through to the end of the program.

```
13
14   lightValue=SensorValue(lightSensor);
15
16   while (SensorValue(touchSensor)==0)
17   {
18   }
19
20   darkValue=SensorValue(lightSensor);
21
22 }
```

Line that was run
When you pressed Step, this line was run. The (condition) was False because the touchSensor value was 1 (and not 0), so the program exited the loop and moved on.

8c. Press Step button again
The program moves to the next line of code, making the variable darkValue equal to the Light Sensor value the moment the Touch sensor was pressed.

```
13
14   lightValue=SensorValue(lightSensor);
15
16   while (SensorValue(touchSensor)==0)
17   {
18   }
19
20   darkValue=SensorValue(lightSensor);
21
22 }
```

8d. Press Step button again
The program moves to the next line of code, the last curly bracket, and the program ends.

Automatic Threshold Variables and the Debugger (cont.)

Checkpoint
Do you see what the problem is? When the Touch Sensor is **held down**, the program shoots straight through to the end of the program without stopping.

Why does it do this? Because we told it to. When the Touch Sensor was pressed, it took the program out of the first while loop. This was what we intended. But then, it quickly set the lightSensor variable, and then waited for the button to be pressed... which it still was, from the first press! The program immediately jumped past the second while loop. This is what we said, though certainly not what we wanted!

With the Step function, you could see this happening one step at a time. At normal speed, all this happens before you can take your finger off the button from the first press!

---

9. Place a command between the while() loops telling the robot to wait for 1 second before looking for the Touch Sensor value again. This allows the human operator enough time to push, **and release**, the Touch Sensor.

```
2  task main()
3  {
4
5   int lightValue;
6   int darkValue;
7   int sumValue;
8   int thresholdValue;
9
10  while (SensorValue(touchSensor)==0)
11  {
12  }
13
14  lightValue=SensorValue(lightSensor);
15
16  wait1Msec(1000);
17
18  while (SensorValue(touchSensor)==0)
19  {
20  }
21
22  darkValue=SensorValue(lightSensor);
23
24 }
```

9. Add this code
Tells the robot to wait for 1 second before it starts looking for the Touch Sensor again.

End of Section
In this lesson, the debugger was used as a tool to diagnose why a program was not working properly. Stepping through the commands in a program one at a time allows you to slow down the program so the problem can be found.

Automatic Threshold Threshold Calculations

About half of the autothreshold calculator program is complete. In the previous lessons the Light and Dark values were recorded and stored in variables. In this lesson, you will use them to calculate the threshold value for the robot's environment.

Checkpoint

This is what the current program should look like.

```
2  task main()
3  {
4
5   int lightValue;
6   int darkValue;
7   int sumValue;
8   int thresholdValue;
9
10  while (SensorValue(touchSensor)==0)
11  {
12  }
13
14  lightValue=SensorValue(lightSensor);
15
16  wait1Msec(1000);
17
18  while (SensorValue(touchSensor)==0)
19  {
20  }
21
22  darkValue=SensorValue(lightSensor);
23
24 }
```

Automatic Threshold Threshold Calculations (cont.)

1. Starting at the end of the program, just before the closing brace of the task main pair, set the sumValue equal to the sum of lightValue and darkValue. The variable sumValue is now being used to store the result of lightValue plus darkValue.

```
2  task main()
3  {
4
5   int lightValue;
6   int darkValue;
7   int sumValue;
8   int thresholdValue;
9
10  while (SensorValue(touchSensor)==0)
11  {
12  }
13
14  lightValue=SensorValue(lightSensor);
15
16  wait1Msec(1000);
17
18  while (SensorValue(touchSensor)==0)
19  {
20  }
21
22  darkValue=SensorValue(lightSensor);
23
24  sumValue = lightValue + darkValue;
25
26 }
```

1. Add this code
Add lightValue and darkValue together, and store the result in the variable sumValue.

Automatic Threshold Threshold Calculations (cont.)

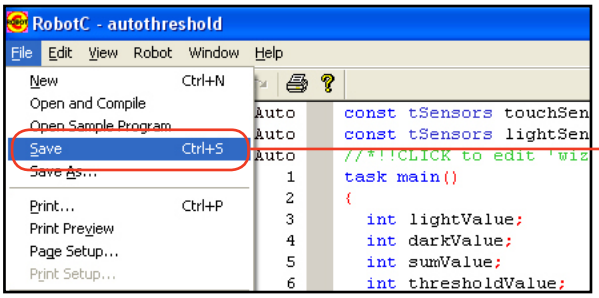
2. Set thresholdValue equal to sumValue divided by two. The variable thresholdValue now stores the threshold value calculated from the readings of light and dark surfaces.

```
2  task main()
3  {
4
5   int lightValue;
6   int darkValue;
7   int sumValue;
8   int thresholdValue;
9
10  while (SensorValue(touchSensor)==0)
11  {
12  }
13
14  lightValue=SensorValue(lightSensor);
15
16  wait1Msec(1000);
17
18  while (SensorValue(touchSensor)==0)
19  {
20  }
21
22  darkValue=SensorValue(lightSensor);
23
24  sumValue = lightValue + darkValue;
25  thresholdValue = sumValue/2;
26
27 }
```

2. Add this line of code
Divide sumValue by 2, and store the result in the variable thresholdValue.

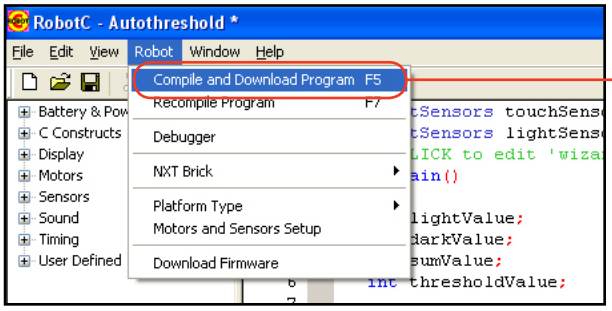
Automatic Threshold Threshold Calculations (cont.)

3. Save the Autothreshold program.



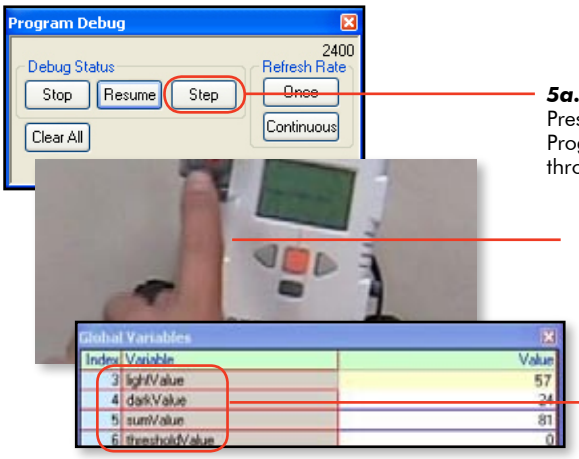
5. Save program
File > Save, to save your current autothreshold program.

4. Compile and download your program.



4. Compile and download
Robot > Compile and Download Program

5. Step through the program using the debugger, pushing the Touch Sensor at the appropriate times. Observe the variables window as sumValue stores the sum of lightValue and darkValue; and thresholdValue stores sumValue divided by two.



5a. Press Step button
Press the Step button in the Program Debug window to step through the program.

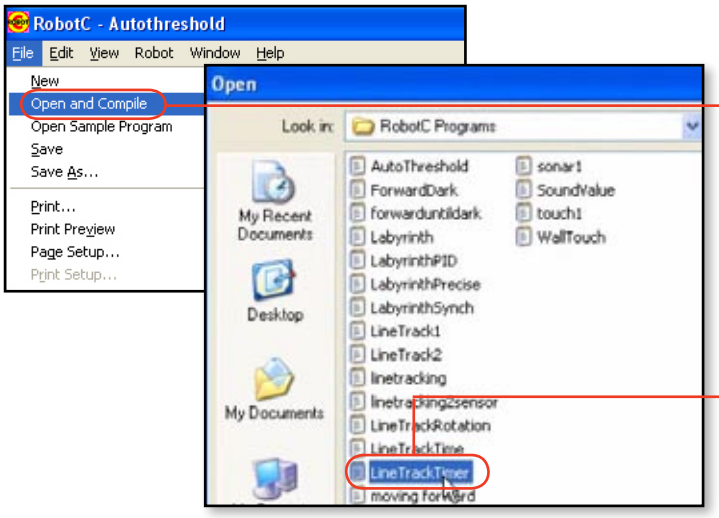
5b. Push the Touch Sensor
Push the Touch Sensor over light and dark surfaces at the appropriate times when you step through the program.

5c. Observe variables
Observe the variables window as lightValue, darkValue, sumValue and thresholdValue are calculated.

Checkpoint
The threshold is now being calculated as the average of the other two values. The debugger window shows the values of all the variables as they are collected and/or calculated.

Automatic Threshold Threshold Calculations (cont.)

6. Open your LineTrackTimer program.



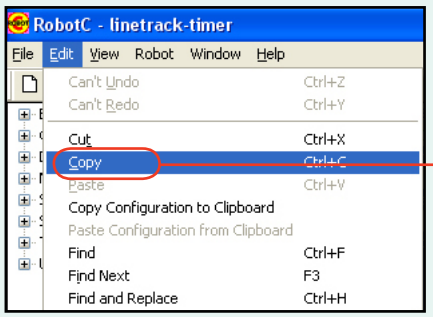
6a. Open and Compile
Select File > Open and Compile to be prompted to open a file.

6b. Select the program
Select LineTrackTimer from your previously saved programs, then double click to open it.

7. Copy the code highlighted below, from lines 5 to 29 of the LineTrackTimer program. Be careful to copy exactly this portion of the program.

```
4
5     ClearTimer(T1);
6
7     while (time1[T1] < 3000)
8     {
9
10      if (SensorValue(lightSensor) < 43)
11      {
12
13        motor[motorC]=0;
14        motor[motorB]=80;
15
16      }
17
18      else
19      {
20
21        motor[motorC]=80;
22        motor[motorB]=0;
23
24      }
25
26    }
27
28    motor[motorC]=0;
29    motor[motorB]=0;
30
```

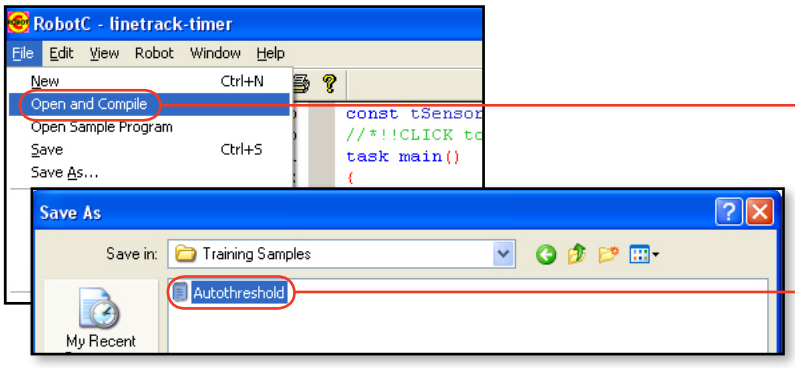
7a. Highlight code
Highlight exactly this section of code in the LineTrackTimer program.



7b. Select Copy
Select Edit > Copy to copy the highlighted code.

Automatic Threshold Threshold Calculations (cont.)

8. Reopen the autothreshold program.

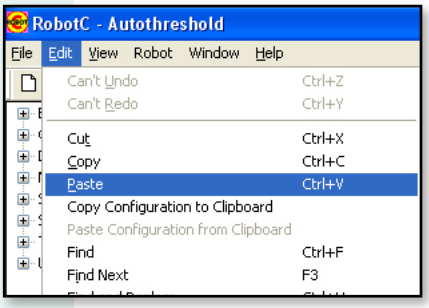


8a. Open and compile
Select File > Open and Compile to open a file.

8b. Select the program
Select the authothreshold program from the previous saved programs.

9. Paste the code you copied between "sumValue/2;" and the concluding curly brace.

```
2  task main()
3  {
4
5   int lightValue;
6   int darkValue;
7   int sumValue;
8   int thresholdValue;
9
10  while (SensorValue(touchSensor)==0)
11  {
12  }
13
14  lightValue=SensorValue(lightSensor);
15
16  wait1Msec(1000);
17
18  while (SensorValue(touchSensor)==0)
19  {
20  }
21
22  darkValue=SensorValue(lightSensor);
23
24  sumValue = lightValue + darkValue;
25  thresholdValue = sumValue/2;
26
27  |
28  }
```



9. Paste the copied code
Place the cursor right before the last curly brace and select Edit > Paste to paste the code.

Automatic Threshold Threshold Calculations (cont.)

10. Change the condition of the "borrowed" if-else statement so that instead of comparing the light sensor value to a set number, it checks it against the "thresholdValue" variable calculated in the Autothreshold program.

```
26
27  ClearTimer(T1);
28
29  while(time1[T1] < 3000)
30  {
31
32     if(SensorValue(lightSensor) < thresholdValue)
33     {
34
35        motor[motorC] = 0;
36        motor[motorB] = 80;
37
38     }
39
40     else
41     {
42
43        motor[motorC] = 80;
44        motor[motorB] = 0;
45
46     }
47
48  }
```

10. Modify code Replace the condition, which had contained a number, with the variable "thresholdValue", that holds the calculated threshold value.

Automatic Threshold Threshold Calculations (cont.)

Checkpoint

Your final program should look like the one below, and on the following page.

```
2  task main()
3  {
4
5   int lightValue;
6   int darkValue;
7   int sumValue;
8   int thresholdValue;
9
10  while (SensorValue(touchSensor)==0)
11  {
12  }
13
14  lightValue=SensorValue(lightSensor);
15
16  wait1Msec(1000);
17
18  while (SensorValue(touchSensor)==0)
19  {
20  }
21
22  darkValue=SensorValue(lightSensor);
23
24  sumValue = lightValue + darkValue;
25  thresholdValue = sumValue/2;
26
27  ClearTimer(T1);
```

Automatic Threshold Threshold Calculations (cont.)

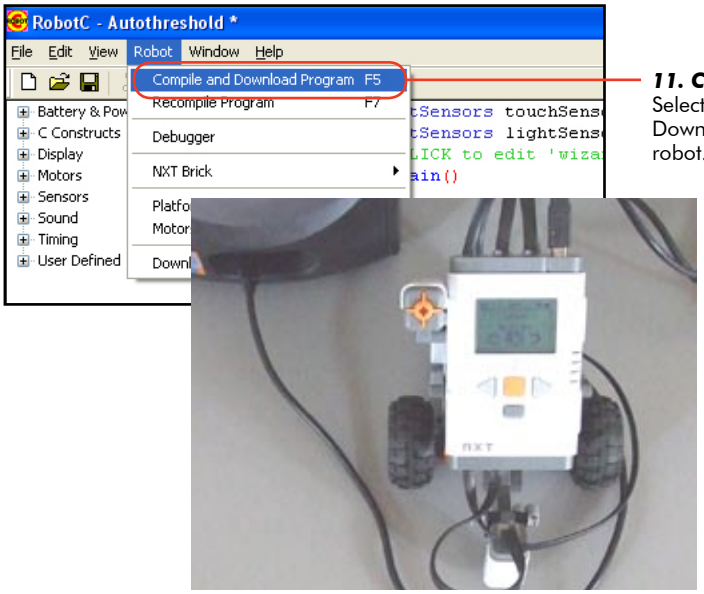
Checkpoint

Your final program should look like the one below. (continued)

```
29  while (time1[T1] < 3000)
30  {
31
32   if (SensorValue(lightSensor) < thresholdValue)
33   {
34
35     motor[motorC]=0;
36     motor[motorB]=80;
37
38   }
39
40   else
41   {
42
43     motor[motorC]=80;
44     motor[motorB]=0;
45
46   }
47
48  }
49
50   motor[motorC]=0;
51   motor[motorB]=0;
52
53 }
```

Automatic Threshold Threshold Calculations (cont.)

11. Compile and Download to your robot.



11. Compile and download
Select Robot > Compile and Download Program to run your robot.

Checkpoint

Test your program. Find a line you can track in a place where you can turn the lights on and off. Run your program and press the Touch Sensor once with the Light Sensor over light, to read the value of the light surface. Move the robot so that it is in line tracking position, with the Light Sensor over the line.

Pressing the Touch Sensor for the second time should not only read the dark value and calculate the threshold, but should also make the robot track the line for three seconds. Now turn the lights off, and run the program again. The robot should still be able to track the line!

Test program with lights on
Show the robot what the light surface looks like, then the dark one, and it should track the line for three seconds.

Test program with lights off
Change the light in the room and test the program again. The robot should again be able to track the line, demonstrating its ability to calculate a threshold in different conditions.

Automatic Threshold Threshold Calculations (cont.)

The program works, but does need to be made more user-friendly. Right now, the robot will not tell you what to do, or when. Place simple instructions in the code to solve this problem.

12. While the robot is waiting for the Touch Sensor to be pushed, program the robot to display a message telling a user to press the button over a light surface. This command makes the NXT display, on its screen, the words "Read Light Now" at position 0, 31 (that's the left edge, about halfway down). Place a similar line in the second while() loop that does the same thing, but says "Read Dark Now".

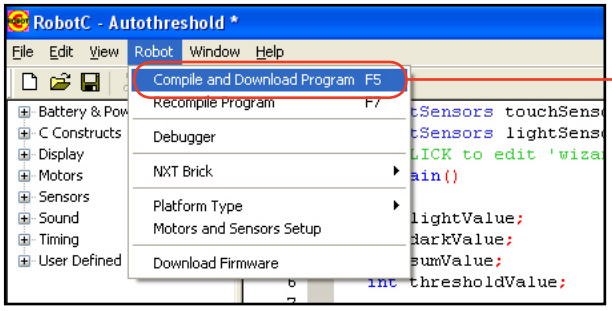
```
2  task main()
3  {
4
5   int lightValue;
6   int darkValue;
7   int sumValue;
8   int thresholdValue;
9
10  while (SensorValue(touchSensor)==0)
11  {
12    nxtDisplayStringAt(0, 31, "Read Light Now");
13  }
14
15  lightValue=SensorValue(lightSensor);
16
17 wait1Msec(1000);
18
19  while (SensorValue(touchSensor)==0)
20  {
21    nxtDisplayStringAt(0, 31, "Read Dark Now");
22  }
23
24  darkValue=SensorValue(lightSensor);
25
26  sumValue = lightValue + darkValue;
27  thresholdValue = sumValue/2;
28
```

12a. Add this code
Tells the NXT to display, on its screen, the words "Read Light Now" at the beginning of the program.

12b. Add this code
Tells the NXT to display, on its screen, the words "Read Dark Now" after the Touch Sensor has been pushed and released once.

Automatic Threshold Threshold Calculations (cont.)

13. Compile and Download your program to the robot.



13. Compile and Download
Select Robot > Compile and Download Program.

---

14. Test the program. After the program starts, the message, "Read Light Now" should appear on the NXT screen. After the Touch Sensor is pushed and released, the NXT screen should display the message, "Read Dark Now." As you did previously, place the robot so that its Light Sensor is directly over the line, and its chassis roughly parallel with the line so that it is in good position to track it. When you press the button, the threshold should be calculated, and the robot should track the line for three seconds.

14a. Read light
When the NXT displays "Read Light Now", record the light surface value.

14b. Read dark
When the NXT displays "Read Dark Now", place the robot in position to track a line.

14c. Autothreshold line track
The robot should track a line for three seconds and end the program.

Automatic Threshold Threshold Calculations (cont.)

End of Section
This is the complete code for the Automatic Threshold program.

```
2  task main()
3  {
4
5   int lightValue;
6   int darkValue;
7   int sumValue;
8   int thresholdValue;
9
10  while (SensorValue(touchSensor)==0)
11  {
12     nxtDisplayStringAt(0, 31, "Read Light Now");
13  }
14
15  lightValue=SensorValue(lightSensor);
16
17  wait1Msec(1000);
18
19  while (SensorValue(touchSensor)==0)
20  {
21     nxtDisplayStringAt(0, 31, "Read Dark Now");
22  }
23
24  darkValue=SensorValue(lightSensor);
25
26  sumValue = lightValue + darkValue;
27  thresholdValue = sumValue/2;
28
29  ClearTimer(T1);
30
31  while (time1[T1] < 3000)
32  {
```

Automatic Threshold Threshold Calculations (cont.)

```
33
34    if (SensorValue(lightSensor) < thresholdValue)
35    {
36
37      motor[motorC]=0;
38      motor[motorB]=80;
39
40    }
41
42    else
43    {
44
45      motor[motorC]=80;
46      motor[motorB]=0;
47
48    }
49
50   }
51
52    motor[motorC]=0;
53    motor[motorB]=0;
54
55  }
```

The robot now tracks a line with its own calculated threshold, and can advise users what to do, and when.