Line Counting Counting

In this lesson, we're going to investigate the meaning of this curious line of code:

```
n = n + 1;
```

Reading it in the normal mathematical sense, this is a contradiction… an impossibility. There's no number out there that can be one more than itself. Of course, that would be misreading what the line says entirely. In fact, this is not an equation, but a command in ROBOTC, and a perfectly sensible one, when you understand what it's really saying.

Line Counting Counting (cont.)

In this lesson, we're going to learn how to use a robot to count. Using code we have already discussed, along with some new stuff, we will find out what we can do with this "line counting" concept.

Let's back up a step. Where have we seen something like this before?

motor[motorC] = 50;
...sets a motor power setting to the numeric value 50.

motor[motorB] = SensorValue(soundSensor);
...sets a motor power to match the value of a sensor reading.

thresholdValue = sumValue/2;
...sets one variable to be equal to another variable divided by two.

In all of these situations, the command is to **set a value to something**. To the **left** of the equal sign, is the variable or other quantity that is set. To the **right** of the equal sign, is the value that it will be set to.

n = n + 1; is part of the same family of commands. It is clearly not meant to say that "n is equal to n plus one," but rather that the program should set n equal to n plus one. How does that work? Well, if n starts at zero, then running this command sets n to be equal to 1. Let's substitute 0 for the n on the right side and see what happens.

n starts at 0, so...
n=n+1; *becomes* **n=0+1;**

---

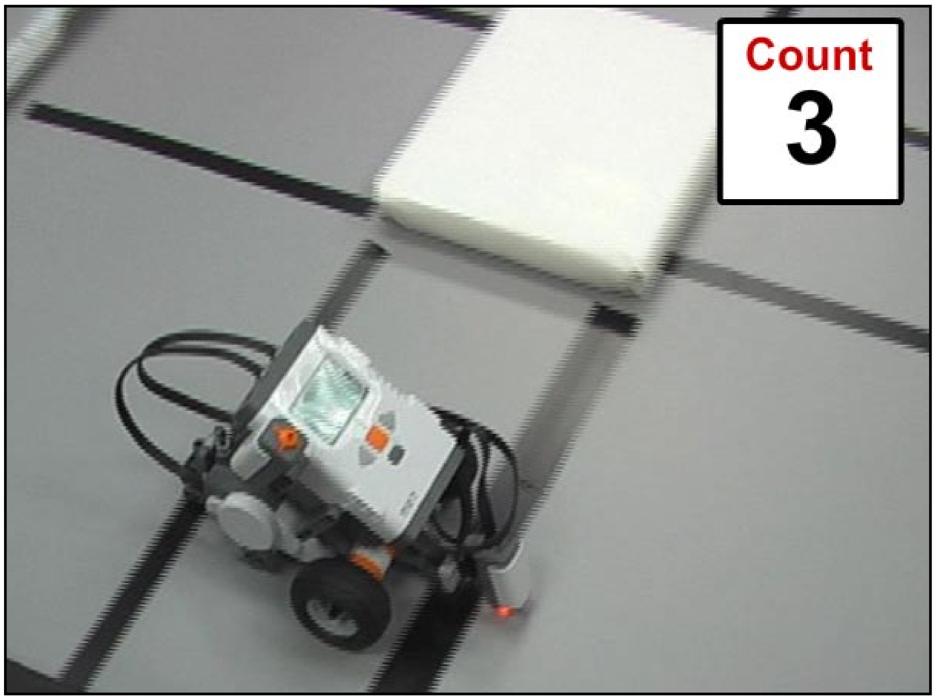
n is set to 1, so now...
n=n+1; *becomes* **n=1+1;**

... and n is set to 2, so now...
n=n+1; *becomes* **n=2+1;**

And so on! Each time you run the command `n=n+1;` the value in the variable n is increased by 1!

Line Counting Counting (cont.)

When would this be useful? Let's examine the warehouse task in more detail.



To get around the warehouse, the robot needs to count lines. Every time you reach a new line, you add one to the number of lines that you've seen. In command form, that looks like:

```
count = count + 1;
```

The new count equals the current count plus one. Commands of the form n = n + 1, like this count = count + 1, add one to the value of the variable each time the command is executed, and can be used over and over to count upwards, leaving the current count in the variable each time. By running this line once each time you spot an object that you want to count, you can keep a running tally in your program, always stored in the same variable. **Your robot can count lines!** This will come in quite handy for this project.

Line Counting Line Counting (Part 1)

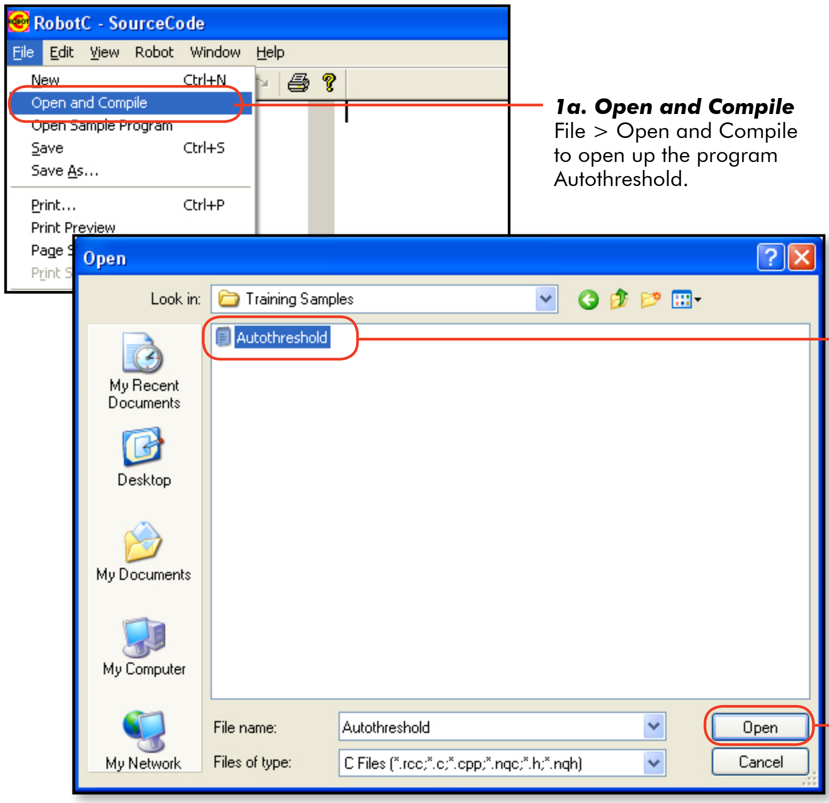
In this lesson, we're going to start teaching the robot to count lines. The eventual goal of this robot is to have it travel to a certain destination by counting special navigation markers on the floor.

We have one piece of the puzzle now, we know how to count.

What we still need to figure out are:

- When to count
- When NOT to count
- How to stop, based on the count

1. Start with your automatic threshold calculation program, the one that asks you to push the button over light and dark, and then tracks the line.



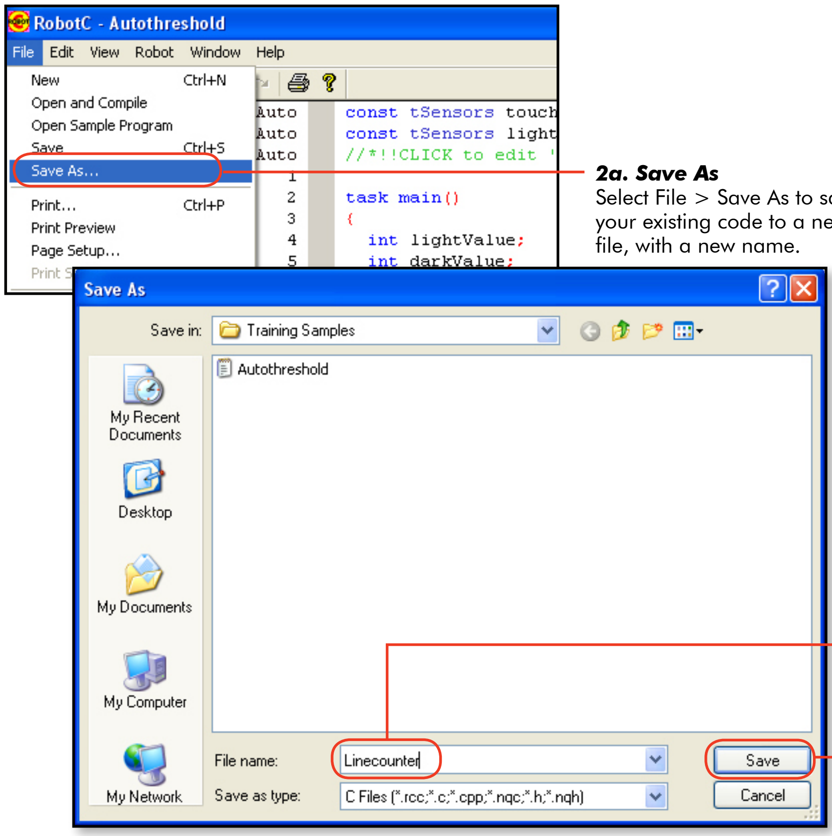
1a. Open and Compile
File > Open and Compile to open up the program Autothreshold.

1b. Find Autothreshold
Find Autothreshold and click on the program previously saved.

1c. Open Autothreshold
Press the Open button to open the program.

Line Counting Line Counting (Part 1) (cont.)

2. For this lesson, this program will be saved as, “Linecounter”.



```
const tSensors touch
const tSensors light
//*!!CLICK to edit '

task main()
{
  int lightValue;
  int darkValue;
```

2a. Save As
Select File > Save As to save your existing code to a new file, with a new name.

2b. Name the program
Name the new program file “Linecounter”.

2c. Save the program
Press the Save button to save the new program.

Line Counting Line Counting (Part 1) (cont.)

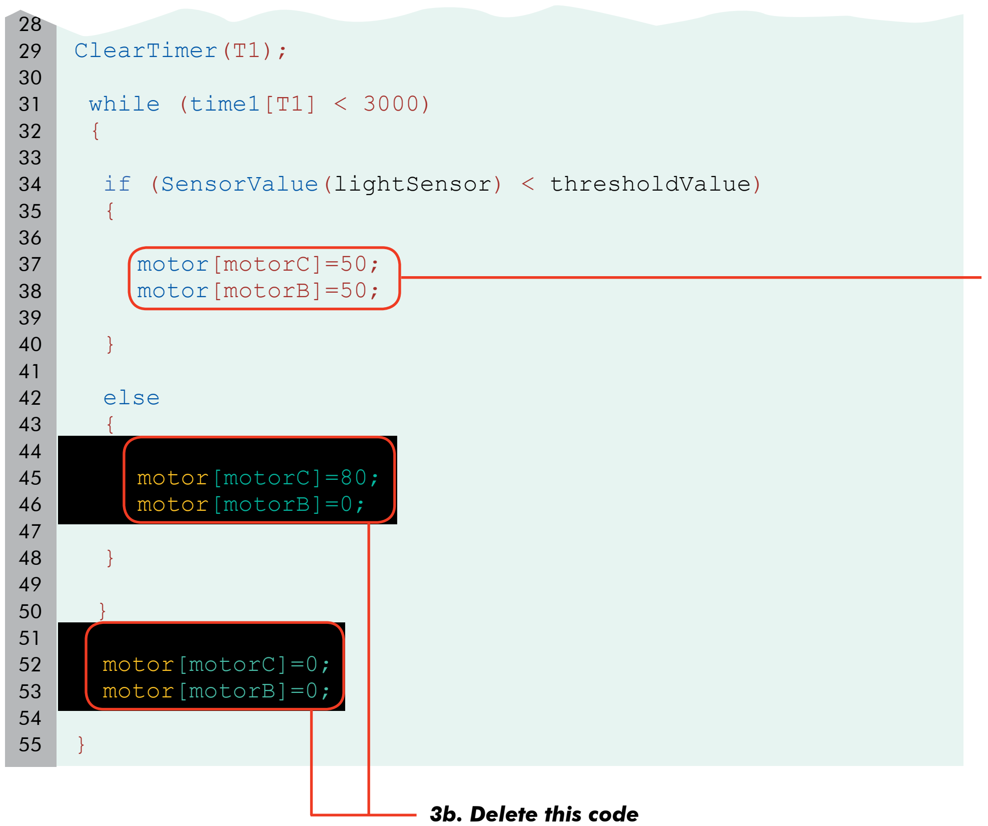
Checkpoint
This is what the program should look like before modifications.

```
task main()
{

  int lightValue;
  int darkValue;
  int sumValue;
  int thresholdValue;

  while (SensorValue(touchSensor)==0)
  {
     nxtDisplayStringAt(0, 31, "Read Light Now");
  }

  lightValue=SensorValue(lightSensor);

 wait1Msec(1000);

  while (SensorValue(touchSensor)==0)
  {
     nxtDisplayStringAt(0, 31, "Read Dark Now");
  }

  darkValue=SensorValue(lightSensor);

 sumValue = lightValue + darkValue;
 thresholdValue = sumValue/2;

 ClearTimer(T1);

 while (time1[T1] < 3000)
 {

  if (SensorValue(lightSensor) < thresholdValue)
  {

     motor[motorC]=0;
     motor[motorB]=80;

  }

  else
  {

     motor[motorC]=80;
     motor[motorB]=0;

  }

 }

  motor[motorC]=0;
  motor[motorB]=0;

}
```

Line Counting **Line Counting (Part 1)** (cont.)

The existing program already has the sensors configured, and finds a nice threshold value so we don't have to worry about either of those. The task at hand, counting lines, involves looking for light or dark just like the Line Tracker did. But unlike the line tracker, our robot only needs to move straight forward, so let's convert over the parts of the code that do steering.

3. Change the first movement portion of the Line Tracking if-else statement to just make the robot go straight instead. Remove the other movement-related commands.



```
  ClearTimer(T1);

   while (time1[T1] < 3000)
   {

    if (SensorValue(lightSensor) < thresholdValue)
    {

      motor[motorC]=50;
      motor[motorB]=50;

    }

    else
    {

      motor[motorC]=80;
      motor[motorB]=0;

    }

   }

   motor[motorC]=0;
   motor[motorB]=0;

  }
```

3a. Modify this code
Change both motorB and motorC to equal power levels. We will use 50.

3b. Delete this code
Delete both these sections of code, which steer the robot in the original line tracking program.

Line Counting Line Counting (Part 1) (cont.)

Checkpoint
This is what the program should look like after modifying the steering.

```
task main()
{

  int lightValue;
  int darkValue;
  int sumValue;
  int thresholdValue;

  while (SensorValue(touchSensor)==0)
  {
     nxtDisplayStringAt(0, 31, “Read Light Now”);
  }

  lightValue=SensorValue(lightSensor);

 wait1Msec(1000);

  while (SensorValue(touchSensor)==0)
  {
     nxtDisplayStringAt(0, 31, “Read Dark Now”);
  }

  darkValue=SensorValue(lightSensor);

 sumValue = lightValue + darkValue;
 thresholdValue = sumValue/2;

 ClearTimer(T1);

 while (time1[T1] < 3000)
 {

  if (SensorValue(lightSensor) < thresholdValue)
  {

     motor[motorC]=50;
     motor[motorB]=50;

  }

  else
  {

  }

  }

}
```

Line Counting Line Counting (Part 1) (cont.)

4. Now let's add the lines to turn on PID control for both motors to help keep the robot moving in a straight line. If you need a refresher you can review PID in the improved movement section.

```
task main()
{

  int lightValue;
  int darkValue;
  int sumValue;
  int thresholdValue;

  nMotorPIDSpeedCtrl[motorC] = mtrSpeedReg;
  nMotorPIDSpeedCtrl[motorB] = mtrSpeedReg;

  while (SensorValue(touchSensor)==0)
  {
     nxtDisplayStringAt(0, 31, "Read Light Now");
  }
```

4. Add this code
Add these two lines to turn on PID control for both motors.

5. Because we do want to look at light and dark for counting purposes, let's keep the light sensor if-else statement in place.

```
  ClearTimer(T1);

  while (time1[T1] < 3000)
  {

    if (SensorValue(lightSensor) < thresholdValue)
    {

       motor[motorC]=50;
       motor[motorB]=50;

    }

    else
    {

    }

  }

}
```

Line Counting Line Counting (Part 1) (cont.)

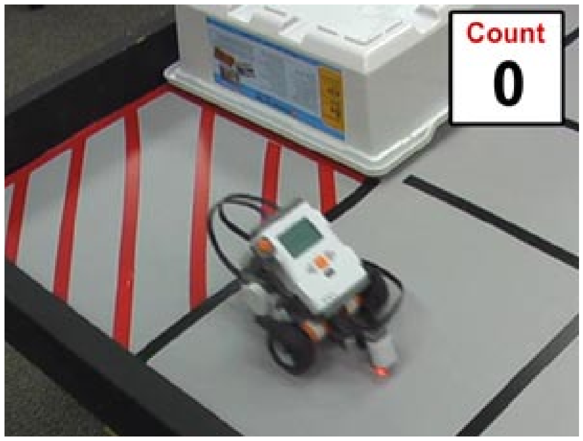
6. We're definitely going to be **counting** (lines), and we don't have a counter variable, so let's create one. It has to be an integer – it's a numeric value, and it won't have decimals – and we'll call it "countValue". After the name, add "= 0" before the semicolon. This statement declares an integer named "countValue" and assigns it an initial value of 0.

```
task main()
{

  int lightValue;
  int darkValue;
  int sumValue;
  int thresholdValue;
  int countValue = 0;

  nMotorPIDSpeedCtrl[motorC] = mtrSpeedReg;
  nMotorPIDSpeedCtrl[motorB] = mtrSpeedReg;

  while (SensorValue(touchSensor)==0)
  {
     nxtDisplayStringAt(0, 31, "Read Light Now");
  }

  lightValue=SensorValue(lightSensor);

 wait1Msec(1000);

  while (SensorValue(touchSensor)==0)
  {
     nxtDisplayStringAt(0, 31, "Read Dark Now");
  }
```

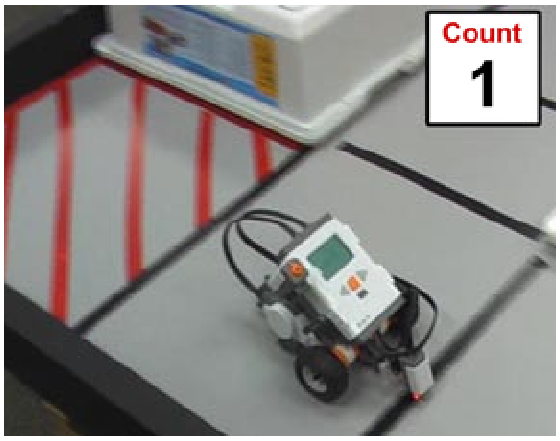
6. Add this code
Declare an integer variable named "countValue", with a value of 0. This variable will be used to count the number of lines we have passed.

Line Counting Line Counting (Part 1) (cont.)

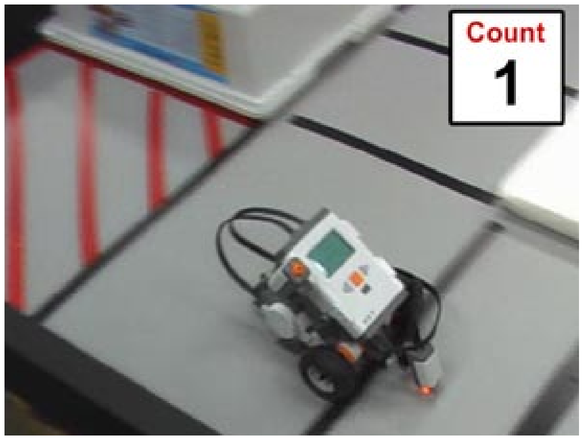
Checkpoint
Now, let's see what we should be doing in terms of counting. Suppose the robot starts running, and...



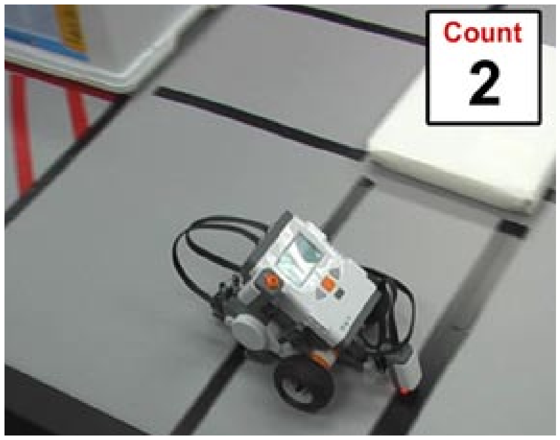
Light Sensor runs over light
Since the robot is running over light, it hasn't reached a line, and the line count should remain at zero.



Robot crosses first line
The robot has crossed its first line, so the line count should now increase to one.



Light Sensor runs over light
The robot is running over light again, and the line count remains at one.



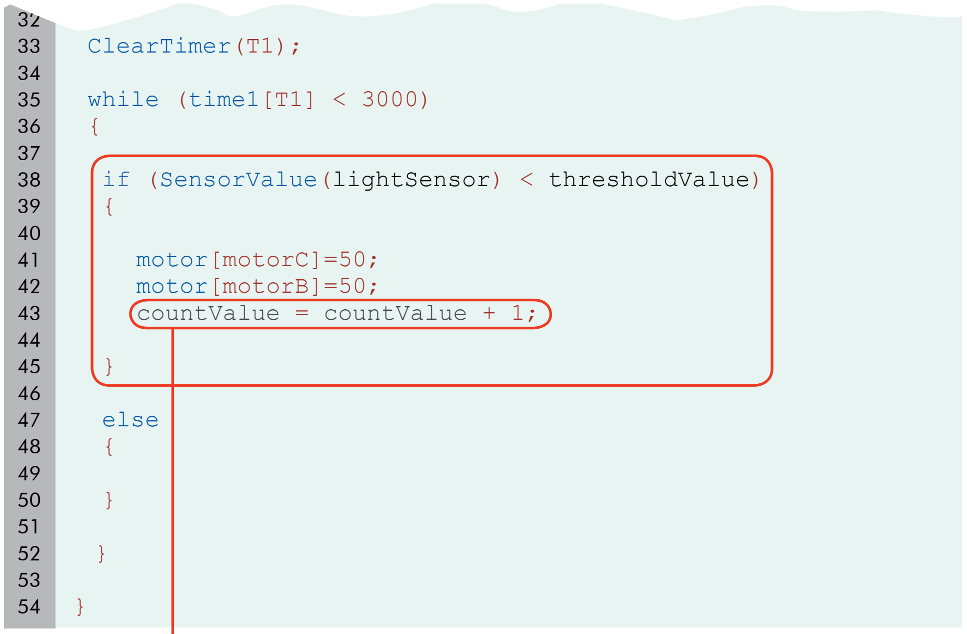
Robot crosses second line
The robot has crossed its second line, so the line count increases again to two.

And so on. It looks like **dark means a line**, and that's when we want to count.

Count when dark... "If the light sensor value is lower than the threshold, count." The adding-one code should go in the part of the code that is run **when the value of the Light Sensor is below the threshold**: inside the {body} of the if-else statement starting on line 38.

Line Counting **Line Counting (Part 1)** (cont.)

7. Put the add-one code **`countValue = countValue + 1;`** in the “seeing dark” part of the if-else statement. The “else” block of code should remain empty so that the robot does nothing when it’s over a light area, just like we want.



```
  ClearTimer(T1);

  while (time1[T1] < 3000)
  {

    if (SensorValue(lightSensor) < thresholdValue)
    {

      motor[motorC]=50;
      motor[motorB]=50;
      countValue = countValue + 1;

    }

    else
    {

    }

  }

}
```

7. Add this code
Insert the add-one code here, so that the robot adds one to the line count whenever it sees dark.

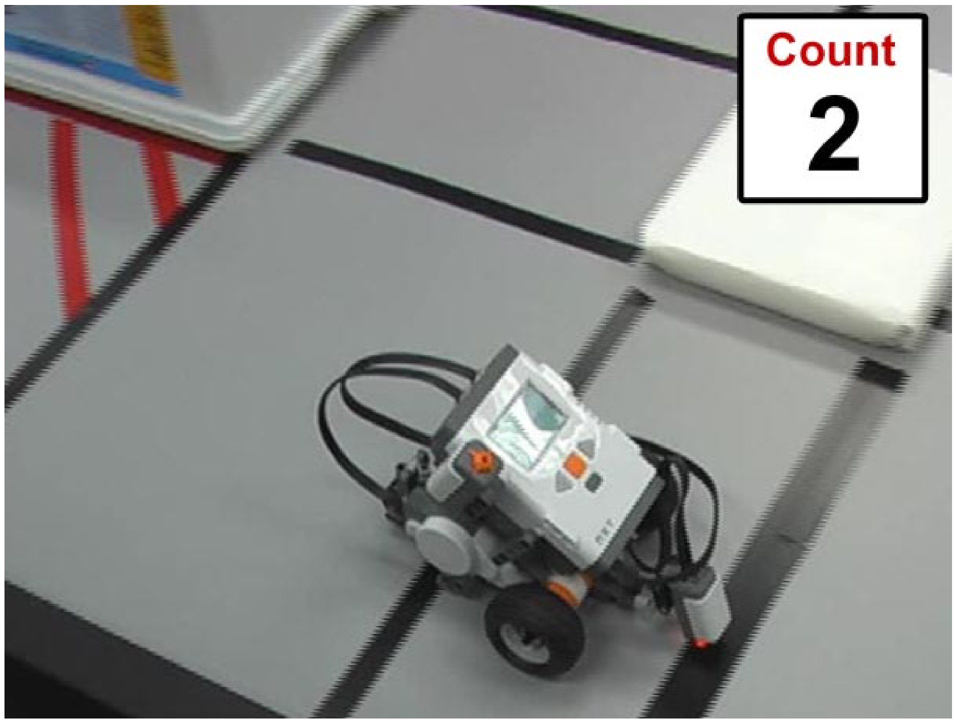
Line Counting Line Counting (Part 1) (cont.)

Checkpoint
This is what the program should look like after adding the add-one code.

```
task main()
{

  int lightValue;
  int darkValue;
  int sumValue;
  int thresholdValue;
  int countValue = 0;

  nMotorPIDSpeedCtrl[motorC] = mtrSpeedReg;
  nMotorPIDSpeedCtrl[motorB] = mtrSpeedReg;

  while (SensorValue(touchSensor)==0)
  {
     nxtDisplayStringAt(0, 31, "Read Light Now");
  }

  lightValue=SensorValue(lightSensor);

 wait1Msec(1000);

  while (SensorValue(touchSensor)==0)
  {
     nxtDisplayStringAt(0, 31, "Read Dark Now");
  }

  darkValue=SensorValue(lightSensor);

 sumValue = lightValue + darkValue;
 thresholdValue = sumValue/2;

 ClearTimer(T1);

 while (time1[T1] < 3000)
 {

  if (SensorValue(lightSensor) < thresholdValue)
  {

     motor[motorC]=50;
     motor[motorB]=50;
     countValue = countValue + 1;

  }

  else
  {

  }

 }

}
```

Line Counting Line Counting (Part 1) (cont.)

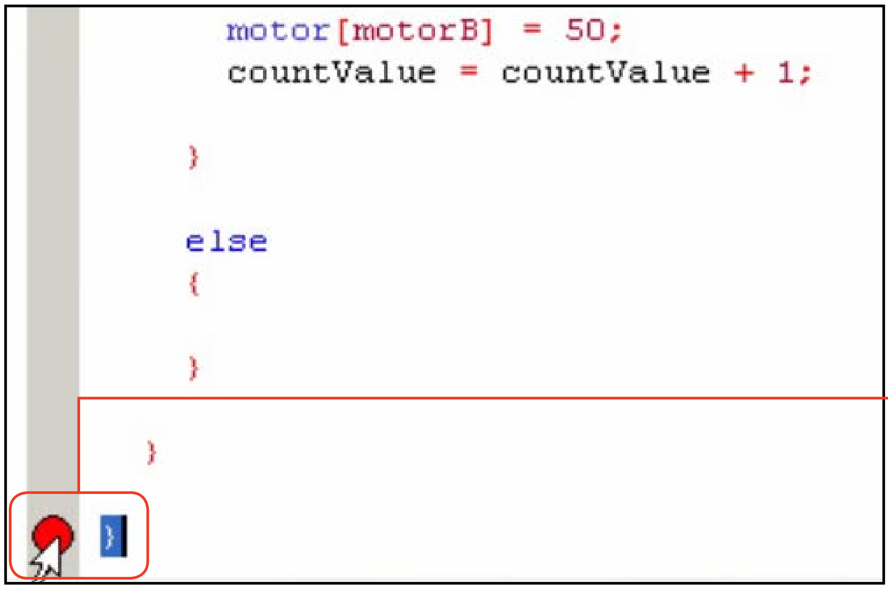
End of Section
We've written code that tells the robot when to count.
Still to come: testing and debugging the program.



Line Counting Line Counting (Part 2)

In this lesson, we're going to learn how to use a breakpoint to debug the line counting program.

1. Before we test, we need to add something that will help us determine whether the program is working or not, before it just reaches the end and erases all the data. Since we have the **debugger** on our side, there's a trick we can use. On the very last line of your program, click once in the grey bar between the code and the line number. A **red circle** will appear, marking the creation of a **breakpoint**.

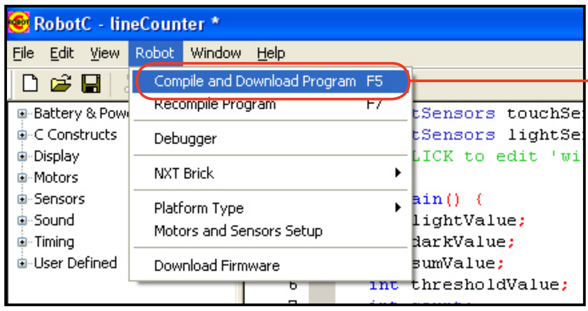


```
        motor[motorB] = 50;
        countValue = countValue + 1;

      }

      else
      {

      }

    }

}
```

1. Add breakpoint
Place your cursor next to the last curly brace, then click in the grey bar to create a breakpoint, marked by a red circle.

A breakpoint is a spot you designate in your code where the robot should automatically go into a time-stop state, as it did while using the step command. The advantage to using a "breakpoint" rather than the "step" approach allows your robot to run the program at normal speed until the program reaches the break point.

2. Compile and Download the program to the robot to begin the test!

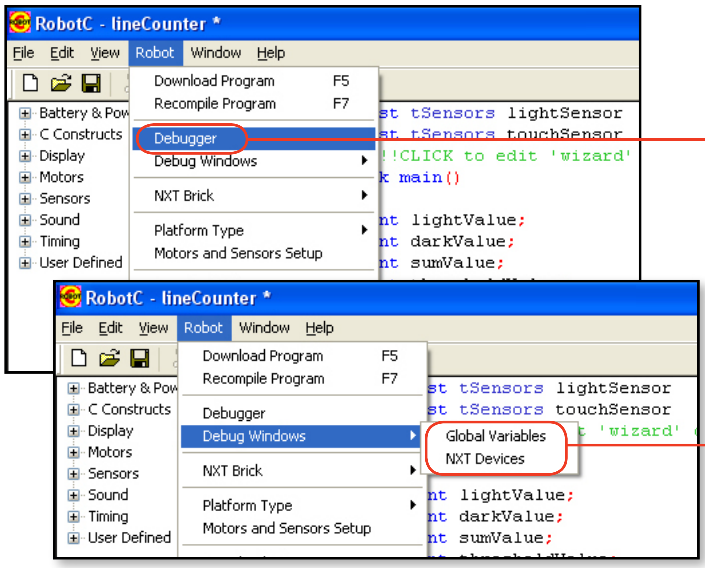


```
tSensors touchSe
tSensors lightSe
LICK to edit 'wi

ain() {
lightValue;
darkValue;
sumValue;
int thresholdValue;
```

2. Compile and Download
Robot > Compile and Download Program

Line Counting Line Counting (Part 2) (cont.)

3. Open up the Debugger, then select both the Global Variables and the NXT Devices options so both these windows are visible.

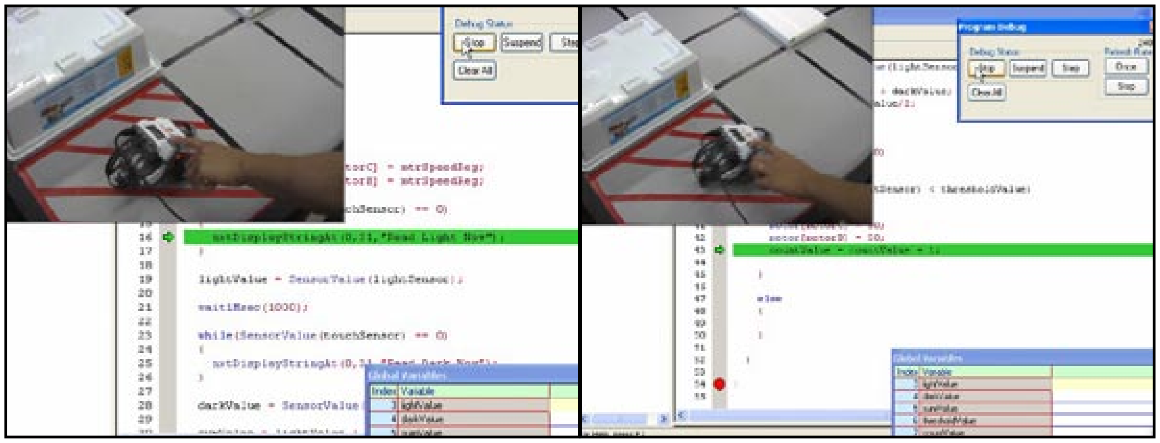


```
st tSensors lightSensor
st tSensors touchSensor
!!CLICK to edit 'wizard'
k main()

nt lightValue;
nt darkValue;
nt sumValue;
```

3a. View Debugger
Select Robot > Debugger to open up the Program Debug window.

3b. View Debugger Windows
Select Robot > Debug Windows and select **both** Global Variables and NXT Devices.

4. Start the program, then follow the prompts on the NXT screen to press the Touch Sensor to store the values of light and dark surfaces in the variables lightValue and darkValue. The second time you press the Touch Sensor, the robot should begin moving forward.

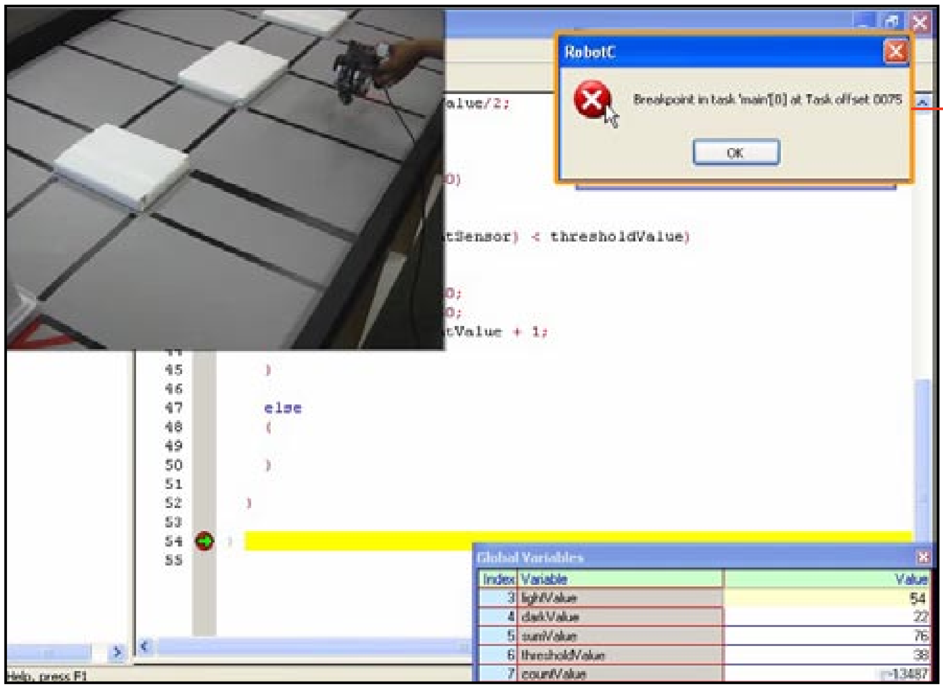


4a. Find lightValue
Push the Touch Sensor while the robot's Light Sensor is over a light area.

4b. Find darkValue
Wait 1 second, position the robot with the Light Sensor over the first line and positioned to go forward, and press the Touch Sensor again. As soon as the Touch Sensor is pressed, the robot will begin to move forward, counting lines as it goes.

Line Counting Line Counting (Part 2) (cont.)

Checkpoint
Since the line tracker we originally borrowed the code from was the Timer version, this behavior should run for a set amount of time, then hit the breakpoint. The program state freezes when it hits the breakpoint, so the motors keep running – they were running when we froze the program, so they'll keep running because there's nothing to tell them to stop.



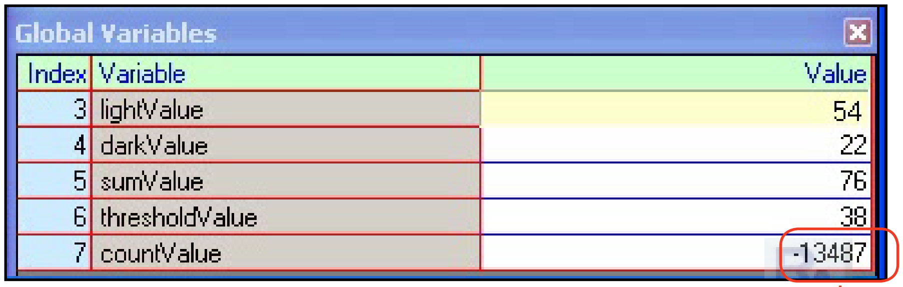
Breakpoint in task 'main'[0] at Task offset 0075

```
alue/2;

0)

tSensor) < thresholdValue)

0;
0;
tValue + 1;
      }

      else
      {

      }

    }

}
```

Index	Variable	Value
3	lightValue	54
4	darkValue	22
5	sumValue	76
6	thresholdValue	38
7	countValue	-13487

Breakpoint
This dialog tells you that your program has reached the breakpoint you set.

5. Observe the variables window, and find the value of your variable "countValue", which should be the number of lines your robot passes over. The number of lines the robot has passed appears to be... negative 13,487.

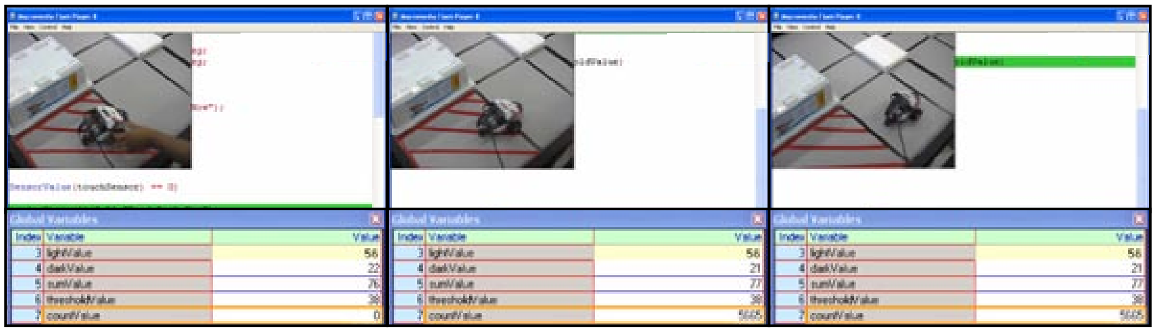


Index	Variable	Value
3	lightValue	54
4	darkValue	22
5	sumValue	76
6	thresholdValue	38
7	countValue	-13487

5. Observe "countValue"
Observe the value of the last variable, "countValue" in the Global Variables window.

Line Counting Line Counting (Part 2) (cont.)

6. Run the program again with the robot connected, this time watching to the value of the variable "countValue" in the variable window as the robot runs.


Global Variables

Index	Variable	Value
3	lightValue	56
4	darkValue	22
5	sumValue	76
6	thresholdValue	38
7	countValue	0

Global Variables

Index	Variable	Value
3	lightValue	56
4	darkValue	21
5	sumValue	77
6	thresholdValue	38
7	countValue	5665

Global Variables

Index	Variable	Value
3	lightValue	56
4	darkValue	21
5	sumValue	77
6	thresholdValue	38
7	countValue	5665

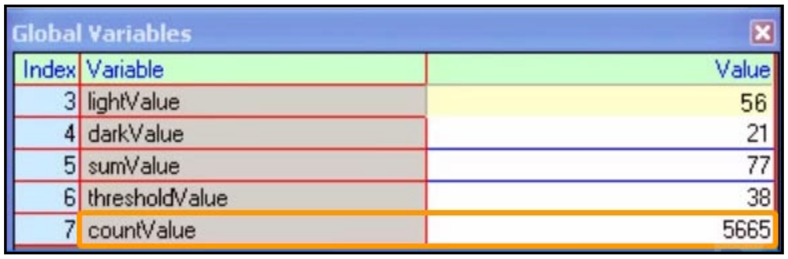
Checkpoint
Look what happens to the variable "countValue." It doesn't move when we're over light, which it shouldn't. But when you place it over the dark line and press the Touch Sensor it counts more than once – thousands of times, actually. The number gets so big that it confuses ROBOTC and wraps around into the negative numbers!

Counting more than once is the same problem we had when we were trying to detect the Touch Sensor press to read Thresholds! Remember back when the program zipped through both readings too quickly because the Touch Sensor was still held when the program reached the second check?

Flashback: Counting too fast
Back in the Automatic Thresholds section, you had another situation where the robot counted too many times, too quickly, and did not work correctly as a result.

Line Counting Line Counting (Part 2) (cont.)

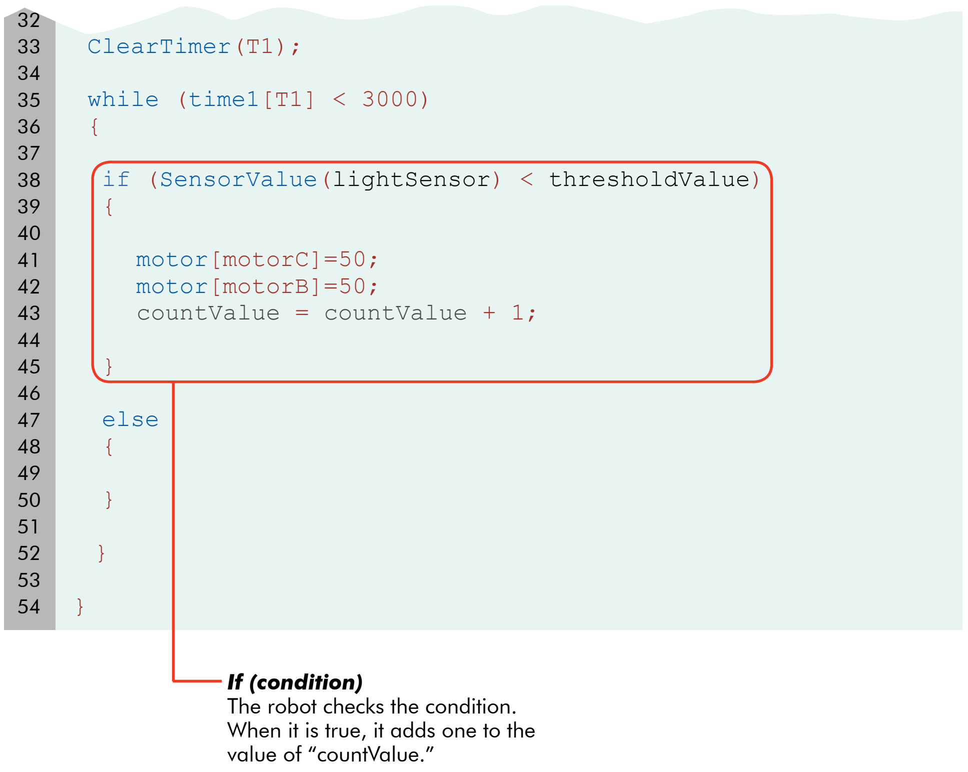
It looks like the same thing is happening here, but about 10,000 times worse. Per second.



Global Variables

Index	Variable	Value
3	lightValue	56
4	darkValue	21
5	sumValue	77
6	thresholdValue	38
7	countValue	5665

Let's look at what happens when the robot crosses a dark line. It checks the if-condition, which is true, and adds one to the variable "countValue".



```

  ClearTimer(T1);

  while (time1[T1] < 3000)
  {

    if (SensorValue(lightSensor) < thresholdValue)
    {

      motor[motorC]=50;
      motor[motorB]=50;
      countValue = countValue + 1;

    }

    else
    {

    }

   }

}
```

If (condition)
The robot checks the condition. When it is true, it adds one to the value of "countValue."

Line Counting **Line Counting (Part 2)** (cont.)

Then it skips the else, and moves back up to the top of the while() loop.

```

  ClearTimer(T1);

  while (time1[T1] < 3000)
  {

    if (SensorValue(lightSensor) < thresholdValue)
    {

      motor[motorC]=50;
      motor[motorB]=50;
      countValue = countValue + 1;

    }

    else
    {

    }

  }

}
```

Top of the while() loop
After adding one to "countValue", the robot moves back to the top of the while() loop.

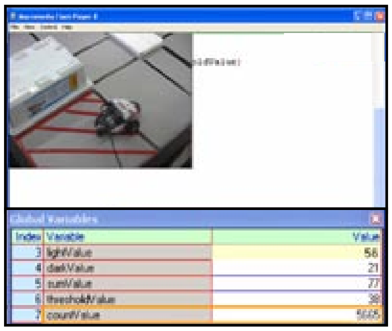
Then it does what it did before: it checks the condition... **which is still true even on this second pass because the sensor is still over the line**, and adds *another* 1 to "countValue".

```

  ClearTimer(T1);

  while (time1[T1] < 3000)
  {

    if (SensorValue(lightSensor) < thresholdValue)
    {

      motor[motorC]=50;
      motor[motorB]=50;
      countValue = countValue + 1;

    }

    else
    {

    }

  }

}
```

If (condition) again...
The robot again checks the condition, which is still true, and adds one to the value of "countValue."

Line Counting Line Counting (Part 2) (cont.)

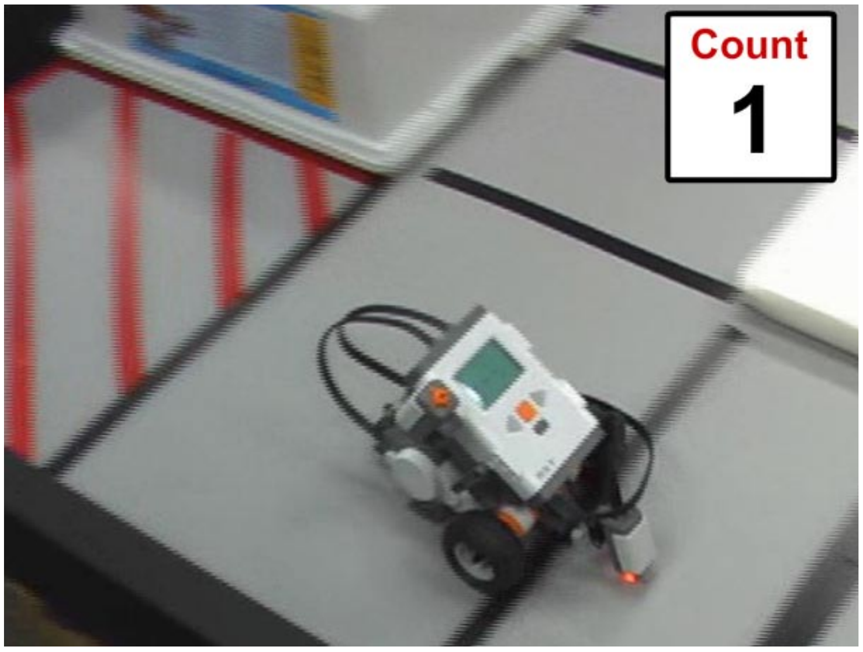
The robot keeps cycling through the while loop over and over again, and keeps adding one to "countValue" every time it does. And this is the problem: the robot is seeing, and hence **counting**, the **same black line** for what seems to be **thousands of cycles** in the amount of time it takes to pass over it.


Global Variables

Index	Variable	Value
3	lightValue	56
4	darkValue	21
5	sumValue	77
6	thresholdValue	38
7	countValue	5605

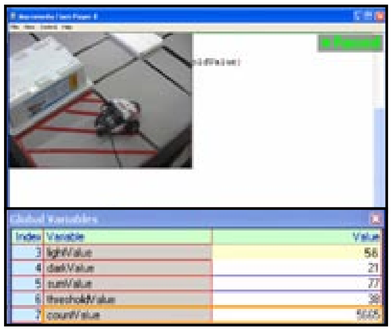
End of Section

We've found the problem: the robot counts one black line thousands of times when we only want to count the line only once. In the next lesson you will use a variable to put a stop to the double counting.



Line Counting Line Counting (Part 3)

We need to find some way to make the robot count the line only once.



Global Variables

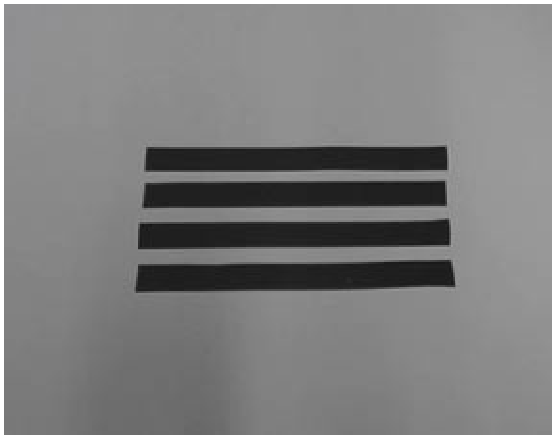
Index	Variable	Value
3	lightValue	56
4	darkValue	21
5	sumValue	77
6	thresholdValue	38
7	countValue	5665

In the Autothreshold program, we solved the problem by putting in a one second delay to allow you to take your finger off the button before the program moves to the next line of code.

```
task main()
{

  int lightValue;
  int darkValue;
  int sumValue;
  int thresholdValue;

  while (SensorValue(touchSensor)==0)
  {
     nxtDisplayStringAt(0, 31, "Read Light Now");
  }

  lightValue=SensorValue(lightSensor);

 wait1Msec(1000);

  while (SensorValue(touchSensor)==0)
  {
     nxtDisplayStringAt(0, 31, "Read Dark Now");
  }

  darkValue=SensorValue(lightSensor);

 sumValue = lightValue + darkValue;
 thresholdValue = sumValue/2;
```

Line Counting Line Counting (Part 3) (cont.)

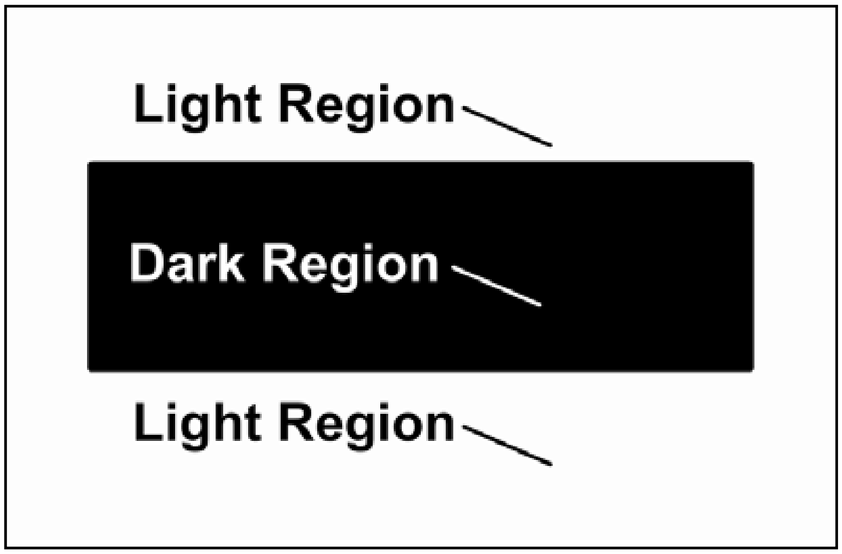
A one-second pause worked well for the button-pushing situation, but is it really appropriate here? What if the lines are close together? The robot could miss a lot of lines in that one second gap. Or what if the line is really huge? It would still count more than once.



Multiple close lines
The robot should count all of these lines separately, but could potentially drive over all of them during the "don't count again" period, and end up counting them as only one line.

Single thick line
If a line is thick enough for whatever reason, the robot may still not get past it before counting again, and it would be counted twice.

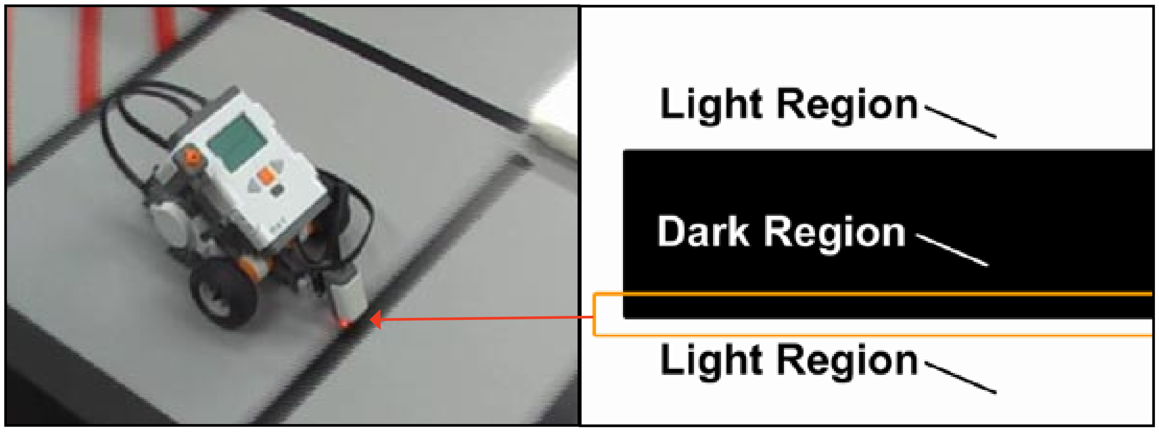
It looks like we'll have to come up with something more creative. We could look at this line as being **made up of several distinct regions**: one light region where you come in from, a dark region, and then another light region.



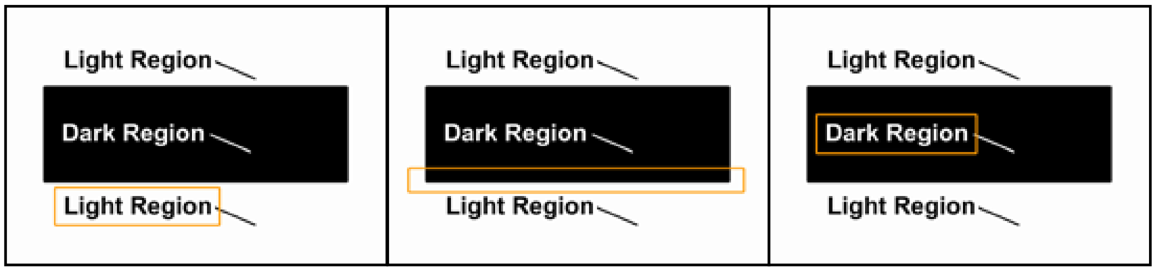
Anatomy of a Line
A line is composed of a light region, followed by a dark region, followed by another light region.

Line Counting Line Counting (Part 3) (cont.)

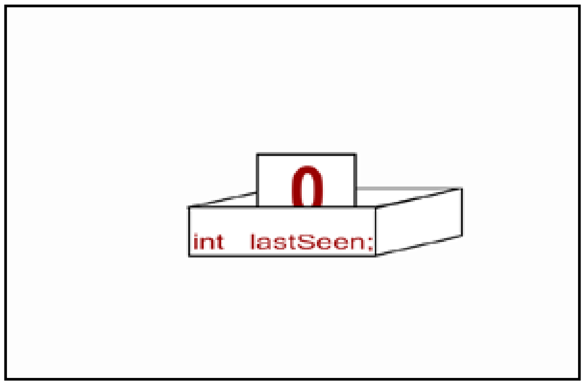
Why not let the robot count only on the **transition** from light to dark? If you look at this picture there is only one light to dark transition per line. And exactly one. So you can count every line and never count the same line twice. What we want to count is not "seeing dark", but "**seeing the transition to dark**."



What does this transition look like? The transition is when you **used to** be seeing light, as in the picture below left, and **now** are seeing dark, as in the picture below center.



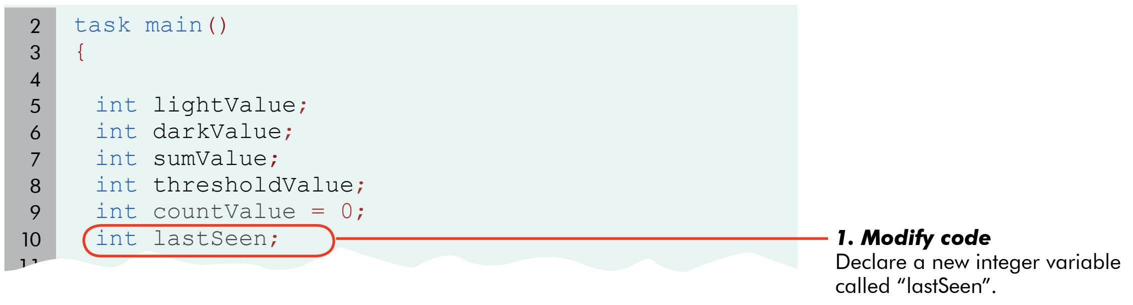
But how do we keep track of that? What we need is **a variable** to store **the color of the region that we saw last**.



Line Counting Line Counting (Part 3) (cont.)

In this lesson, we're going to learn how to make our line counter count a line only once, by counting only the transition to dark. A variable will be used to keep track of the previously seen color.

1. Declare a new integer variable, int, and call it "lastSeen".



```
task main()
{

  int lightValue;
  int darkValue;
  int sumValue;
  int thresholdValue;
  int countValue = 0;
  int lastSeen;
```

Checkpoint
We'll decide now that "lastSeen" is going to have 0 in it if the last thing it saw was dark, and a 1 in it if the last thing it saw was light. This is an arbitrary choice, but one that must be kept consistent after this point!

0 = dark
1 = light

Line Counting Line Counting (Part 3) (cont.)

2. Just before the while() loop, start the value of lastSeen at 1 (which is "light") so that we are able to count the first line

```
sumValue = lightValue + darkValue;
thresholdValue = sumValue/2;

ClearTimer(T1);
lastSeen = 1;

while (time1[T1] < 3000)
{

 if (SensorValue(lightSensor) < thresholdValue)
 {

    motor[motorC]=50;
    motor[motorB]=50;
    countValue = countValue + 1;

 }
```

2. Modify code
Assign the variable "lastSeen" the value of 1 just before the while() loop.

3. The rest of the program needs to make sure this variable stays up to date. In the block of code corresponding to the "**dark**" area of the if-else loop, add the line "lastSeen = 0;" And in the block for the "**light**" area (inside the else block), add the line "lastSeen =1;".

```
sumValue = lightValue + darkValue;
thresholdValue = sumValue/2;

ClearTimer(T1);
lastSeen = 1;

while (time1[T1] < 3000)
{

 if (SensorValue(lightSensor) < thresholdValue)
 {

    motor[motorC]=50;
    motor[motorB]=50;
    countValue = countValue + 1;
    lastSeen = 0;

 }

 else
 {
    lastSeen = 1;
 }

 }

}
```

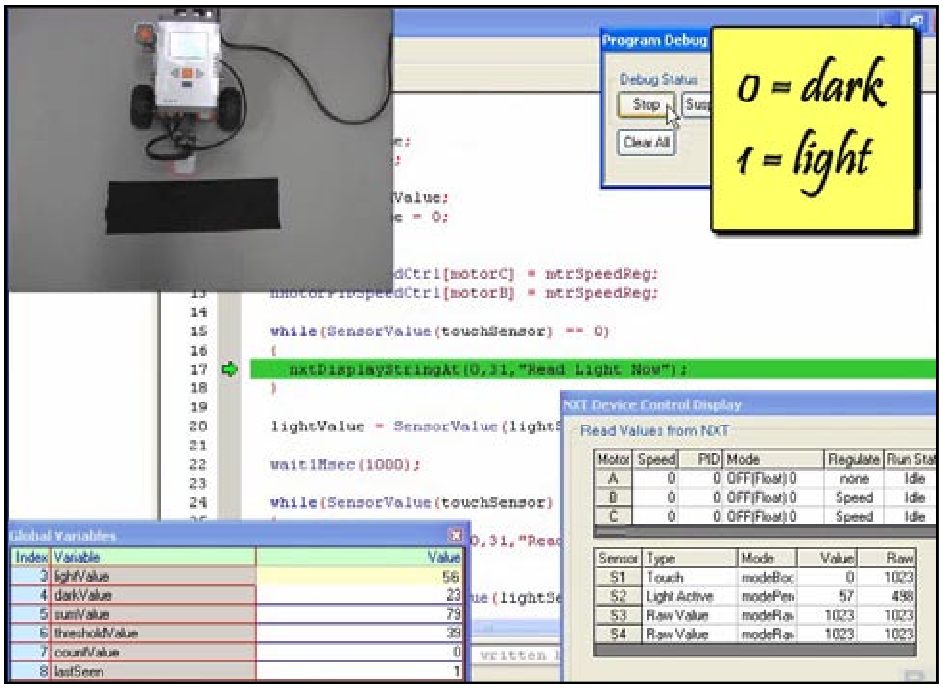
2. Modify code
Assign the variable "lastSeen" the value of 1 just before the while() loop.

3a. Modify code
Assign the variable "lastSeen" the value of 0 in the "dark" area of the if-else structure.

3b. Modify code
Assign the variable "lastSeen" the value of 1 in the "light" area of the if-else structure.

Line Counting Line Counting (Part 3) (cont.)

4. Save, compile and download the program to the robot to see how we are doing. Bring up the **Debugger, the Global Variables Window and the NXT Device Control Display**.



```
                             e:
                             ;

                             Value;
                             e = 0;

                             dCtrl[motorC] = mtrSpeedReg;
    nMotorPIDSpeedCtrl[motorB] = mtrSpeedReg;

    while(SensorValue(touchSensor) == 0)
    {
      nxtDisplayStringAt(0,31,"Read Light Now");
    }

    lightValue = SensorValue(lightS

    wait1Msec(1000);

    while(SensorValue(touchSensor)
```

Motor	Speed	PID	Mode	Regulate	Run Stat
A	0	0	OFF(Float) 0	none	Idle
B	0	0	OFF(Float) 0	Speed	Idle
C	0	0	OFF(Float) 0	Speed	Idle

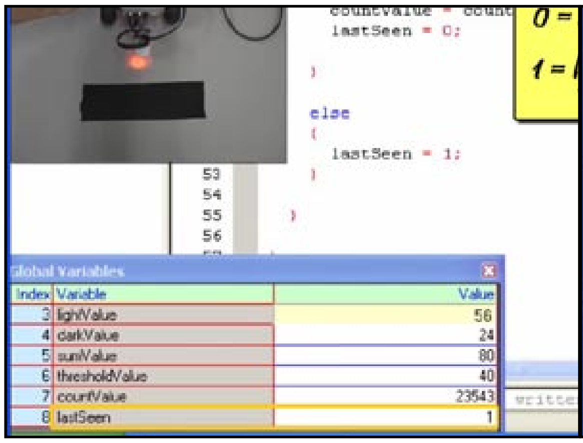
Sensor	Type	Mode	Value	Raw
S1	Touch	modeBoc	0	1023
S2	Light Active	modePer	57	498
S3	Raw Value	modeRa	1023	1023
S4	Raw Value	modeRa	1023	1023

Index	Variable	Value
3	lightValue	56
4	darkValue	23
5	sumValue	79
6	thresholdValue	39
7	countValue	0
8	lastSeen	1

Debuggers
Compile and Download the program, then make sure the debugger windows are still open.

Checkpoint

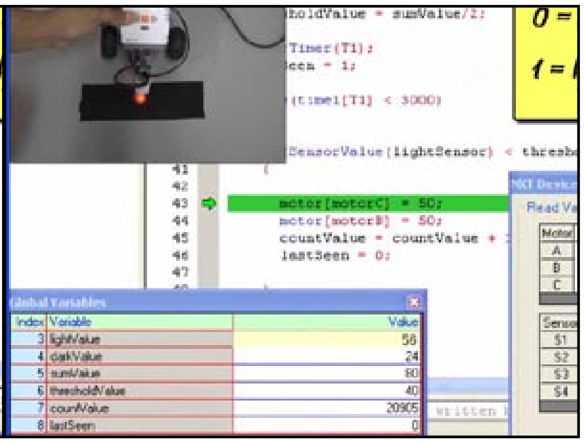
Run the robot, but **pick it up and hold it over either the dark or light areas**. Whenever it's over the dark area, "lastSeen" should be 0. Whenever it's over the light area, "lastSeen" should be 1.



```
      countValue = count
      lastSeen = 0;
    }
    else
    {
      lastSeen = 1;
    }
  }
```

Index	Variable	Value
3	lightValue	56
4	darkValue	24
5	sumValue	80
6	thresholdValue	40
7	countValue	23543
8	lastSeen	1

Robot held over Light
When the Light Sensor is held over the light area, the lastSeen variable in the Global Variables window should be 1.



```
holdValue = sumValue/2;

Timer(T1);
een = 1;

(time1[T1] < 3000)

SensorValue(lightSensor) < thresh
  {
      motor[motorC] = 50;
      motor[motorB] = 50;
      countValue = countValue +
      lastSeen = 0;
```

Index	Variable	Value
3	lightValue	56
4	darkValue	24
5	sumValue	80
6	thresholdValue	40
7	countValue	20905
8	lastSeen	0

Robot held over Dark
When the Light Sensor is held over the dark line, the lastSeen variable in the Global Variables window should be 0.

Line Counting Line Counting (Part 3) (cont.)

5. A light-to-dark transition will be marked by a "**last seen**" color of light, and a "**currently seeing**" color of dark. Therefore, the counting must be in the seeing-dark portion of the code, but should **also check** that the "lastSeen" value is light, a value of 1.

Create an if-else structure (beginning with the line "if (lastSeen == 1)" **around** the existing code. The "else" portion is actually optional, and is left out here to save space.

```
  sumValue = lightValue + darkValue;
  thresholdValue = sumValue/2;

  ClearTimer(T1);
  lastSeen = 1;

  while (time1[T1] < 3000)
  {

   if (SensorValue(lightSensor) < thresholdValue)
   {

     motor[motorC]=50;
     motor[motorB]=50;

     if (lastSeen == 1)
     {
       countValue = countValue + 1;
       lastSeen = 0;
     }

   }

   else
   {
     lastSeen = 1;
   }

  }

}
```

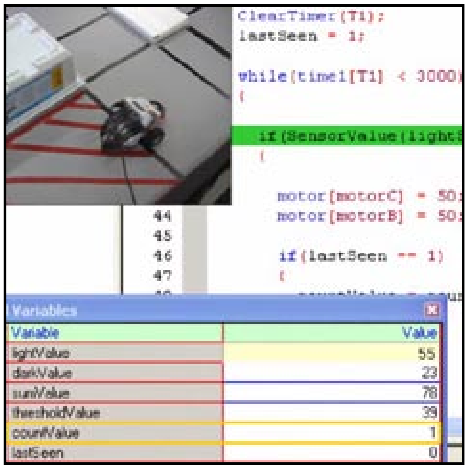
5. Modify code
Create an if structure which checks if the variable "lastSeen" is equal to 1. The add-one code should become its {body}.

Line Counting Line Counting (Part 3) (cont.)

6. Save, download, and run. Watch the count variable in your program as the robot travels over lines, and with any luck, the count will match the number of lines!

Success
The robot now travels for 3 seconds, counting appropriately only when it has reached a new line (a light-to-dark transition).

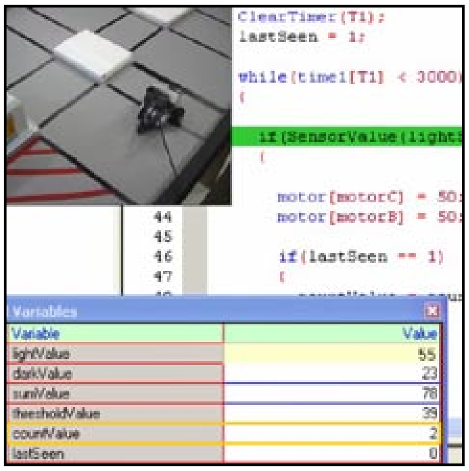
Observe the value of "countValue" in the debug window for each position of the robot shown below.



```
ClearTimer(T1);
lastSeen = 1;

while(time1[T1] < 3000
{

  if(SensorValue(light
  {

    motor[motorC] = 50;
    motor[motorB] = 50;

    if(lastSeen == 1)
    {
```

Variables

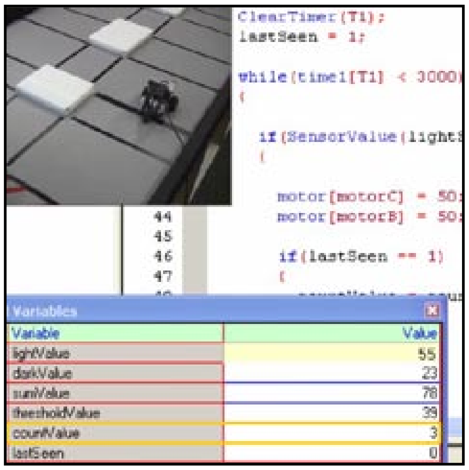
Variable	Value
lightValue	55
darkValue	23
sumValue	78
thresholdValue	39
countValue	1
lastSeen	0



```
ClearTimer(T1);
lastSeen = 1;

while(time1[T1] < 3000
{

  if(SensorValue(light
  {

    motor[motorC] = 50;
    motor[motorB] = 50;

    if(lastSeen == 1)
    {
```

Variables

Variable	Value
lightValue	55
darkValue	23
sumValue	78
thresholdValue	39
countValue	2
lastSeen	0



```
ClearTimer(T1);
lastSeen = 1;

while(time1[T1] < 3000
{

  if(SensorValue(light
  {

    motor[motorC] = 50;
    motor[motorB] = 50;

    if(lastSeen == 1)
    {
```

Variables

Variable	Value
lightValue	55
darkValue	23
sumValue	78
thresholdValue	39
countValue	3
lastSeen	0

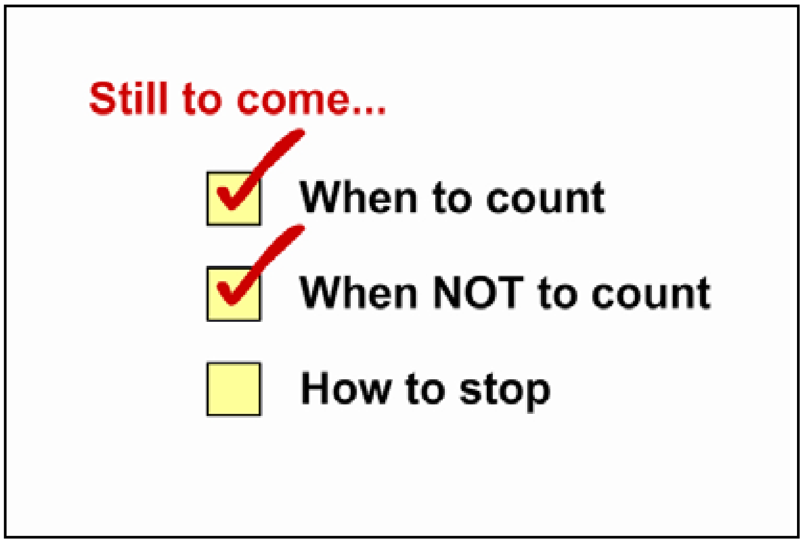
Line Counting **Line Counting (Part 3)** (cont.)

Checkpoint. This is what the program should look like after all your modifications.

```
task main()
{

  int lightValue;
  int darkValue;
  int sumValue;
  int thresholdValue;
  int countValue = 0;
  int lastSeen;

  nMotorPIDSpeedCtrl[motorC] = mtrSpeedReg;
  nMotorPIDSpeedCtrl[motorB] = mtrSpeedReg;

  while (SensorValue(touchSensor)==0)
  {
     nxtDisplayStringAt(0, 31, "Read Light Now");
  }

  lightValue=SensorValue(lightSensor);

  wait1Msec(1000);

  while (SensorValue(touchSensor)==0)
  {
     nxtDisplayStringAt(0, 31, "Read Dark Now");
  }

  darkValue=SensorValue(lightSensor);

 sumValue = lightValue + darkValue;
 thresholdValue = sumValue/2;

 ClearTimer(T1);
 lastSeen = 1;

 while (time1[T1] < 3000)
 {

  if (SensorValue(lightSensor) < thresholdValue)
  {

     motor[motorC]=50;
     motor[motorB]=50;

     if (lastSeen == 1)
     {
        countValue = countValue + 1;
        lastSeen = 0;
     }

  }

  else
  {
     lastSeen = 1;
  }

 }

}
```

Line Counting Line Counting (Part 3) (cont.)

End of Section
We've covered the first two items we need for our line counting program.
In the next section, we'll learn how to stop.


Still to come...

- ✓ When to count
- ✓ When NOT to count
- ☐ How to stop

Line Counting Line Counting (Part 4)

In this lesson, we're going to learn how to make our line counter stop when it has passed over a specific number of lines, instead of stopping after a specific amount of time has elapsed.

The Line Tracking code we originally borrowed was the Timer version, which works by running while the elapsed time value is **less than the time limit**. Right now it loops until 3000 milliseconds have passed. What we really want is for this robot to move until it has **passed 7 lines**.

```
  ClearTimer(T1);
 lastSeen = 1;

 while (time1[T1] < 3000)
 {

  if (SensorValue(lightSensor) < thresholdValue)
  {

    motor[motorC]=50;
    motor[motorB]=50;

    if (lastSeen == 1)
    {
      countValue = countValue + 1;
      lastSeen = 0;
    }

  }

  else
  {
    lastSeen = 1;
  }

 }

}
```

While loop
The (condition) in the while loop determines whether the move-and-count behavior continues, or whether the program moves on to the next behavior.

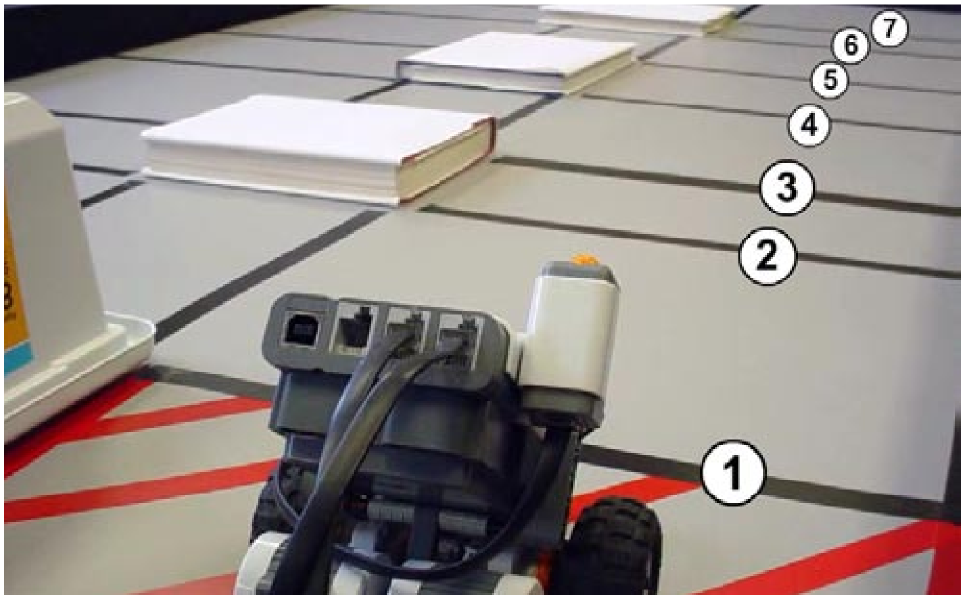
Line Counting **Line Counting (Part 4)** (cont.)

1. Replace the “Timer < 3000” condition with “countValue < 7”.

```

 ClearTimer(T1);
lastSeen = 1;

while (countValue < 7)
{

  if (SensorValue(lightSensor) < thresholdValue)
  {

     motor[motorC]=50;
     motor[motorB]=50;

     if (lastSeen == 1)
     {
        countValue = countValue + 1;
        lastSeen = 0;
     }

  }

  else
  {
     lastSeen = 1;
  }

```

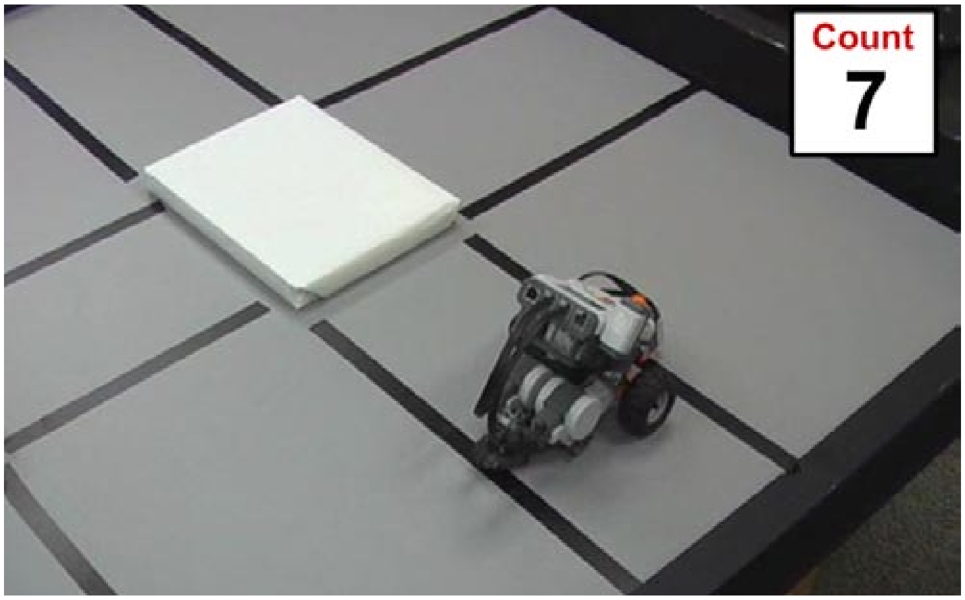
1. Modify code
Change the condition the while() loop checks from “Timer < 3000” to “countValue < 7”.

2. Make sure your table has at least seven lines on it.



Line Counting Line Counting (Part 4) (cont.)

3. Save, download and run.



Line Counting Line Counting (Part 4) (cont.)

End of Section If this is what your program looks like, you've finished the line counting program!

```
task main()
{

  int lightValue;
  int darkValue;
  int sumValue;
  int thresholdValue;
  int countValue = 0;
  int lastSeen;

  nMotorPIDSpeedCtrl[motorC] = mtrSpeedReg;
  nMotorPIDSpeedCtrl[motorB] = mtrSpeedReg;

  while (SensorValue(touchSensor)==0)
  {
     nxtDisplayStringAt(0, 31, "Read Light Now");
  }

  lightValue=SensorValue(lightSensor);

  wait1Msec(1000);

  while (SensorValue(touchSensor)==0)
  {
     nxtDisplayStringAt(0, 31, "Read Dark Now");
  }

  darkValue=SensorValue(lightSensor);

 sumValue = lightValue + darkValue;
 thresholdValue = sumValue/2;

 ClearTimer(T1);
 lastSeen = 1;

 while (countValue < 7)
 {

  if (SensorValue(lightSensor) < thresholdValue)
  {

     motor[motorC]=50;
     motor[motorB]=50;

     if (lastSeen == 1)
     {
        countValue = countValue + 1;
        lastSeen = 0;
     }

  }

  else
  {
     lastSeen = 1;
  }

 }

}
```