An introduction to Robotics with LEGO® MINDSTORMS (XIV)

Multiple calibrations of light sensors

By Koldo Olaskoaga

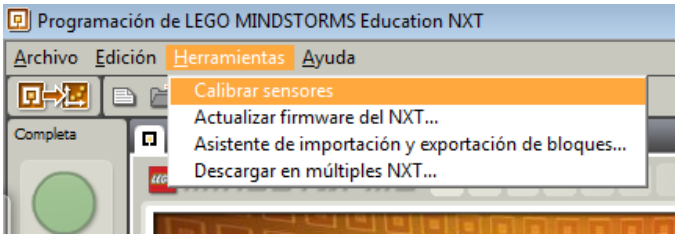
A sensor (also called detector) is a converter that measures a physical quantity and converts it into a signal which can be read by an observer or by an (today mostly electronic) instrument[1]. These signals can correspond to light intensity, distance, acceleration, inclination, temperature, sound pressure...

In the real world, the environmental conditions can affect the readings of sensors and sometimes this can cause real headaches. In the case of light sensors, the light conditions can cause a robot that performed its task perfectly in the space where it was created, to be incapable of doing the same in another place: what the sensor saw as white in one place it sees as grey in another.

So, in the real working space it is necessary to tell the robot what is white and what is black, i.e. to calibrate the sensor.

Basic calibration of a light sensor

There are two standard methods of calibrating a light sensor in NXT-G. The first is by means of the **Calibrate sensors** option that can be found in the **Tools** menu of NXT-G and the second is to create a small program with the **Calibrate** block from the **Advanced** menu.



These tow methods allow for a single calibration for all light sensors attached to the NXT, so if we wish to use different calibrations for different sensors or two different calibrations for a single sensor we'll need to use a different strategy; we'll do so using the possibility the NXT has to store data in a file and read it later.

Before we get into this let's have a look at how files can be used.

Use of files in NXT-G

NXT-G allows you to store information in text files. These files can be read from the same or from another program and be transferred to the computer to analyse the data.

This data can be text or numbers, for example, the readings of a sensor...

In general terms, the steps to do this are as follows:

1. Write to the file
2. Close the file
3. Read from the file.

In each case the same block is used: **File access**, configured according to the need for each case.

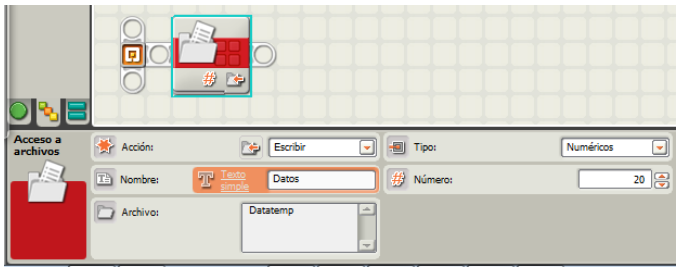
Creating a file

The first step is to create a file. Since we are likely to run the program more than once, the first thing to do is to delete the file that was created previously, since otherwise the information will be added to the existing file.

In the picture you can see a **File access** block that has been configured to delete a file named **MiArchivo**.

Once the file has been eliminated we can create the file again and start to write to it. There is no specific mode for creating a file: the file is created the first time you write to it.

the data the file can hold can be text or numbers. The block in the following image writes a numeric value to a file named **Datos**.

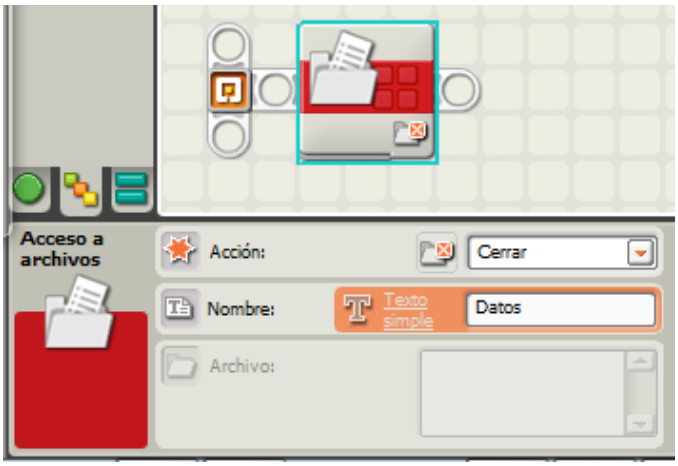


[1] http://es.wikipedia.org/wiki/Sensor

We'll use this file as many times as necessary to store new values in.

Closing the file

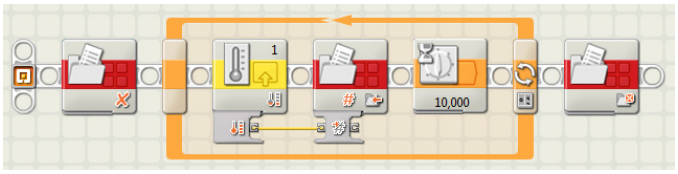
Before reading the data you need to close the file, configuring the File access block as you can see in the following image.



Example

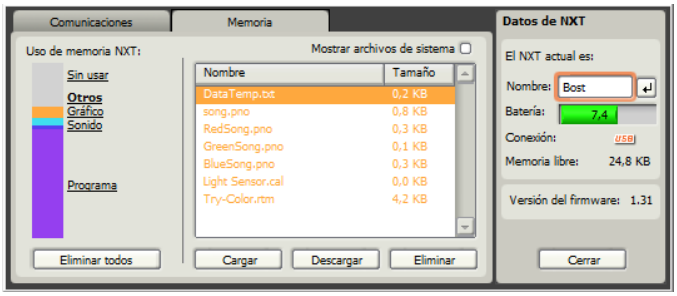
Let's see how this technique can be applied with an example. The ideas is to register the variation of the temperature in time: during a heating or cooling process, room temperature... To do this we'll use the LEGO MINDSTORMS NXT temperature sensor.
After registering the values we'll transfer the file to the computer and convert it into a graph that makes interpreting the data easier.
This is something that can be dode more easily with the data logging mode of NXT-G Edu, but not with the Retail version. On the other hand, the use of a file allows for a more flexible data collection.

In the example, for which you can see the code in the following image, the temperature is measured every second during 100 seconds. This means there will be 100 values registered.



This program first deletes the file (named DatosTemp in this case). Then it opens a loop (Loop block configured with a counter) that will repeat 100 times. The loop reads the temperature sensor that is connected to port 1, writes the value to the file and waits 1 second before repeating the operation. After 100 times it closes the file and the program exits.
To transfer the file to a computer first you need to open the NXT window, clicking on the top left button in the controller.

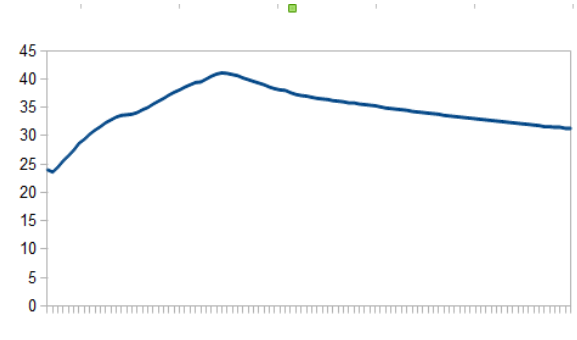
In the NXT window select the **Memory** tab and inside that **Others** and you'll be able to see the files on your NXT, including the one created in this example.



If you click on Load you can transfer the file (DataTemp.txt) to the folder of your choice on your computer.

After this you can open the file in a spreadsheet, in my case LibreOffice Calc. AFter opening the file you will observe that NXT-G uses a period to separate decimals, but the SPanish version of LibreOffice Calc will not interpret those values as numerical, so we need substitute the periods with commas (Find and replace). Using the Insert Graph tool you can get a graphical representation of the temperatures as can be seen in the following image.

1	24
2	23,6
3	24,5
4	25,6
5	26,5
6	27,5
7	28,7
8	29,4
9	30,3
10	31
11	31,6
12	32,3
13	32,8
14	33,3
15	33,6
16	33,7

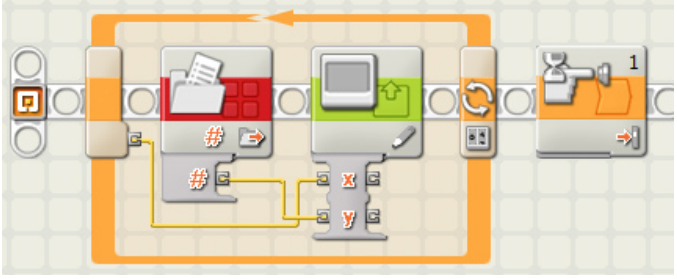


Reading the data

Now let's see how you can read data from a previously stored file. Before doing so I should point out that reading is done sequentially, so data is read in the order it was written: you cannot read the third value before reading the first and second.

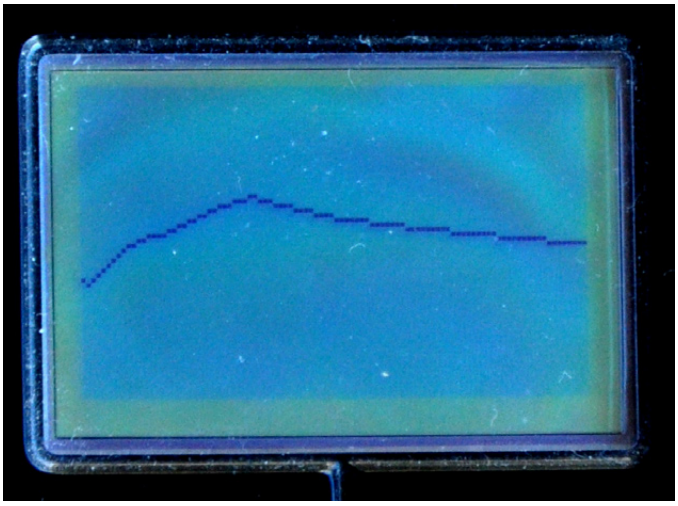
You can read the values from a file in the same program it was created in or on any other program. In this case we'll do so from a second program; we will read the values that were stored in DataTemp and represent them on the screen of the NXT.

The program is as follows:



The program consists of a loop that is repeated as many times as there is data to read. To ensure the graph stays on the screen for as long as needed, a **Wait** block has been added, configured for a touch sensor. In fact none needs to be connected, as the program will not end until the grey NXT button is pushed.

The **View** block allows you to draw dots at any desired location, based on the x and y coordinates of the point. In this case the “x” will be the number of the order of the measurements and the “y” the temperature. The **File access** block is configured in read mode and the Number exit is connected to the “y” entry of the **View** block. The value for “x” will be obtained from the counter of the loop.
In this way we’ll get the same graph we’ve seen earlier on the computer, displayed on the screen of the NXT.



Calibrating more than one sensor

Now that we know how to use files in NXT-G let’s see another way of calibrating a sensor.

In the calibration modes I described at the beginning of the article, the program automatically creates a file and reads it each time the light sensor is used without us noticing it. What we will do now is create a file with the necessary data for calibration and use it each time we need it.

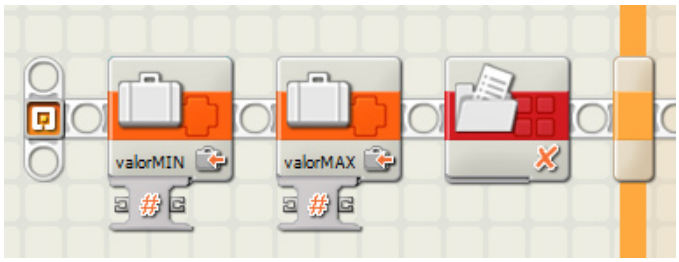
Creating a calibration file

To create the calibration file we need to read and save the maximum and minimum values the sensor reads in the real situation. This is usually done by placing the sensor over first the lightest and then the darkest area, but in this case we’ll do it differently: we will move the robot in the same way it does when it carries out its task and register the maximum and minimum values. Next we will save those values in the corresponding file.

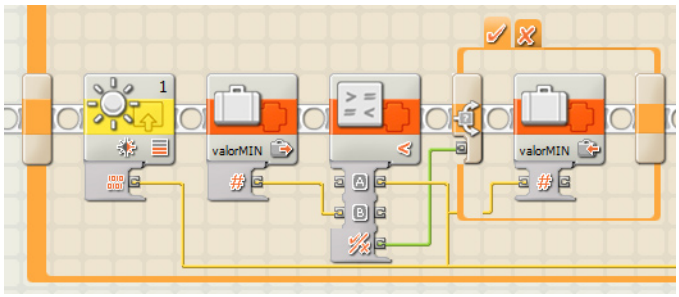
First let’s see what steps need to be taken.

1. Create two variables (**valorMAX** and **valorMIN**) to store the maximum and minimum light values.
2. Initialise the variables with valorMAX=0 and valorMin=1023 (in stead of using a percentage we will use the direct value with is somewhere between 0 and 1023).
3. Eliminate the calibration file (**Calibra1**).
4. Start the loop, which will run for 5 seconds, with the following steps:
   a. read the value of the light sensor
   b. compare the value to **valorMAX**; if the value is greater, store the new value in the variable
   c. compare the value with **valorMIN**. If it is less than the stored value, save the value to the variable.
5. After closing the loop, write the contents of **valorMIN** to **Calibra1** and do the same for **valoMAX**.
6. Close the file **Calibra1**.

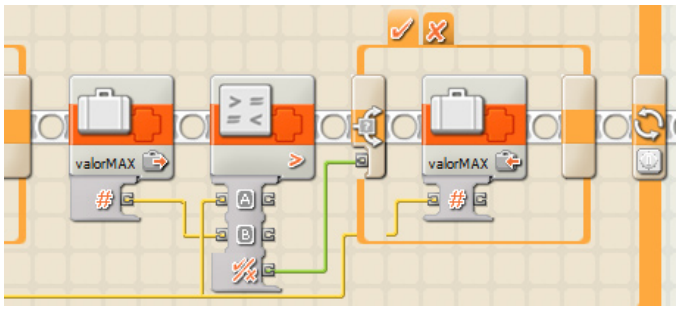
Now we have a calibration file stored on the NXT. This is a basic algorithm to which movement can be added, so that instead of the user moving the robot over light and dark areas, the robot itself can move (ensuring that the movement of the robot over the line is completely accurate).



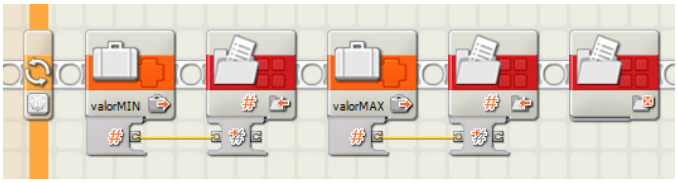
After creating the variables in the Define variables option of the Edit menu, we will execute steps 2 and 3 as in the previous image.



In this code fragment (steps 3a and 3b) you can see how the value of the reading from the light sensor on port 1 is obtained and compared to the value of the variable **valorMIN**. If it is less than this value it is stored in the variable, else it continues (the wiring is slightly mixed up due to the automatic wire organisation of NXT-G).



Next the program compares the reading to the value of **valorMAX** and if it is greater it is saved to the variable (step 3c).



After storing the contents of the two variables in the file and closing it, this first part is finished.

The same could be done for sensors connected to other NXT ports. We would create a new calibration file for each sensor we would want to calibrate differently. As for the program that would use these calibrations, I have assigned the name **Calibra1** to the file corresponding to the sensor connected to port 1, **Calibra2** for port 2 etc.

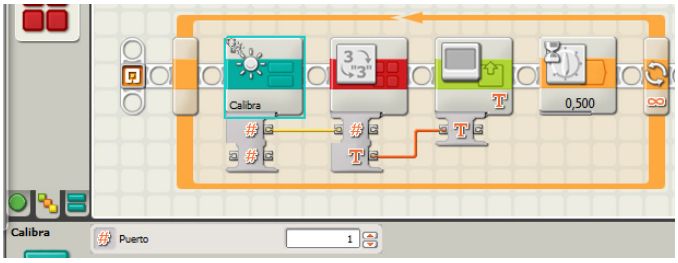
Using the calibration file

Let's see how we can use the values saved in the file. We have save the maximum and minimum values that were read under specific light conditions. Now we are going to use some mathematics to convert these two values into 0 (corresponding to the darkest reading, or "black") and 100 (corresponding to the lightest reading, or "white").

We can do this in two steps:

1. After reading the light value with the sensor, apply a proportional rule to calculate the result in a range of 0-100.
2. The direct light value the sensor gives is somewhere between 0 and 1023 (and since we have used that range in the file we need to use it here also), but a value of 0 corresponds to high luminosity (white) and 1023 to lack of light (black). If we want to convert this to the standard used in NXT-G (from 0 for dark to 100 for bright) we need to subtract 100 from the previous value.

Let's see an example program that serves to test all this and that can be recycled for any other application. The program, as seen in the following figure, will show the calibration value of a read on the screen in real time.



In the program in the image, the new block called **Calibra** reads the sensor that is connected to port 1 and returns the calibrated value. The rest of the code is to continuously show the readings on the NXT screen. So let's see the inside of the **Calibra** block.

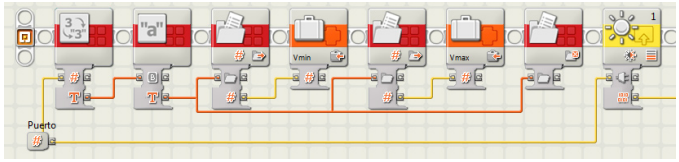
This block takes a number as an entry that corresponds to the port the sensor we are reading is connected to. If you name the files as described previously, it can return the calibrated value of sensors connected different ports with different calibrations.

The algorithm is as follows:

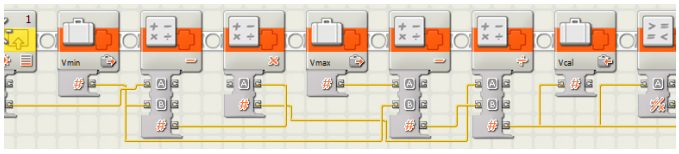
1. Create a file name with a text operation: convert the port number into text and add it to the word **Calibra** so that if the sensor port is 1, the file will be **Calibra1**.
2. Read the first value in the file and store it in the variable **Vmin** that was previously created.
3. Read the second value in the file and store it in the variable **Vmax** that was previously created.
4. Close the file.
5. Read the value of the light sensor connected to the port that corresponds to the entry (direct value).
6. Convert the reading into a calibrated value between 0 and 100 to store it the variable Vcal, with the following formula:

$$Vcal=[(Lectura-Vmin)/(Vmax-Vmin)]*100$$

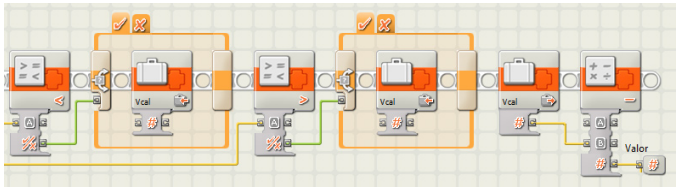
7. Check if the result is less than 0, in which case it becomes 0.
8. Check if the result is more than 100, in which case it becomes 100.
9. Subtract the previous value from 100 to get the exit value.



In this first image you can see the first 5 steps.



After that the calculation is made.



Finally, test to see if the result is in the normalised range.

Final remarks

From here each one can adapt the program to his needs: different calibrations for different sensors or different calibrations for the same sensor. This means you need to create as many files as you need different calibrations.

#