<http://www.legoengineering.com/inside-a-two-step-simple-line-follower/>

**Inside a two-step simple line follower**

In this unit, we will use the datalogging tools available in the EV3 Software to study the internal operation of a two-step simple line follower with one Light sensor. It is a very simple but powerful example that can help students to understand and to correctly program other situations.

One of the most important functions in both robotics and computer science is the capacity to record information to analyze it in real time or later on. Although it is possible to use the EV3 brick’s screen to display messages and sensor readings, it is not always possible to stop the robot to read its screen. In those cases, we need either to record the data to analyze it later, or display data in real time on the computer, if it is connected to the EV3 brick.

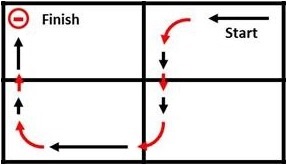
This unit explains in detail the concept of datalogging by means of manually accessing files, the management of these files from the EV3 Software using uploads and downloads, and their subsequent query and analysis. We will use data conversion, logic, and data wires to control the program flow.

It also takes advantage of the datalogging feature within the EV3 Software to automatically record data and visualize it graphically in the computer screen. This capacity to generate and relate the sensors’ readings to different curves as a function of time can be extremely useful in many and diverse situations.

Finally, it should be mentioned that the datalogging can be a very useful way of debugging a computer program, given that, in many occasions, it allows to understand much better the program flow and the reasons of its decision making process, like for example, how a switch works inside a loop.

This unit is based on my “[Teachers Introduction Course to LEGO® Mindstorms NXT & EV3](http://www.bogatech.org/index.htm?inicio=curs/)” at[BOGATECH](http://www.bogatech.org/)’s website, where you can find NXT versions of these programming examples and the differences between the NXT and EV3 versions, if you are interested.

## Exercise overview and preparation

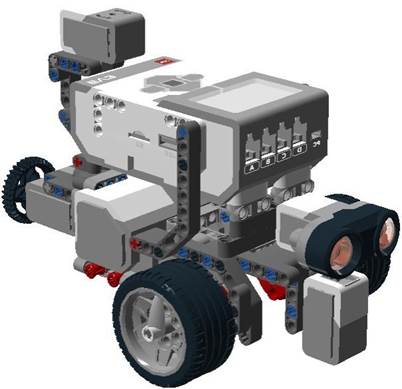


Challenge exercise field setup

The first exercise we will do is very simple, we just need a line with an intersection. It will, however, be useful to construct the field for this unit’s final challenge exercise, which is a circuit I like to call a “cage” that the robot has to follow and that will be very useful for all the exercises.

Use black tape on a white background to build the black line the robot will follow, as shown in the image to the right, where we can also see the path the robot will have to follow to do the final challenge exercise.

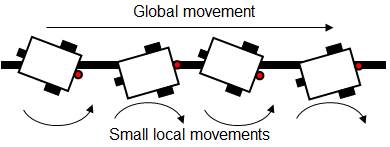
Each team will need a simple two-wheeled robot with a Color sensor (to be used as a Light sensor) attached to the front of the robot, as shown in the image (e.g. LEGO MINDSTORMS Education EV3 Software > Robot Educator > Driving Base + Color Sensor Down).



## ****A little bit of theory: Basic line following****

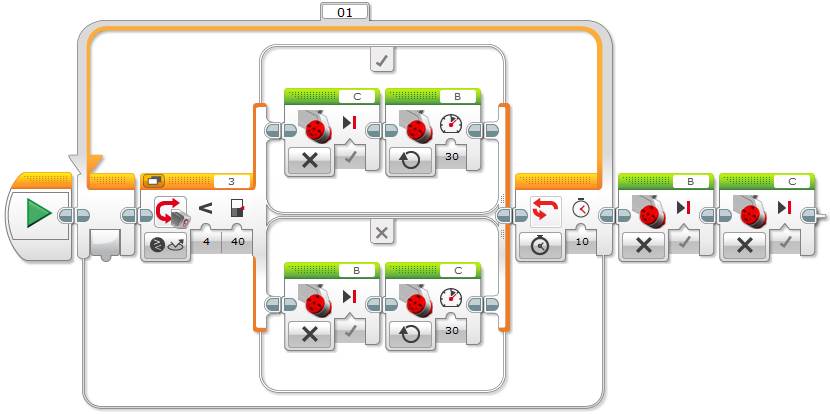
How can we make a robot follow a black line on a white background?

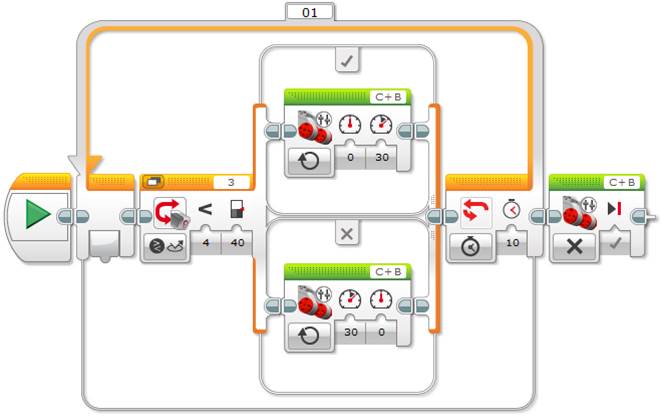
There are several solutions, but with only one Color sensor (used as a Light sensor) the simplest solution is probably to make it follow one of the edges of the line. Because the line has thickness, we can program the robot to follow the line in such a way that when the sensor “sees” white, the robot makes a point turn (a turn with a stopped wheel) towards black, and when it “sees” black it does the opposite turn. This algorithm is called a “two-step simple line follower” because it only has two actions. In addition, it demonstrates how the combination of small local movements, apparently nonsense (turns to the right or to the left), imply a global movement with a very special sense, in this case following the line. To program this algorithm, we just need a Switch block set to Color sensor mode (to make the robot turn to each side), inside a Loop.



Two-step simple line following algorithm

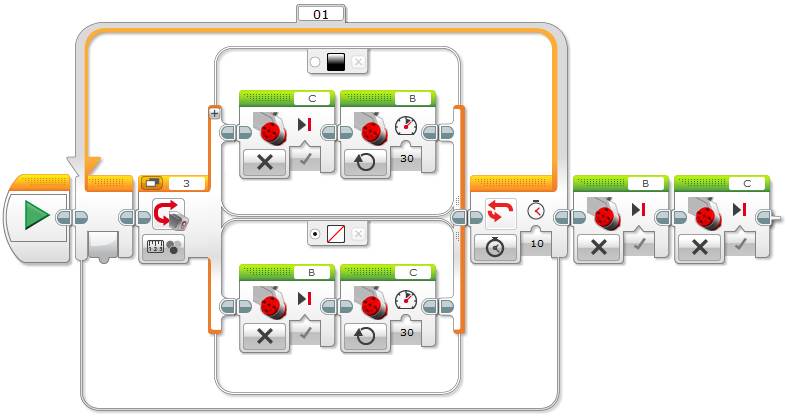
If we study how a two-step simple line follower works, as shown in the image below, we can ask ourselves how many times the program passes through each one of the switch options before changing the option, while the time loop of our example lasts.

[](http://www.legoengineering.com/wp-content/uploads/2014/12/image005.jpg)

Two-step simple line follower with one color sensor, with Motor blocks[](http://www.legoengineering.com/wp-content/uploads/2014/12/image006.jpg)

Two-step simple line follower with one color sensor, with Move Tank block

In the previous examples the motors are configured with 30% power, to avoid swerve movements, and the time loop runs for 10 seconds. The Switch block is associated with the Color sensor in the “Compare – Reflected Light Intensity” mode. In the next example, the Color sensor uses the “Measure – Color” mode to measure the color.

[](http://www.legoengineering.com/wp-content/uploads/2014/12/image007.jpg)

Two-step simple line follower with one color sensor, with “Measure – Color” mode

**Tip**: When using the Color sensor with the “Measure – Color” mode to measure the color, it is important to select the Switch default case, for example, the white color. We can also select “No Color” as default in such a way that any detected color by the Color sensor not being black will execute this case. How will the robot behave if we do not configure the “No Color” case as default? Well, since you need to select a default case, the only other option will be to select the black color case, thus the program will always execute this branch, even while detecting white, because any detected color is different from “No Color”! It is important that the students make tests to understand how the Switch block works.

At this point it is important to make the students give their opinion about what’s happening inside the Loop and, for example, make them replicate the robot behavior with their body. We will see that the majority of them do not clearly understand how the algorithm operates. For example, they may not fully appreciate that as the robot moves from one color to the other, the program will execute the sameoption or branch of the Switch many times, until the robot moves onto the other side – many  faster than the robot’s apparent physical execution of the Loop. The following exercises show  how the Switch inside the Loop algorithm works for the two-step simple line follower with one Color sensor.

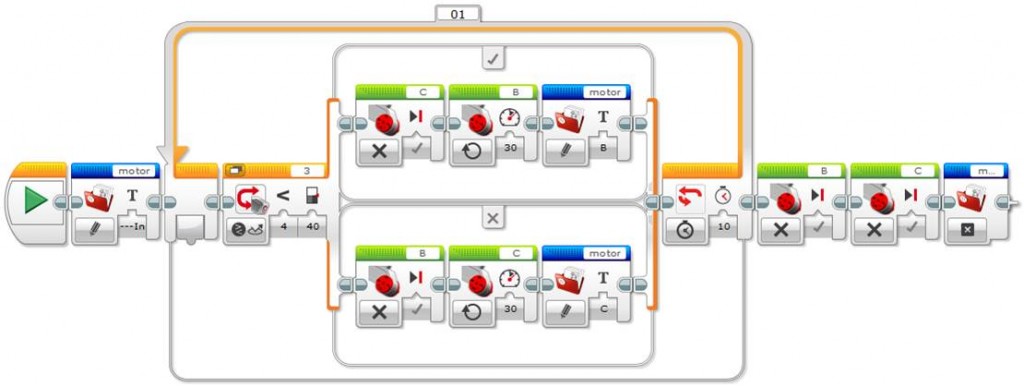
## Exercise 1. Data logging using file access: A numerical and manual debugging strategy

The simplest way to demonstrate that the EV3 brick executes the program much faster than the robot’s apparent physical execution of the same program is to record the Switch branch that is being executed in each iteration of the Loop.

This exercise uses a File Access block to record in a text file the Switch branch that the program has just executed. The final step is to check the generated text file. Create a new program, build the line follower basic algorithm and add the File Access blocks, as shown below.

First we need to add an initial block configured with the “Write” action to write to a file. Set the name of the file we want to create, for example “motor”, and the type of information we want to add, then choose the option “Text” of the “Type” pulldown menu, to add an initial text “—Start—”, that will help us to know when the data logging starts in the text file.

Next we will add two “File Access” blocks similar to the previous one at the end of both branches of the switch to write in the file the branch that the program just passed through. For example, when the program goes through the option where the motor B runs we will add “B” and when it goes through the other option we will add “C”, in the textbox “Text”. To finish the program, we need to close the file before stopping the program. For this we need to add a final block with the action “Close” to close the file. Be careful to always write in the appropriate filename, “motor” in our case. To avoid errors, a good option is to copy any previous block and modify the desired options.

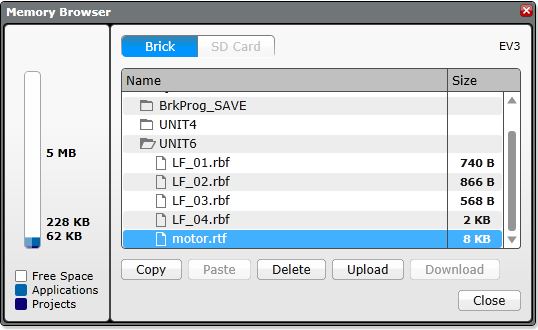
[](http://www.legoengineering.com/wp-content/uploads/2014/12/image008.jpg)

Simple line follower program with “File Access” blocks

Once the program has been compiled and executed by the robot, connect the robot to the computer to find the generated file. For this click on the “Open Memory Browser” button in the program’s window lower right corner to open the EV3 memory management window where we will find the text file “motor.rtf”.

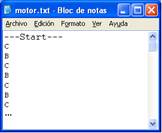


Button to open the intelligent brick memory management



Intelligent brick memory management window

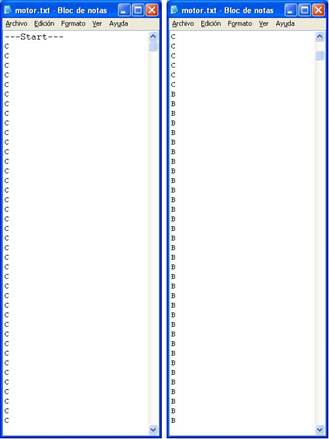
Now just upload the file from the EV3 brick by clicking on the “Upload” button, and save it in the computer to the desired location.



Supposed contents of the “motor.rtf” file

Open the “motor.rtf” file (e.g. with Microsoft Word) to view its contents. Note that the generated text file has been saved with a “.rtf” extension. Before opening the file, it can be interesting to ask the students about what they think its contents will be and make them fill in the file manually on the black board. Any students who think that each switch’s branch will be executed only once while the robot goes from one color to the other will presumably write: “B, C, B, C, B, …”, as shown to the right.

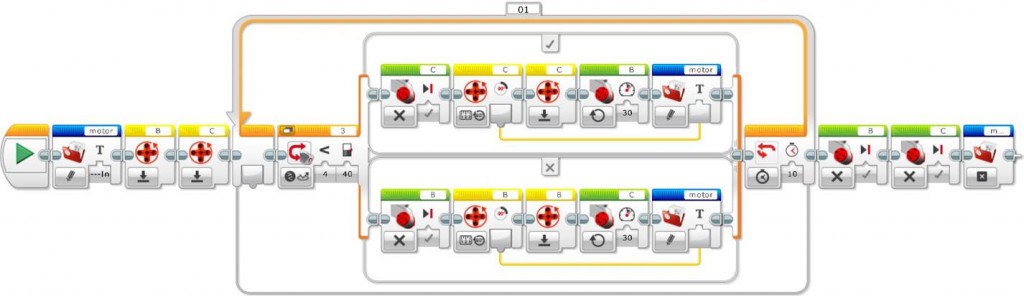
When opening the file we can observe that each letter is repeated many times, indicating that the program goes through the same Switch’s branch many times before changing to the other branch inside the Loop. Thus, the program executes the Loop much faster than the robot movements, as shown in the images below.



Real contents of the “motor.rtf” file

The next challenge is to count how many degrees each motor turns before changing the branch of the Switch inside the Loop. The number of degrees turned by each motor will mainly depend on its power, as well as on the robot design. Initially we will suppose a power of 30% to avoid the robot slipping. To accomplish this we will explicitly use the internal rotation sensor of each motor. An important aspect before starting the exercise is to ask the students where to add the internal rotation sensor blocks of each motor. Given that the program passes through the same Switch branch several times before changing branch inside the Loop, as it has been proved, it is important to realize that we need to record the rotations of a motor just before changing the active motor in the switch.

The following image shows the previous line follower program where internal rotation sensor blocks for each motor have been added. It is important to realize that when a motor stops we need to count its rotation degrees, we need to reset the sensor, and finally, we need to record these rotation degrees in the text file. To accomplish this process we need to connect the data hub plug coming out of each motor internal rotation sensor block, corresponding to the motor degrees of rotation, to the corresponding “File Access” block data hub text input.

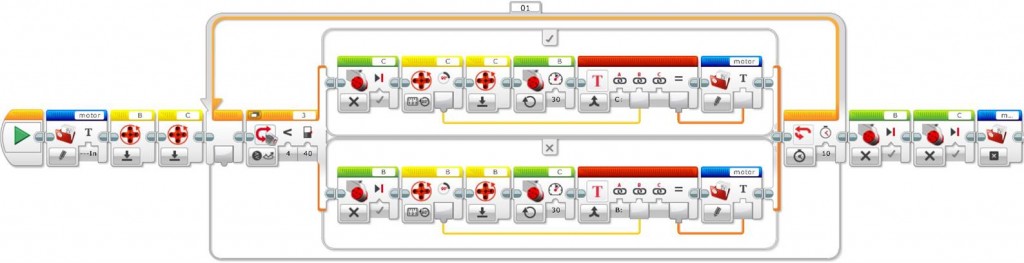
[](http://www.legoengineering.com/wp-content/uploads/2014/12/image013.jpg)

Initial line follower program with file access and internal rotation sensor blocks

Note: As we can see in the previous image, the “File Access” block is capable of automatically converting the numeric values of the rotation degrees into a text string and we do not need to use a conversion block, as we might need to in other programing languages.

But have we obtained the desired result yet? If we analyze in some detail the result we want to obtain in the text file, we will see that we not only need to record the rotation degrees, but we also need to record which motor has turned these degrees. Otherwise, we will obtain a bunch of numbers and we will not know which motor they refer to. For this, we need to add the letter of the Motor “B” or “C” to the rotation degrees in each case. Thus, we need to add a new block to concatenate these text strings. This block is called “Text” and allows concatenating up to 3 text strings.

**Tip**: It is interesting to observe that every time we use the file access block, this writes a new line into the data logging text file. Thus, if we do not concatenate the text strings and we use the “File Access” block two times in a row, one for each text, we will obtain the motor letter and the rotation degrees in different lines, which will make it difficult to read the datalog file.

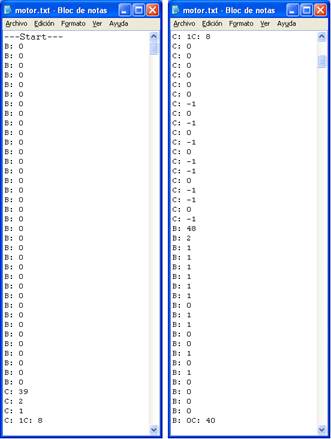
[](http://www.legoengineering.com/wp-content/uploads/2014/12/image014.jpg)

Correct line follower program with file access and internal rotation sensor blocks

As mentioned before, when recording the data in the text file we can observe that, to distinguish each motor’s rotation degrees, the motor is shown as well as its rotation degrees. For this, we need to combine the rotation degrees with a text that shows the corresponding motor, by means of a “Text” block, for example using the text string “B: ” or “C: ” for each case, as shown in the above image. Next, compile and download the program to the robot.

**Tip**: Every time a data log is generated with the same name, its size gets incremented, and that can end up by filling up all the EV3 brick’s memory. Thus, before the robot executes a program that generates a data log, it is important to delete the previous file from the memory to avoid the program adding data and incrementing the file size, whenever the previous data is not important. If you want, this can be achieved programmatically by adding a “File Access” block at the beginning of the program with the action “Delete” and selecting the appropriate filename. If we do not delete it, we will easily find each test start by locating the initial text string that the program adds just before starting.

Finally, upload the “motor.rtf” file corresponding to the data log to observe and analyze its contents. Before opening the file, ask the students to specify with the maximum detail possible what its contents will be.

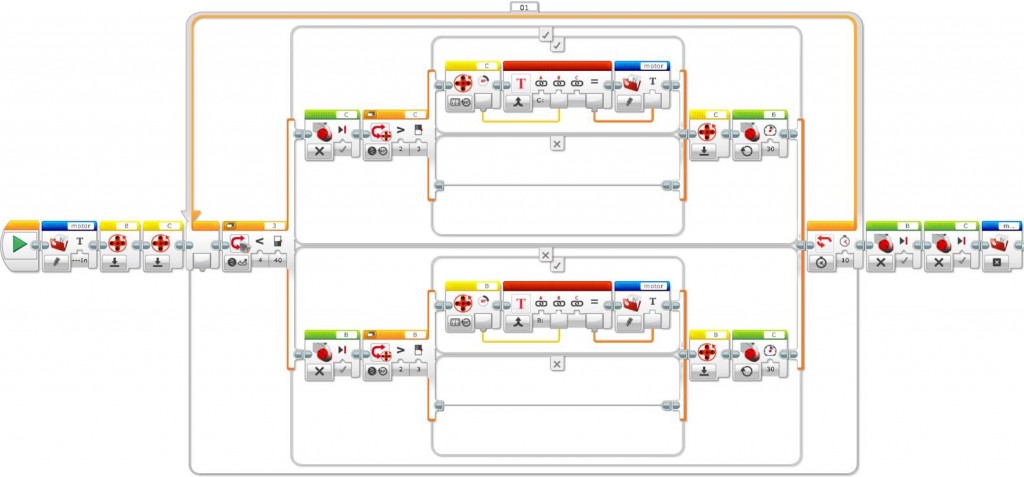


“motor. rtf” file contents with each motor’s rotation degrees

As we expected, we can see that every file entry which captures motor rotations just before changing the Switch branch inside the Loop contains rotation degrees that range from 30 to 50 degrees. But the question is why there are some entries equal to “0” or very small, “1” or “2”? The answer is very simple… When we count the rotations of one motor, the Switch writes the rotations of the opposite motor, which is obviously stopped. If this motor slips a little bit it might actually make one or two degrees of rotation, or it can even slip backwards, and that implies negative degrees, e.g. “-1”!

How can we improve our data logging file? Let’s eliminate all the data from it that does not provide meaningful information. For this we need to use the internal rotation sensor comparison attribute or threshold value of each motor, compare it to the values we want to record, for example only the ones bigger than 3, and finally, record only these data for the desired case.

To record only the desired data, one less efficient solution that students should be able to find by themselves with the acquired knowledge, is to use a Switch associated to the appropriate internal rotation sensor. It is important to leave the students find this solution, or at least part of it.

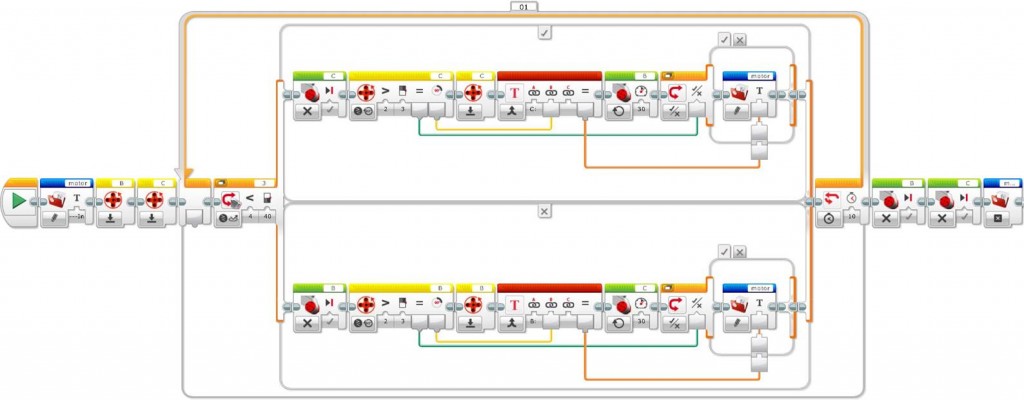
[](http://www.legoengineering.com/wp-content/uploads/2014/12/image016.jpg)

Line follower program with a strategy to avoid null rotations, with file access blocks and a switch associated to the internal rotation sensor

A more efficient strategy to record only the significant data is to use the internal rotation sensor logic plug out “Yes/No” specifying the desired rotation degrees threshold, for example bigger than 3, and connect this data plug out to a logic switch data plug in to record in the file only the values bigger than the specified threshold.

Finally, to connect the text of the “File Access” block with the motor rotation degrees we need to “cross” the switch, something that can be only accomplished by unselecting its “Flat View”.

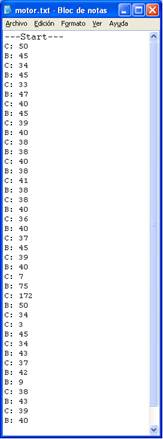
**Tip**: Switches can be visualized either with a split sequence beam under the so called “Flat View” or in the shape of “Tabs”. To wire from a block located outside the Switch to a block  inside the Switch, the “Flat View” needs to be unchecked to enable the “Tabbed View”.

[](http://www.legoengineering.com/wp-content/uploads/2014/12/image017.jpg)

Line follower program with file access blocks, internal rotation sensor blocks and efficient strategy to avoid non-significant rotations

But why is this programing strategy more efficient than the previous one? This program is more efficient than the previous one because it writes to the file at the end of the Switch branch after activating the opposite motor, and because it does not duplicate blocks unnecessarily.

Once the program is completed, compiled, and executed by the robot, the file “motor.rtf”, corresponding to the data logging record, only shows the rotation degrees accomplished by each motor while following the line.



“motor.rtf” file contents only with relevant rotation degrees of each motor

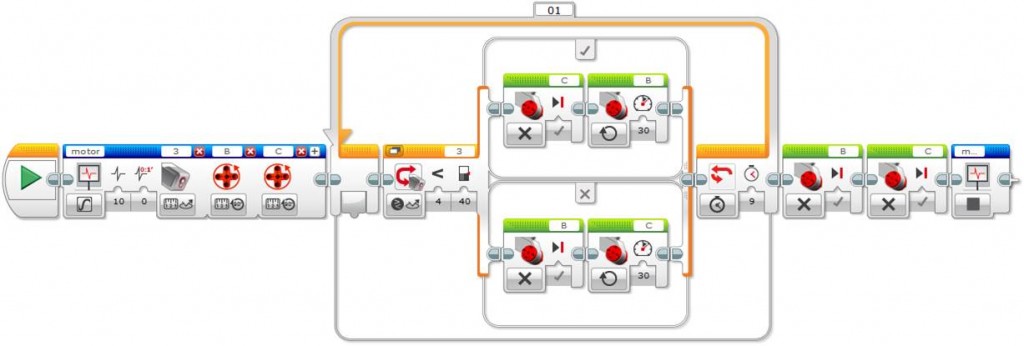
Is there anything that calls the attention in this file? Why is there a point where the rotations are bigger than in the general case? Can we identify this point on the field? This example, run on the challenge exercise field, where the robot follows the line on its internal side, and during a line following time that allows to surpass the first intersection line, numerically illustrates the point where the robot crosses this intersection line. This point corresponds the biggest rotations, that is, B=75 and C=172 to recover the line following path.

Thus, we have just designed a strategy to make a two-step simple line follower with one Color sensor more intelligent, by tracking each motor’s rotations, so as to make it capable to knowing when it encounters a line intersection!

## Exercise 2. Data logging: A graphical debugging strategy

This exercise uses the Data Logging block to record the sensor readings of Color and internal rotations associated with motors “B” and “C” in a file with an “.rdf” extension.

To start coding, create a new program, add the basic line follower algorithm including data logging blocks, as shown below.

[](http://www.legoengineering.com/wp-content/uploads/2014/12/image019.jpg)Line follower with “Data Logging” blocks to start and stop data log recording

When configuring the Start Datalogging block it is important to pay attention to different aspects.

You need to give it a name that will be used later to stop the datalogging with a corresponding block, in our example “motor”. You need to define the duration to “On” to pass the time control to the Loop that will repeat for 9 seconds –this allows robot line following to start immediately after the datalogging begins–, and the rate with the number of samples per second to 10.

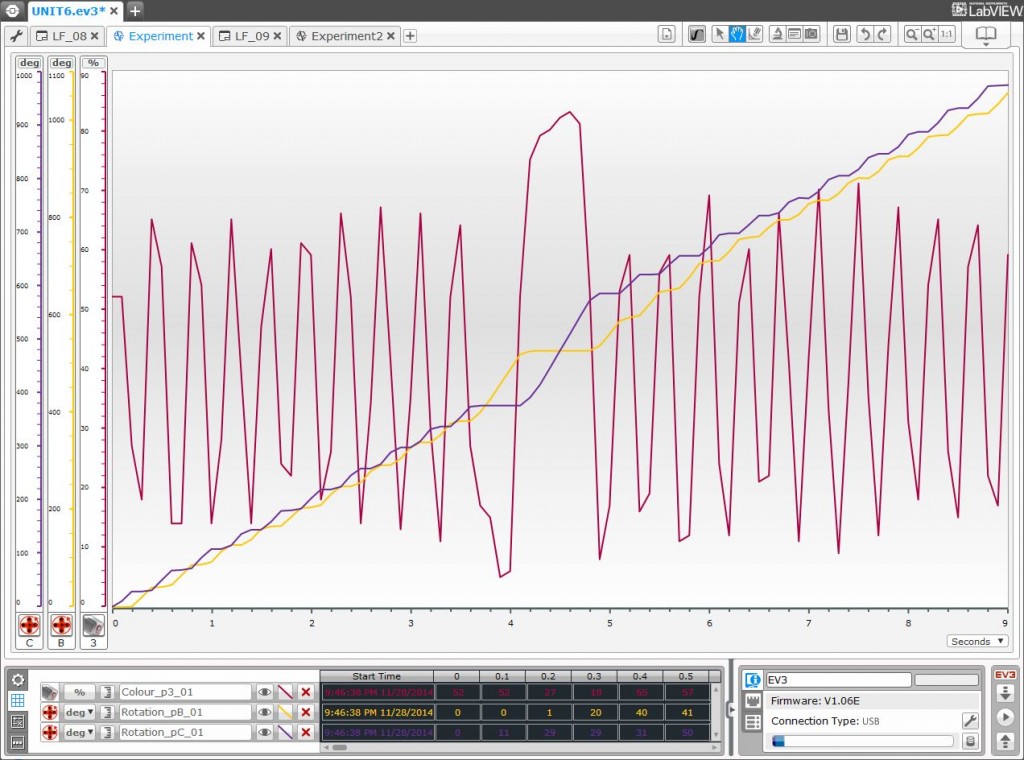
Finally, we have to configure the desired sensors’ data capture. First we have to add the Color sensor and after this the internal rotation sensors associated with each motor. We can add sensors by clicking on the upper right “+” sign icon of the block. Note that the internal rotation sensors units are degrees and not rotations, later on we will see the difference.

Connect the EV3 brick to the computer to download the program, and make the robot execute it, having previously disconnected the robot from the computer. You need to place the robot in the same field of the previous challenge exercise to be able to compare the results.

Once the robot has completed the line following exercise, connect it to the computer, and from the Experiment interface, open the generated “.rdf” file, “motor.rdf”, by clicking on the up arrow icon, in the application lower right corner, to do a file “Upload”.

We will see that the window allows loading the “.rdf” file from the computer or directly from the EV3 brick, by selecting it from the list. In the lower part, the window shows the available files to select. In our case, select the file “motor.rdf”.

**Tip**: The “.rdf” files not only can be opened and deleted from the experiments graphical interface, but also, you can manage them from the EV3 window, as we have done in the previous exercise with the text file. The “.rdf” files are binary files that can be only opened from the EV3 Software. These files have a specific structure that allows making an “intelligent” reading of them.

[](http://www.legoengineering.com/wp-content/uploads/2014/12/image020.jpg)“.rdf” file with rotation sensors units expressed in degrees

This first datalogging file shows the data recorded from the Color sensor in red and the rotation sensors in purple. To better read the graph, we can change the color of one of the rotation sensors by clicking the  corresponding color square.

To better differentiate the motors’ movements and to obtain more relevant differences, we have chosen degrees instead of rotations, but why? Because 1 rotation is equivalent to 360 degrees, that is 360 units, thus the differences are bigger. As you can see, the system chooses automatically this option.

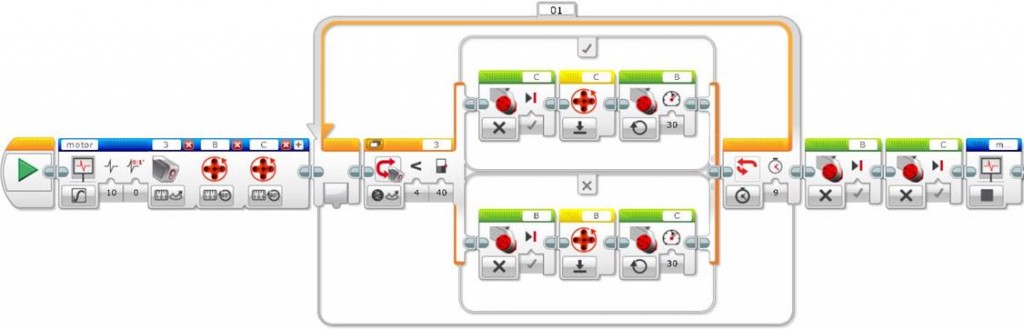
We can select the units, rotations or degrees, on the left side of each curve of the “Dataset Table” tab. We can also choose the default desired units in the “Data Logging” block attribute definition, as explained previously. It can be interesting to let the students test different units to better adjust the curves visualization, and take into account that you can even manually modify the coordinate system minimum and maximum values by writing the desired value in.

It is important to make the students interpret and explain the generated graph relating to the different curves. First we can observe the Color sensor readings in red, which range from black to white (see the reflected light intensity in the Y-axis). One first question is why are the lines inclined in a “toothed saw” shape and not vertical? The answer is that near the threshold between the black and white colors, the color sensor detects progressive changes of the reflected light intensity while the robot goes forward from one color to the other. See [Exploring Thresholds](http://www.legoengineering.com/exploring-thresholds/) for more about this topic.

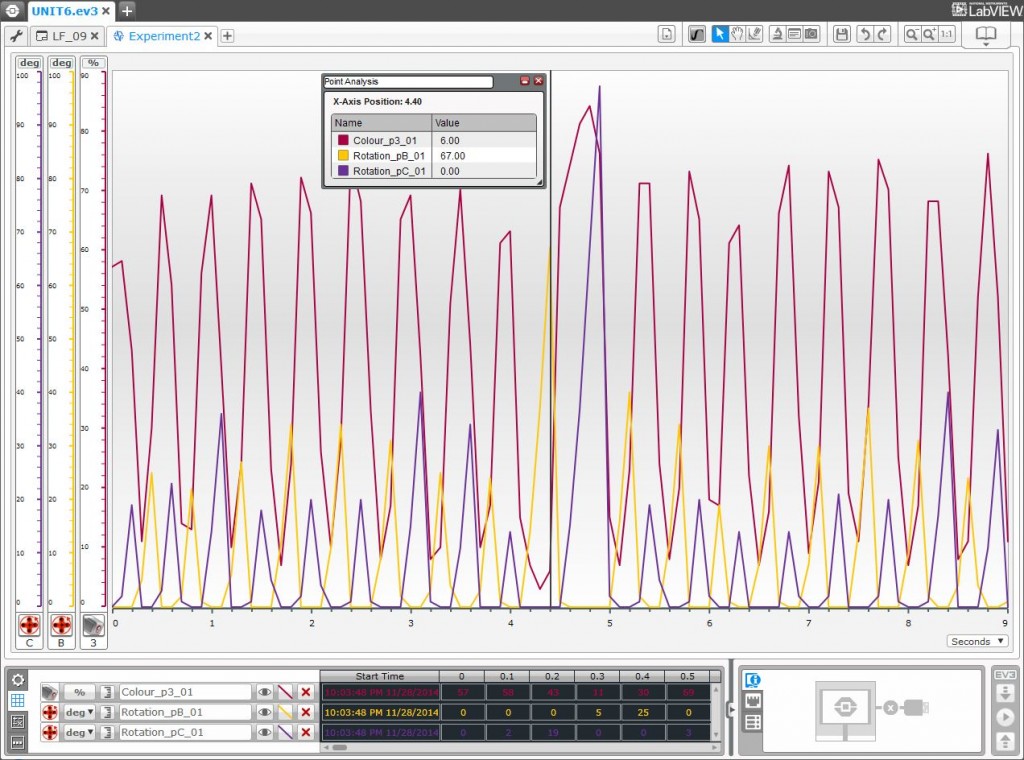
In relation to the motor curves, we can ask the students why the corresponding curves have an “S” shape that tends to go up. The answer is because, on the one hand, when a motor starts the other one stops, and on the other hand, the rotation sensors accumulate rotations, and thus, the line has a positive trend.

Finally, if we observe the color sensor curve, we can see that up to a certain point there is a higher peak. What does it correspond to? This peak corresponds to an intersection on the line the robot follows, that in our case, it is the field middle line of the challenge exercise. If we pay attention, even if it is a little bit hard, we can also see that the motors’ rotation degrees, especially the “B” ones in orange color, look also bigger in the vertical of this same point, which approximately corresponds to the second 4.5.

To finish the exercise, what do we need to do to obtain only the rotations the robot does when changing the color in the loop iterations? To obtain only the rotations that each motor does in each Switch phase inside the Loop, we only need to reset the internal rotation sensors after stopping them inside each branch of the switch, by using the corresponding internal Rotation Sensor block.

[](http://www.legoengineering.com/wp-content/uploads/2014/12/image021.jpg)Line follower with data logging blocks and reset of each motor internal rotation sensors

Before executing this exercise it can be interesting to ask the students what will be the differences between this new graph and the previous one. As we can see below, the graph corresponding to the new program allows  much better interpreting of the results obtained.

[](http://www.legoengineering.com/wp-content/uploads/2014/12/image022.jpg)

“.rdf” file resetting the motors internal rotation sensors and with units expressed in degrees

Given that the motor turn is reset between each line follower Switch branch of the Loop, the previous graph shows only the degrees turned by the motors inside each branch, which is exactly the information we are interested in to find out the point of the field where a line intersection lies.

At this point it is interesting to ask the students to interpret the results. In the graph we can see that the motors rotation degrees also show a “toothed saw” shape curve and that the curves of each motor alternate, given that when a motor runs the other stops (or almost, because it can slip), corresponding to the cycles between the field background white color and the black color of the line to follow.

Finally, we can observe that the peaks corresponding to the internal rotation sensors do not correspond exactly with the peaks of the color sensor, why? This is due to the robot inertia and to the program and sensors readings execution speed, when each motor is reset it is just when the color changes according to the specified threshold between white and black. Thus, the peaks of each rotation sensor tend to coincide with a point previous to the “rise” or “descent” of the color sensor curve, that is, with the threshold between white and black. If we take many more samples per second this fact would be more obvious and, in fact, if the program was very fast, the internal rotation sensors curves will show a completely vertical descent, because the motors stop at this point.

Another fact that we can observe in the example is that Motor B seems to make a little bit more rotation degrees than Motor C through the whole robot’s path. Why does a motor seem to turn a little bit more than the other one?

Assuming the robot is following a straight line segment, motors should turn equally and even if the difference is very small it can happen that a motor turns more than the other one due any of a number of factors. These might include robot geometry, or if the color sensor is not well located in the middle of the robot, or if the robot weight is not evenly distributed, or if the wheels are not well built and arranged at the same distance from the robot axis, or if one wheel slips more than the other one on the field surface (the one that moves as well as the one that should be stopped, in fact we have seen that a supposedly stopped wheel can actually turn backwards…), or if the field surface is not clean enough or if it has an unnoticeable lean towards one side. If the differences are big, you can revise the robot construction and repeat again the experiment, and probably you will obtain a slightly different result with very similar rotations between both motors. This is a very important point to take into account, especially when going to a competition or when high precision of movements is required.

Finally, the Experiment mode of the EV3 Software also provides a very interesting interface tool that allows querying the sensors values at a specific point, in our case the position around the 4.5 second where motor “B” rotations are the biggest and which corresponds to a line intersection on the line the robot follows. This tool can be accessed from the menu icon “Analysis” with the option “Point Analysis”.

The line corresponding to the analyzed point in a specific time can be dragged to the desired point, in our case near the second 4.5. We can also name it, generate as many analysis points as we want, and save all these analysis points with the experiment for future use. The Experiment interface also offers other tools like an area analysis tool, a magnifying glass, to zoom into the image on a specific area, and a prediction tool, all of them very easy to use and that can be very helpful.

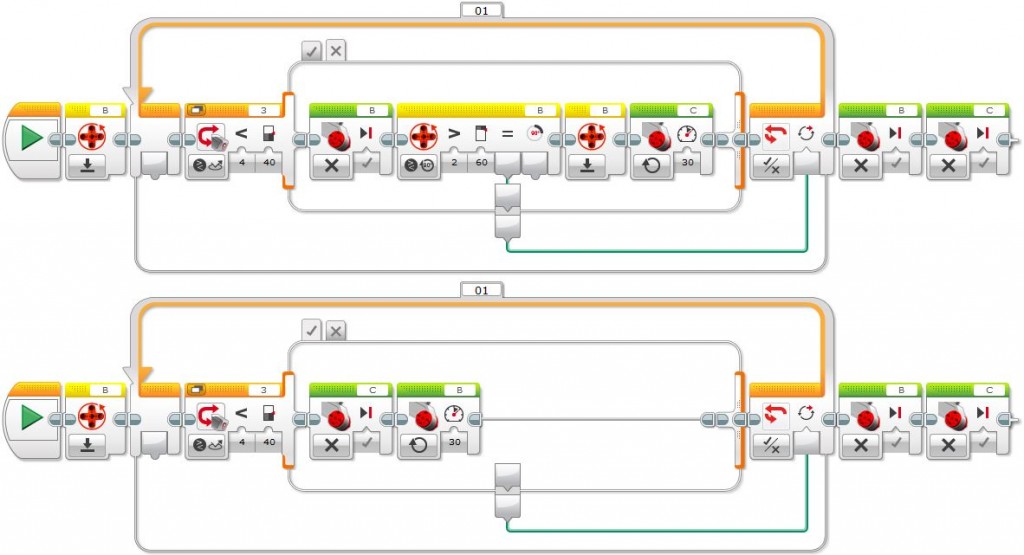
## Challenge: Get to the end of the cage!

As we have seen previously, when following the line, the robot makes small movements to the right and to the left. When it encounters an intersection, however, to continue following the branch of the line to follow, one wheel does more rotations than those needed to follow the straight line, while the other wheel remains stopped. To control the global robot movement, we can detect when this occurs, stop the line following, turn the robot to the appropriate side (the one towards this intersection) and continue following the line. To do the challenge exercise, we only need to monitor one wheel. In our case, we only need to control motor “B” rotations on the right, since the robot follows the black line on the left, that is, in the interior side of the field, and when finding the intersection line that goes to the left, wheel “C” stops and wheel “B” does more rotations than when it follows the straight line, to find the “white” color.

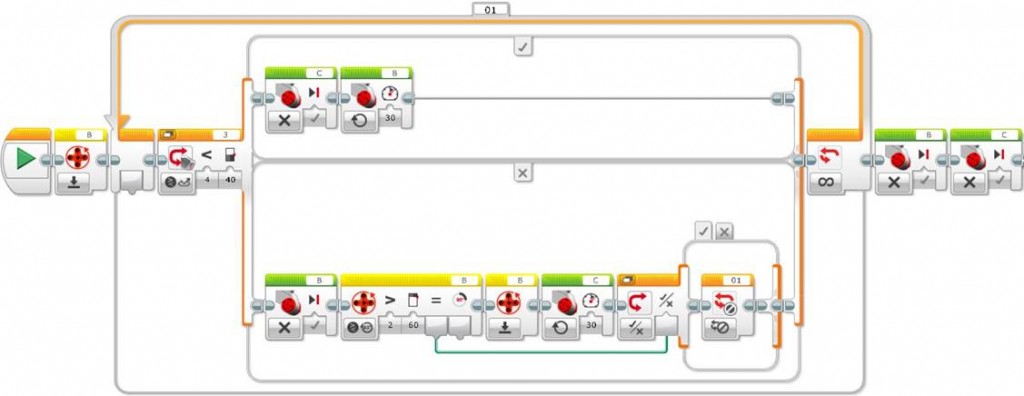
As we have seen, to control a motor’s degrees or number of rotations, the internal rotation sensor block provides a way to compare the rotations with a specific threshold, 60 degrees in our case. If the rotations are bigger to the activation or threshold value, then the attribute or plug out “Yes/No” of the data hub will return the value “True”, which we can connect to the loop logical plug in, using a data wire, to stop the line follower by getting out of the loop.

**Tip**: Remember that to connect a block logical attribute, which is inside a Switch, to the logical connector of a Loop that contains the Switch, “Flat View” needs to be unselected. Alternatively you can use a “Loop Interrupt” block to force the exit of an unlimited or forever Loop.

Note that the rotation sensor needs to be reset after reading the supervised wheel rotations, for each line following cycle. To count the rotations, the rotation sensor needs to be placed just after stopping the supervised motor and before starting the opposite motor rotations.

[](http://www.legoengineering.com/wp-content/uploads/2014/12/image023.jpg)

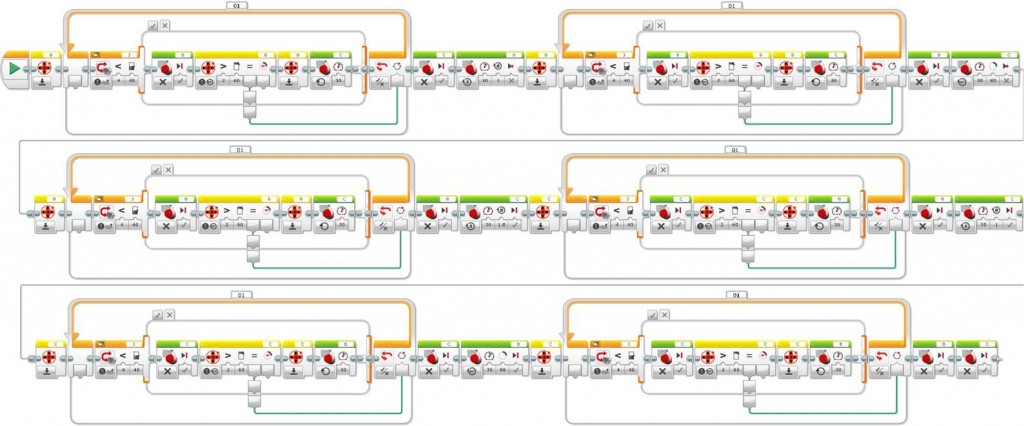
Line follower that stops when detecting a rotations threshold bigger than 60 degrees in the internal rotation sensor associated to motor “B”

[](http://www.legoengineering.com/wp-content/uploads/2014/12/image024.jpg)

Line follower alternative that stops when detecting a rotations threshold bigger than 60 degrees in the internal rotation sensor associated to motor “B”, using a loop interruption

At a global level, to do the challenge exercise we need a line follower every time there is a direction change or a line intersection. To get over the points of direction change we only need either to execute some rotations in the appropriate wheel to turn the robot and start a new line following path (making a point turn with a stopped wheel), or to go forward to surpass the intersection line (in this case since the robot will have turned, we only need to make it turn to the opposite side to recover the robot direction, surpassing the intersection).

Observe as well that, as a function of the robot’s turning direction, sometimes we need to monitor the right wheel and sometimes the left one, as shown below. For this last point, we can ask the students what needs to be changed in the program to make the robot follow the line from the other side.

[](http://www.legoengineering.com/wp-content/uploads/2014/12/image025.jpg)

Challenge exercise programing using a simple line follower with rotations control to detect each intersection line

What are the advantages of using a rotation sensor to do the challenge exercise? The big advantage of this solution is that it can even work if we modify the size of the field or the robot starting point. Thus, this solution is very generic and robust although a little bit complex compared to other straight forward solutions. What disadvantage can we observe in the previous program? What we can see is that the program duplicates the line follower code after each intersection. Further on we will see how we can encapsulate this code in a user block or subprogram to make it much more compact and efficient.

## Acquired knowledge

Datalogging, either by writing to a file directly using the File Access block or through the Data Logging block, not only allows us to record data dynamically while the program executes, but also allows us to observe in detail, study, and understand how a program works, as well as providing a very good method of program debugging. In addition, students learn how to use Logic Switches, how to pass information between program blocks using data wires, how to access and manage the EV3 brick’s memory, and how to interpret numeric data in tables and graphs.