An introduction to Robotics with LEGO® MINDSTORMS (XI)

Basic Robot Movements

By Koldo Olaskoaga

This is the third article in a series of challenges to be solved using LEGO® MINDSTORMS. The idea responds to the questions some FLL teams have asked me about how to navigate the table. We are going to look at creating a robot that can move wherever we want and some proposals for improved use of the NXT-G Move block.

Challenge

This proposal is very basic, the typical starting point for programming the robot. It is a challenge in which we use only the motor ports of the robot.
The idea is to build a robot that move along a predetermined path that can be defined at will. In my case what I will do is place three cans on the floor and we will see how to solve some common problems.

The robot

Let's start by building the robot. Before we start programming with sensors it is a good idea to practice with robots that move along a predetermined path, much the same way you would move through a well-known space without being allowed to touch anything and blindfolded. This means we are not going to ask too much, just the following:
● Capacity to move in a straight line and turn.

With this starting point, the first decision we need to take is what locomotion system we are going to use.

Locomotion systems

The choice for one system or another conditions both the structural complexity of the robot and its programming, and in addition, its maneuverability. Without trying to be exhaustive, let's look at the following systems.

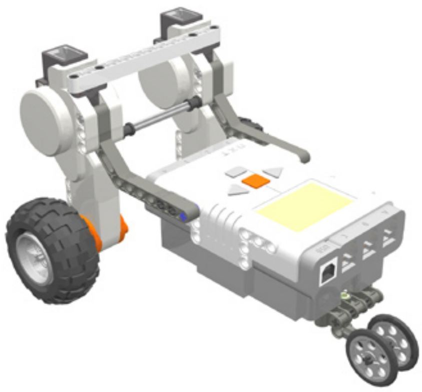
Differential drive

This system is based on connecting a motor to the wheel or wheels on each side of the robot. If both motors turn in the same direction the robot will go straight forward, if the turn in opposite directions the robot will turn to one side or the other. It will even be able to turn in place.
Depending on the surface the robot will move on there can be differences in the way this system is implemented. Let's see three of them.

● Robot with two wheels
This first construction is the the most basic, simple to build and program, although it doesn't move well on irregular surfaces or surfaces on which it cannot slide well. The robot in the picture is supported on two wheels and two additional sliding points which will work better or worse depending on the characteristics of the surface it moves on.

● Robot with two wheels and a third support
The frontal support of the previous system can be substituted by a third support , although this makes the build a little more complex.

● Robot with two wheels and an idler wheel
In this case the robot is also supported on three points, with a wheel that changes its position to adapt to the turning. In this case the design of the idler wheel conditions its efficiency. It is also possible to use two idlers in the same way, just like wheelchairs or baby strollers have.



● Robot with tracks
This system gives us a very stable base. It is an easy to build robot that is relatively stable. The track system is compatible with different types of floors, like carpet, ceramic, wood... and allows the robot to overcome obstacles. This is the system used by the starter bot from LEGO® MINDSTORMS 2.0

Other locomotion systems

Another well known system is the one used in cars and which requires 2 motors; one for propulsion and one for steering. The robots have reduced maneuverability.
Walking robots deserve a separate mention and correspond to a more advanced level.

The centre of gravity

There is an important factor that must always be taken into account in any mechanical design: the position if the centre of gravity. The centre of gravity will influence the behaviour of the robot, especially at start, stop and turning. A robot that is apparently stable may sway or even topple over when starting if the centre of gravity is too far to the back. It can topple over at turning if the centre of gravity is too high.
The centre of gravity should always be located inside the polygon described by the supports on the floor (triangle, rectangle...), but not only that. When the robot starts, stops or turns there are new forces that can also make the robot sway or topple over and for this reason - just like in cars - it is convenient to have the lowest possible centre of gravity.

My choice

I have built the robot you can see in the first picture, simple, and easy to build. Nothing further is required to experiment with motors.

Programming

Let’s start with a basic program and then improve on it.

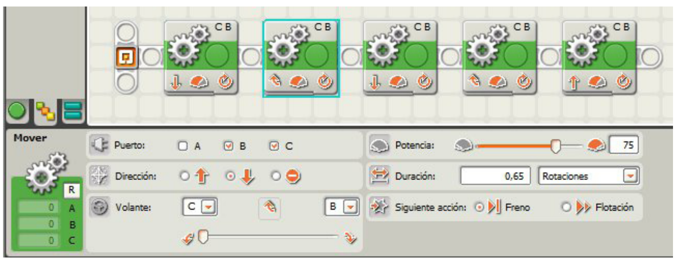
Basic challenge #1

We draw a straight line and place a can at a predetermined distance. The robot starts at the line, turns around the can and returns.

The algorithm in this case is very simple:

1. Straight forward
2. Turn around the can
3. Straight forward

When we turn this into an NXT-G program, after going straight forward and although this could be done differently we’ll do it as follows: turn 90º, go forward past the can and another 90º angle turn. In this way you can make a smaller turn that by doing it in a single movement. The program will be as follows:



To determine how much the motors must turn, NXT-G allows you to use time, degrees (with a precision of 1 degree) and rotations. A fourth option , unlimited, is very useful when working with sensors. In this program I have used the rotation option in the control panel of the Move block. This option allows for more precision, since the time option may not work the same way if the batteries are low. The necessary rotation has been determined by trial.

Improvement #1

Although the program carries out the assigned task, the movement is not fluent and when the execution of the program passes from one block to the next the motors stop and start again. By default the Move block has the Brake selected in the control panel, which means it ends by blocking the motors and starts them again with the next block.
This behaviour can be improved somewhat selecting the Coast instead of Break. In this way at the end of each block instead of blocking the motors, power to the motors is cut and they could continue turning if they have enough inertia.

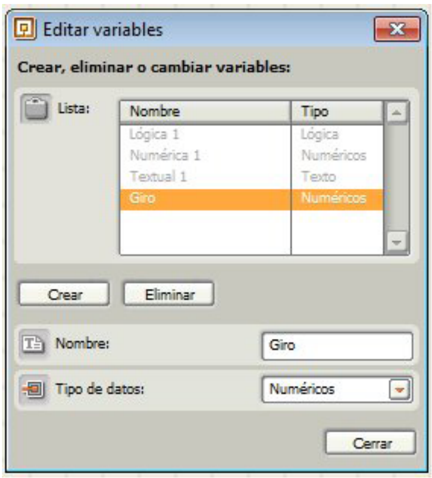
Challenge

Another interesting challenge is placing several cans on the floor to create a slalom. I will leave this as an exercise for you. What you need to do is place the necessary Move blocks (straight and turn) to solve the challenge.

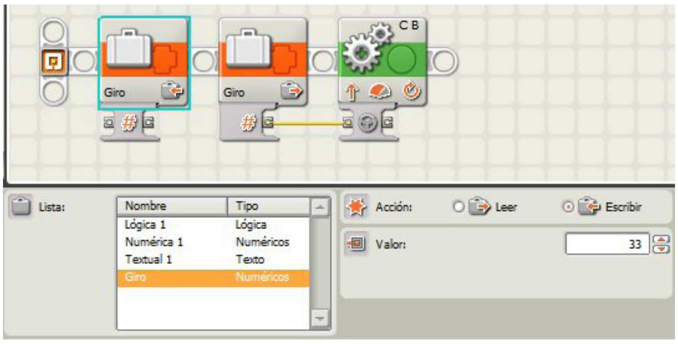
Improvement #1 Improving precision in the turns

The slider in the control panel for the Move block allows 10 different positions for turning left and 10 for turning right. This will be enough in many occasions, but in some cases one position will be too little and the next too much.

Turning can also be done with a numeric value between 0 and 100 in one direction and between 0 and -100 in the other. Remember that a value of 100 means the robot will turn in its footprint in one direction and with -100 in the other direction. To assign this value we will have to use a variable in which to store the number for the desired turn. After opening a new program we will go to Edit > Define new variable. We create a new variable called Giro (Turn - it will hold a numeric value).



In the following image you can see a fragment of the code that shows how to store a value in the previously defined variable and use it to establish the turning radius. The value of a variable can be changed at any time.



The first block stores the desired value in the variable "Giro". The second reads the variable and sends this information to the plug with the steering wheel on the hub of the move block.

Improvement #2 Ramping starting and stopping

While it is good practice to have a low centre of gravity, at times there may be circumstances that impede this. It isn't the case in the robot we are using, but in other cases starting and stopping may topple over the robot (as will happen to a motorcycle or a bike if you suddenly block the front wheel). One way to avoid this situation is to ramp up speed, i.e. in stead of going to full power at once to do so progressively until you get to the desired value. The motor block has this option, but since the Move block doesn't we are going to do it another way. Let's see the algorithm to make the robot increase speed progressively and reduce it in the same way.

1. Define a variable to store the value of the power and send it to the Move block.

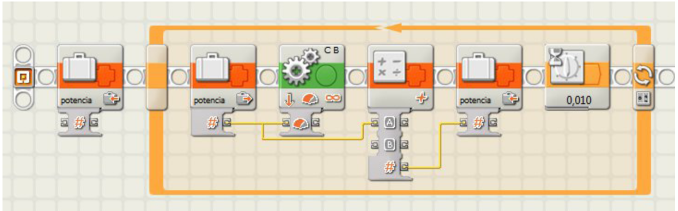
2. Give it an initial value of 0

3. Repeat the following 100times (to get to power level 100)
   a. Send the stored value to the Power plug on the Move block
   b. Add one to the variable
   c. Wait 0.01 s (this way, since we will repeat this 100 times we will go from 0 to 100 in 1s).

4. Advance at full power during 2 seconds.

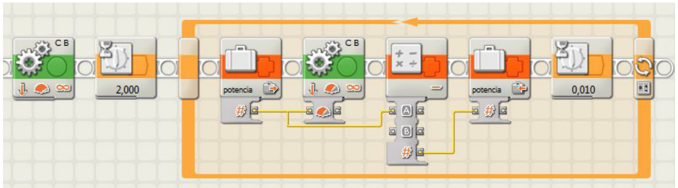
5. Repeat the following 100times (to get to power level 0)
   a. Send the stored value to the Power plug on the Move block (the last value was 100)
   b. Subtract one from the variable
   c. Wait 0.01 s (this way, since we will repeat this 100 times we will go from 100 to 0 in 1s).

After opening a new program we will have to create a new variable. I have called it "potencia" (power) and it is a Numeric variable (it will store numbers).
In the following image you can see steps 2 and 3 of the algorithm.



A value of 0 is assigned by the variable and the loop is started. In the control panel of the loop you select the option Count and enter 100. Internally, the sequence of blocks starts to assign the value contained in the variable to the Move block, add one, wait one hundredth of a second and start over.

The rest of the program:



To advance we will use the Move block with power level 100 and for duration the option unlimited. If instead of this option we were to use just the Move block with a duration of 2 seconds we would not be able to control the braking in this way. The rest is similar to the first loop, with the exception of the subtraction instead of the addition.

Proposal 1

It's time to practice: try to do the same without using a variable. To this end, look at what happens in the control panel of the Loop structure if you ckeck the box at the bottom of it.

Proposal 2

Although the power of the robot can be set between 0 and 100, the robot doesn't start to move at power level 1 as it needs a minimum power level to start moving which depends on its weight and mechanical design. If the robot uses different accessories for different tasks, like in FLL, this minimal power may vary. Develop a program to find out this power level and show it on the display.
#