2.4 Sensorized robots

2.4.1 Introduction

The main objectives (competences or skills to be acquired) behind the problems presented in this section are:

- The students have to know the features of the Lego Mindstorms NXT sensors (measured magnitudes, calibration, programming blocks, etc)
- The students should be able to design and build "good" robots, where the sensors are used and placed in the best possible places
- The students should be able to combine appropriately different programming techniques (Loops, If's, variables, user blocks, etc….) with the use of one or more sensors.
- The students should be able to decide which sensors to use and how to program them in order to convert their robot into a reactive robot (as adaptive and autonomous as possible)

2.4.2 NXT-G necessary programming Features

To work on the proposed problems, we will use the following NXT-G features: Loops , if, variables, Myblocks, Miscellaneous blocks (arithmetical, logic operations, time etc). But first of all, let us see the different possibilities of the sensor blocks and the programming options.

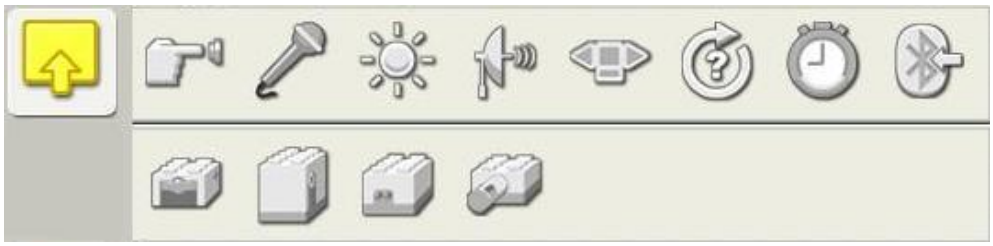

Figure 2.4.1*: The Sensor group of blocks*

The sensor group (fig 2.4.1) has several blocks (one for each sensor). These blocks allow the user (the programmer) to measure the sensor value and, for example, to write it into a variable (fig 2.4.2):

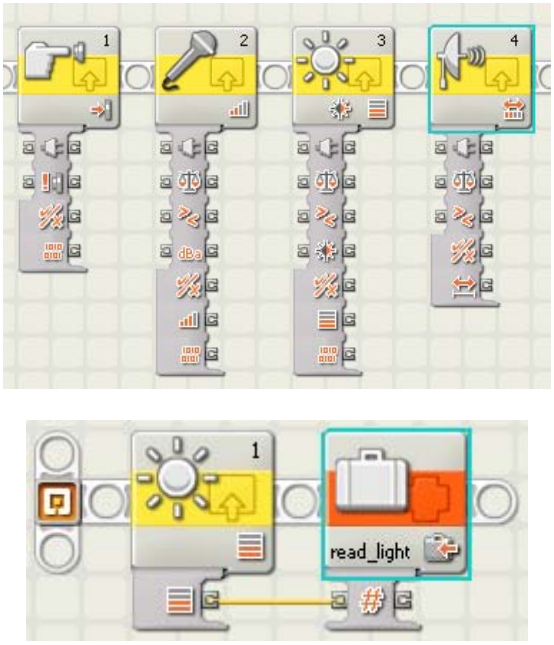

Figure 2.4.2: *Reading and Storing sensor values: Reading the sensor value of port 1 and storing it into the variable read_light.*

We can use also the sensors within a loop, as it is shown in the next figures:

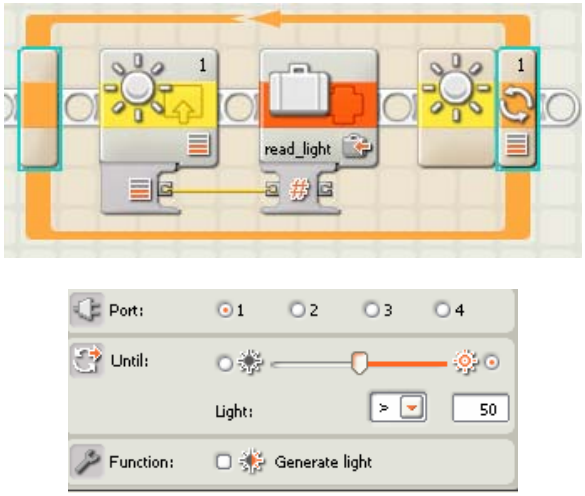

Figure 2.4.3: *Reading the sensor value of port 1 and storing it into the variable "read_light" until the read value of the light sensor (intensity) is higher than 50*

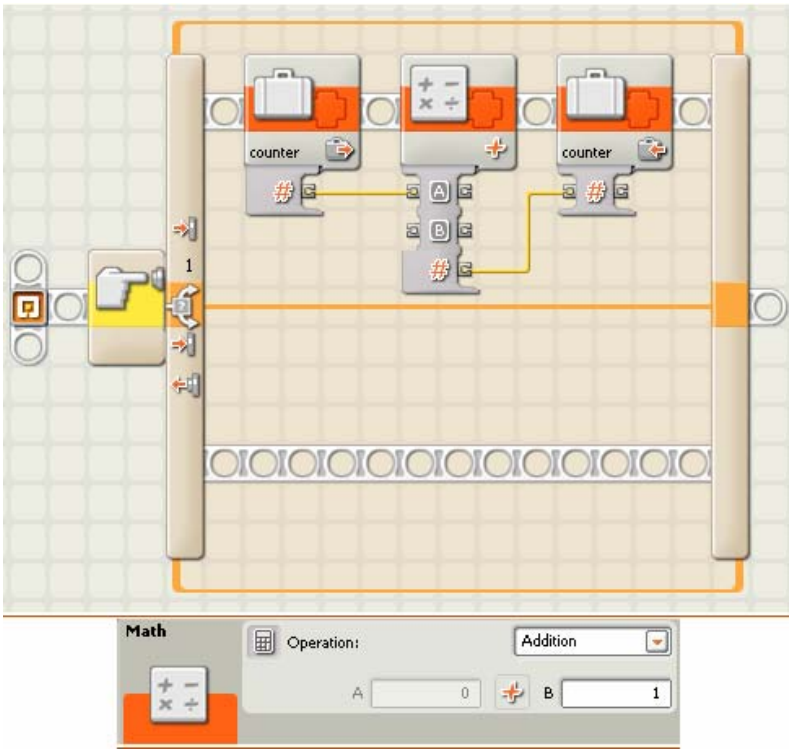

Figure 2.4.4: *Counting the number of times that the sensor touch is pressed*

2.4.3 The Tunnel problem

These problems aim at learning to work with the light sensor and to write a normal program (normally an infinite loop). The main motivation for the student is to propose to him/her to try to construct a mechanism for cars that make them possible to detect whether they need to switch lights on and whether they must switch lights off.

We propose the following sequence of steps:

a. To use the light sensor as a "photometer", just to get to know how the sensor works.

b. To design a program which detects when the light is needed or not (depending on the light sensor values received) and switch the light on or off correspondingly.

For the first problem, we are going to use the *Try Me* option of the NXT brick with one light sensor directly connected with it. Then, we need to use one robot with 2 motors and the already acquired competencies about using the motors for straight-line motion and steering purposes. We will use a bulb actuator, conditional statements (if) and loops. We could formulate our problem as follows: "How can we

construct an automatic device that could automatically switch on-off the light of our robot?"

The students should:

a. Try to understand how to interpret the light sensor values (intensity, reflected light, ambient light, etc)

b. Do that with the Try Me process, but also with a NXT-G program that can write the results on the display.

c. Understand the following combinations of blocks:

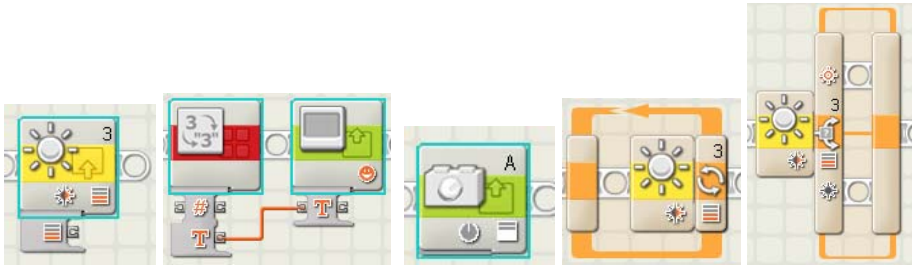

Figure 2.4.5*: NXT-G features needed to solve the problem*

The conclusion should be that it is possible to solve the problem for a given context as it is shown in the following figure, where the car/robot has to cross a tunnel and the goal is just to cross it turning the light on or off whenever needed. In order to generalize the use of such "security devices for cars/robots" we should integrate this "gadget" with others related to mobility (straight-line and turning motion) or to integrate other sensors in order to avoid obstacles and to be able to follow a path (surrounded by walls, for example).

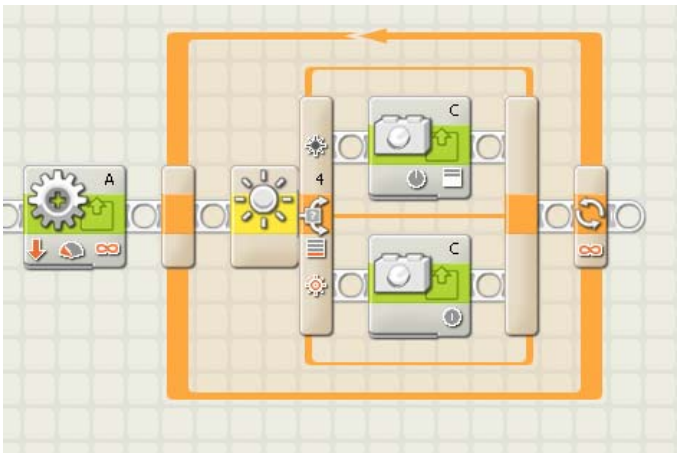

Figure 2.4.6: *a solution to the Tunnel Problem*

2.4.4 Counting black Lines on white background with a Light Sensor

The main objective is to use the light sensor to distinguish between black and white lines; this is an important feature in order to follow lines, or to keep the robot within a ring (a white one limited for example by a black line). In this case, we have to use a light sensor able to "read" light floor values and to react correspondingly.

We will use a mobile robot, for example the robot used for the tunnel problem. We need to place the new light sensor in the right position in order to read reflected light values from the floor. In fact the problem can be formulated as it follows: "How to count black stripes onto a white floor?"

We could undertake, for example, the following explorations: Calibrating our light sensor, counting one line, counting 2 lines, counting n lines, counting lines until some final condition is reached. We have to deal with the entire program setting in order to use the sensor, but, also, we need to store the "counting". We still need, as well, a kind of loop to "include" all our counting readings.

In the next figure we see one example of one program that counts n black stripes. This version of the program counts 3 black stripes (the external loop verifies this end condition). The robots go forward until he detects (with the light sensor) a black stripe. At this point it produces a sound and increments the value of a variable which is counting the number of black stripes and is directly linked with the loop.

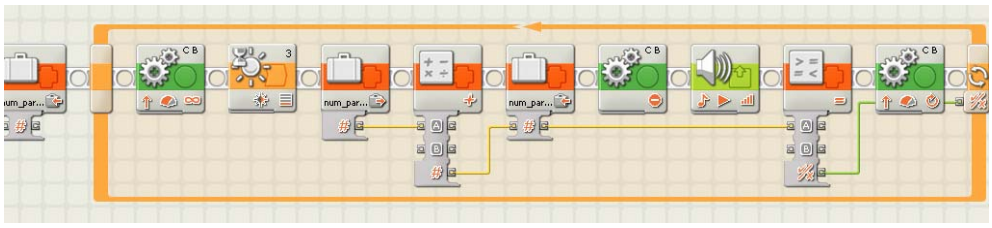

Figure 2.4.7: *a program counting a finite number of black stripes*

We can generalize this kind of programs: either finding an ending condition (for counting) or, maybe, making a "*MyBlock*" that counts a stripe every time it encounters one.

In any case, our robot is able to "read" black lines, which means it is able to detect if it goes out of or into a part of a circuit or similar, while, at the same time, this kind of behaviour (detection of black stripes) could be used to construct some kind of "language" (black on white) to tell the robot to do something special.

2.4.5 Detecting obstacles "My robot does not hit any obstacle"

In order to enable our robot to detect any obstacle before hitting it, we have to use either the ultra sonic sensor and/or the touch sensor. In the following examples, we suggest to use the commands "loop" and "switch", even if the "wait" command might also be used.

To solve this problem, students could use a touch sensor placed in the front part of the robot, which, when it is pressed, makes the robot stop. As you can see in the figure *2.6.a*, the program could be realized with two instructions:

- Feed the motor connected with port A
- Go on until the sensor connected with port 1 is pressed.
- After the last operation, stop the execution of the program

As you can observe in the figure 2.4.8 the command "move" is not posed inside the loop. We adopted this solution because, the command "move", if is posed in the loop, causes a discontinuity in feeding the motor every time the loop starts and the motor is turned on again. If the command "move" is outside the loop, the motor is turned on only once and then it is powered by the same value of power.

Motor (Port=A, Dir=FD, Pwr=75, PwrCtrl=OFF, Dur=FOREVER)

LP1:Loop(Ctrl=SENSOR, Sensor=TOUCH, Dis=OFF, Port=1,Act=PRESS) [LP1]

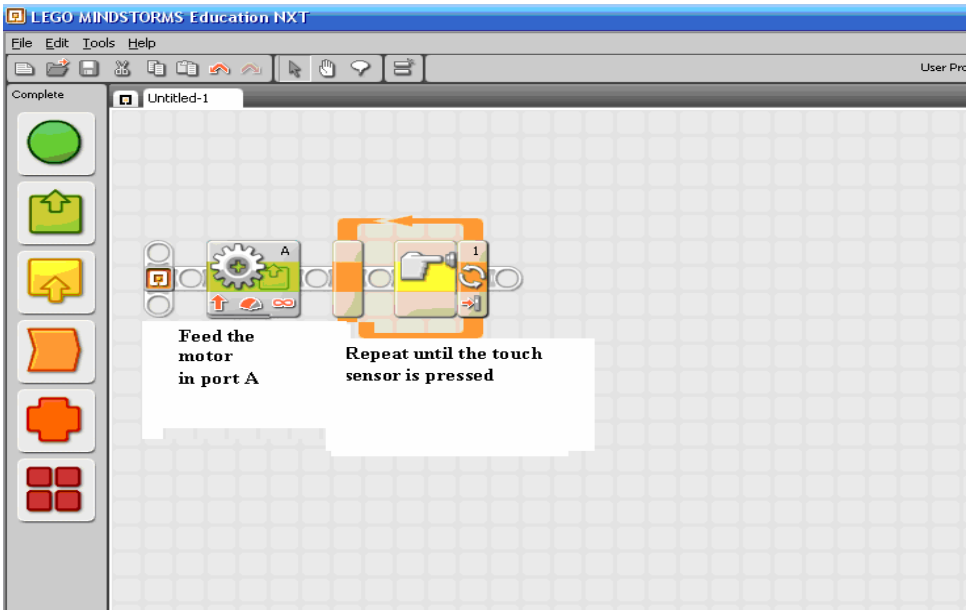

Figure 2.4.8: *"My robot does not hit any obstacle" program*

nt value is
old

The problem could be solved also by using an ultra sonic sensor and defining a threshold. The robot feeds the motor until the distance value is less than the threshold and then the execution of the program stops. The use of the light sensor is also interesting. In fact, this sensor can measure the light reflected by a surface. So students can use the sensor to "see" the light reflected by an obstacle.

The next step is to add another sensor to the robot, thus enabling it to reverse the sense of motion when it encounters an obstacle. In this way we can solve the following problem: *"a robot moving between two fixed points"*. To solve this problem, students can use different kinds of sensors attached to different points of the robotic trolley. They can choose, for example, an ultra sonic sensor for the front side and a touch sensor for the backside of the robot, thus enabling it to move forward until the distance is lower than a fixed threshold, to reverse the motion and to go back until the touch sensor is pressed.

After solving the problem with the two sensors, the students should think about a solution of the same problem using only one distance sensor or a light sensor. The distance sensor could be placed in the front part of the robot, thus enabling it to move forward until the distance is lower than a fixed threshold, and then reverse the motor and go back until the distance exceeds another fixed value.

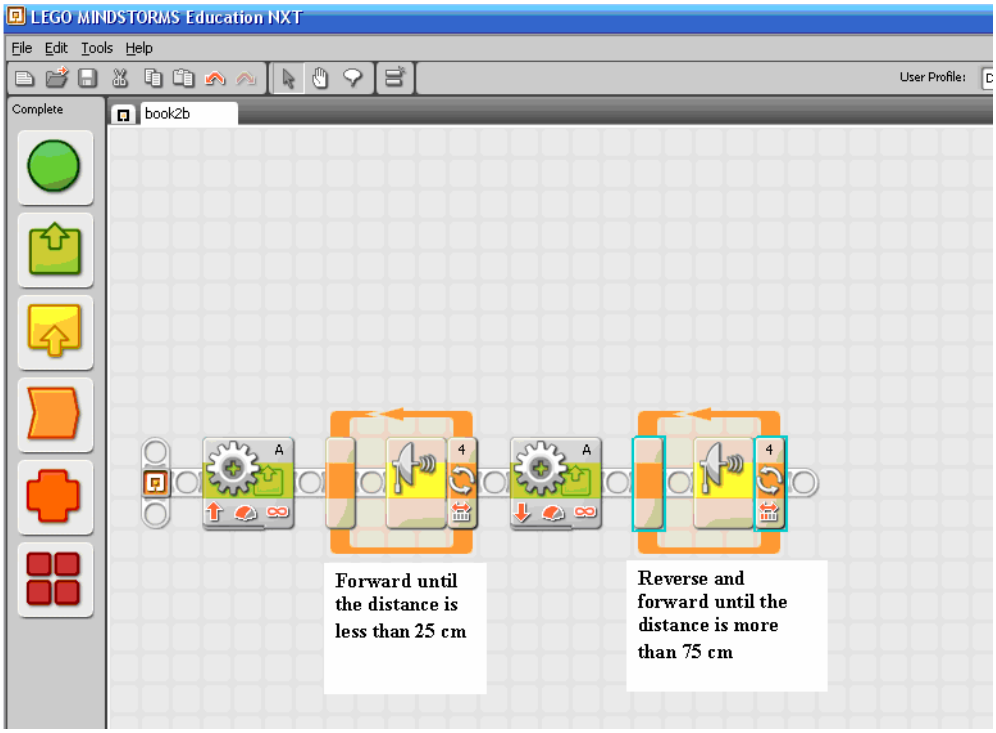

Figure 2.6.9 *Using only one distance sensor*

In figure 2.6.10 you can see a program for a robot moving on a white surface between two black lines. Note that, when the motor is reversed, the robot has to proceed for a brief step without interrogating the light sensor; in fact, because of its inertia, the trolley doesn't stop immediately, when it detects the change of light. So during the backward motion, it has to keep moving until it has moved away from the black line and then to start again detecting the light reflected by the surface behind it. Without the "blind" phase, the robot would detect immediately the same black line and remain on it. This is an example of another way to solve the same problem of the robot moving between two lines.

Another example of sensorized robot is the one which can change direction when it meets an obstacle. We can attach a distance or a touch sensor in the frontal part and program the robot to power the two motors at the same time on the same verse until the distance is lower than a fixed threshold or the touch sensor is pressed. Then the motors are reversed backwards for a short time, they are reversed forwards again with two different values of power for a second short time so that the robot moves avoiding the obstacle and finally it starts again the straight motion.

An interesting evolution of this problem is to program the robot to do "slalom" between some obstacles. It has to alternate turning right with turning left. The simplest solution is to add to the program an equal sequence of instructions and just change the turning direction in the last one.

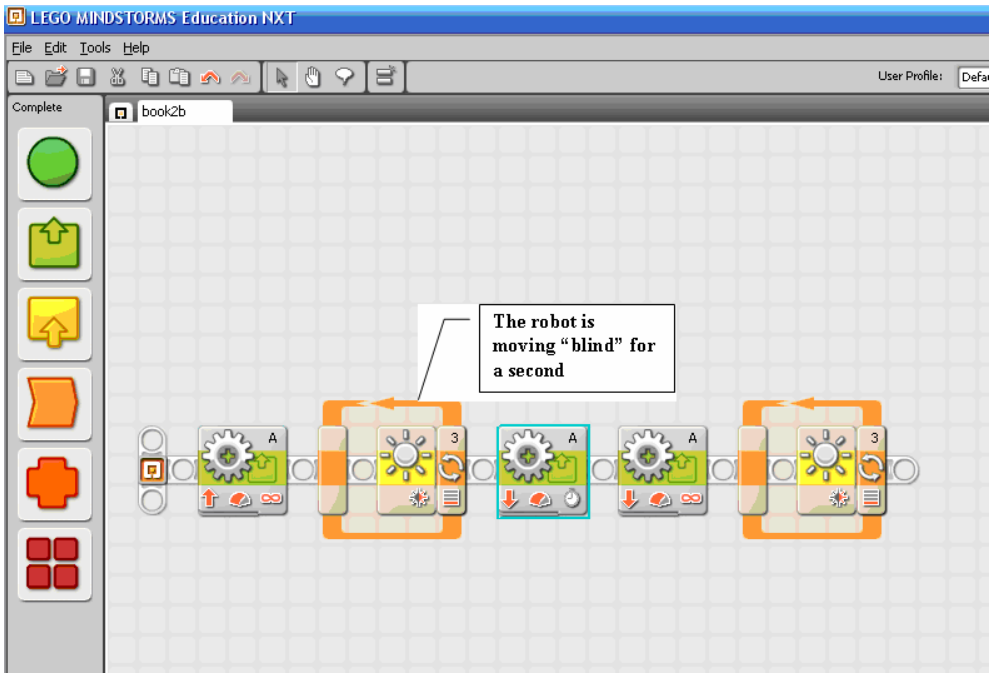

Figure 2.6.10 *A program for a robot moving between two black lines*

2.4.6 The Bee Problem

A particular example of complex turning robot is the “Bee robot”. The goal is to create a robot based on a strategy used by animals looking for food. The example was developed by the educational section of the Town Museum of Rovereto (Italy). Some researchers at the museum are interested in the behavior of insects. So, they projected an educational activity to study the bees simulating them by robots. The adopted strategy consists of tracking circular paths increasing periodically the radius until a flower is found.

In our activity the flowers are black spots fixed on white floor. We propose that students build the simplest robot they can imagine. So, the final model may be a robot with two-motorized wheels and a half sphere as a third touch with the floor. The most important problem is to translate the strategy in a program for robots. We can divide the problem in sub-problems:

- The robot has to move in circular motion
- The robot has to increase the radius of the circle
- The increasing of the radius must be a periodical action
- The robot has to be able to recognize a flower

We can solve the first problem feeding the motors in the same verse with different values of power. We can increase the radius increasing the power of the slower motor.

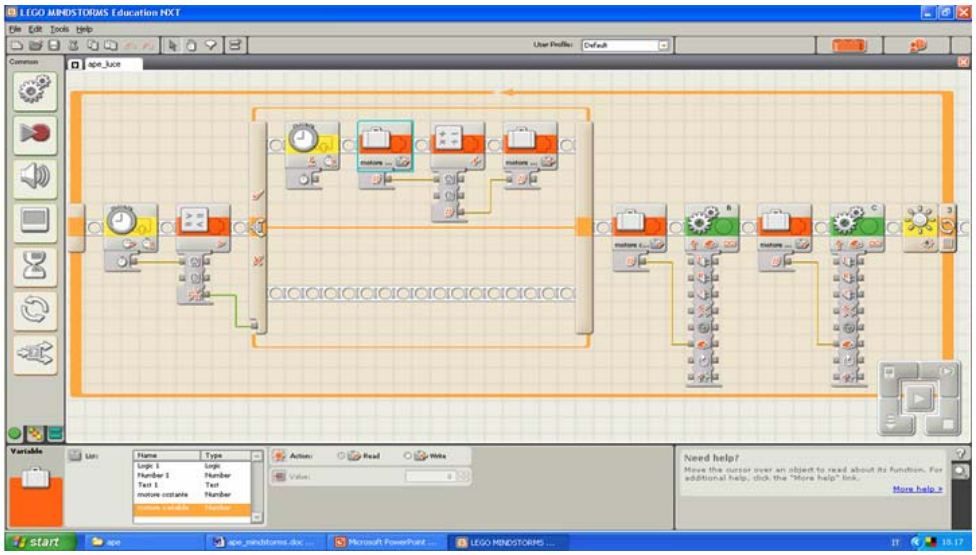

Figure 2.6.11 *A solution for the bee- problem*

The most intuitive method to make this change periodical is to wait until a period is over and then to increase the value of the power for a constant quantity each period. So we have to save the actual value of the power on a variable which is increased every period. In order to “look for “the flowers, the robot can use a reflected light sensor pointed to the floor.

In the program shown in figure *2.6.11*, the two starting values for power are 80 and 40 percent. The robot moves, every 5 seconds the lower power is increased by 5 units and the process is repeated until the light sensor measures a value under the threshold of 40% of the saturation.

We can observe that the problem of tracking circles with regularly increased power could be an interesting topic for maths activities; in fact, it might be useful to study methods to approximate the spiral by circular arches of increasing radius. Consequentially, this kind of robot-enhanced activities might be useful to study general methods whereby to approximate mathematical curves.