The Line Follower robot, *"Samianator"*

By: Vogiatzis Dimitrios, Vasileias Panagiotis, Konsta Eleni, Papatheofanous Themistoklis

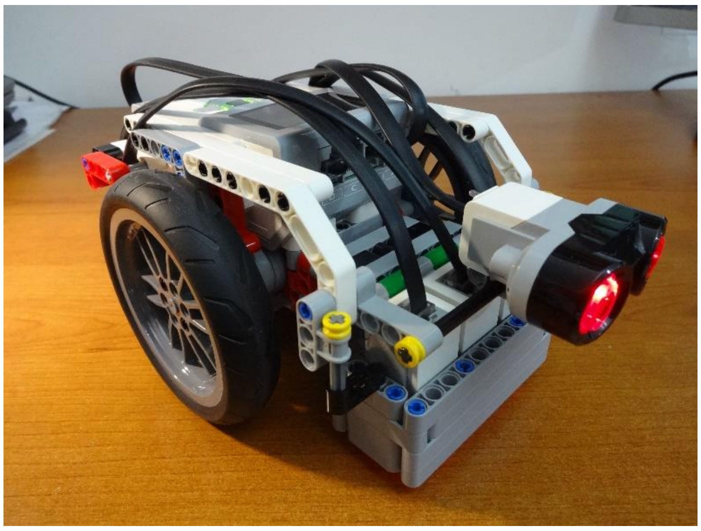
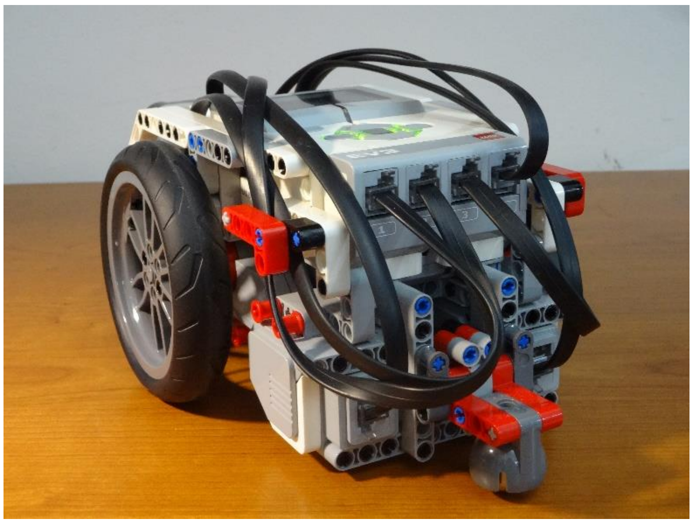
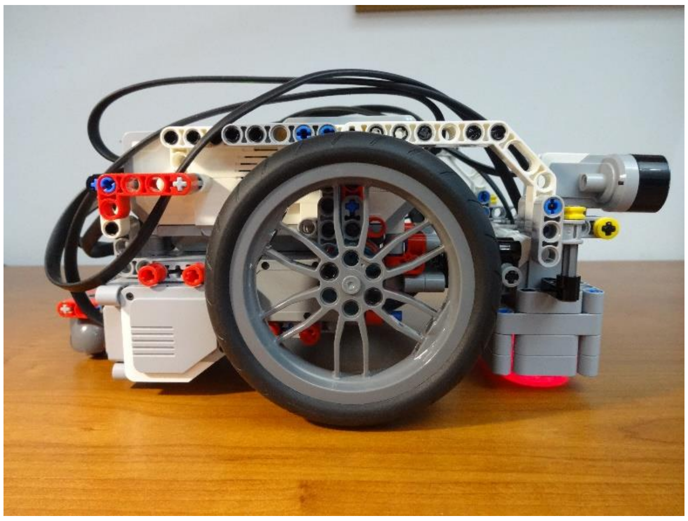
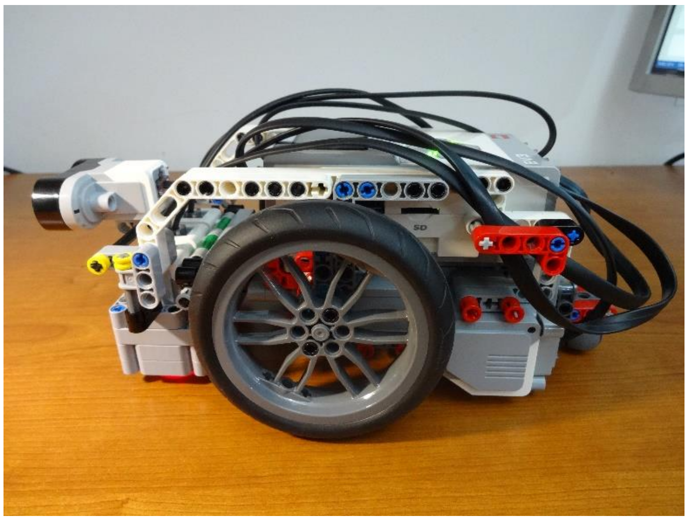
Instructor: Mparekos Vasileios 2o Gymnasio Samou, GREECE

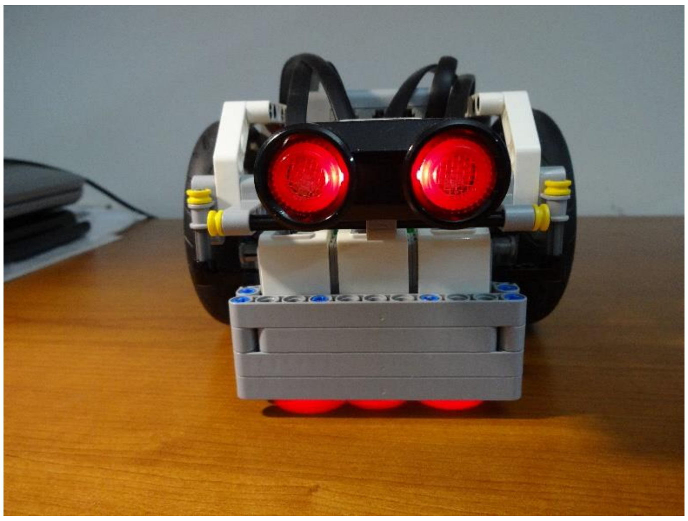
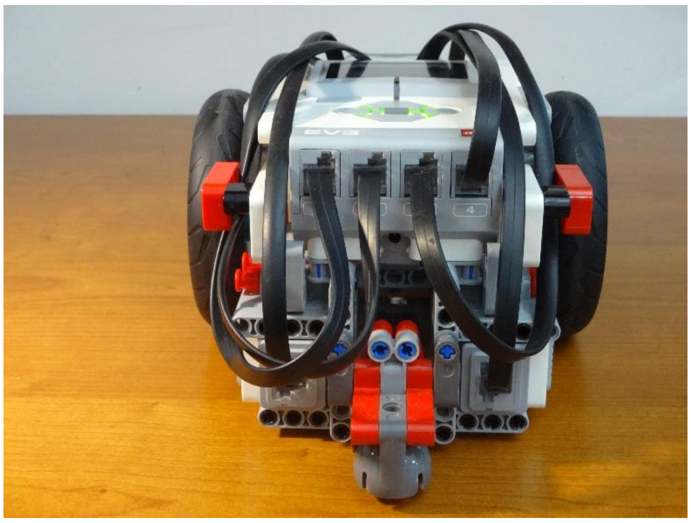
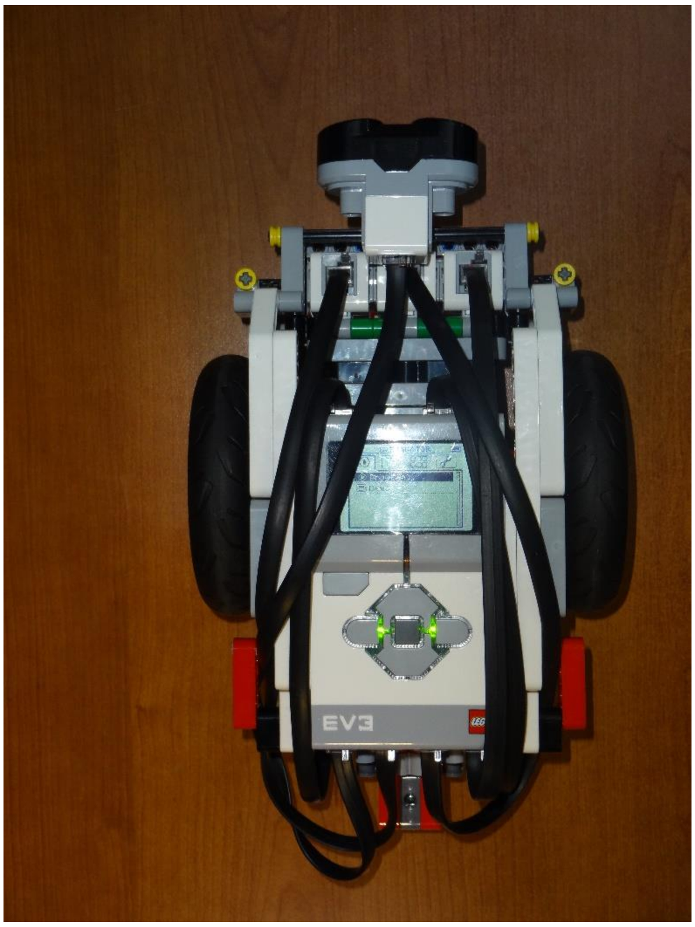
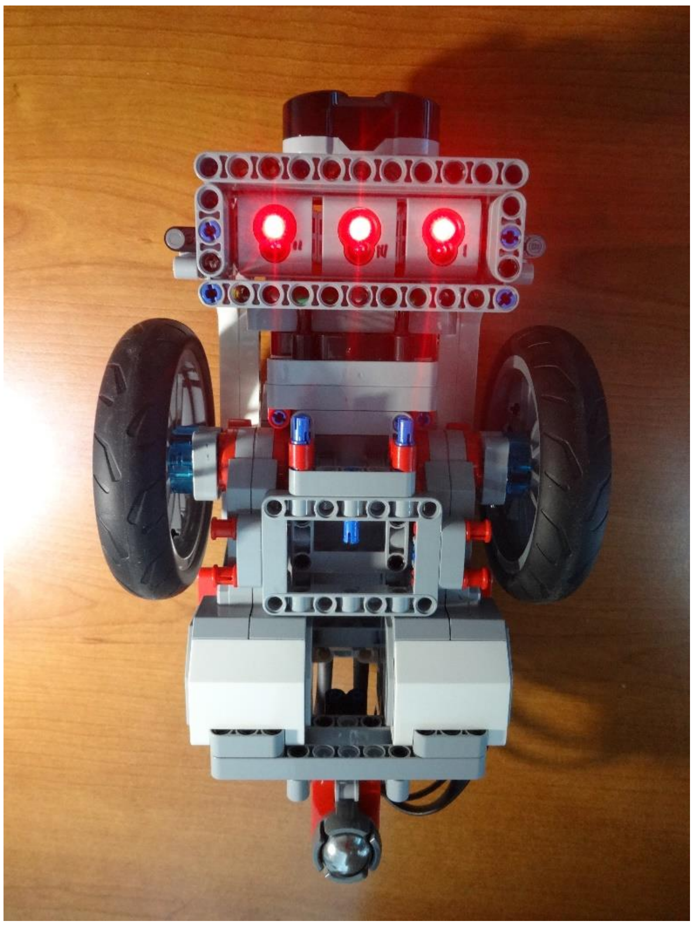
*A Participation in the **Robotic Day 2017** international event for the Line Follower competition.*

ID #131: Samianator **(kit: Lego EV3)** **TEAM: Samians**, ORGANIZATION: 2o Gymnasio Samou (2nd Junior High School of Samos), HOME: Samos, GR

Our LEGO EV3 Line Follower robot follows the black line that has several curved turns and may split and re-join. It uses the 2 LEGO Large motors in reversed position with big LEGO wheels.
The robot can also cope with track obstacles like objects laid on the track or line cut-offs.
For the line following we use three EV3 Color Sensors that measure reflected light intensity. To avoid obstacles a single EV3 Ultrasonic Sensor is used.
Robot size 14.5(w)x10.3(h)x25(l) cm. Weight 832 g. Tire diameter 94 mm.

Our final Design

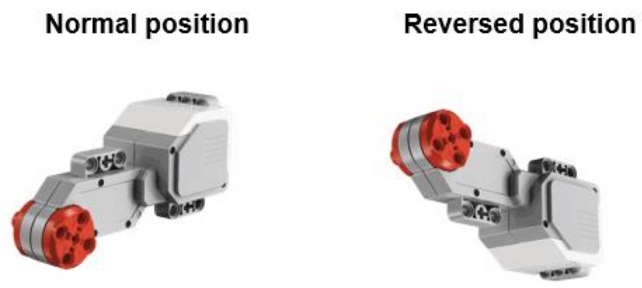





Design Issues

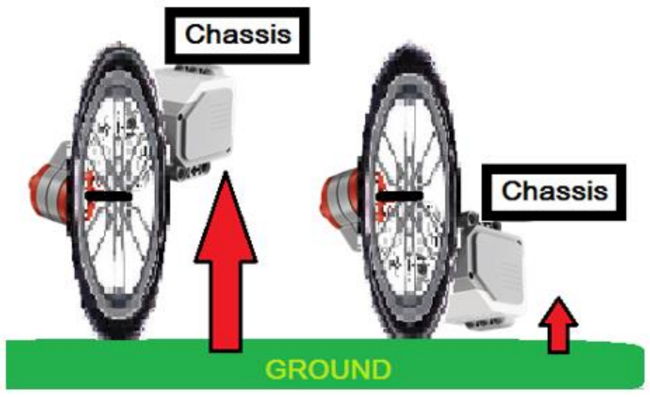
Kits used

Lego MindStorms Education EV3 Core set (**45544**) comes with just one Colour Sensor. For our design, we bought separately two more colour sensors. We also used parts from the Lego MindStorms Education EV3 Expansion Set (**45560**). We have chosen some big tires from the Lego Technic (**42036**) Street Motorcycle so that our robot can move faster due to the larger wheel radius without having to increase the motor power too much thus preserving its battery life.

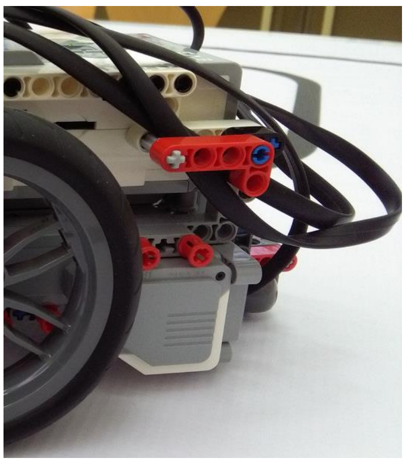
Motors



In our robot design, the 2 LEGO Large motors are used in a reversed position, so that we have a chassis that is positioned as lower as possible to the ground. Lowering the center of gravity and overall height of the robot increased speed and maneuverability. Due to the reversed motor position all motor powers inside the EV3 code are of negative value.



Organize the Wires

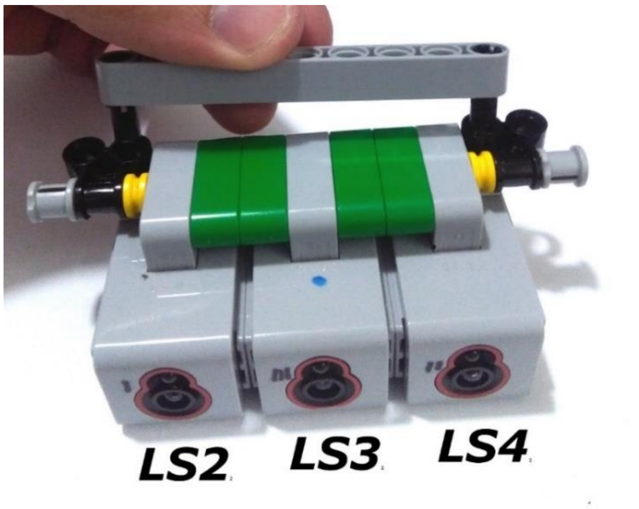


Using all four sensors (3 Colour Sensors and the Ultrasonic) on the EV3, plus the two motors, results in 6 wires around the robot. Cable management is needed in order to avoid cables interfere when the bot moves around. We used “wire barrettes”.

EV3 Colour Sensor

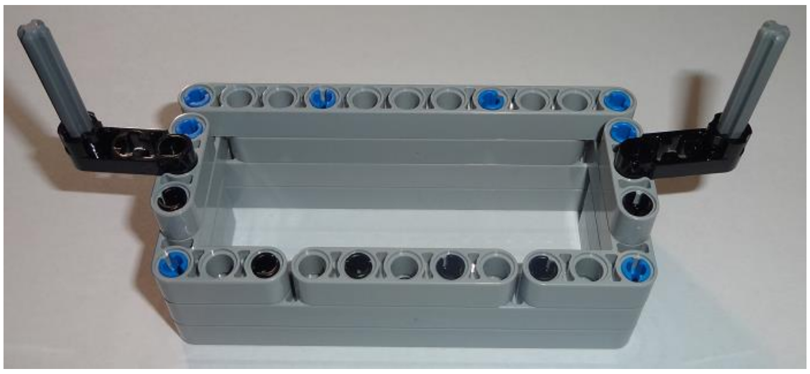
Used as Light Sensor by detecting reflected light intensities. It can tell the difference between colour or black and white, or between blue, green, yellow, red, white and brown.

Line Following with 3 Light Sensors



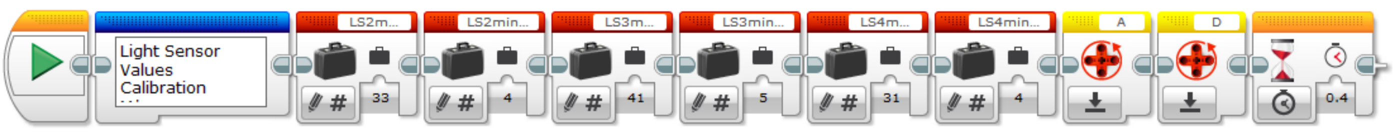
For our Line Follower we used 3 Light Sensors, named **LS2**, **LS3** and **LS4**, all of them aligned in series close together. The light sensors need to be perpendicular to the game field and at a relative low distance from the track. We positioned them at about 1.8 cm distance from the game field. The stacked together light sensors allow for a wider tracking area coverage over the black line. This means that our robot will not lose the black line so easily. The sensors are stacked together as tight as possible since we wish the black line to be always detected, even partially, by any of our 3 light sensors. In other words, if all our 3 light sensors lose at the same time completely the track's black line, then our robot is off the track and special care has to be taken so that our robot manages to seek & recover its path!

Shielding the Light Sensors is important because ambient light in the facility where we compete may cause variable results to our sensors. This is especially true if we follow a line and make several runs at different times of the day. A special LEGO shield was built around those sensors as shown below.



Calibration of ratings

When in a new Venue or track, a sensor calibration has to take place in order to record light intensities "seen" by each of the 3 sensors on the White and Black colours. This takes place manually (ie LEGO Port View function can be used) and all values, for all our 3 sensors, are stored in variables as shown.



Normalization of ratings

Every LEGO Light Sensor is unique meaning that there are considerable value variations in measurements taken between different LEGO sensors. Normalization is the process of adjusting values measured on different scales to a notionally common scale, in our case 0 to 100. Eliminating light measurement variations, results to a more robust line follower. In order to bring all 3 light sensors we use, to the same scale and also have a wide value range ie 0 to 100 of readings, the following scheme is used.

$\text{Max}_{\text{value}}$: is the Light measured on White colour

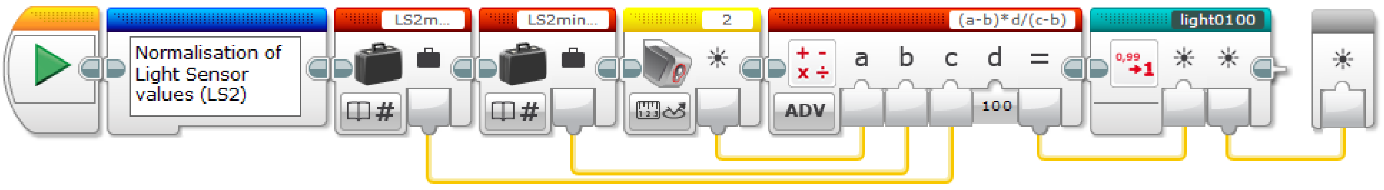
$\text{Min}_{\text{value}}$: is the Light measured on Black colour

Normalised Light = $\dfrac{light - Min_{value}}{Max_{value} - Min_{value}} * 100$

An Example:
for light measured = $\text{Min}_{\text{value}}$, numerator becomes $\text{Min}_{\text{value}} - \text{Min}_{\text{value}} = 0$

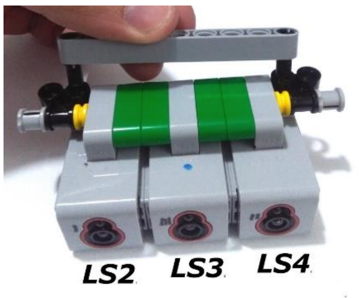
and for light = $\text{Max}_{\text{value}}$, becomes $\dfrac{\text{Max}_{\text{value}} - \text{Min}_{\text{value}}}{\text{Max}_{\text{value}} - \text{Min}_{\text{value}}} * 100 = 100$

EV3 code



Block named «light 0 to 100» is used to truncate values after the normalization, to range 0 to 100 exclusively (in case out of range numbers appear after the calculations), and also to round the result to an integer value.

Line Following with 3 Light Sensors – Error Calculation

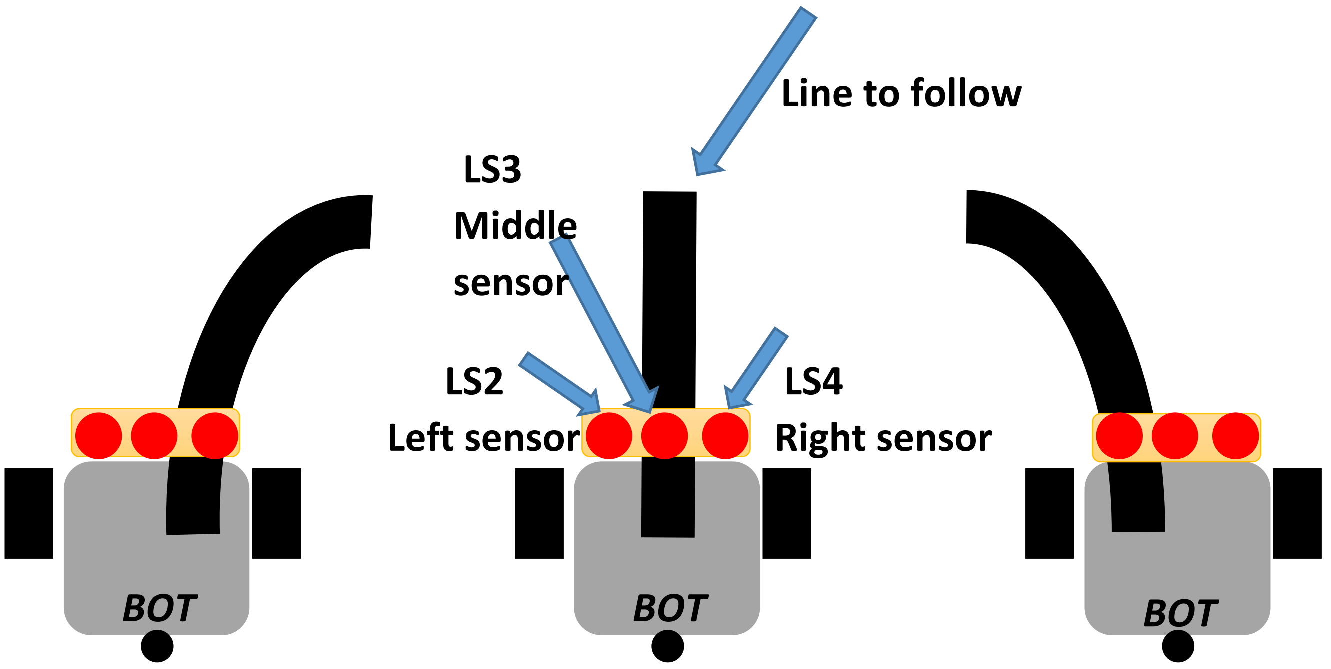


For our Line Follower we used 3 Light Sensors.

A simple approach to Line Following is using a proportional system. This is a system that uses a bit of math to calculate the amount of correction that is needed to get the robot back on the line that is following. Instead of using a set value to correct the direction of the robot (ie 0 left, 1 right, 2 straight), the actual direction change is calculated based on the values read from the light sensors. If the error value is small, the robot corrects very slightly, whereas a larger value results in a stronger correction. For our Error Calculation approach we used the formula.

Error = (LS2 – LS3) + (LS3 – LS4)

An **Example** of our **Error Calculation approach** follows.



action TURN RIGHT

Error = (100 – 90)+(90 – 10)=

= 10 + 80 = 90

action MOVE STRAIGHT

Error = (100 – 0)+(0 – 100)=

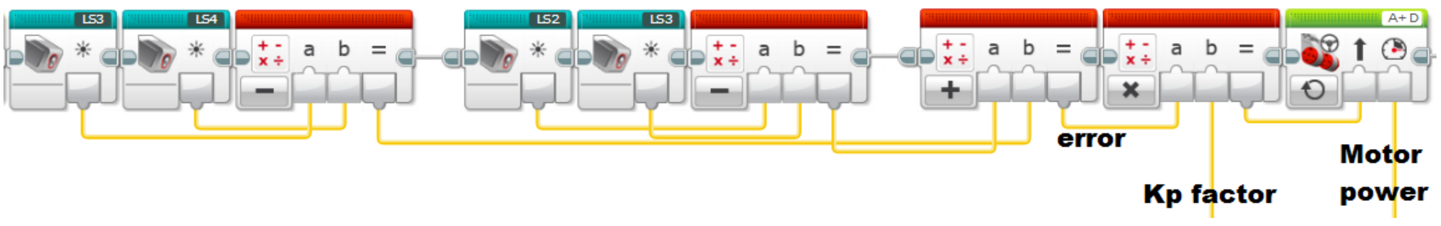
= 100 + (- 100) = 0

action TURN LEFT

Error = (10 – 90)+(90 – 100)=

= -80 + (-10) = -90

Our Error is the calculated sum of differences of the light intensities. This, in a proportional system, after being multiplied by a suitable Kp factor can become the steering value for the motors as shown in the EV3 code below. In this, the LS2, LS3 and LS4 blocks are the corresponding My Blocks, created for the normalized ratings of the 3 sensors. The Kp is an arbitrary value, the higher its value the steeper the steering. Obviously for a track with a lot of straight track lines Kp can be low and motor speed high, whereas for a track with curved lines the motor speed is lower and Kp increases as curve diameter decreases.



PID Controller

Line Following is achieved as robot moves and always trying to minimize the Error value to zero (that is the black line to be positioned in the Middle sensor). As a controller we decided to use the **PID algorithm** that is more sophisticated than a simple proportional system and provides smoother line following when it is fine-tuned correctly.
Our PID controller will also try to make **Error = 0**.
The term PID stands for Proportional, Integral and Derivative, ie it has 3 parts! All three parts contribute to the final Correction value.

The Proportional part (P) is already explained. Basically it is the Error calculated, multiplied by a Kp factor. P = Kp * Error
If Kp increases the robot takes sharper turns, if Kp decreases the robot takes smoother turns.

The Integral part (I) is calculated as: I = I + (Ki * Error) . ie the sum of all "error" values.
It is used to solve the problem where the output of a control system never reaches the goal (ie it corrects the steady-state error fast enough). By summing the total error over time, the output must reach the set point with some overcompensation, ie the Integral term allows the output to reach goal. This accumulated error when the robot drives a straight line should be zero, this is because slight movements to the right produce a positive error and slight turns to the left of the line produce a negative error. When all these add over a long period of time, for a straight line movement, should add to zero.

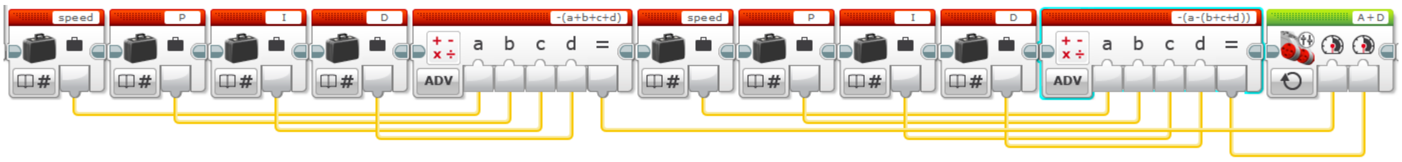
The Derivative part (D) is calculated as: D = (Current Error – Previous Error) * Kd
It uses the past and current error values to anticipate the next error. The Derivative term decreases oscillation especially caused by disturbances, thus allowing our robot to limit its annoying squiggly motion.

Kp, Ki, Kd are arbitrary values used to scale the Correction of the robot.

Correction = _P_ + _I_ + _D_

In our code this Correction value is applied to our motors Power values to the Motor block. The trick is to invert the value between the two motors, so for our Left Motor we add the Correction to the Power, and for our Right Motor we subtract the Correction from the Power.

Left Motor Power = speed **+** (**_P_** **+** **_I_** **+** **_D_**)   **Right Motor Power =** speed **-** (**_P_** **+** **_I_** **+** **_D_**)



The last thing is of course tuning the PID controller ie discovering the optimal Kp, Ki and Kd. This is a trial and error operation that can take many iterations. The values of Kp, Ki and Kd will depend on the actual robot design and the type of track. The procedure we followed for tuning is:

- First set Ki = Kd = 0 thus temporarily eliminating the integral & derivative parts of the correction, leaving just the proportional part.
- Then increased Kp until the robot can cope with the curve turns and until oscillation, then decreased it a bit.
- Having set Kp, it's the turn of the Ki. We increased Ki until loss of stability, then back off.
- Finally increased the Kd to increase performance in response to disturbance, ie until robot oscillations disappear.

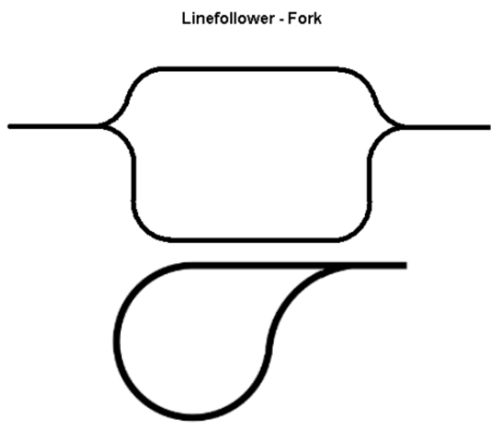
Consolidate Logic into My Blocks

A My Block is a combination of one or more blocks of code that are grouped into a single block. Once a My Block is created, you can use it in multiple programs. The My Blocks function in EV3 programming is useful because it:

- reduces the space needed to run programs, thus makes our program using less memory,
- is a good programming practice because it makes the code reusable. Making a change in one program will automatically update all the other programs that use the same My Block,
- makes a program more readable since it fragments the code to simpler blocks and at the same time the number of blocks in the program is reduced,
- allows to build fully custom code, since various value parameters can be passed into a My Blocks.

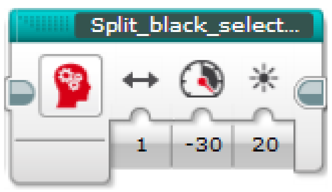
Playing field

Black Line Splits (Forks)



According to the competition rules, the black line may split and re-join at some points.

To tackle this we created in our program a My Blocks block with the name “Split_black_selection”. What it does is, when a line split is detected, the robot selects either the right or the left path. This selection is preconfigured with the parameter of the block named direction. When is set to 1 the right path is selected, when set to -1 the left one. In the code this takes place with a switch block that simply changes the power to the motors so that the robot turns accordingly. The other 2 parameters passing in, are the motor speed variable value (say 30) and the light intensity threshold value for the black line detection (say 20).

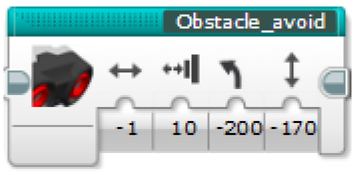


The detection of a split of the black line takes place by our two far most located sensors (LS2 and LS4) when they both “see” black line at the same time. Also special care is taken for the rare condition that all 3 light sensors “see” black line at the same time (this may be the case when the robot is moving perpendicular towards a black line or when the line split junction is very “thick”), in these cases the robot has to turn either left or right to recover the track.

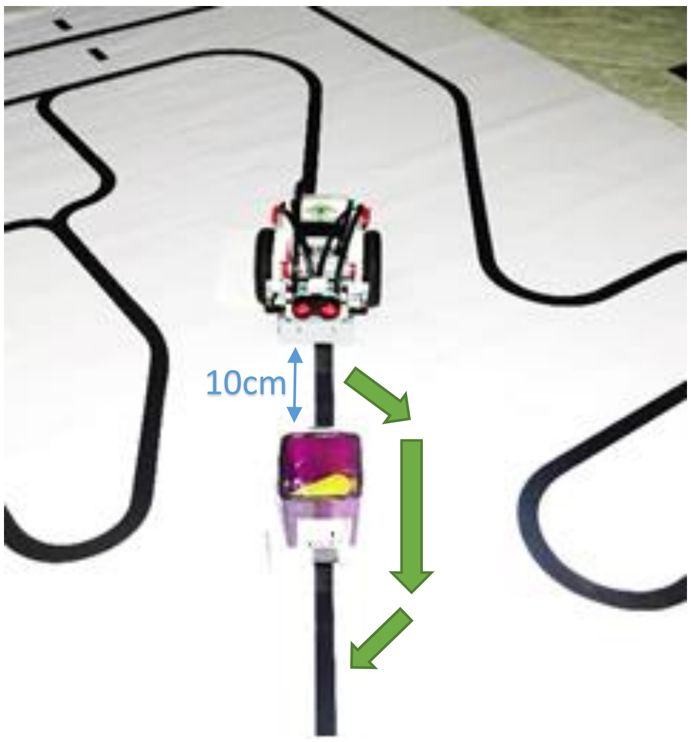
Obstacle Avoidance - EV3 Ultrasonic Sensor

At the competition, obstacles will exist that are laid on the track. Minimum obstacle size is 10(w)x8(h)x2(l) cm and the robot has to bypass the obstacle without moving it and then recover the black line at latest after 30 cm from its end. For the detection, one **EV3 Ultrasonic Sensor** was used, positioned in the middle front side of the robot, at a height from ground less than 8 cm, so that it could detect without problem an obstacle of the minimum height. This sensor measures distances between 1 and 250 cm with an accuracy of +/- 1 cm.

The basic principal of an Ultrasonic Sensor is that generates sound waves and reads their echoes to detect and measure distance from objects. It can also send single sound waves to work as sonar or listen for a sound wave that triggers the start of a program. In our case it is used to measure the distance from the obstacle.



All obstacle avoidance operation is taken care inside the relative “Obstacle_avoid” My Blocks block with the parameters passed to it as shown below (direction_of_turn, distance_from_obstacle, degrees_to_turn, far_out_distance_in_cm).



For example, when the sensor detects an obstacle in front, in a distance equal or less than 10cm, then line follow stops. Then the robot turns a predefined degrees angle (say 200) and then leaves the black line for a distance in motor degrees (say 170). These values are parameters since the actual size of the obstacle is unknown. Using these parameters our robot bypasses the obstacle by following the route depicted with the green arrows.

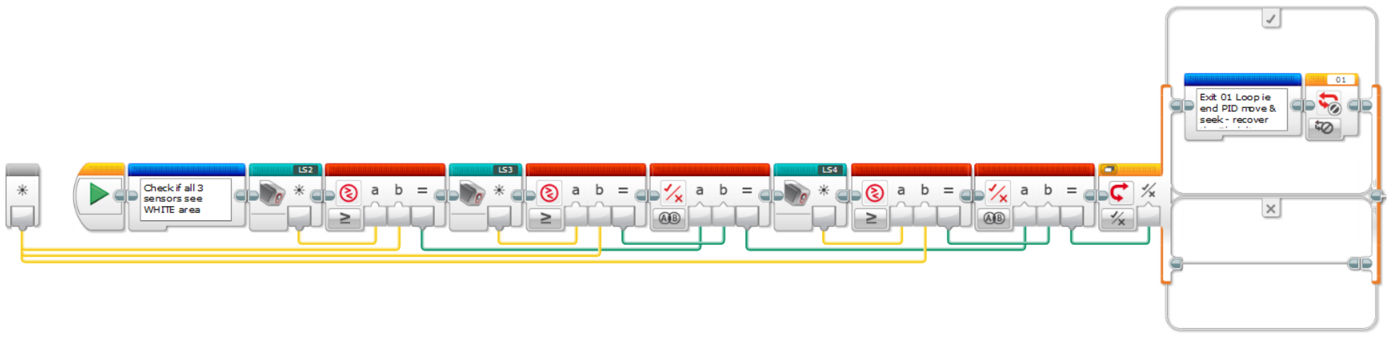
In the competition that has several tracks, the direction of the track is not known in advance. That means that in some tracks, the movement of the robot will be clockwise and in some others anticlockwise.

The reason this is mentioned, is that we wish our LEGO robot to bypass the obstacle from the inner side of the obstacle (towards the track) otherwise if it moves along the outer side of the obstacle, we may hit the playing field border. This is because rules state that minimal distance between the line and playing field border is 15 cm, that means that there is no space left there to bypass an obstacle outside (our robot width is 14.5 cm).

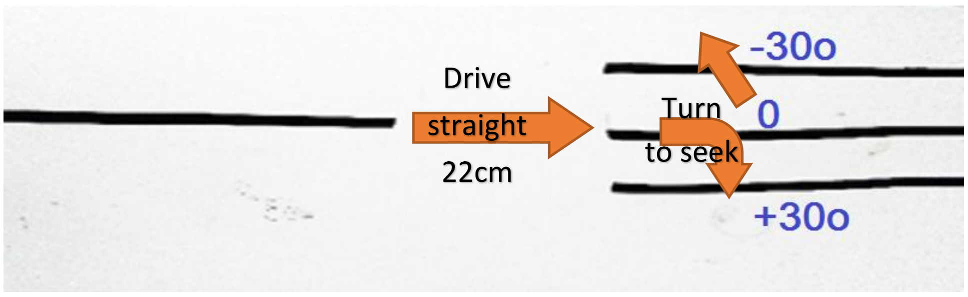
Again it is a matter of setting in the code to tackle the obstacle bypassing from the left or the right side of it. This is taken care by the parameter direction_of_turn. When a value of -1 is set to it, then it is a left bypass as depicted by the green arrows. When a value of 1 is set, then a right side bypass is performed.

Line Cut-off

The line of the track may be discontinued at some points. In our algorithm, when all our 3 light sensors of the robot detect white area (ie light intensity > 80), then the line follow using the PID algorithm stops and the robot goes to a procedure to seek and recover the black line. Below is the logic of our block “No_line_stop” that detects a line cut-off.


Check if all 3 sensors see WHITE area
LS2
LS3
LS4
Exit 01 Loop ie end PID move & seek - recover
01

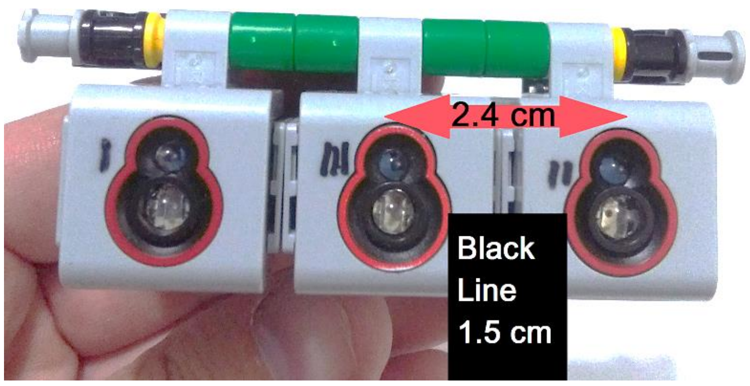
According to the contest rules, line cut-off can be of a maximum distance of 20 cm. After the gap, the line may continue anywhere within $\pm 30^o$ from the original direction. These extreme line repositions are illustrated below. To tackle this, our robot first drives straight for a safe distance of about 22 cm and if the black line is nowhere to be found then it has to search for it either right or left of this point. This search is done be turning the robot right & left, until it meets the black line.



Robot Testing – Black Line Width

Robot Testing was performed to several tracks as time permitted. Black line width is a key factor. For LEGO sensors, the standard tape width is about 20mm (wro2017.org , robogames.net). For this competition, the Robotic Day 2017 (roboticday.org) rules for the Line Follow are that the line is approx. 15mm. Thus we also tested our robot to some tracks with a black PVC tape of 16mm width, where we observed a poorer line follow performance when compared to tracks with wider black lines.

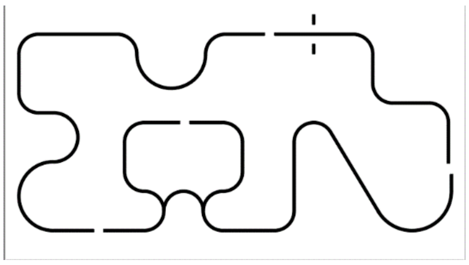
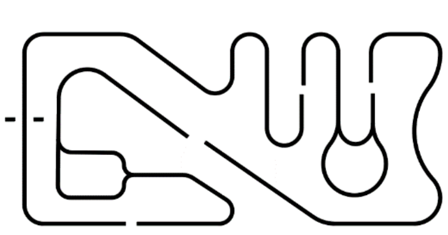
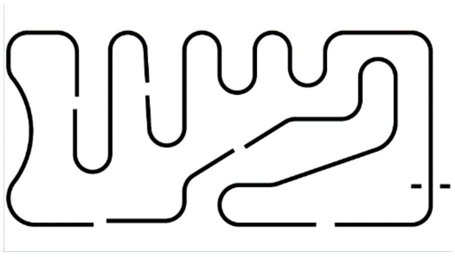
This can be explained from the fact that the LEGO Light Sensors are very thick by construction, and they cannot be placed one next to another in a distance less than 2.4 cm apart, as shown in the picture.



This means that if the black line is too narrow, the sensors have to be moved to a higher height from the ground, so that the sensors light red spots do not leave a blind spot in-between and in this way ensure that the line is always detected when in-between the sensors. If the sensors are not high enough from the ground, when the black line is going between the 2 sensors it may become undetected! Of course there is a design limit on the maximum height of the sensors from ground so that they can detect things and perform reliably! Our team tried to find the optimum solution to this problem.

This is not a problem if someone is using just one LEGO Light Sensor and doing a line edge detection, but it is in the case that someone tries to do line following with 2 or more sensors. Again it is not a problem for the Arduino type of robot designs and their sensors, since these sensors are tiny and numerous (ie arrays of 15 or more sensors are used) and are stacked very close together.

Some of our testing tracks follow. The first 3 tracks were challenges of the **Aegean Robotics Competition 2017** in Samos, Greece.

Some more of our testing tracks follow. Obstacles have been placed to them at several places.