The DIY automobile project

Project description and guidelines for teachers for circuit making and programming

Content

General Approach

RoboScientists aims at engaging secondary school students in robotic artefact construction through interdisciplinary in nature projects. The set of the projects (that are going to be carried out) offer students opportunities to explore different aspects of the field of Science, Technology, Engineering, Arts and Maths. Crafting/ handcrafting is a pivotal point in all the projects. Through the crafting process (highly interwoven in the robotic artefact construction) it is likely that the students will explore a number of engineering and design concepts, confront challenges and consider multiple solutions in order to achieve the results that they want.

About the DIY Automobile project

This project revolves around the creation of a robotic artefact that can freely move around space. For this reason, this artefact will be called automobile. Apart from moving around space, the automobile will be also able to detect and avoid obstacles, perform specific movements, as well as being remotely controlled. The following image (*Figure 1*) is indicative since their main goal is to graphically illustrate the concept of the project. In details, the basic structure of the automobile is consisted of two wheels that are powered by two DC gear motors, a caster that assists the mobility and the stability of the artefact, a driver for the motors as well as a Shield that expands Arduino board, facilitating the circuit making process.

Gradually, two more modules are added to the Automobile, namely the Ultrasonic sensor which will allow the artefact to detect obstacles, and the Bluetooth module that will facilitate the remote controlling operation. All the aforementioned components should be firmly assembled in a robust structure, highlighting the advanced needs as far as crafting is concerned, while reflecting upon engineering concepts and issues. Alternative scenarios as far as power supply is concerned are implemented, raising awareness and questions relevant to ecology and ecological energy solutions.

In general, the project is related to emerging issues in regards to engineering and ecology and consequently brings up issues related to environmentally friendly solutions, innovation and engineering, as well as environmental policies, citizen engagement and smart cities (i.e. creating solar cars, creating remote controlled devices that can reach areas of low accessibility etc.). Also, it is related to alternative learning methodologies that tend to engage students through rather playful, but highly interrelated to physics and mathematics, scenarios and procedures (i.e. finding ways to instruct automobile to move on specific shapes, angles etc.).

When developing the designing and setting up the project, it is important to have in mind the presentations provided and discussions that took place during the training sessions. It is recommended to the educators that for a project to be delivered it is important to: employ the “makeology approach”, work in teams, encourage experimentation, involves crafting and coding, apply the Engineering design process is employed, encourage sharing, employ the STEAM approach, design and develop robotics models and artefacts, use various tool, equipment and materials, involve students as makers.

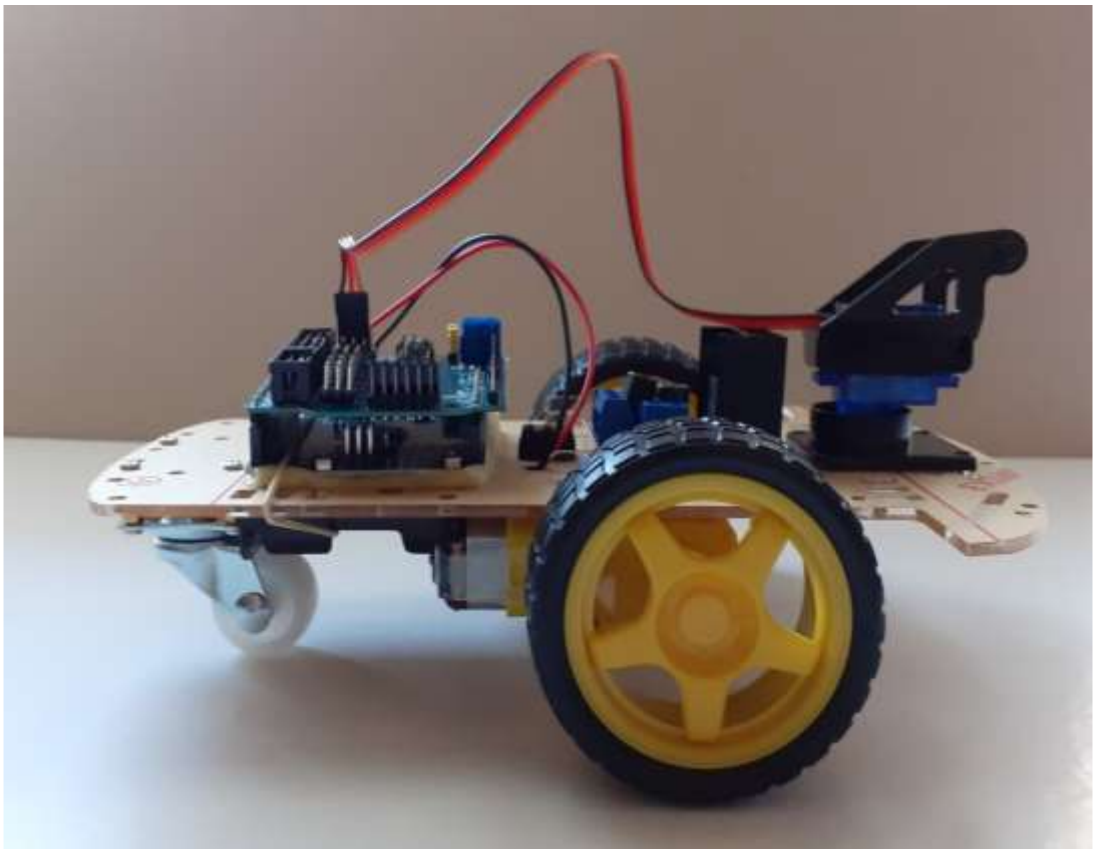
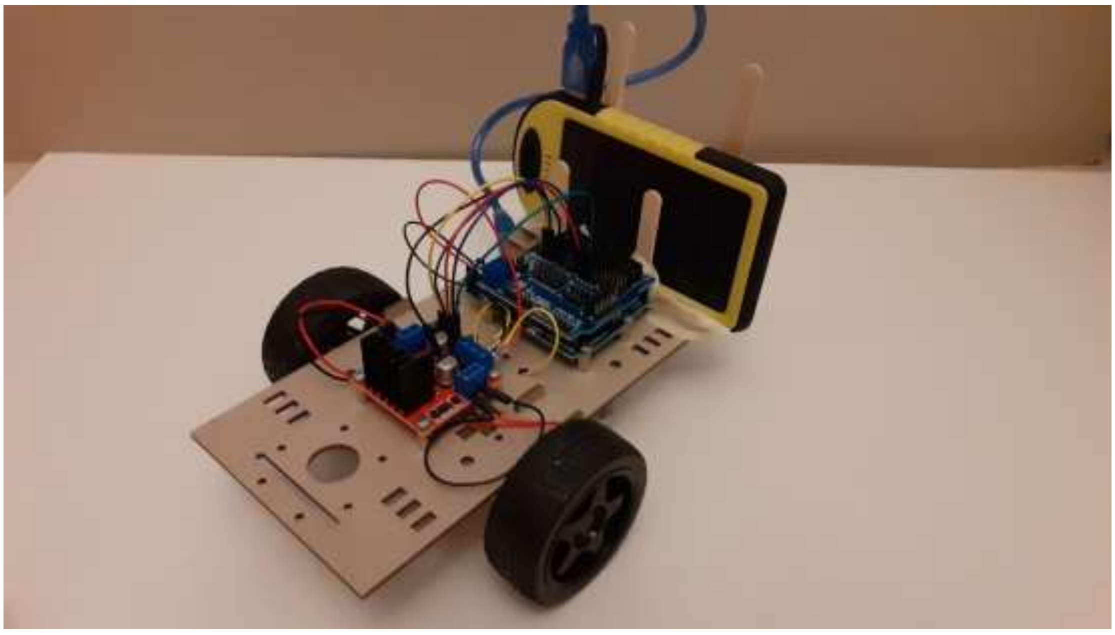



Figure 1: Images of a DIY Automobile - the DIY Automobile can take several forms based on students' imagination and preferences.

Guidelines for circuit making

The DIY Automobile is an advanced project and therefore there are significant differences compared to previous projects as far as the circuit making are concerned. Unlike previous projects, in DIY Automobile there is a need of an extra/special module through which the DC gear motors can be powered and controlled. This module is called DC motor driver. There are different types of motor drivers but, for the needs of this project, the **L298n DC motor driver** is used/implemented. Another extra module that is introduced in the present project is the **Arduino Sensor Shield V5**. As it has already been mentioned (i.e. in guidelines for assembling the DIY Automobile[1]), the Shield is an expansion that facilitates the circuit making process since it provides an easier way to connect sensors, motors and servos, by expanding the digital and analog input pins of Arduino board with Power (V) and Ground (GND), while providing separate PWM pins. Therefore, the need of a breadboard is eliminated.

The cabling and circuit making process is divided in three parts:

a. the first one concerns the power supply,

b. the second one is about the connection of DC gear motors to the DC motor driver module, while

c. the third one depicts the way that DC motor driver is connected to the Shield.

[1] http://edumotiva.eu/edumotiva/wp-content/uploads/2021/03/Guidelines-for-assembling-the-DIY-automobile.pdf

L298n DC motor driver module: ports and pins diagram and analysis

The following diagram (*Figure 2*) and description briefly present the use of motor driver's embedded ports and pins

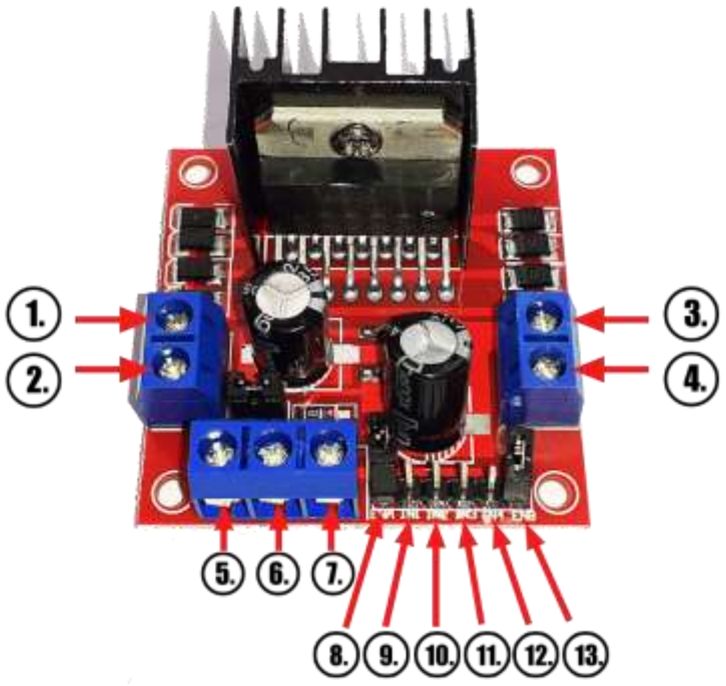


Figure 2: The L298n motor driver's ports and pins

Ports 1, 2, 3 and 4 (OUT 1, OUT 2, OUT 4 and OUT 3 respectively) provide connection to the DC gear motors. Normally, ports 1 and 4 are for power (+), while 2 and 3 for ground (-).

Ports 5, 6 and 7 are for power supply. In particular, port 5 provides 12V, port 7 5V while port 6 provides ground to the circuit.

Pins 8 to 13 are for connecting motor driver to Arduino/Shield. Pins 8 (EnA) and 13 (EnB) are respectively for DC gear motor A and B, and should be connected to PWM pins. These pins are used to control motors' speed (*see appendix 1. for details*). Pins 9 (IN1),10 (IN2), 11 (IN3) and 12 (IN4) are respectively for DC gear motor A and DC gear motor B, and should be connected to digital pins. Through high and/or low signals, these ports are responsible for controlling motors' spinning direction. When one of them is HIGH and the other is LOW the corresponding motor will spin. If both are set to LOW then the motor stops.

Part One: Power supply circuit

The following diagram (*Figure 3*) presents the circuit for providing power supply to the **L298n DC motor driver**. Connect the 5V (**1.**) of Shield to the 12V (**3.**) of motor driver. Then connect the Shield’s ground (**2.**) to motor’s driver ground (**4.**). Finally, create a close circuit by connecting the 12V (**3.**) of motor driver to the 5V (**5.**).

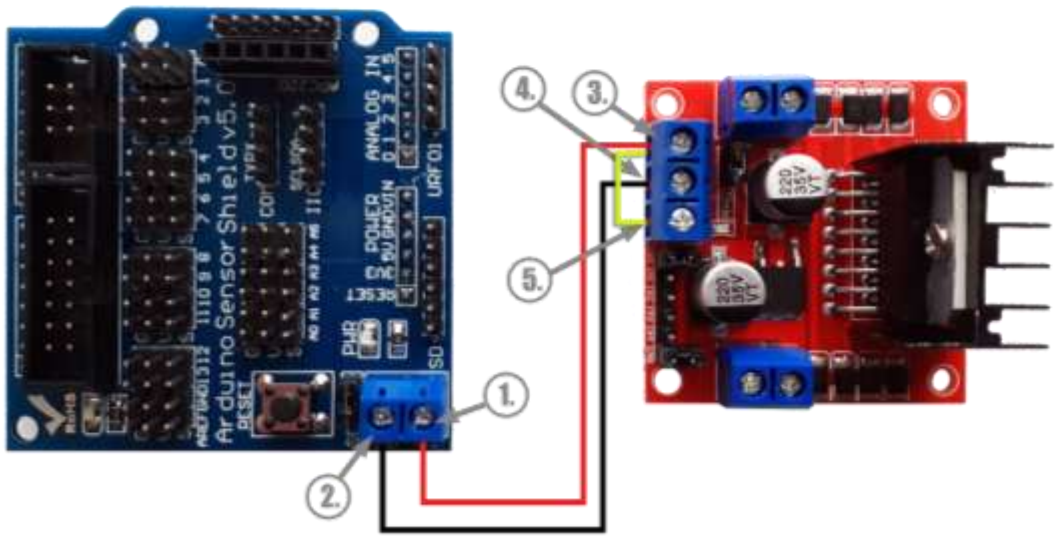


Figure 3: Circuit for providing power to DC motor driver.

Since we want – at some point – for our Automobile to be independent from our computer, we need to provide an extra source of power. As has already been mentioned (i.e. Guidelines for Assembling the DIY Automobile[2]), there are different options on the power source that will be supplied. In case you are using a solar bank or a battery holder with a barrel-jack connector, you just need to connect it to Arduino’s power supply port. In case you want to use a battery holder with free cables and an on-off switch you need to follow the instructions illustrated on the diagram depicted in Figure 5.

Specifically, connect the ground of the battery holder (**6.**) to Shield’s ground (**2.**). Then connect the 5V (**7.**) of the battery holder to Shield’s 5V (**1.**) through the two pins (**8.**, **9.**) of the switch. For this last step, you will probably need a wire stripper or a cutter and a soldering iron or a hot glue gun (Figure 4).

[2] http://edumotiva.eu/edumotiva/wp-content/uploads/2021/03/Guidelines-for-assembling-the-DIY-automobile.pdf

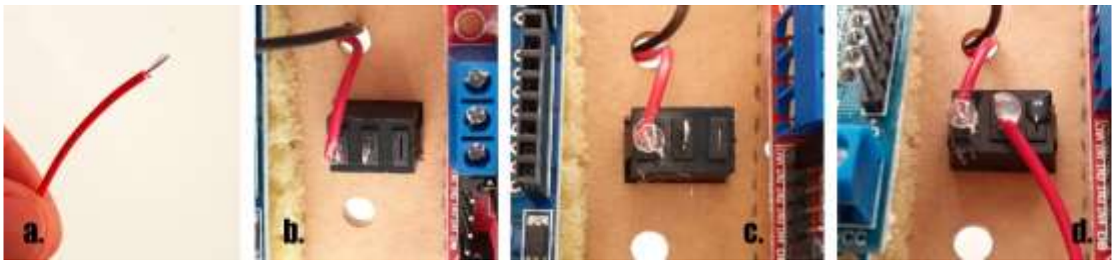


Figure 4: a. Revealing the wire of the cable using a wire stripper or a cutter; b. Folding the wire to the one pin of the switch; c. Gluing the wire by using hot glue gun; d. Gluing another wire that will connect the second pin of the switch to Shield's 5V

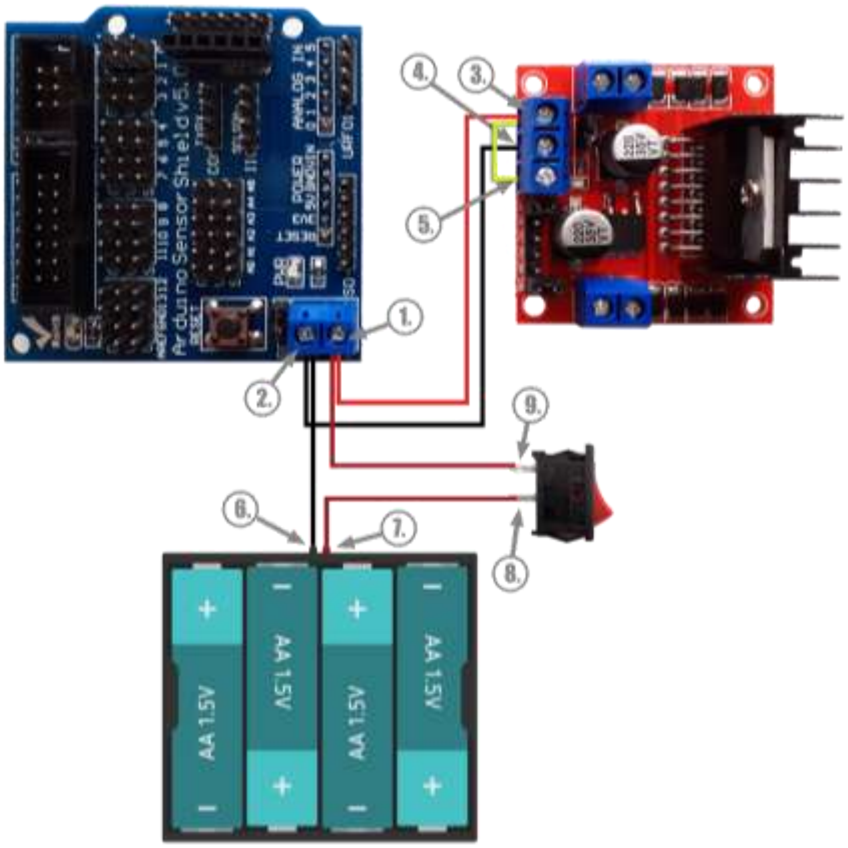


Figure 5: Circuit for providing power by using a 4 AA battery holder and a switch

Part Two: DC gear motors to DC motor driver circuit

The following diagram (*Figure 6*) - which is complementary to the one presented in Figure 3 - depicts the way that the DC gear motors are connected to the L298n DC motor driver.

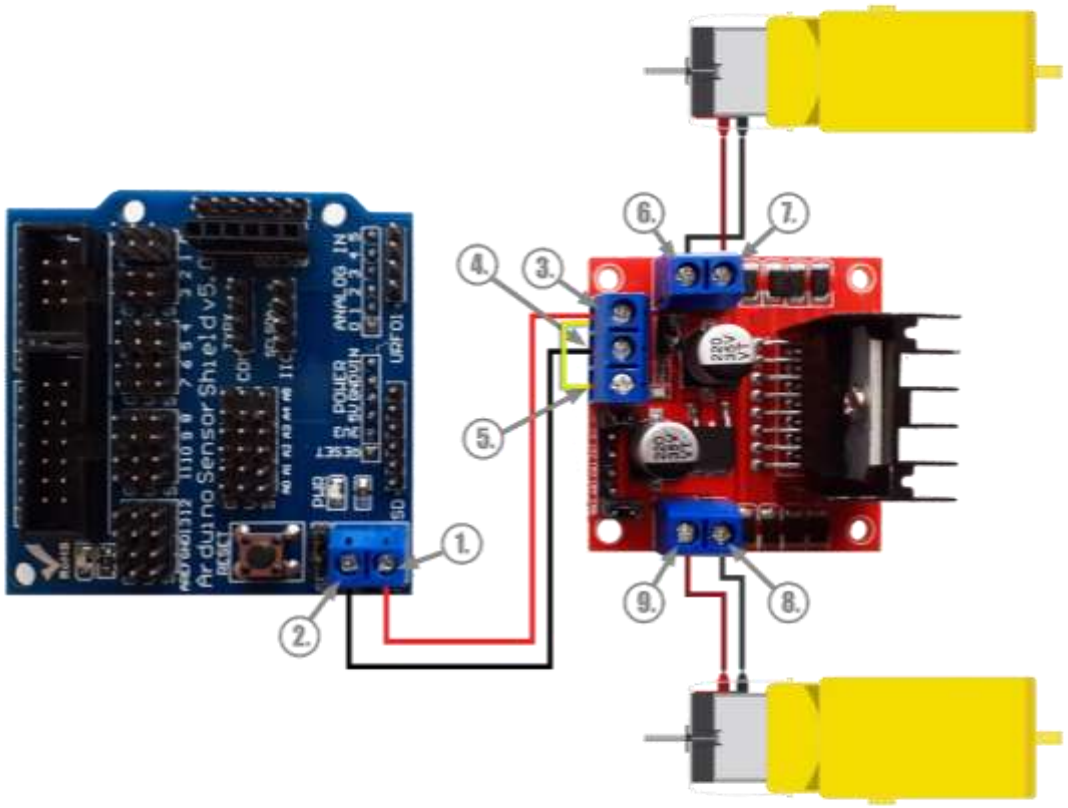


Figure 6: Circuit for connecting the DC gear motors to the L298n DC motor driver

Each DC gear motor has two small pins/joints through which it can be connected to the motor driver. Normally, there is no indication for which should be connected to power and which to ground. However, for the needs of the diagram the joints that should be connected to power are colored in red, and those that should be connected to ground, in black. Therefore, connect the pins for power to OUT1 (**7.**) and OUT3 (**9.**), and the pins for ground to OUT2 (**6.**) and OUT4 (**8.**). If no wires are already attached to the DC gear motors, then you will also need four wires (or M-M jumpers) as well as soldering iron or hot glue gun, to attach them. In case that no wires are attached to the gears, keep in mind to symmetrically connect the wires (i.e. if you choose to connect Motor's A upper pin to power, you should also connect Motor's B upper pin to the corresponding power pin). If two wires are already attached to your DC gear motor, simply follow the directions of the aforementioned diagram.

Part Three: Shield circuit making process

The following diagram[3] (*Figure 7*) indicates how the **DC motor driver** can be connected to **Shield**. Connect the EnA (**1.**) and EnB (**4.**) pins to two PWM pins (pin 5 and pin 11 in the example) by using two F-F jumpers. Then, connect IN1 (**2.**), IN2 (**3.**), IN3 (**5.**) and IN (**6.**) to any digital pin you wish (pins 6, 7, 12 and 13 in the example) by using four F-F jumpers. EnA, IN1 and IN2 pins will control the way that DC gear motor A is moving, while EnB, IN3 and IN4 will control DC gear motor B.

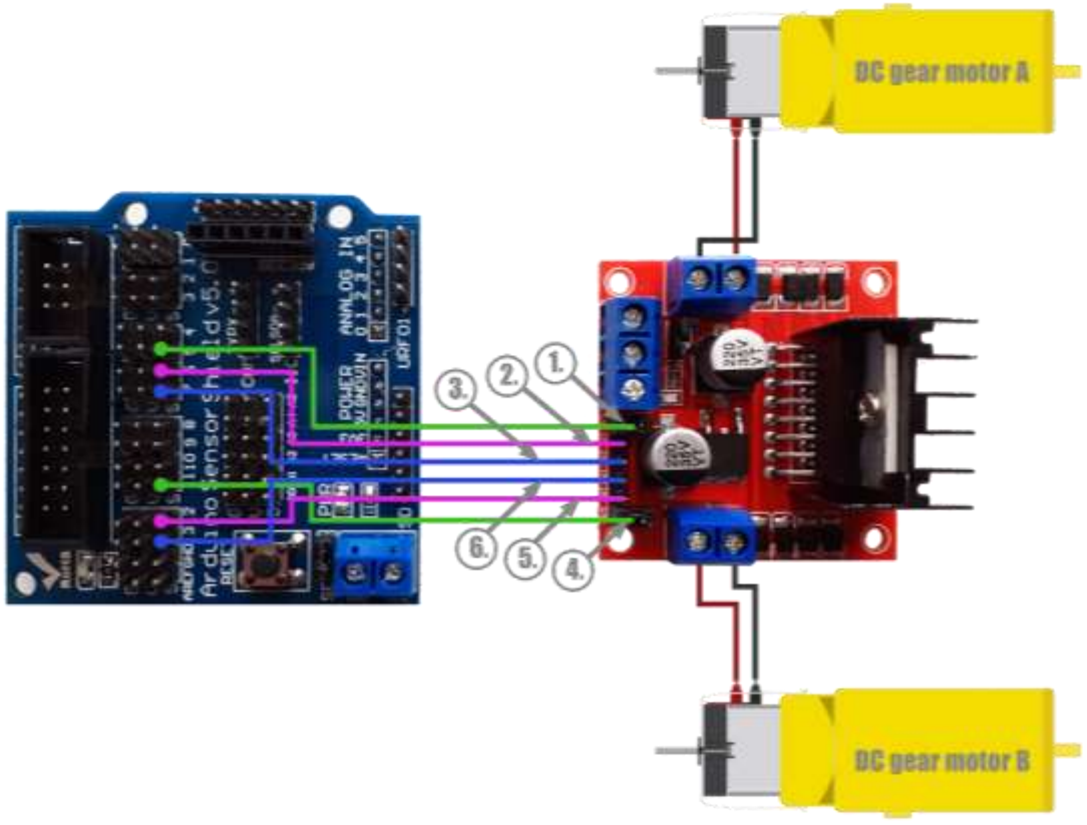


Figure 7: Circuit for connecting Arduino Sensor Shield to DC motor driver.

[3] For reasons of readability, the circuit of power supply (*Figure 3*) is not depicted in the present diagram.

Guidelines for DIY automobile Programming

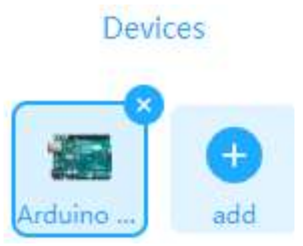
Level 1: A DIY automobile that moves forwards, left, right and backwards

Toward a block-based programming solution with mBlock[4]

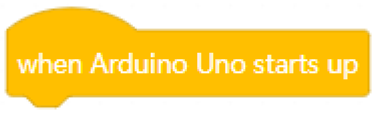
At this level, the students should be encouraged to breath some life to DIY automobile through the implementation of programming. For the needs of this project, the mBlock software – a similar to Snap4Arduino block-based programming environment – will be used. Therefore, through the implementation of the appropriate block-commands the DIY automobile will be instructed/scripted to move forwards, backwards, left and/or right.

Firstly, in the "Devices" menu of mBlock we are adding Arduino extension, by choosing the Arduino Uno device from the pop-up "Device Library" menu (*Figure 8*). The process of connecting Arduino board to mBlock is rather straightforward. Therefore, and unlike Snap4Arduino, there is no need to use Arduino IDE as an intermediate software, in order to upload to Arduino board a Firmata file or any other extra library.

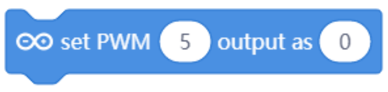
The blocks that appear below will be needed for assembling your scripts: The yellow blocks are from the Events palette. The blue blocks (Pin) control the Arduino Input/Output Pins. The pink blocks (My Blocks) and specifically the "Make a Block" command, enable you to define your own block of code. Like in the majority of block-based programming environments, a script in *mBlock* is assembled by dragging blocks from a palette and dropping them into the scripting area (exactly like in Snap4Arduino).



This is an *mBlock* extension that imports Arduino board block-commands palette, while allowing the connection of the board to the software. You can connect the device directly to the computer by using a USB data cable. The connect button achieves the communication with the Arduino



This is an Arduino extension Event block that executes the subsequent script when Arduino board starts up.



This block sets the output of the selected PWM pin to the specified value.

PWM signals can be used to control the speed of DC motors. Pins 3, 5, 6, 9, 10, and 11 of Arduino Uno can be used as PWM output. The range of values varies from 0 to 255, where 0 indicates the duty cycle of 0%, and 255 the duty cycle of 100%

[4] https://www.mblock.cc/en-us/

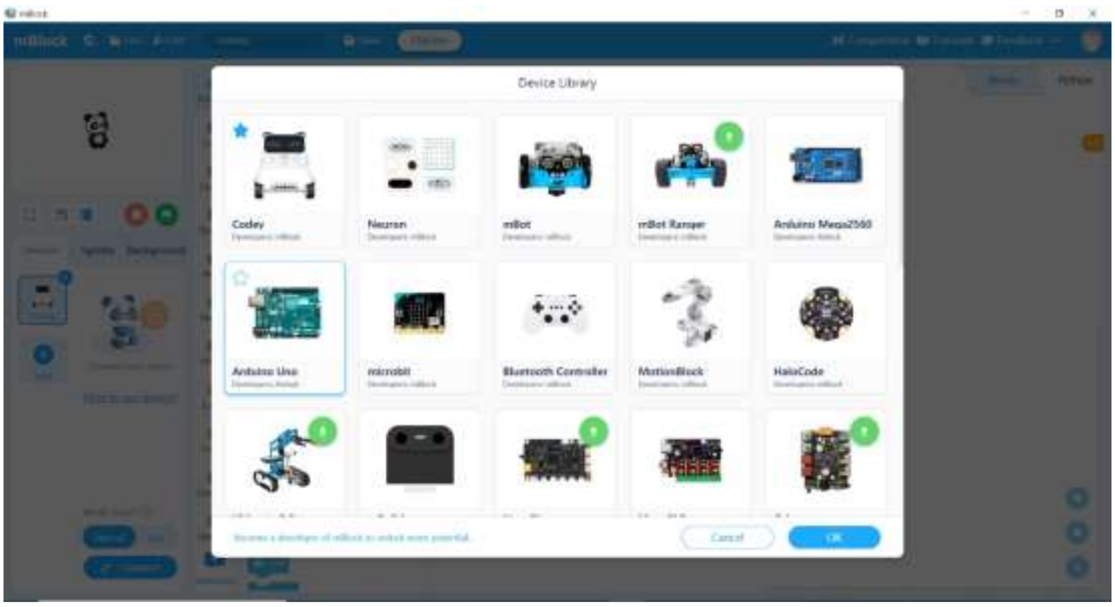
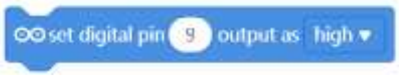
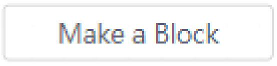
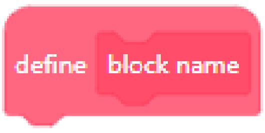
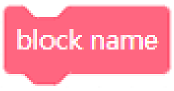
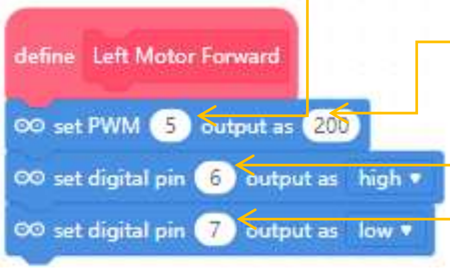


Figure 8: "Device Library" pop-up menu (selection of the corresponding Arduino board)

	Sets the output of the selected digital pin to low (false) or high (true) level.
	Click on **Make a Block** command to create a procedure that contains a number of consecutive commands (i.e. Move Forward).
	Drag the needed function blocks and assemble them under the hat block **"define()"** to set a new procedure (i.e. all the needed functions to make your Automobile move forward).
	Use the created procedure (i.e Move Forward) into the main code, under the Event hat block. When the procedure runs, mBlock will run the blocks below the corresponding Define block.

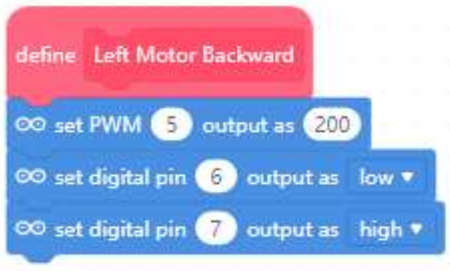
Programming DC Gear Motor A (Left Motor):



```
define Left Motor Forward
set PWM 5 output as 200
set digital pin 6 output as high
set digital pin 7 output as low
```

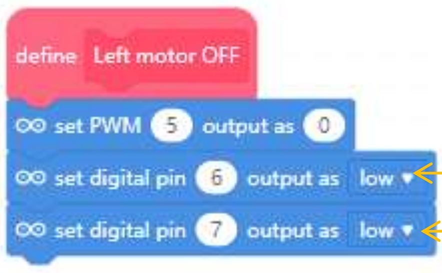
The PWN **Pin (**5 in the example**)** corresponds to the **ENA** pin of L298N motor driver that controls **motor's speed** (in the value range 0-255). The **IN1** and **IN2** pins (6 and 7, respectively, in the example) of L298N motor driver) control the spinning direction of the **motor A.**

Through this block, a number of consecutive commands are assigned to the "Left Motor Forward" procedure, instructing the DC Gear Motor A to move forward (conventional direction).



```
define Left Motor Backward
set PWM 5 output as 200
set digital pin 6 output as low
set digital pin 7 output as high
```

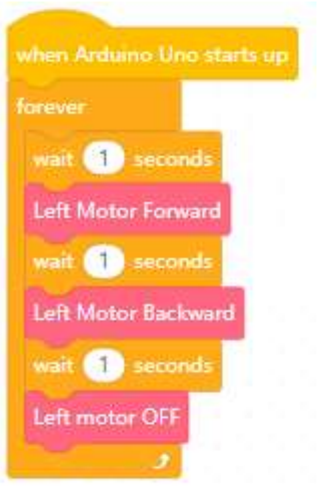
Through this block, a number of consecutive commands are assigned to the "Left Motor Backward" procedure, instructing the DC Gear Motor A to move backward (conventional direction).



```
define Left motor OFF
set PWM 5 output as 0
set digital pin 6 output as low
set digital pin 7 output as low
```

When one of the pins is set to HIGH and the other to LOW, the motor will spin. If **both of them**
(pins 6 and 7 in the example) are set to equal value (Low/Low or High/High) then the **motor stops.**

Through this block, a number of consecutive commands are assigned to the "Left Motor OFF" procedure, instructing the DC Gear Motor A to stop.



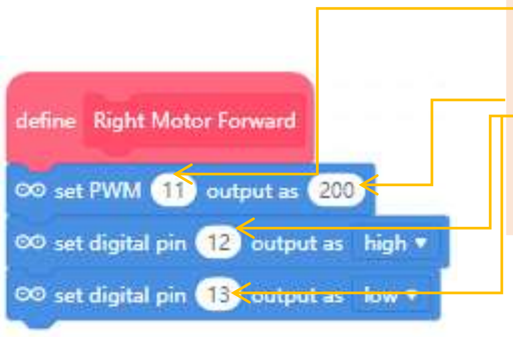
```
when Arduino Uno starts up
forever
    wait 1 seconds
    Left Motor Forward
    wait 1 seconds
    Left Motor Backward
    wait 1 seconds
    Left motor OFF
```

This script allows you to check Left's Motor mobility. Initially, (when Arduino starts up) the Left Motor is moving forward continuously. After a second the spinning direction changes to backwards, and after that it stops for 1 sec.

Tips:
Encourage your students to freely experiment with the values included in the aforementioned scripts or/and use a table to record their observations

EnA (pin 5)	*IN1 (pin 6)*	*IN2 (pin 7)*	*Motor A status*
200	*High*	*Low*	*Motor A is turning forwards*
200	*Low*	*Low*	*Motor A is stopped*
….	*….*	*….*	*….*

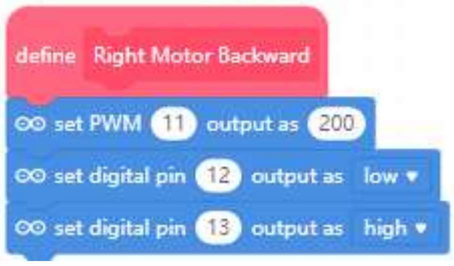
Programming DC Gear Motor B (Right Motor):



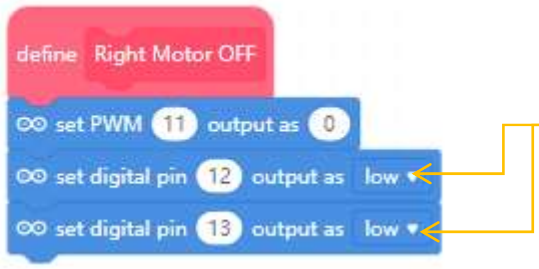
```
define Right Motor Forward
set PWM 11 output as 200
set digital pin 12 output as high
set digital pin 13 output as low
```

The **PWM Pin** (pin 11 in the example) corresponds to the **ENB** pin of L298N motor driver that controls **motor speed** (in the value range 0-255). The **IN3** and **IN4** pins (12 and 13 respectively, in the example) of L298N motor driver) control the spinning direction of the **motor.**

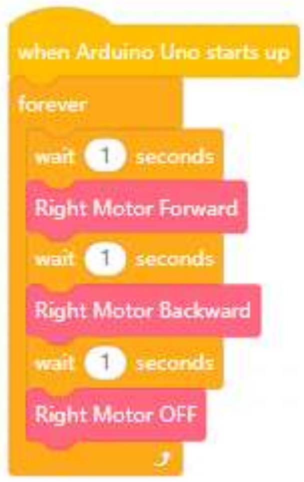
Through this block, a number of consecutive commands are assigned to the "Right Motor Forward" procedure, instructing the DC Gear Motor B to move forward (conventional direction).


define Right Motor Backward
set PWM 11 output as 200
set digital pin 12 output as low
set digital pin 13 output as high

Through this block, a number of consecutive commands are assigned to the "Right Motor Backward" procedure, instructing the DC Gear Motor B to move backward (conventional direction).


define Right Motor OFF
set PWM 11 output as 0
set digital pin 12 output as low
set digital pin 13 output as low

When one of the pins is set to HIGH and the other to LOW, the motor will spin. If **both of them** (pins 12 and 13 in the example) are set to equal value (Low/Low or High/High) then the **motor stops.**



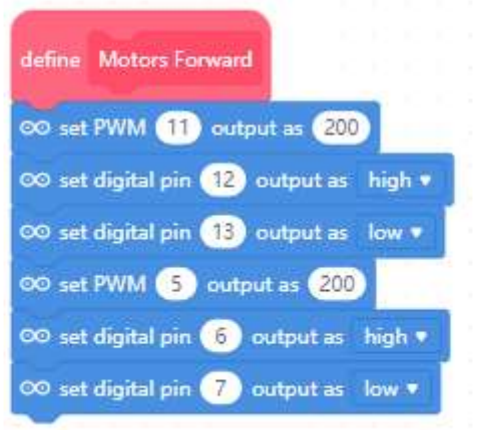
This script allows you to check Right's Motor mobility. Initially, (when Arduino starts up) the Right Motor is moving forward continuously. After a second the spinning direction changes to backwards, and after that it stops for 1 sec.

Tips:
Encourage your students to freely experiment with the values included in the aforementioned scripts or/and use a table to record their observations

EnB (pin 11)	*IN3 (pin 12)*	*IN4 (pin 13)*	*Motor B status*
200	*High*	*Low*	*Motor B is turning forwards*
200	*Low*	*Low*	*Motor B is stopped*
….	*….*	*….*	*….*

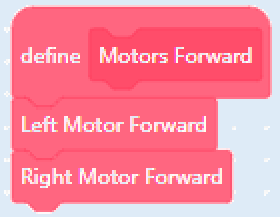
Define both Motors functionality:

The aforementioned scripts are suggested as a familiarization stage with the functionality of DC motors, and as a preparation step toward the main goal of the present level. Through the next steps/scripts, the students should be encouraged to test how the automobile respond when both motors are set in motion.

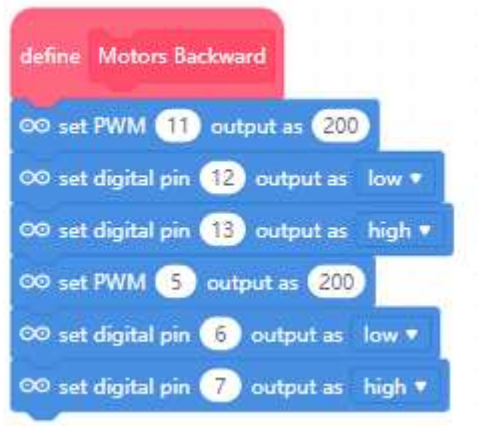


```
define Motors Forward
set PWM 11 output as 200
set digital pin 12 output as high
set digital pin 13 output as low
set PWM 5 output as 200
set digital pin 6 output as high
set digital pin 7 output as low
```

A number of consecutive commands are assigned to “Motors Forward” procedure, instructing the DIY automobile to move forwards (conventional direction). An alternative and simplified approach is depicted below.

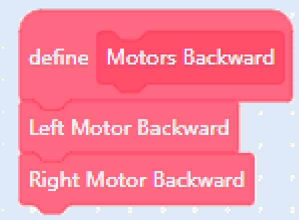


```
define Motors Forward
Left Motor Forward
Right Motor Forward
```

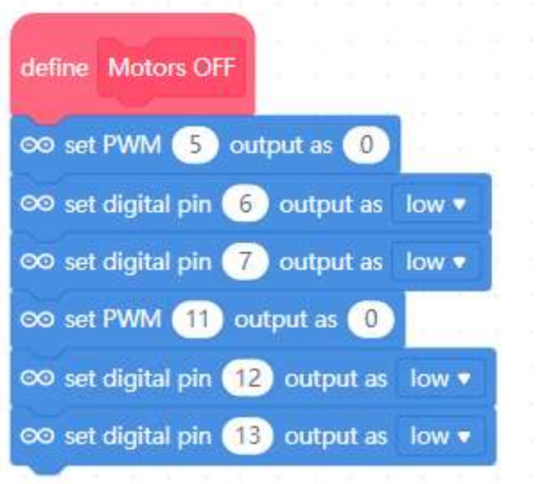


```
define Motors Backward
set PWM 11 output as 200
set digital pin 12 output as low
set digital pin 13 output as high
set PWM 5 output as 200
set digital pin 6 output as low
set digital pin 7 output as high
```

A number of consecutive commands are assigned to “Motors Backward” procedure, instructing the DIY automobile to move backwards (conventional direction). An alternative and simplified approach is depicted below.

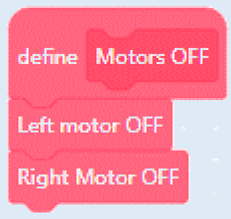


```
define Motors Backward
Left Motor Backward
Right Motor Backward
```



```
define Motors OFF
set PWM 5 output as 0
set digital pin 6 output as low
set digital pin 7 output as low
set PWM 11 output as 0
set digital pin 12 output as low
set digital pin 13 output as low
```

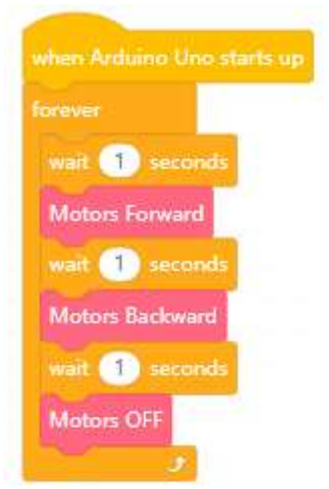
A number of consecutive commands are assigned to “Motors Backward” procedure, instructing the DIY automobile to stop. An alternative and simplified approach is depicted below.



```
define Motors OFF
Left motor OFF
Right Motor OFF
```

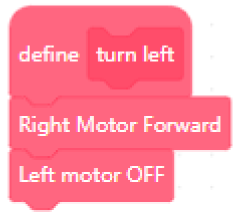
Moving the DIY automobile Forwards, Backwards, on the Right and on the Left:

Through the previous scripts, the automobile was able to move forwards, backwards, or stop. This is a major step towards the scope of the present level. Next, the students should be encouraged to combine all the aforementioned functionalities and observe how they can affect the mobility of the robotic artefact.



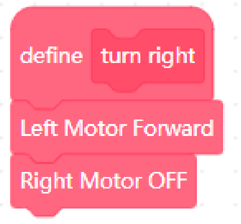
```
when Arduino Uno starts up
forever
    wait 1 seconds
    Motors Forward
    wait 1 seconds
    Motors Backward
    wait 1 seconds
    Motors OFF
```

This script allows you to set in motion both Motors. Initially, (when Arduino starts up) both Motors are moving Forward for 1 second. Then, their spinning direction changes to backwards and stops for 1 sec.



```
define turn left
Right Motor Forward
Left motor OFF
```

Through this block, a number of consecutive commands are assigned to the “turn left” procedure, instructing the DIY automobile to turn left (pivot).



```
define turn right
Left Motor Forward
Right Motor OFF
```

Through this block, a number of consecutive commands are assigned to the “turn right” procedure, instructing the DIY automobile to turn right (pivot).

Questions to raise in the class:

- *How could you make the automobile move forward for longer time?*
- *How does the value of the speed argument affect DIY automobile’s movement?*
- *Does the type of surfaces (table, floor etc.) affects the distance that the automobile covers when the same value of speed is applied?*
- *What will happen to DIY’s mobility if you assemble a script that includes the “Move Forward” and “turn right” block commands?*

Level 2: A DIY automobile that detects and avoids obstacles

At this level, the students should be encouraged to enhance the abilities of the DIY automobile by adding and programming the Ultrasound Sensor module. Therefore, through the implementation of the appropriate block-commands the DIY automobile will be instructed/scripted to detect and avoid obstacles.

Adding the Ultrasonic sensor to the circuit

The following diagram (*Figure 9*) illustrates the way that the Ultrasound sensor is connected to the Shield. For reasons of readability, the diagram does not illustrate all the full circuit. In particular, the Vcc (**7.**) and Ground (**10.**) pins are respectively connected to a 5V and a Ground pin of the shield (pin 8 in the example). The Trigger pin (**8.**) is connected to a PWM pin (pin 9 in the example) and the Echo (**9.**) to one of digital pins (pin 8 in the example)[5].

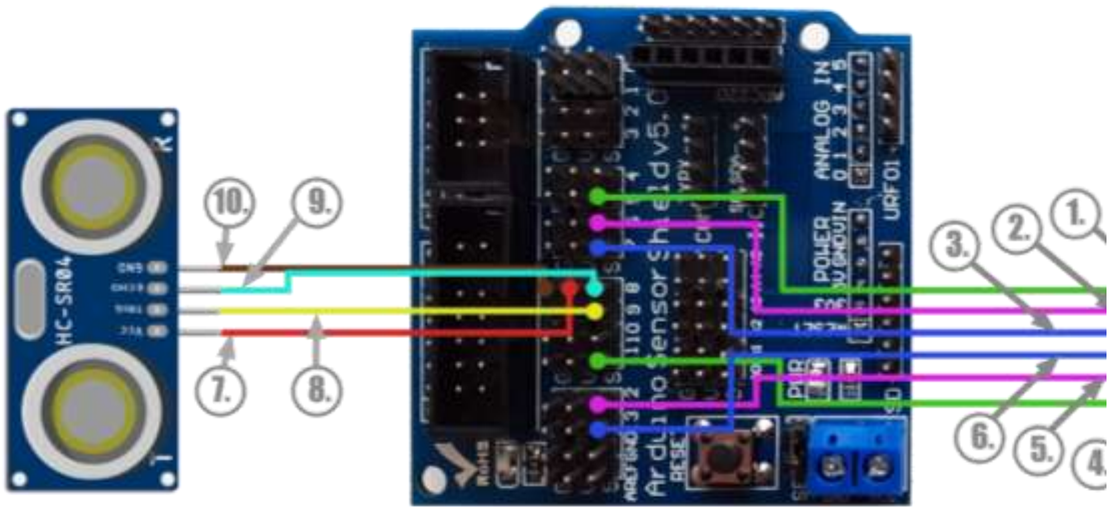
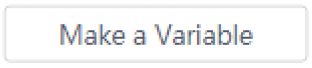


Figure 9: Circuit for connecting the Ultrasound sensor to the Shield

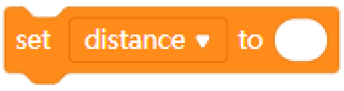
Toward a block-based programming solution with mBlock

The blocks that appear below will be needed for assembling your scripts: The yellow blocks are from the Events palette. The orange blocks are related to control of the program. The blue blocks (Pin) control the Arduino Input/Output Pins. The pink blocks (My Blocks) and specifically the "Make a Block" command, enable you to define your own block of code. The light blue (Sensor) blocks control the Arduino sensors, like ultrasound sensor. The green blocks are responsible for mathematical operations execution. The dark orange set of blocks concerns the variables creation. Like in the majority of block-based programming environments, a script in *mBlock* is assembled by dragging blocks from a palette and dropping them into the scripting area (exactly like in Snap4Arduino).

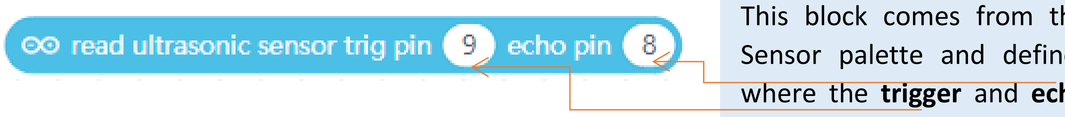
[5] Unlike Snap4Arduino, in mBlock there is no restrain on which pins you should choose for connecting the trigger and the echo pins.



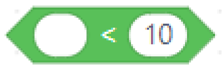
Click on **Make a Variable** command to create a variable that contains the ultrasound sensor’s distance value.



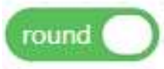
This block comes from the Variables palette and sets the variable ‘***distance***’ to a specific value. This value can be inserted manually or can be linked to the values received from specific sensors (i.e. an ultrasound sensor).



This block comes from the Sensor palette and defines where the **trigger** and **echo** pins of Ultrasound sensor are connected. The return value of the block is the ‘***distance***’ of the obstacle in cm.

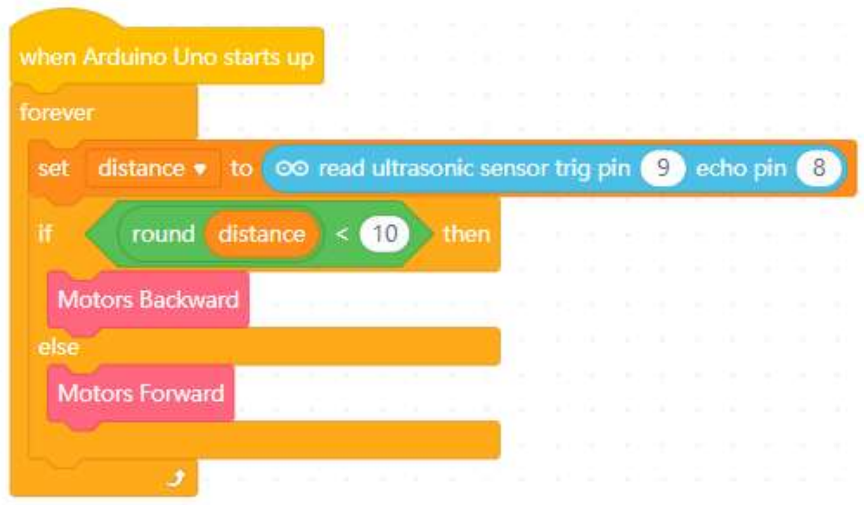


This block comes from the Operators palette and executes the subsequent script if the value of the specified parameter (i.e distance) is smaller than the manually specified value (distance of 10cm in the example).



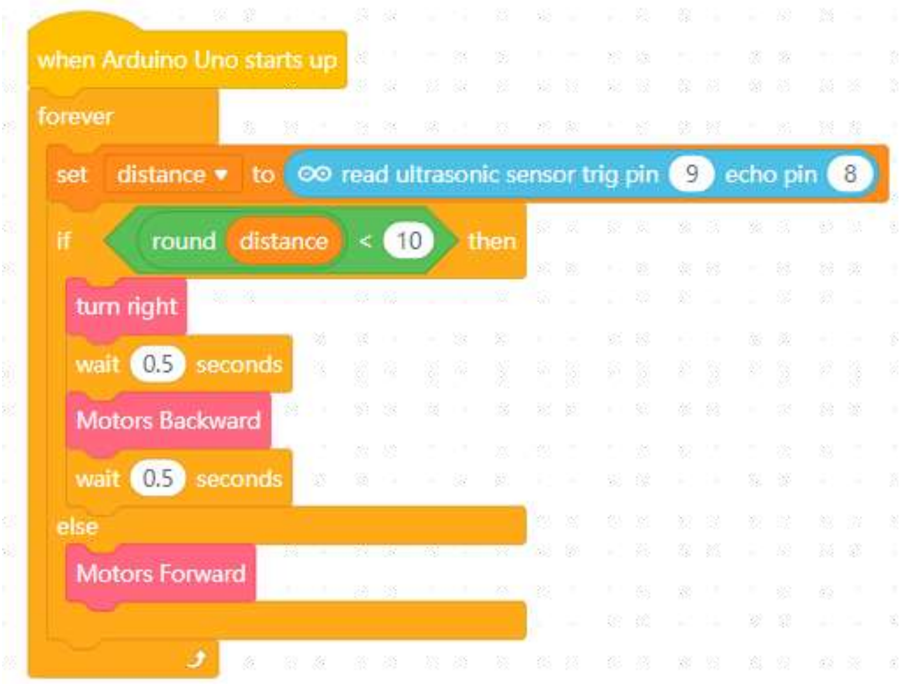
This block comes from the Operators palette and rounds the number to the nearest integer.

The DIY automobile avoids obstacles:



```
when Arduino Uno starts up
forever
    set distance ▾ to read ultrasonic sensor trig pin 9 echo pin 8
    if round distance < 10 then
        Motors Backward
    else
        Motors Forward
```

This script allows the DIY automobile to detect and avoid obstacles. When Arduino starts up, the value of the distance (which is assigned to the values received by the Ultrasound sensor) is repeatedly compared to a user defined value (i.e. 10). Therefore, if the value of the distance is smaller than 10 then, the DIY automobile is moving backwards. Otherwise, the automobile is moving forwards.



```
when Arduino Uno starts up
forever
    set distance ▾ to read ultrasonic sensor trig pin 9 echo pin 8
    if round distance < 10 then
        turn right
        wait 0.5 seconds
        Motors Backward
        wait 0.5 seconds
    else
        Motors Forward
```

This is an alternative script that allows the automobile to detect and avoid obstacles. When Arduino starts up, the value of the distance (which is assigned to the values received by the Ultrasound sensor) is repeatedly compared with a user defined value (i.e. 10). If the value of the distance is smaller than 10, then the DIY automobile is initially turning right for 0.5 sec and then is moving backwards. Otherwise it is moving forwards.

Questions to raise in the class:

- *What will happened if you change the value of the distance (i.e. from 10 to 20)?*
- *What will happened if you change the duration between “turn right” and “Motors Backward” blocks?*

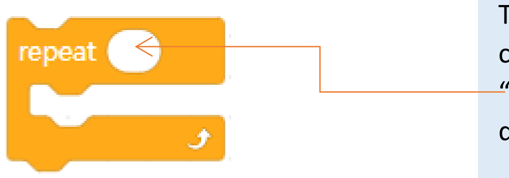
Level 3: A DIY automobile that moves on different angles and/or geometrical shapes

Toward a block-based programming solution with mBlock

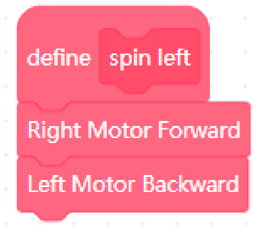
At this level, the students should be encouraged to program the DIY automobile moving on different angles and/or geometrical shapes (like square, triangle).

Tip: You can sketch the shapes on a paper or on the floor, by using a tape, in order to make the task visual perceptible.

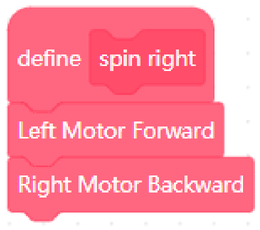
The blocks that appear below will be needed for assembling your scripts: The yellow blocks are from the Events palette. The orange blocks are related to control of the program. The blue blocks (Pin) control the Arduino Input/Output Pins. The pink blocks (My Blocks) and specifically the "Make a Block" command, enable you to define your own block of code. The green blocks are responsible for mathematical operations execution. The dark orange set of blocks concerns the variables creation.



This is a repeat loop from the Control menu. The commands/blocks that will be placed in the "repeat construct" are repeated based on a defined value of times.



A number of consecutive commands are assigned to "spin left" procedure, instructing the DIY automobile to make a rather quick turn on the left, by setting the right motor to move forward and the left to move backwards.

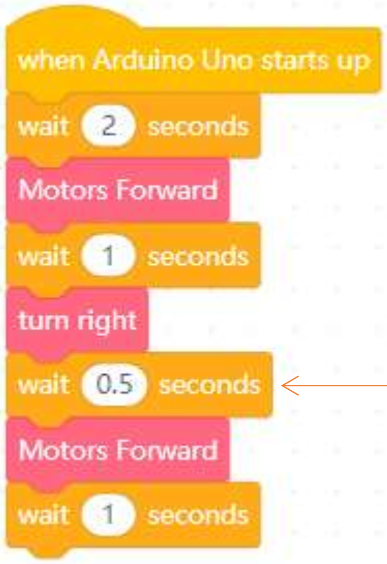


A number of consecutive commands are assigned to "spin right" procedure, instructing the DIY automobile to make a rather quick turn on the right, by setting the left motor to move forwards and the right to move backwards.

The DIY automobile moves forward and turns right for 90^0 (vertical angle):

Until now, we have managed to control the ways that our automobile can move forwards and backwards. We have also managed to make our automobile turn right or left, but – since the DC motors do not support the programming parameter of angle – we still haven't been able to control the angle of turning. However, recalling the second script that was introduced in Level 2, we can realize that we managed to make our automobile to change the angle of its direction before it moved backwards, by including in the script the "turn right" block of commands. With this as a solid base, the students should be encouraged to come up with solutions on how they could control this directional change in order to make their automobile move in specific angles. Therefore, they are encouraged to experiment with the parameter of time and see how it can affect the results of their script.

Note: *An alternative way of introducing the concept of angular turning is by using the sprites of mBlock. For more details on this approach, please check the appendix 2.*



```
when Arduino Uno starts up
wait 2 seconds
Motors Forward
wait 1 seconds
turn right
wait 0.5 seconds
Motors Forward
wait 1 seconds
```

This script allows the automobile to take a turn. When Arduino starts up, the automobile waits for 2 sec (in order to prepare the status of its position). Then it is moving forward for 1 sec. After that it turns right for 0.5 sec, making a pivotal turn and finally it is moving forward, again for 1 sec.

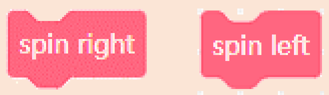
- How many degrees did the automobile turn when you set the waiting duration to 0.5 sec?
- What will happen if *you change the value of the time argument to 0.3 sec?*
- *Can you find the optimum duration in order to succeed a 90^0 pivotal turn?*
- *Is the result affected by different surfaces (i.e. floor, table etc.)?*

Tip*: Try to use a tape to mark the angle of pivotal turning. You can also create a table and note down your observations.*

Time (sec)	0.3	0.5	0.8	1.0
Angle (degrees)				

Questions to raise in the class:

- *How does the DIY automobile behave on different surfaces (i.e. table, floor etc)?*
- *Can you make the DIY automobile turn for 90^0 degrees using the blocks for spinning (turn quickly) Right and/or Left?*

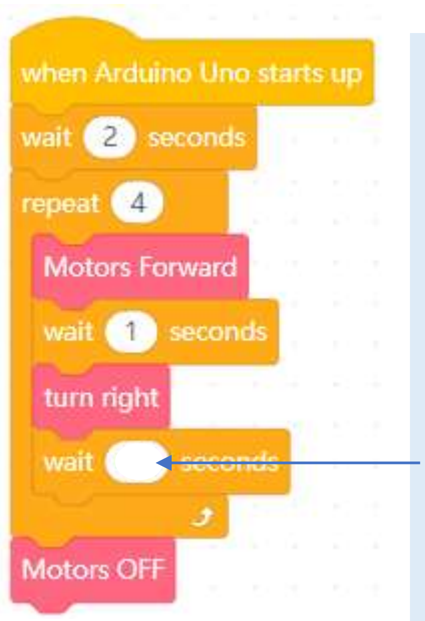


```
spin right
spin left
```

The DIY automobile moves on square:

Now, let’s try to instruct our automobile to move in such a way that will abstractly inscribe a square shape (meaning a shape with four equal sides and four equal angles of 90 degrees each). As it was previously mentioned, since the DC motors do not support the parameter of angular turn, the parameter of time together with the block commands of turn are suggested to be implemented instead.

Note: *An alternative way of introducing the concept of square shaping is by using the sprites of mBlock. For more details on this approach, please check the appendix 3.*



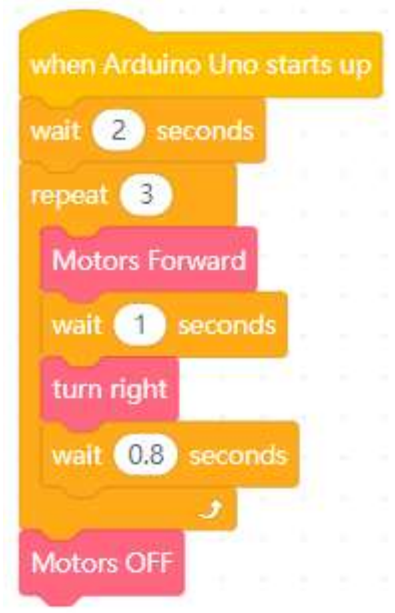
```
when Arduino Uno starts up
wait 2 seconds
repeat 4
    Motors Forward
    wait 1 seconds
    turn right
    wait ( ) seconds
Motors OFF
```

Under certain conditions, this script allows the DIY automobile to abstractly inscribe a square. To do that, the automobile should move on a constant speed (i.e. 200) for a fixed amount of time (i.e. 1 sec) and make a pivotal turn of 90 degrees. This procedure should be repeated for four time. When Arduino starts up, the automobile waits for 2 sec (in order to prepare the status of its position). Then, both motors are moving forward for 1 sec, and after that only the right one turns for some seconds (*insert the value that you found from the previous task i.e. through the script that allows DIY to take a turn*), in order to make the pivotal turn of 90 degrees. This procedure is repeated for 4 consecutive times and after that both Motors stop.

Questions to raise in the class:

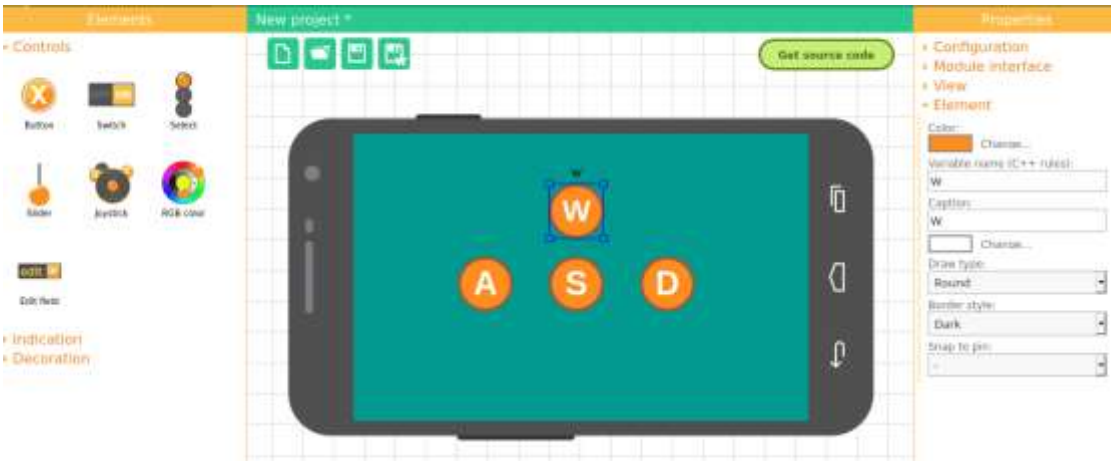
- What will happened if you were using the turn left block of commands instead of the turn right block?
- *Can you make the DIY automobile to abstractly inscribe a square by using blocks for spinning (turn quickly) (i.e. blocks spin right and spin left)? How does the value of the corresponding wait block is affected?*

Encourage your students to experiment with more shapes. How could they instruct the automobile to move in a triangle shape? Does the following script can make the automobile to abstractly inscribe a triangle? How many degrees should be the turning angle?



```
when Arduino Uno starts up
wait 2 seconds
repeat 3
    Motors Forward
    wait 1 seconds
    turn right
    wait 0.8 seconds
Motors OFF
```

Level 4: Controlling remotely the DIY automobile (optional)



You can find a detailed presentation of level 4 here: http://edumotiva.eu/edumotiva/wp-content/uploads/2021/03/Automobile_level4.pdf

Appendix

1. Details for EnA and EnB pins

Each of the aforementioned pins, is covered with a jumper (*Figure 10, left*). Remove them, by using a tweezers (*Figure 10, Center*), in case you want to use the EnA and EnB pins so as to control the speed of your motors.

When you are removing the jumper, two pins are revealed. To control the speed of the motor, use the one that is collinear to IN pins, as it is indicated in the right image of Figure 10.

Please, do not remove these jumpers in case you are not interested to control the speed of your motors.

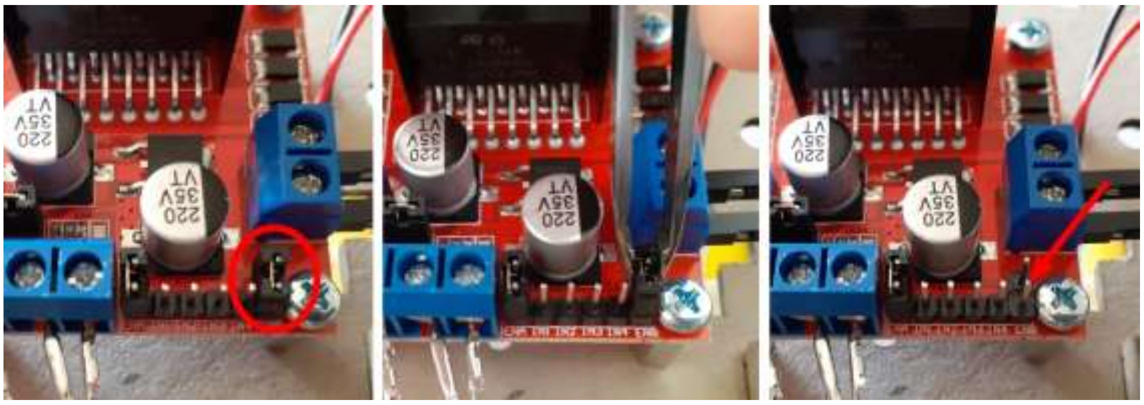
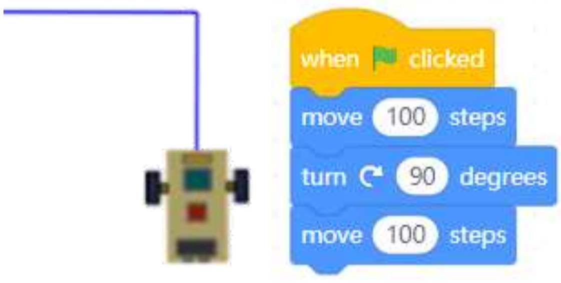


Figure 10: Uncovering the EnB pins for controlling speed of Motor B; Left: Jumper that covers the EnB pin; Center: Removing the jumper with a tweezers; Right: Two pins are revealed. Connect the one that is indicated with the red arrow, to one of PWM pins of your Arduino or your Shield

2. The DIY automobile moves forward and turns right for 90° (vertical angle): Using Sprites to introduce the concept of angular turning



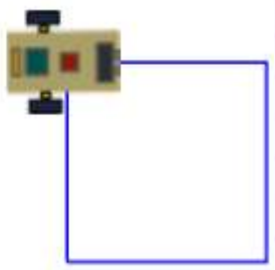
```
when 🏳 clicked
move 100 steps
turn ↻ 90 degrees
move 100 steps
```

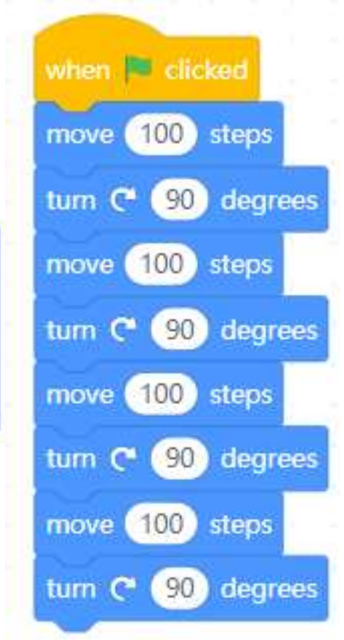
In this step, the concept of angular turning is introduced through the use of mBlocks sprite. For this purpose, the move and the turn blocks from **Motion blocks (blue blocks)** are assembled in order to make the sprite (an image of the DIY in the example) to move for 100 steps, make a pivotal turn of 90 degrees and move again for 100 steps. The present script can be the first step for the construction of a squared shape.

The aforementioned paradigm is suggested as a familiarization (intermediate) stage towards the concept of angular turning.

3. The DIY is moving on a square: Using sprites to introduce the concept

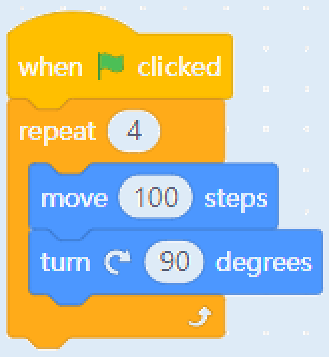
Let's return to the sprites programming menu and try to make the sprite inscribe a square shape (meaning a shape with four equal sides and four equal angles of 90 degrees each). Again, for this purpose, the move and turn blocks from **Motion blocks (blue blocks)** are implemented.





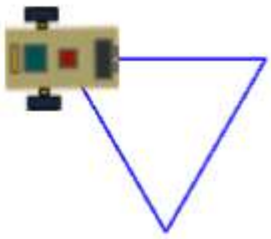
```
when 🏁 clicked
move 100 steps
turn ↻ 90 degrees
move 100 steps
turn ↻ 90 degrees
move 100 steps
turn ↻ 90 degrees
move 100 steps
turn ↻ 90 degrees
```

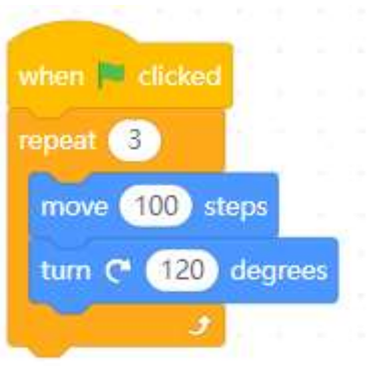
This script helps to understand the concept of a squared shape construction through the implementation of mBlock sprites block commands. The sprite is scripted to *move for 100 and* make a pivotal *turn of* 90 (degrees), for four consecutive times. You can either choose of repeating the aforementioned block for four times, creating an 8-line code, or simplify the procedure by using the **Control block "Repeat"**.



```
when 🏁 clicked
repeat 4
    move 100 steps
    turn ↻ 90 degrees
```

The DIY automobile moves on triangle:





```
when 🏁 clicked
repeat 3
    move 100 steps
    turn ↻ 120 degrees
```

This script helps to understand the triangle construction. For this purpose, the instructions *move 100 and turn* 120 degrees are repeated three (3) times.

ROBOSCIENTISTS PROJECT

Motivating secondary school students towards STEM careers through robotic artefact making

Erasmus+ KA2 2018-1PL01-KA201-051129

Creators

Rene Alimisi, Chrysanthi Papasarantou, Konstantinos Salpasaranis (EDUMOTIVA)

Declaration

This report has been prepared in the context of the ROBOSCIENTISTS project. Where other published and unpublished source materials have been used, these have been acknowledged.

Copyright

Funding Disclaimer

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.