

# **Micromouse**

By Globaltronic

User manual A3

### INDEX

<b>ÍNDICE .....</b>	<b>1</b>
<b>.....</b>	<b>1</b>
<b>1 VERSIONS .....</b>	<b>3</b>
<b>2 CONTEXTUALIZATION .....</b>	<b>3</b>
2.1 ROBOTICS .....	3
2.2 MICROMOUSE PORTUGUESE CONTEST .....	3
<b>3 FUNCTIONS AND FEATURES OF MMkit.....</b>	<b>4</b>
3.1 CARACTERÍSTICAS DE HARDWARE .....	4
<b>4 PROGRAMMING ENVIRONMENT .....</b>	<b>5</b>
4.1 INSTALLING THE ARDUINO IDE .....	6
4.2 ARDUINO IDE INTERFACE .....	6
<b>5 MMkit INSTALLATION AND CONFIGURATION .....</b>	<b>11</b>
5.1 EQUIPMENT .....	11
5.2 MANUAL INSTALLATION OF DRIVERS .....	12
5.3 MMkit IDENTIFICATION .....	13
<b>6 LIBRARIES INSTALLATION .....</b>	<b>13</b>
<b>7 USE THE MMkit.....</b>	<b>15</b>
7.1 UPLOAD THE FIRST SKETCH (LED flashing) .....	15
7.2 SKETCH EXAMPLES FROM THE MMkit LIBRARY .....	16
7.2.1 PROJECT 1: Turn on LED with button .....	16
7.2.2 PROJECT 2: Measure distances using IR LED .....	17
7.2.3 PROJECT 3: Move Forward (Move the MMkit) .....	19
7.2.4 PROJECT 4: Move Forward (With interruptions) .....	20



# MICROMOUSE

## 1 VERSIONS

HARDWARE	FIRMWARE	SOFTWARE
A8	1.1	A4

## 2 CONTEXTUALIZATION

Before we take a closer look at MMkit, it is important to understand the context in which it is embedded and the scope in which it arose. For this we will briefly cover terms such as "Robotics" and "*Micromouse Portuguese Contest*", concepts that will later allow us to better understand the MMkit.

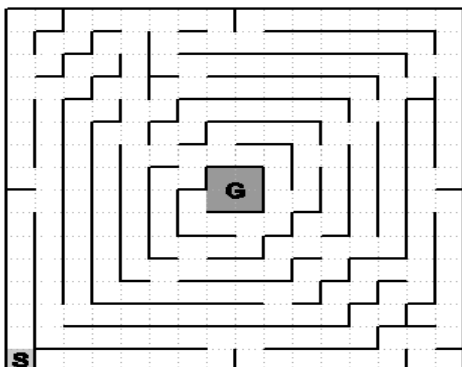
### 2.1 ROBOTICS

Robotics is a branch of technology that is responsible for planning and building automated machines, covering areas such as mechanical, electrical and electronic engineering, including also various branches of physics and computer science. They deal with systems composed by machines and mechanical parts that are automated and managed by integrated circuits.

Robotics works with the mechanical part of the movements, in other words, it works with the devices that act in their displacement or in some functionality that makes it interact with the environment, as well as with the computational part that develops all the programming inherent to the movement itself. It develops electromechanical devices, capable of performing tasks in a pre-programmed or autonomous way.

### 2.2 MICROMOUSE PORTUGUESE CONTEST

The "*Micromouse Portuguese Contest*" is a robotics contest where competitors put their robots to compete for intelligence and speed in solving a specific maze. robot participating in this contest is called "*Micromouse*". The contest is organized annually by the Robotics Nucleus (EEC/NEUTAD) of the University of Trás-os-Montes and Alto Douro. This is the portuguese version of the contest started in Japan in the 1970s. "*Micromouse*" events are held all over the world, being the most popular in the United Kingdom, USA, Japan, Singapore, India and South Korea.



Img 1 Example of a 16 by 16 matrix maze.

The labyrinth (img 1) is composed by a matrix of 16 by 16 cells, each cell is a square of 180 mm, having the walls being 50 mm high. "*Micromouses*" are completely autonomous robots that must find the path of the labyrinth, from a predetermined position to the central area of the same.

After the meta is reached, the robot usually performs additional searches of the maze until it finds an ideal route from the beginning to the center, within a limit time of 10 minutes. Once the optimal route is found, the robot will try to execute this route in the shortest possible time.

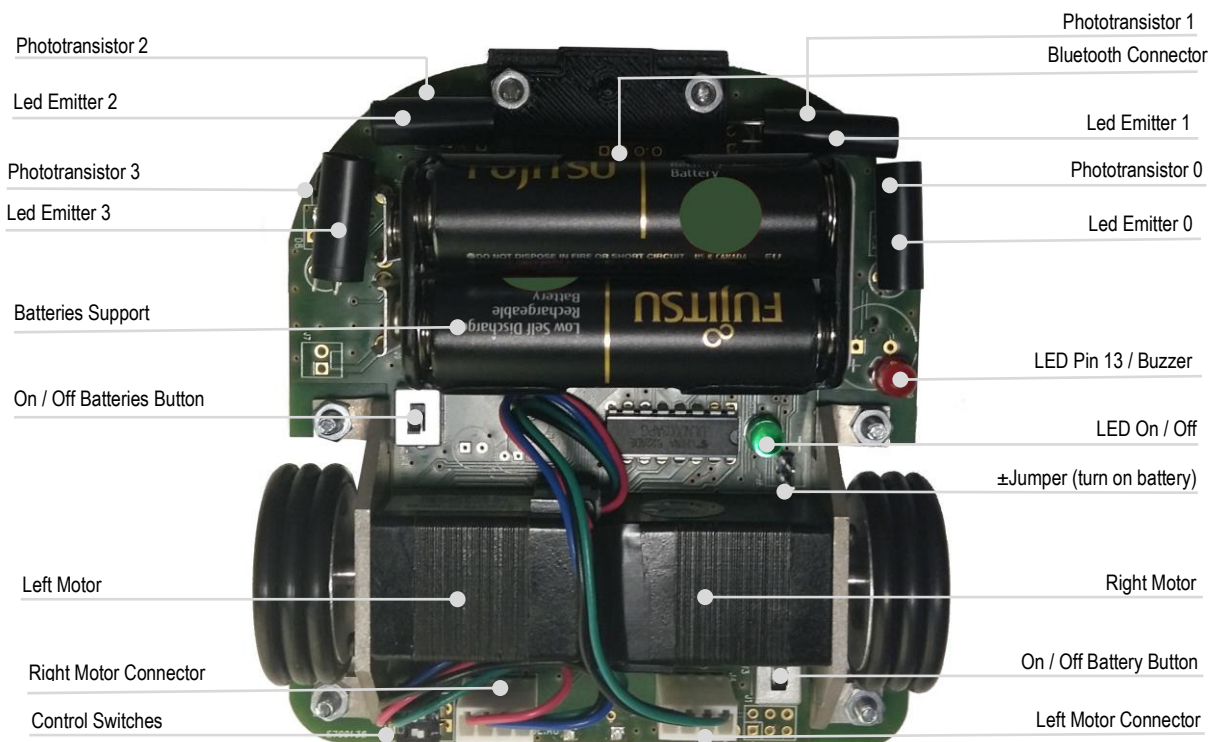
## 3 FUNCTIONS AND FEATURES OF MMkit

MMkit is a physical platform based on an embedded system transformed into a robot (img 2). It aims to be an educational robotics kit intended for the learning and development of robots whose main purpose is to solve a maze. MMkit is a complete and high-performance robotic platform. It involves sensors that acquire information about the robot surroundings, actuators that allow the robot to influence its surroundings, and a controller that gives the robot the ability to process information about it and select “responses” to its stimulus.

It is programmed through the use of C language with the ARDUINO programming environment (see glossary) IDE. The ATmega32u4 model microcontroller present in the robot has the “bootloader” (see glossary) of the Arduino Leonardo, so the robot is programmed as if it were an Arduino. One of the advantages and big differences of the Atmega32u4 is that it has an integrated USB communication port, not requiring a secondary processor used only for communication.

### 3.1 CARACTERÍSTICAS DE HARDWARE

**Dimensions** (length, width, height): 103 x 92 x 33mm | **Microcontroller:** ATmega32u4 (integrated USB communication port) | **Motor:** NEMA 8 motor (hybrid stepper bipolar motor with 1.8° step angle and 200 steps per resolution. Each phase consumes 600 mA at 3.9 V) | **Motor driver:** DRV8834 (with 2 “H bridges” that allow the control of a bipolar stepper motor) | **Infrared emitter led:** SFH4511 (very narrow radiation angle – 4th) | **Phototransistor:** TEFT4300 (with high radiant sensitivity and daylight filter).



Img 2 Explanation of the components from the micromouse robot

## 4 PROGRAMMING ENVIRONMENT

As mentioned previously, MMit uses the **Arduino IDE** as the programming and development environment.

Both the hardware and the software for ARDUINO are “*opensource*”, which means that the code and assembly schematics are accessible to anyone and can be used for free. This means that PCB schemes and designs themselves can be produced (copied or changed) and even sold. This is not only allowed as it is fomented and is at the basis of the “*opensource*” philosophy. The only requirement that the ARDUINO development team does to outsiders is that their name can only be used exclusively by them in their own products, and therefore the cloned boards from an ARDUINO have names such as Freeduino, BoArduino, Roboduino, among others.

The ARDUINO programming language is a “*Wiring*” (open source structured programming platform for microcontrollers), which is based on the concept “*Processing*” (language and multimedia programming environment). To program the ARDUINO microcontroller is convenient to use the **ARDUINO “Integrated Development Environment” (IDE)**, which allows to write programs in **C language**.

In ARDUINO IDE, a program is a sequence of instructions written in the text editor, followed by a compilation process (identification of language errors) and upload to the memory of the microcontroller. The compilation procedure is a conversion from written program text to machine code, capable of being interpreted by the final device. The programa is only executed, when it is successfully loaded into the memory of the device. In the ARDUINO world, programs written by users are referred to as “*sketches*” (see glossary) and the compiled code by “*firmware*”.

**NOTE:** In case of doubt, the MMkit uses mandatorily as development environment the **Arduino IDE** and does not work with any other software type.



Img 3 ARDUINO IDE development environment

The exhaustive description of the ARDUINO language is outside the scope of this manual. For a better understanding of the terms, concepts and languages used, search in:

→ **Introduction: what is an ARDUINO and why will you want to use it**  
<http://Arduino.cc/en/Guide/Introduction>

→ **Software installation**  
<http://Arduino.cc/en/Guide/Windows>

→ **Description of the IDE (integrated development environment)**  
<http://Arduino.cc/en/Guide/Environment>

→ **ARDUINO language reference**  
<http://Arduino.cc/en/Reference/HomePage>

→ **Sketches examples**  
<http://Arduino.cc/en/Tutorial/HomePage>



## 4.1 INSTALLING THE ARDUINO IDE

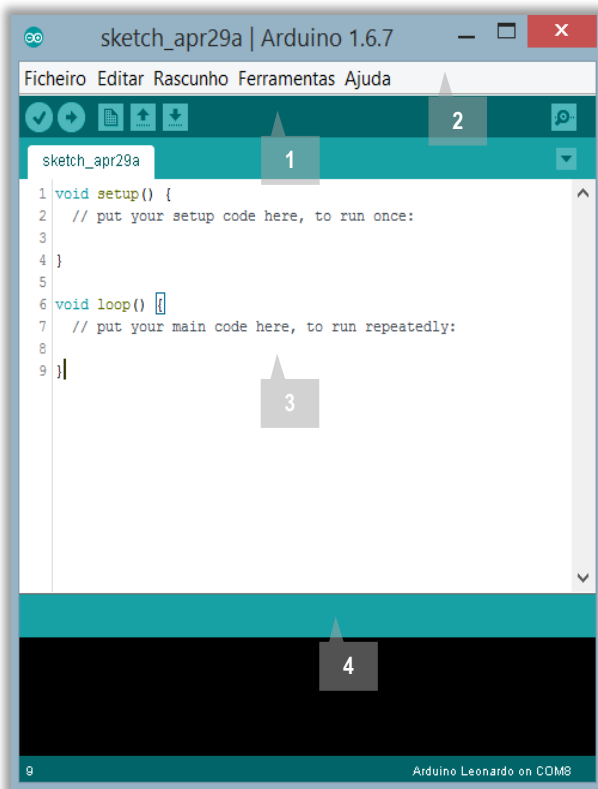
### → IDE INSTALLATION

**Software download:** <http://Arduino.cc/en/Main/Software>

**NOTE:** You must choose the specific installer for your operating system. Although ARDUINO IDE versions are available for various operating systems, this manual will use Windows as a reference. In this case you can choose to install the IDE by one of two methods: via installer (Windows Installer version) or via “*portable*” (Windows zip file version). The simplest method is to use the “*portable*” version that can be used without the need of installation.

On PCs with other operating systems where the procedures are slightly different, see the details for other cases in: <http://www.arduino.cc/en/Guide/HomePage>

## 4.2 ARDUINO IDE INTERFACE



Img 4 Arduino IDE interface

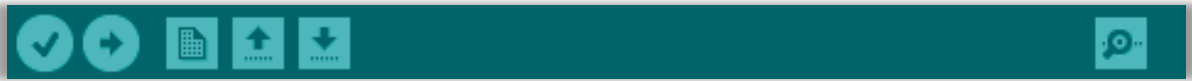
Behind the seemingly simplistic interface (Img 4), a powerful programming tool is available.

The IDE is organized as follows: a toolbar at the top, the code edit window in the middle, and the series output window at the bottom. The toolbar has 6 buttons and underneath it, a strip with tabs appears in the case of the “sketch” (see glossary) having multiple files. The text in each tab will be the file name of the respective “sketch”.

There is also an additional button placed at the right side that allows you to perform operations on the tab strip. The toolbar buttons allow quick access to frequently used menu functions.

1

### TOOLBAR



The **Verify** button (“Verify/Compile”) is used to verify that the written code text is correct.



The **Upload** button, when pressed, loads the current code into the ARDUINO memory. In order for this to be successful, it is necessary to previously make sure that the selected board and port (in the Tools menu) are correct. It is recommended that you save the sketch file in a disk before sending. We also recommend checking the code before doing it so, to ensure that there are no errors.



The **New** button allows you to create a new sketch. You must enter a name and the path to save it. Whenever possible, choose the directory displayed by default for this operation. After doing this, the tab bar above the code window will now display a tab with the name given to the “sketch”.



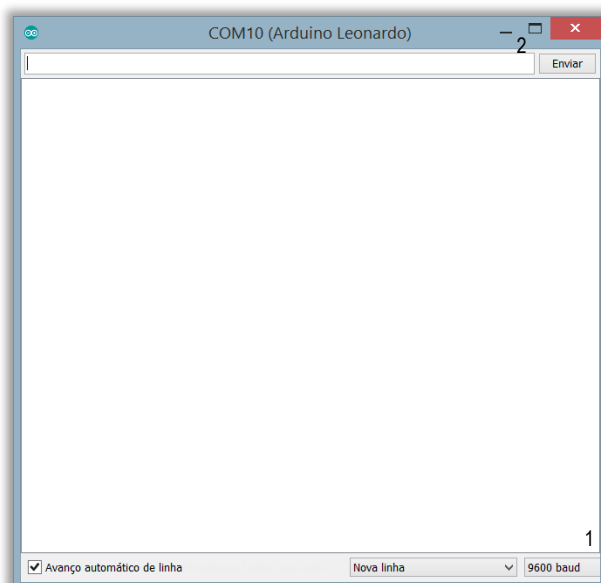
The **Open** button lists the “sketches” stored, the “sketches” examples and also allows to open a sketch in any location of the file system.



The **Save** button allows you to save the code. If the file has already been created it is updated, otherwise it works as **Save As**, in which case it is necessary to indicate the name and the path to be given to the new file. When the save is completed the message appears at the bottom of the code window: **Done saving**



The **Serial Monitor** (img 5) is a very useful tool, especially for clearing the code (debug (see glossary)). The monitor shows the serial data transmitted by MMkit (through the USB port or serial port). You can also send serial data to MMkit using the Serial Monitor. When the button is selected a new window is opened (img 5), in this case with the title COM10, since this is the serial port used in connection with the MMkit.



Img 5 Serial Monitor

**1** On the lower right side of the Serial Monitor you can select the bidirectional **serial communication** (see glossary) data rate between MMkit and PC. The default setting for the baud rate is 9600 baud, which means that if we want to send a certain text using this serial communication line (in this case the USB cable), will be sent 9600 bits per second.

**2** At the top of the window there is a blank text box that allows you to set text to be sent to MMkit when you press the **Send** button. Note that no serial data will be received by the Serial Monitor or MMkit, unless this functionality has been properly programmed in the sketch code that is loaded.

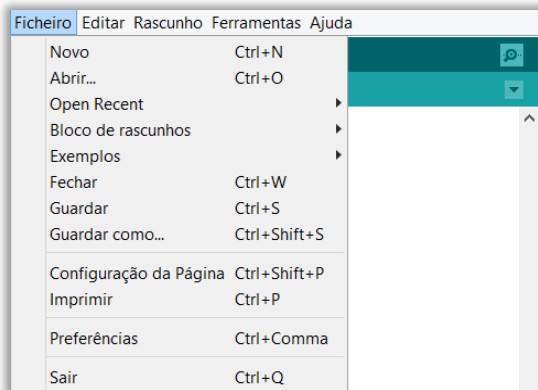
**3** Finally, the central area is where the serial data received from MMkit are displayed.



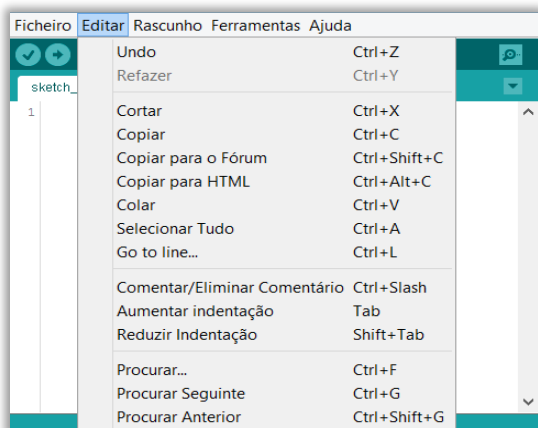
## MENU BAR

Ficheiro Editar Rascunho Ferramentas Ajuda

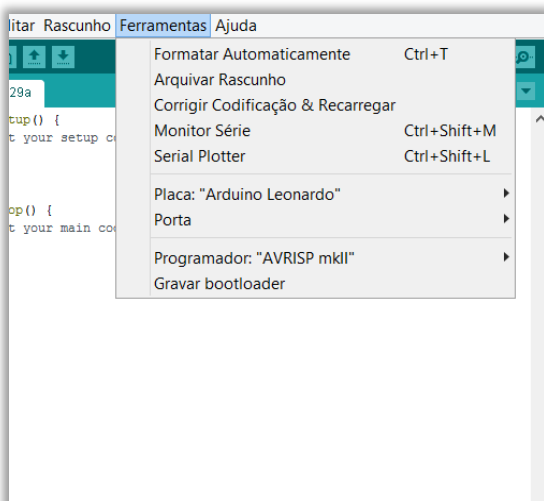
At the top of the window there is a menu, as shown above, with the following items **File** (Img 6), **Edit** (Img 7), **Tools** (Img 8), **Sketch** (Img 9) and **Help** (Img 10), each of them being a drop-down menu (when we click a panel with options unrolls).



Img 6 File menu



Img 7 Edit menu

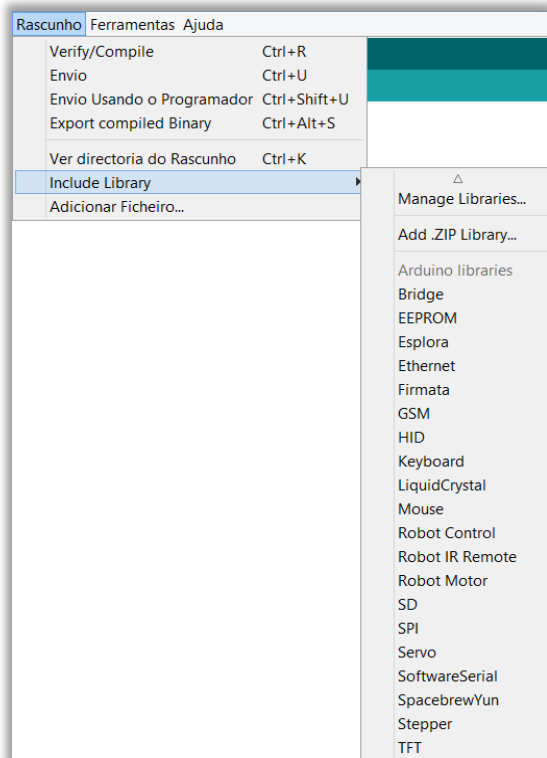


Img 8 Tools menu

The **FILE** menu contains the options: create **New** “sketch”; **Open** and search “sketches” existing in Sketchbook as well as **Examples**; **Save** or **Save As** the actual sketch; **Upload** the sketch to the board; **Print**; **Preferences**, where you can change various IDE options; **Quit** which closes the IDE.

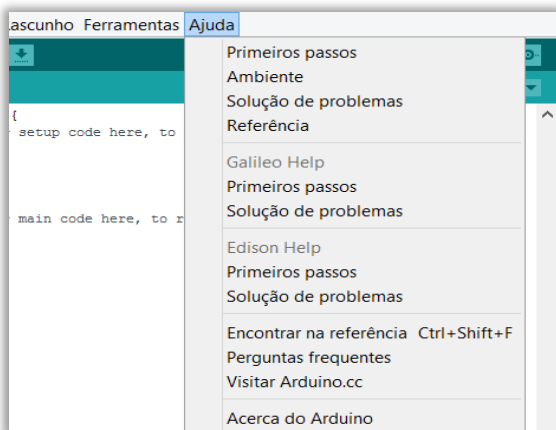
The **EDIT** menu provides the editing options that allow you to **Cut**, **Copy** and **Paste** sections of code, **Select All** the code, as well as **Find** certain words or phrases. The **Copy for Forum** option will copy the code inside the sketch window, but in a format that when pasted in the ARDUINO Forum will be similar to the IDE. It also includes the **Undo** and **Redo** options for the latest changes.

In **TOOLS**, there are the options to select the **Board**: **“Arduino Leonardo”** and the **Serial Port** that will be used to communicate with the board. Here is also available the **Auto Format** function that formats the code to a more readable. The **Archive Sketch** option allows you to compress and save the sketch in a ZIP file. Finally, the option **Burn Bootloader** that can be used to record an ARDUINO Bootloader (see glossary).



Img 9 Sketch menu

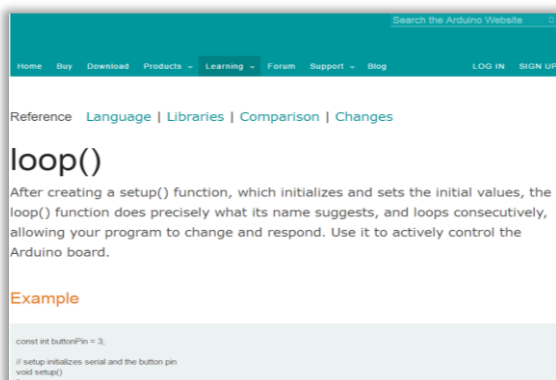
In the **SKETCH** menu we have access to the functions **Verify/Compile** and **Include Library** which shows a list of the available libraries, stored inside its folder of libraries. For example, one of the libraries that can be found, **Stepper**, which is a set of functions that can be used in the code of a sketch to control a stepper motor. Thus, including the Stepper library in a "sketch" we are able to use the functions to control a motor. The user can create his own function libraries, reuse them in different projects, and thus avoid repeating complicated code writing. There are also available the **View Sketch Folder** and **Add File** options, which will be useful in large projects in which it is convenient to break them into several sketches, each one in your file and all in the same directory.



Img 10 Help menu

The **HELP** menu presents a series of options that are no more than a connection table to help on the **Environment IDE**, the programming language of the **ARDUINO Reference**, and other related aspects. The **About** option shows various information about the **ARDUINO** project, such as the current version and the list of people involved in the development of this device.

One of the options available here, **Find in Reference** (or by the shortcut **Ctrl+Shift+F**), is very useful since it allows access to the information about a certain word of the language from the code itself. To test this functionality, in an open sketch in the IDE, place the cursor over the word **loop** (see glossary) and simultaneously press the **Ctrl+Shift+F** keys. A page of the documentation of this function will appear in your browser (**Img 11**).

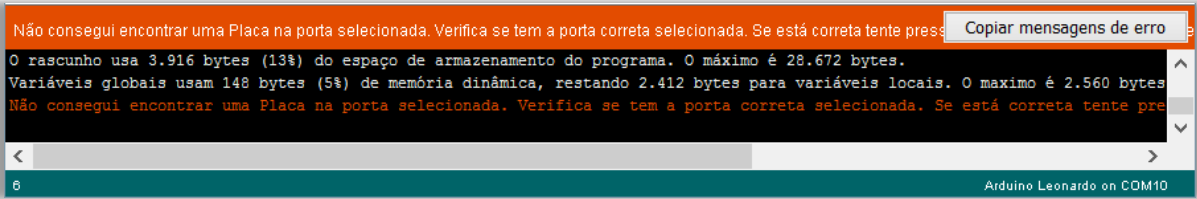


Img 11 Page of the documentation loop opened through the shortcut **Ctrl+Shift+F**

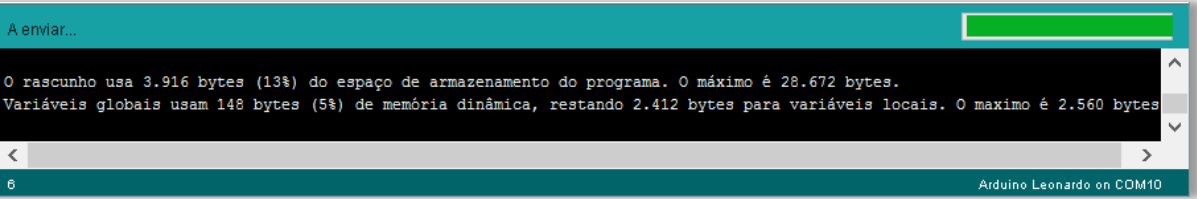


## 2 SERIAL OUTPUT WINDOW

Back in the IDE window, there is an area in the lower part of the window where there are error messages (in red) that may occur when trying to connect to the board, to compile or to load the code (img 12). In the lower left corner, a number appears that corresponds to the current cursor line. This information can be useful when looking for the line where there is a certain error pointed out by the compiler in an error message.



Img 12 Error message example



Img 13 Example of uploading the sketch to the board

## 5 MMkit INSTALLATION AND CONFIGURATION

The first time the PC is used to work with ARDUINO on WINDOWS or another operating system, some settings have to be made. Once the PC is set up, there is no need to go through these procedures again.

### 5.1 EQUIPMENT

Computer with USB port | USB cable type A - B | 2 Batteries type AA 1.5V | Arduino IDE | MMkit



Img 14 Identification of mandatory equipment

- 1 Place the MMkit board and the Micro USB type B cable on the table. Connect the cable to the Micro USB B port of the MMkit.
- 2 Then plug the end of the USB cable (type A) into the USB socket of your PC. A small green LED that says "ON" lights up to show that the board is powered. In WINDOWS is still missing a step to complete the use of MMkit for the first time: the installation of the board drivers.
- 3 As the board does not have the "Plug and Play" feature, ARDUINO will not be recognized automatically and the process will end with an error window. The solution involves manual installation of the drivers.



Img 15 Connecting MMkit to PC

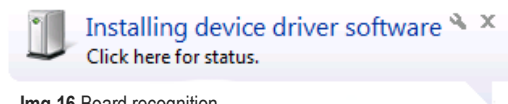


## 5.2 MANUAL INSTALLATION OF DRIVERS

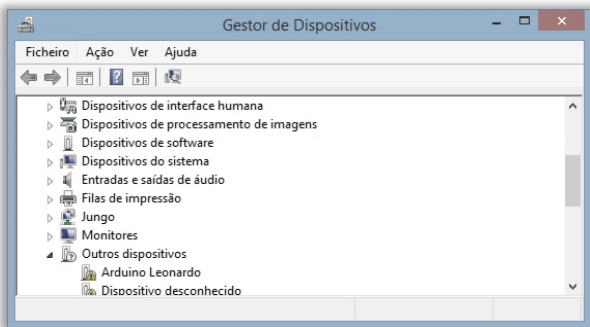


Img 17 Software is not installed

When connecting MMkit to the PC for the first time, it will try to recognize the board by displaying a window (Img 16). Sometimes MMkit will not be able to self-install (Img 17) and must follow the following steps to install it.



Img 16 Board recognition



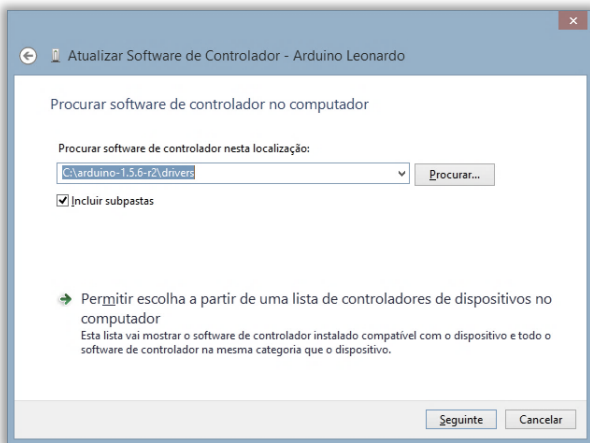
Img 18 Device Manager: Identifying "other devices"

### OPEN DEVICE MANAGER

1 In the 'Other devices' (Img 18) >> Choose 'Arduino Leonardo'

2 Right click on the device 'Arduino Leonardo' >> choose the option "Update Software Driver..."

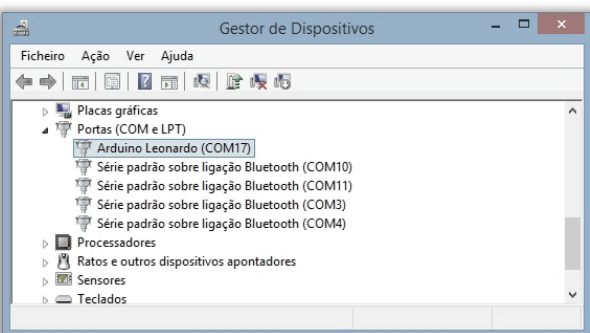
3 A window appears (Img 19) to search for the Arduino IDE installation folder (usually in C:\ or C:\Programs or where you unzipped it, if you chose the zip version of the IDE) >> select the folder Drivers that is inside. (This is the drivers folder that is inside the folder where you saved the previously decompressed files. If you are following the indications in the manual should correspond to: C:\Arduino-1.6.7\drivers)



Img 19 Arduino IDE installation folder lookup window

4 When you click "Next" (Img 19), the controller must be installed and a message of success will be shown.

5 If the "software" is installed correctly, the list of devices should show the Arduino Leonardo device in the Ports (COM and LPT) (Img 20). If the board isn't recognized, disconnect the USB port and reconnect and repeat the process again from the beginning.

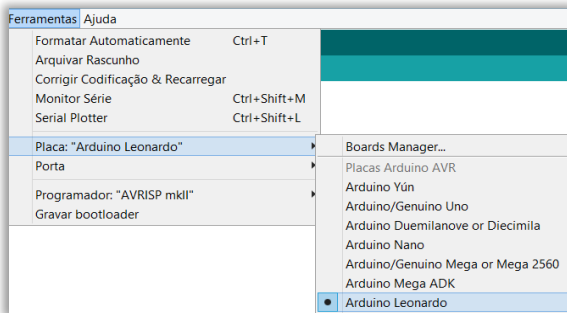


Img 20 Introducing Arduino Leonardo in the list of devices

**NOTE:** Identifying this port is important since you will need to use it in the serial port configuration in the IDE.

### 5.3 MMKit IDENTIFICATION

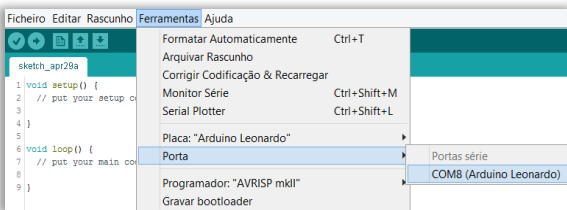
After installing the “hardware”, the MMkit must be properly identified and selected in the Arduino IDE settings.



Img 21 Selecting the board, in this case “Arduino Leonardo”

1 Select version of the ARDUINO board (img 21). In the **Tools** menu, confirm if the board Arduino Leonardo is selected in the section **Board: “Arduino Leonardo”**. If this is not the case, select it.

2 To configure the USB port (img 22), Click **Tools** again and select the **Port** option. A list of the serial ports available on your computer will be displayed, choosing the one referring to the USB cable that connects to the MMkit. You should select the port that was set when installing the drivers.



Img 22 USB port configuration

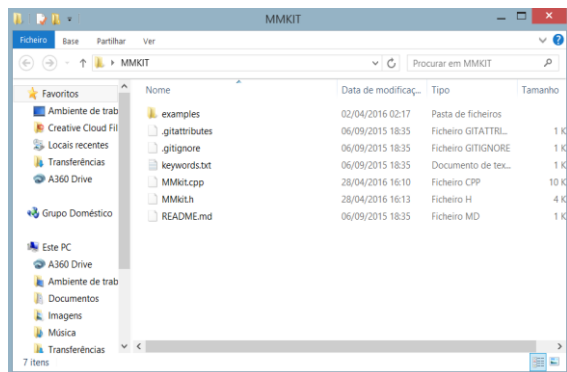
## 6 LIBRARIES INSTALLATION

As explained previously in the **Sketch** menu we have access to the **Include Library** function. In the following steps we will download and import the **MMkit library**, **AccelStepper** and **TimerOne**.

1 **Libraries download** <https://github.com/MMkit/MMKit>

<https://code.google.com/p/arduino-timerone/downloads/list>

2 Identify a folder corresponding to the library or a set of libraries by checking the structure of the folder (img 23).



Img 23 Type of a library folder

Normally, an Arduino library is composed by a folder with a structure identical to that of the represented figure, which includes:

Example Folder – Optional

File `<library>.cpp`

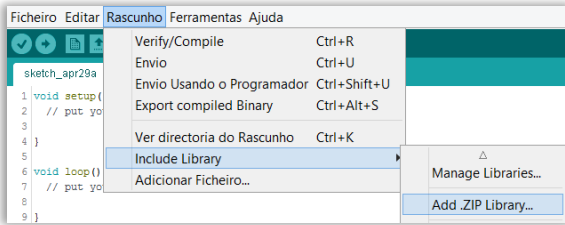
File `<library>.h`

keywords.txt – Optional

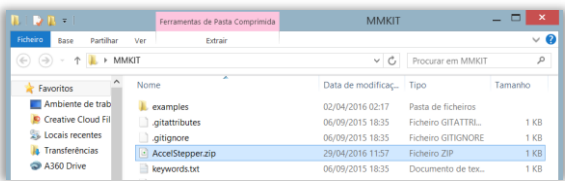
library.properties – Optional



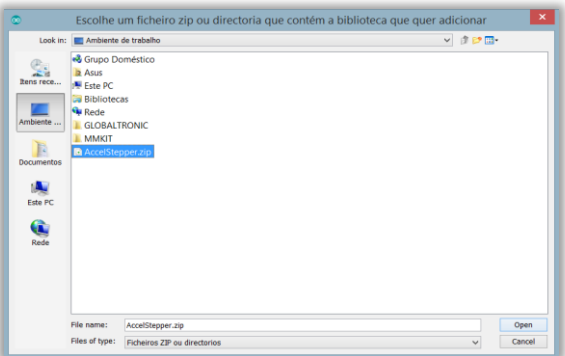
3 Once the libraries have been downloaded and the corresponding folders have been identified, we will proceed to import them, taking into account one of the following procedures.



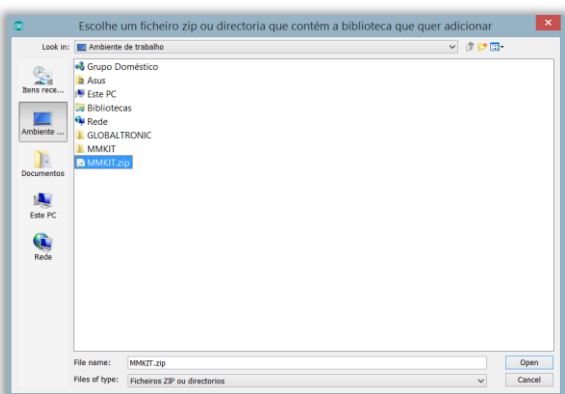
Img 24 Import library



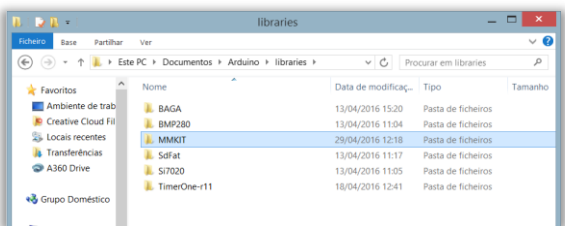
Img 25 File to be deleted inside the MMKIT library



Img 26 File to import



Img 27 Folder to extract



Img 28 Unzipped folder to store in the Arduino library

### VIA file.ZIP

1 After the MMkit library is downloaded, the **AccelStepper** library needs to be extracted from the MMkit zip folder.

Since this library is required for the control of stepper motors. (for compatibility reasons we decided to include it in the download).

2 Then delete the AccelStepper zip from the MMkit zip folder (Img 25).

3 Proceed with the imports into the Arduino software as follows: open the **Sketch** menu >> **include Library** >> **add Library** (Img 24).

4 Choose the AccelStepper zip file that you just extracted and import it (Img 26).

5 Repeat steps 2 and 3 for the MMkit and TimerOne library.

### MANUALLY

1 After the MMkit library is downloaded, unzip the library file to be installed and, in this case, extract the folder from the **AccelStepper** library as you did in the previous procedure (Img 27).

2 Identify whether the folder corresponds to an Arduino library or a set of libraries by checking the structure of the folder.

3 Copy the library to the Arduino libraries folder: **Documents** >> **Arduino** >> **libraries** (Img 28).

4 When the **Arduino IDE** restarts, the new library should already be listed.

5 Repeat the same procedures for all libraries.

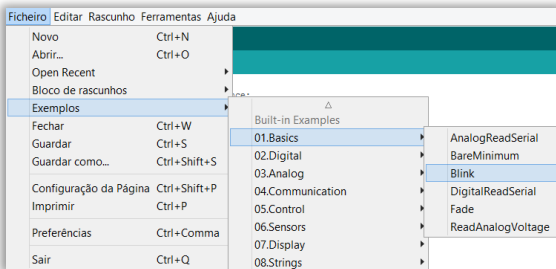


## 7 USE THE MMkit

### 7.1 UPLOAD THE FIRST SKETCH (LED flashing)

Now that MMkit is plugged in and the drivers were successfully installed, you're ready to load your first sketch and try ARDUINO for the first time.

Before creating a new project for the "micromouse", the "sketches" example included in the library should be tested to better understand of its operation. Example of "sketches" can be loaded from the library submenu in **File >> Examples** (Img 29). In this case, we will choose the **Blink** option (Img 30).



Img 29 Open a sketch from the File menu

```

1 /*
2  Blink
3  Turns on an LED on for one second, then off for one second, repeatedly.
4
5  Most Arduinos have an on-board LED you can control. On the Uno and
6  Leonardo, it is attached to digital pin 13. If you're unsure what
7  pin the on-board LED is connected to on your Arduino model, check
8  the documentation at http://www.arduino.cc
9
10 This example code is in the public domain.
11
12 modified 8 May 2014
13 by Scott Fitzgerald
14 */
15
16
17 // the setup function runs once when you press reset or power the board
18 void setup() {
19   // initialize digital pin 13 as an output.
20   pinMode(13, OUTPUT);
21 }
22
23 // the loop function runs over and over again forever
24 void loop() {
25   digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
26   delay(1000); // wait for a second
27   digitalWrite(13, LOW); // turn the LED off by making the voltage LOW
28   delay(1000); // wait for a second
29 }
    
```

Img 30 Blink code



Img 31 Example of MMkit with LEDs turned on

During the loading time of the sketch (in this case lasts a few seconds) notice the small RX and TX LEDs (red and green led) (see glossary). The board is receiving the new program via serial communications. In the lower section of the IDE window, several messages can be read while loading:

1 Compiling sketch

2 While the data is sent to the board, the message "Uploading" is displayed with a progress bar during the process.

3 Finally, the "Done Uploading" confirmation message appears.

At this point the RX and TX LEDs go off, the MMkit automatically resets and starts executing the sketch that was loaded in its memory. The Blink sketch is very simple and only serves to blink the red LED.

If the red LED is blinking with a fixed time interval; congratulations your MMkit, PC and IDE are correctly configured, if not, then we will have to try to reconfigure the board. Next, some brief explanations of sketch examples from the MMkit library in order to know their functions.





## 7.2 SKETCH EXAMPLES FROM THE MMKit LIBRARY

Before creating our own projects for the micromouse, in the following points we will approach in more detail and clarity some sketches of the MMkit library, that will allow us to better understand the practical usability of its functions and how they can interact with each other in order to make the process of developing a project for the micromouse as intuitive as possible.

### 7.2.1 PROJECT 1: Turn on LED with button

In this project, what is intended objectively is by buttons (control switch) to turn the red LED on or off. To do this write the code shown below (img 31).

#### SKETCH

```

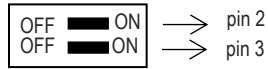
1  /*
2  This example was created by Sérgio Silva
3  on the 20th April 2015
4
5  This code example is in the public domain.
6
7  Description:
8  The robot moves forward the desired distance in cm
9
10 */
11
12 void setup() {
13   pinMode(2, INPUT);
14   pinMode(3, INPUT);
15   pinMode(13, OUTPUT);
16
17 }
18
19 void loop() {
20   //read the pushbutton value into a variable
21   if (digitalRead(3) || digitalRead(2)) {
22     digitalWrite(13, HIGH);
23   } else{
24     digitalWrite(13, LOW);
25   }
26   delay(1000);
27 }

```

Img 32 Example of Sketch (turn on LED with button)

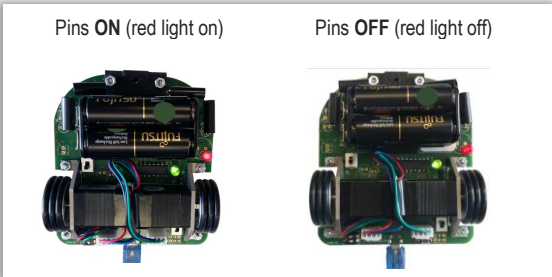
#### DISCUSSÃO DO SKETCH

The MMkit is equipped with a red LED connected to the pin 13 of the microcontroller and the control switch with two buttons connected to the pins 2 and 3, respectively. By its configuration, when the switch has the buttons turned to the right, they are connected, that is, the red LED turns on. If they are turned to the left, they are turned off.



Inside the **void setup** function (img 32) (see glossary) we configure the pins (**pin mode** (see glossary)) that initialize and define the initial values, being executed only once. In this case, we are considering that the buttons connected to the buttons are **INPUT**, which will allow us to read the button (whether it is on or off). The pin 13 (red button) will be configured as **OUTPUT**, that is, it will provide power to turn the LED on or off. We can see this example on **13, 14 and 15 line** (img 32).

→ It is from **line 19**, inside the function **void loop{}** (see glossary), that the program to execute will be written.



Img 33 MMkit examples with the on and off buttons

In the **line 21** (img 32) we start the **if condition** (see glossary) by a boolean operator (see glossary) **OR ( || )**, which tells us if pin 3 or pin 2 has been changed (on or off), through **digitalRead()** (see glossary).

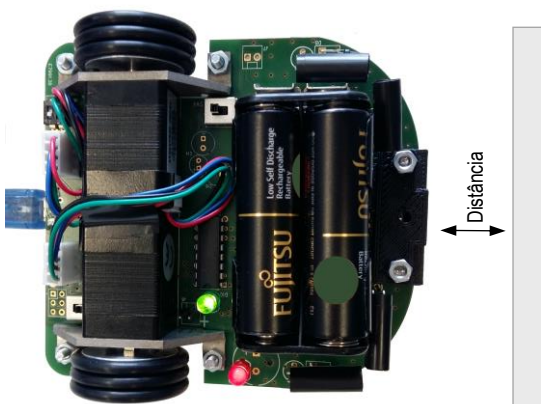
In lines 22, 23 and 24 (img 32), if the above condition is confirmed then we will turn the LED on or off. In this sense if the pins are **ON** the red LED will turn on: **DigitalWrite(13, HIGH)** (see glossary), otherwise, if they are **OFF**, it will turn off: **DigitalWrite(13, LOW)** (see glossary).

### 7.2.2 PROJECT 2: Measure distances using IR LED

In this project you will learn some concepts related to the LEDs (see glossary emitters and the phototransistors mounted on your MMkit. Let's make the brightness of the mounted LED vary in intensity depending on how far a white object is from your MMkit.



Img 34 Identification of phototransistors and emitter LEDs



Img 35 Measurement of the distance between MMkit and the wall

The MMkit is equipped with 4 IR emitters (infrared) (img 34), 2 front and 2 diagonals, these LEDs are connected to pins 4, 5, 6 and of the microcontroller. In the following Sketch we will connect the IR LEDs for 3 micro seconds and turn them off immediately. It is important not to keep these LEDs turned on for too long as they receive about 1 ampere current and would eventually burn. **The IR light is not visible to the naked eye, so to visualize the LEDs in operation, you must use the camera of your mobile phone.**

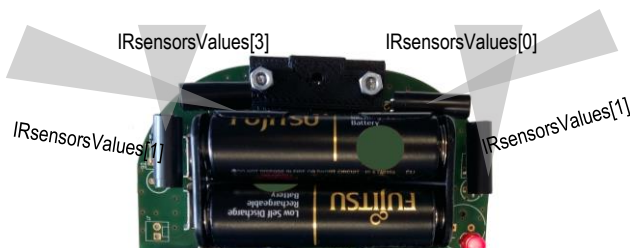
The MMkit is also equipped with phototransistors that are responsible for reading the values of the IR light reflected in the objects, the distance will always be a function of emitted light versus sensitivity of the receiver. The photoreactors (img 34) are connected to the A0, A1, A2 and A3 pins.

One of the most common problems with IR readings is interference from the ambient light. To reduce this, an efficient algorithm is used. First, it gets a reading of the IR sensors (dark\_read), turns on the IR LEDs, waits a little and reads the value on the ADC of the sensors (light\_read) then turns off the LEDs. The value will be the difference between read\_light and dark\_read.



→ In this project we will make the red LED glow depending on how far the object is from the MMkit.

The `readIRSensors()` (see glossary) function performs the procedure for reading the IR sensors referred to above. This function stores the read values into an array (set of values assigned to a variable) `IRsensorsValues[ ]`. Thus, we will have the value corresponding to the right sensor in `IRsensorsValues[1]`, The front right sensor in `IRsensorsValues[0]`, The front left sensor in `IRsensorsValues[3]` and the left sensor in `IRsensorsValues[2]` (Img 36).



Img 36 Example of IR sensors used to read a front wall

In analysis of the figure we verify that the distance from the MMkit to a frontal wall will be given by the values of `IRsensorsValues[0]` and `IRsensorsValues[3]`. In this first example we will use the sum of these values and divide them by 2 to calculate the distance to the obstacle (Img 36).

$$\text{Grigoros.IRsensorsValues}[0] + \text{Grigoros.IRsensorsValues}[3] / 2$$

→ Before we move on to the sketch, we will make a small preamble to talk about the `map()` function (see glossary).

The analog output of the MMkit is 8 bits, that is, it can simulate values between 0 and 255 ( $2^8 - 1$ ), being the inputs 10 bits, that is, the input value will be represented between 0 and 1023 ( $2^{10} - 1$ ). The `map()` function will be used to transform input range values (0 to 1023) into output range values. `map(valorEntrada, 0, 1023, 0, 255)`, where the `valorEntrada` is the value that we want to convert. This way, when the entry has the value 1023 the output of the `map()` function will be 255.

## SKETCH

```

1  /* Projeto 2 - Medir distâncias usando sensores IR
2
3  3  Adaptação: Talents & Treasures, Lda
4  4  @ 2015
5  5  */
6  #include <AccelStepper.h>
7
8  #include <MMkit.h>
9
10 AccelStepper motorLeft(AccelStepper::DRIVER, 9, 8);
11
12 AccelStepper motorRight(AccelStepper::DRIVER, 11, 10);
13
14 MMkit Grigoros(motorLeft, motorRight);
15
16
17 void setup() {
18
19   Grigoros.setupMMkit();
20 }
21
22
23 void loop() {
24
25   Grigoros.readIRSensors();
26
27   Serial.println(Grigoros.IRsensorsValues[0]);
28
29   Serial.println(Grigoros.IRsensorsValues[3]);
30   |
31   int valorEntrada = (Grigoros.IRsensorsValues[0]
32   + Grigoros.IRsensorsValues[3]) / 2;
33
34   analogWrite(13, map(valorEntrada, 0, 1023, 0, 255));
35
36   delay(100);
37 }

```

Img 37 Code to measure distances using the IR LED

## SKETCH DISCUSSION

→ In **line 19** (Img 37) is called the statement that auto configures MMkit. This function is responsible for defining the output ports for controlling the motors, for defining the LED port 13 (which we will use in this example), as well as other functions necessary for its operation.

→ In the **Loop** routine (see glossary) the function of **line 25** (Img 37) `Grigoros.readIRSensors()`; is responsible for the sensor reading algorithm. When 2 phototransistor readings are made, one with the IR LEDs turned on, other with them turned off, the values are subtracted and stored into the `IRsensorsValues[]` array.

→ In **lines 27 and 29** (Img 37) we print the read values to the serial port so we can see the value that each sensor is reading.

→ In **line 31/32** (Img 37) we create a variable with the name `valorEntrada` and we assign the value of the average of the 2 read sensors.

→ In **line 34** (img 37) we mapped the obtained value, so that it is between 0 and 255 and then write it in the LED 13. In this way, as the distance gets smaller, the greater the intensity of the LED's brightness.

→ In **line 36** (img 37), the `delay(100)` (see glossary) serves to read the values in the serial port (can be increased or decreased).

### 7.2.3 PROJECT 3: Move Forward (Move the MMkit)

In this project we will make the MMkit move for the first time. For this we will use 1 of the pre-installed examples of the MMkit library. Choose the menu **File >> Examples >> MMkit >> Basic >> MoveFoward**.

When we open the example, two tabs are opened simultaneously, the **MoveFoward** tab (img 36) that gives us the program to run and the **acceleration** tab (img 39) that defines the `acceleration()` function, which calculates the acceleration based on the speed and distance defined in the **MoveFoward** sketch.

#### SKETCH

```

1 /*
2  This example was created by Sérgio Silva
3  on the 20th April 2015
4  This code example is in the public domain.
5  Description:
6  The robot moves forward the desired distance in cm
7  */
8 #include <AccelStepper.h>
9 #include <MMkit.h>
10
11 AccelStepper motorLeft(AccelStepper::DRIVER, 9, 8);
12 AccelStepper motorRight(AccelStepper::DRIVER, 11, 10);
13
14 MMkit Grigoros(motorLeft, motorRight);
15
16 enum statesRobotMovements {
17     STATE_MOV_IDLE,
18     STATE_MOV_FRONT,
19     STATE_MOV_RIGHT,
20     STATE_MOV_LEFT,
21     STATE_MOV_UTURN,
22     STATE_MOV_STOP
23 };
24
25 unsigned char stateMovement = STATE_MOV_FRONT;
26 unsigned char toMove = STATE_MOV_FRONT;
27 unsigned int velocidad=10;
28 double acceleration=1;
29 void setup() {
30     Grigoros.setupMMkit();
31     Grigoros.goForward(20.0);
32     Grigoros.setForwardMotionSpeed(velocidade);
33     Grigoros.waitForStart();
34 }
35
36 void loop() {
37     if(Grigoros.running()==true) {
38         acceleration();
39         Grigoros.runSpeed();
40     }
41     else{
42         Grigoros.stop();
43     }

```

Img 38 MoveFoward code

```

1 void acceleration(void)
2 {
3     if (acceleration<=velocidade){
4         Grigoros.setForwardMotionSpeed((int) acceleration);
5         acceleration=acceleration+acceleration/100;
6         Serial.println((int) acceleration);
7     }else{
8         Serial.println((int) acceleration);
9     }
10 }

```

Img 39 Acceleration function code

#### SKETCH DISCUSSION

→ In **line 14** (img 38) is called the statement that auto configures the MMkit. This function is responsible for defining the output ports to control the motors.

→ In **line 16** (img 38) the possible conditions for the robot are listed. These movements can also be viewed as numbers from 0 to 5 where 0 is the IDLE (see glossary) and 5 the STOP.

**Line 17** (img 38) STATE\_MOV\_IDLE, does nothing;

**Line 18** (img 38) STATE\_MOV\_FRONT, moves forward;

**Line 19** (img 38) STATE\_MOV\_RIGHT, turn right;

**Line 20** (img 38) STATE\_MOV\_LEFT, turn left;

**Line 21** (img 38) STATE\_MOV\_UTURN, turn back;

**Line 22** (img 38) STATE\_MOV\_STOP, stops;

→ In **lines 25 and 26** (img 38) 2 variables are created that will save the movement to do (**toMove**) and done (**stateMovement**). To walk forward we use the **runSpeed** statement and to bend, the **run** statement.

→ In **line 27** (img 38) we define the speed that MMkit will move in cm per second. Therefore, 10 means that it will walk 10 cm in a second.

→ In **line 31** (img 38) we define the distance that we want to move the MMkit in cm. That means that we will move the MMkit 20 cm forward.



→ In **line 33** (img 38) the **Grigoros.waitForStart()** statement forces MMkit to wait for the hand to be passed in front of the 2 sensors, right and front, allowing adjustment before MMkit starts.

→ Inside the loop we have a conditional statement **if** (see glossary). This statement checks if the condition inside it is true, if it is, executes the first part, **Grigoros.runSpeed()**, that moves the MMkit to the speed specified previously, if it is false then it will make the MMkit to stop.

→ The **MAXIMUM SPEED** that the MMkit allows is limited by the acceleration ramp, without ramp the maximum speed will be 24 cm/s. An acceleration ramp is possible to reach speeds of about 3 meters/s, however, we advise caution in the use of these speeds, as the motors can also jam.

## 7.2.4 PROJECT 4: Move Forward (With interruptions)

In this project we will put the MMkit to move using interrupts. To make it easier, we'll use the **timerOne** library that lets you run interrupts more easily. In this way we open the **ISRblink** example (img 41) using the command **File >> Examples >> TimerOne >> ISRblink** and you will obtain the following sketch.

### SKETCH MoveFoward

```

8 #include <AccelStepper.h>
9 #include <MMkit.h>
10
11 AccelStepper motorLeft(AccelStepper::DRIVER, 9, 8);
12 AccelStepper motorRight(AccelStepper::DRIVER, 11, 10);
13
14 MMkit Grigoros(motorLeft, motorRight);
15
16 enum statesRobotMovements {
17     STATE_MOV_IDLE,
18     STATE_MOV_FRONT,
19     STATE_MOV_RIGHT,
20     STATE_MOV_LEFT,
21     STATE_MOV_UTURN,
22     STATE_MOV_STOP
23 };
24
25 unsigned char stateMovement = STATE_MOV_FRONT;
26 unsigned char toMove = STATE_MOV_FRONT;
27 unsigned int velocidad=10;
28 double aceleration=1;
29 void setup(){
30     Grigoros.setupMMkit();
31     Grigoros.goForward(20.0);
32     Grigoros.setForwardMotionSpeed(velocidade);
33     Grigoros.waitForStart();
34 }
35
36 void loop(){
37     if(Grigoros.running()==true) {
38         acceleration();
39         Grigoros.runSpeed();
40     }
41     else{
42         Grigoros.stop();
43     }
44 }

```

Img 40 MoveFoward code

### SKETCH ISRblink

```

1 #include <TimerOne.h>
2
3 void setup()
4 {
5     // Initialize the digital pin as an output.
6     // Pin 13 has an LED connected on most Arduino b
7     pinMode(13, OUTPUT);
8
9     Timer1.initialize(100000); |
10    Timer1.attachInterrupt( timerIsr );
11 }
12
13 void loop()
14 {
15     // Main code loop
16     // TODO: Put your regular (non-ISR) logic here
17 }
18
19 /// -----
20 /// Custom ISR Timer Routine
21 /// -----
22 void timerIsr()
23 {
24     // Toggle LED
25     digitalWrite( 13, digitalRead( 13 ) ^ 1 );
26 }

```

Img 41 ISRblink code

The **sketch ISRblink** code (img 39) will be diffused in the **Sketch MoveFoward** (img 38) so we can make a move with interruption.

But before we take a closer look at the sketch that allows us to move the MMkit using interrupts, we will briefly explain the **ISRblink** example, so later we will have a better understanding of the next “*sketch*”. This sketch will allow us to check the red LED connected to the pin 13, turning on or off from 100000 in 100000 microseconds.

In **line 7** (img 41), inside the **function void setup ()** (see glossary) we configure the pin 13 by assuming it as **OUTPUT**.

In **line 9** (img 41) we stipulate how often the **timerIsr()** function will be run. The measure unit set for the interrupt is in microseconds.

In **line 10** (img 41) the **timerIsr()** function is executed through the **Timer1.attachInterrupt( timerIsr )** statement.

In **line 25** (img 41), the **timerIsr()** function defines the value of the pin 13, that is, if it is off (LOW) or on (HIGH), via the **digitalWrite( 13, digitalRead( 13 ) ^ 1 )** statement.

#### SKETCH MoveFoward with interruption

```

6 #include <AccelStepper.h>
7 #include <MMkit.h>
8 #include <TimerOne.h>
9 AccelStepper motorLeft(AccelStepper::DRIVER, 9, 8);
10 AccelStepper motorRight(AccelStepper::DRIVER, 11, 10);
11 MMkit Grigoras(motorLeft, motorRight);
12
13 void setup() {
14     //configura o MMkit
15     Grigoras.setupMMkit();
16     Grigoras.goForward(20.0);
17     Grigoras.setForwardMotionSpeed(8);
18     // Grigoras.waitForStart();
19     Timer1.initialize(200);
20     Timer1.attachInterrupt( timerIsr );
21 }
22
23 void loop() {
24 }
25
26 void timerIsr() {
27     if (Grigoras.running() == true) {
28         Grigoras.runSpeed();
29     }
30     else {
31         Grigoras.stop();
32     }

```

img 42 Example of the combined MoveFoward and ISRblink code

#### SKETCH DISCUSSION

→ In **line 20** (img 42), after the **timer1** will be executed every 200 microseconds as defined in **line 19** (img 42). Lower values can lead to the call of a new interrupt without the previous one being finished since the **runSpeed** statement takes between 90 and 150 microseconds according to the adjustments that the MMkit have to make.

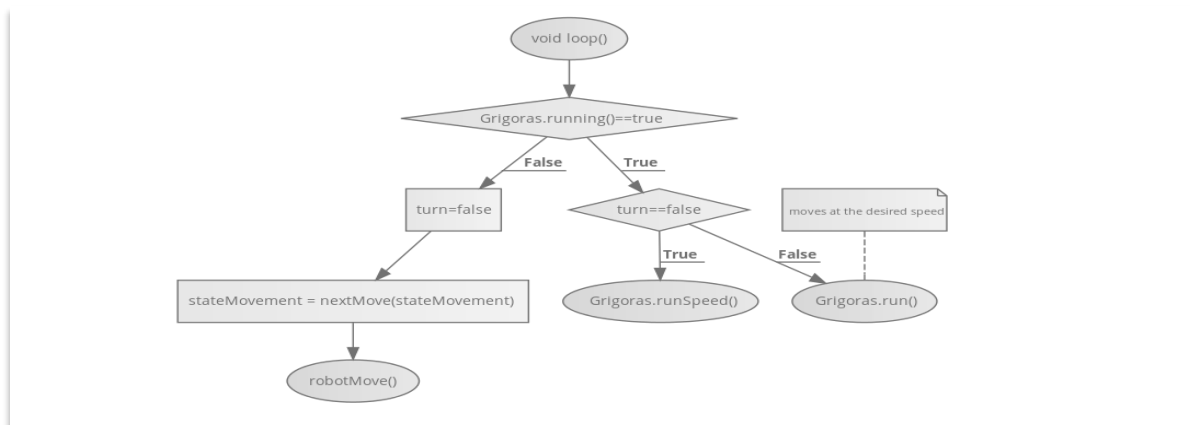
→ In order to combine the 2 projects, we went to the **ISRblink** sketch and place the **line 9 and 10** (img 41) inside the **void setup ()** and the statements that were within the loop of the project 3 are now defined on the **line 26** inside the **timerISR()** function.

### 7.2.5 PROJECT 5: RandomNextMove (algorithm)

The **RandomNextMove** algorithm allows the user a first introduction to coordinate movement of the MMkit. To do this, go to the **File >> Examples >> MMkit >> Advanced >> RandomNextMove**.

In addition to the new variables created, this example differs from the previous one by introducing two new steps in the loop. For a better understanding we created a flowchart (img 43) which will facilitate the reading of the code presented below.





Img 43 Flowchart for a better reading and understanding of the code

## SKETCH

```

11 #include <AccelStepper.h>
12 #include <MMkit.h>
13
14 AccelStepper motorLeft(AccelStepper::DRIVER, 9, 8);
15 AccelStepper motorRight(AccelStepper::DRIVER, 11, 10);
16
17 MMkit Grigoras(motorLeft, motorRight);
18
19 enum statesRobotMovements {
20     STATE_MOV_IDLE,
21     STATE_MOV_FRONT,
22     STATE_MOV_RIGHT,
23     STATE_MOV_LEFT,
24     STATE_MOV_UTURN,
25     STATE_MOV_STOP
26 };
27
28 unsigned char stateMovement = STATE_MOV_FRONT;
29 unsigned char toMove = STATE_MOV_FRONT;
30 boolean turn=false;
31 void setup(){
32     Grigoras.setupMMkit();
33     Grigoras.goForward(18.0);
34     Grigoras.setForwardMotionSpeed(500);
35     Grigoras.waitForStart();
36 }
37
38 void loop(){
39
40     if(Grigoras.running()==true) {
41         if(turn==false){
42             Grigoras.runSpeed();
43         }
44         else{
45             Grigoras.run();
46         }
47     }
48     else{
49         digitalWrite(13,HIGH);
50         turn=false;
51         stateMovement = nextMove(stateMovement);
52         robotMove();
53     }
54 }
55 }
  
```

Img 44 Example of the RandomNextMove code

## SKETCH DISCUSSION

The initial part of the code is very similar to the code **MoveFoward** sketch code, it is introduced a small change inside the **loop()** funtion (see glossary). This way, we will approach these changes taking into account the flowchart above (Img 43).

→ If MMkit has already reached the end of the last statement, the statement in **line 40** (Img 44) returns the value “false”, making us follow the left side of the flowchart running the instructions of lines **50, 51 e 52**.

→ In **line 50** (Img 44) we say that the variable **turn** is false, so if the next statement is not a turn, we use the **runSpeed** statement and not the **run**.

→ In **line 51** (Img 44) we call the **nextMove** function (giving it the value of the last move).

→ In **line 52** (Img 44) we call the **robotMove()** function which will cause the MMkit to move according to the decided in the previous function.

→ Both the **NextMove** and the **robotMove()** functions (Img 44), are defined inside the **loop()** function, which we will explain next.

```

80 unsigned char nextMove(unsigned char State)
81 {
82     unsigned char nMove=State;
83     switch (nMove) {
84     case STATE_MOV_FRONT:
85         nMove=rand()%2;
86         return nMove+2;
87         break;
88     case STATE_MOV_RIGHT:
89         return STATE_MOV_FRONT;
90         break;
91     case STATE_MOV_LEFT:
92         return STATE_MOV_FRONT;
93         break;
94     default:
95         digitalWrite(13,LOW);
96         return STATE_MOV_FRONT;
97         break;
98     }
99
100
101 }

```

Img 45 Example of the nextMove function

```

57 void robotMove(void)
58 {
59     switch (stateMovement) {
60     case STATE_MOV_IDLE:
61         Grigoras.stop();
62         delay(2000);
63         break;
64
65     case STATE_MOV_FRONT:
66         turn=false;
67         Grigoras.goForward(18.0);
68         break;
69     case STATE_MOV_RIGHT:
70         turn=true;
71         Grigoras.rotate(90);
72         break;
73     case STATE_MOV_LEFT:
74         turn=true;
75         Grigoras.rotate(-90);
76         break;
77     }

```

Img 46 Example of the robotMove function

→ In **line 84** (Img 45) if the previous move was to move forward, then:

→ In **line 85** (Img 45) we make a random with 2 hypotheses, either it returns 0 or 1.

→ In **line 86** (Img 45) we add 2, so that, this choice falls to either **STATE\_MOV\_RIGHT** or **STATE\_MOV\_LEFT** and returns the value.

→ This allows us that, whenever we move forward, we have the possibility to turn left or right. If the previous move was to move to the left or to the right, the next move will be to move forward.

→ In **line 59** (Img 46) we have, once again, a switch that allows us to choose between 4 options: IDLE, FRONT, RIGHT, LEFT and by choosing, perform 2 operations:

1 Assign the variable to "turn" the value of "false", when is not to change the direction, and orders it to move forward.

2 rotate depending on the situation in which we find ourselves.

→ In **line 65** (Img 46) where the option will be to move forward, "turn" will be false.

→ In **line 67** (Img 46) is given the instruction to move forward 18 cm.

## 7.2.6 PROJECT 6: RightWallFollow (algorithm)

The **RightWallFollow** algorithm provides to the user a solution to solve the mazes with MMkit. This example can be accessed in the tab "advanced". Choose in the menu **File >> Examples >> MMkit >> Advanced >> RightWallFollow**.

When we open the example, 5 tabs are opened simultaneously, the **RightWallFollow** tab that will give us the program to run, the **acceleration** tab, the **nextMove** tab, the **odometry** tab and the **robotMove** tab. In the last four are defined all the functions that are called in **RightWallFollow**.



## SKETCH

```

1 #include <AccelStepper.h>
2 #include <MMkit.h>
3 #define ANGLE_0 0x01
4 #define ANGLE_90 0x02
5 #define ANGLE_180 0x04
6 #define ANGLE_270 0x08
7
8 AccelStepper motorLeft(AccelStepper::DRIVER, 9, 8);
9 AccelStepper motorRight(AccelStepper::DRIVER, 11, 10);
10
11 MMkit Grigoros(motorLeft, motorRight);
12
13 enum statesRobotMovements {
14     STATE_MOV_IDLE,
15     STATE_MOV_FRONT,
16     STATE_MOV_RIGHT,
17     STATE_MOV_LEFT,
18     STATE_MOV_UTURN,
19     STATE_MOV_STOP
20 };
21
22 unsigned char stateMovement = STATE_MOV_FRONT;
23 unsigned char toMove = STATE_MOV_FRONT;
24 boolean turn=false;
25 unsigned int velocidad=10;
26 double acceleration=1;
27 void setup(){
28     Grigoros.current_cell.x = 5;
29     Grigoros.current_cell.y =10;
30     Grigoros.current_cell.theta = ANGLE_90;
31     Grigoros.setupMMkit();
32     Grigoros.goForward(18.0);
33     Grigoros.current_cell.y--;
34     Grigoros.setForwardMotionSpeed(velocidad);
35     Grigoros.waitForStart();
36 }
37
38 void loop(){
39
40     if(Grigoros.running()==true) {
41         if(turn==false){
42             acceleration();
43             Grigoros.runSpeed();
44         }
45         else{
46             Grigoros.run();
47         }
48     }
49     else{
50         digitalWrite(13,LOW);
51         turn=false;
52         toMove = nextMove(stateMovement);
53         robotMove();
54     }
55 }

```

Img 47 Example of the RightWallFollow code

## SKETCH DISCUSSION

→ Either in the **RightWallFollow** or **RobotMove**, the code is very similar to that discussed in the **RandomNextMove** example. The changes are on the **NextMove** tab. In this case the decision is no longer random and based on the previous instruction, but based in the perception obtained from the walls: **switch (Grigoros.current\_cell.wall)**.

→ It is important to remember that **Grigoros.current\_cell.wall** contains an 8-bit value in which the last 3 represent the left, right and front walls. 0 (zero) corresponds to having no wall, and 1 means that there is wall. By this means of this evaluation, a new decision of the next movement of MMkit is made.

→ In the **Odometry** tab we calculate the position of the robot, that is, it allows us to calculate the positioning measurements.

→ As discussed and explained in section 7.2.3, the **acceleration** tab defines the acceleration function.

### 7.2.7 PROJECT 7: Train the mazes with simulators

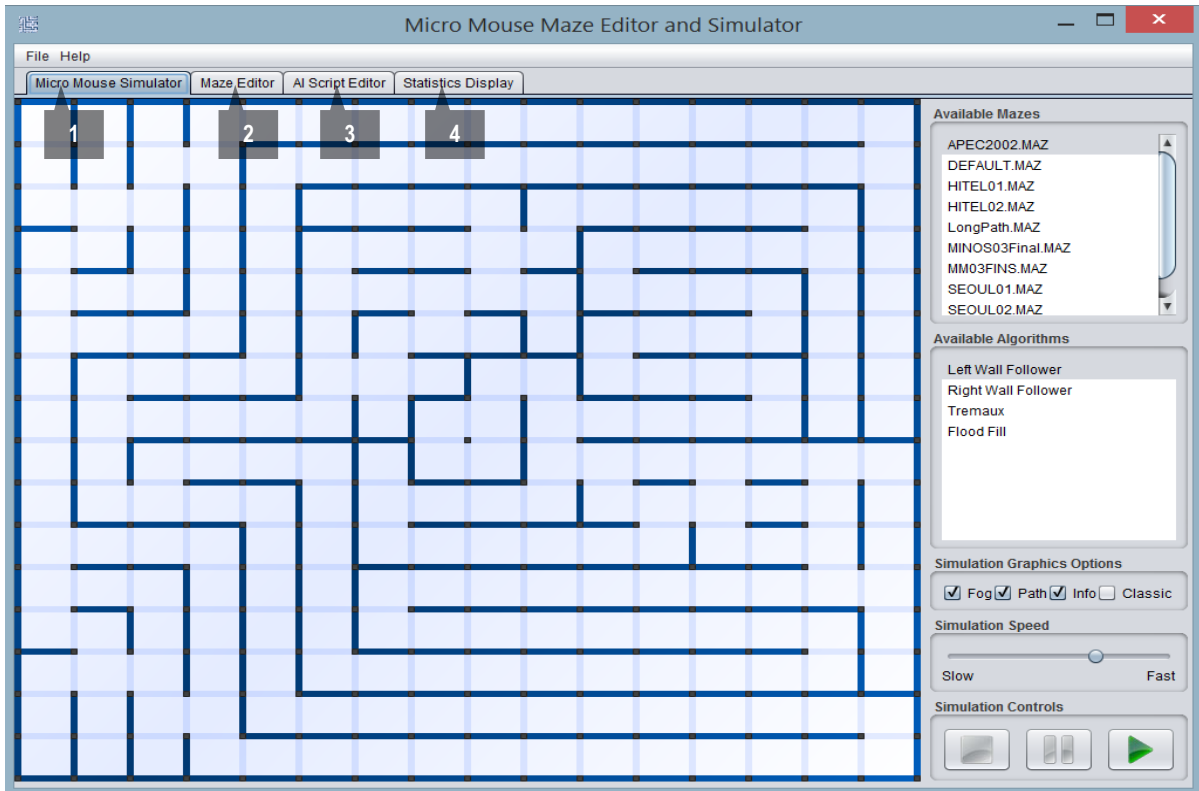
You can test new algorithms before implementing them in MMkit. For this we can use the maze-solver tool that will allow us to simply perform new algorithms and test them.

Available at one of the following addresses: (it is mandatory to have java installed)

→ <https://code.google.com/p/maze-solver/>

→ <https://github.com/MMKit/MMKit>

After downloading and installing the previous software, the following figure shows the maze-solver program running. In the next point we will take a brief approach to the tool interface and how we can take advantage of it.



Img 48 Maze Editor and Simulator interface

- 1 The first tab, **Micro Mouse Simulator** (img 48), presents the current maze and on the right side we have the possibility to choose the algorithms that will be executed when we press play.
- 2 In the second tab, **Maze Editor** (img 48), we can create different mazes, allowing us to add and remove walls and test it later.
- 3 In the third tab, **AI Script Editor** (img 48), beside including templates examples, we can also create our own algorithms and test their operation through the python language. This tab allows us to make quick code changes and immediately execute a new simulation.
- 4 In the fourth tab, **Statistics Display** (img 48), allows us to compare the performance of different algorithms and mazes.

As previously mentioned, the maze-solver simulator is based on the python language, so we will make some considerations on how to facilitate the translation of the C language used in ARDUINO for python. For those who do not know, python is a high-level, interpreted, and multi-paradigm language. One of its main features is to allow easy code reading and requiring few lines of code.

For those who are not so familiar with the language in python, we will make some observations to facilitate the translation into C language (img 49).

In this language, the indentation (space between left margin and the text) fits really well in the Arduino, so that the following instructions are equivalent.

**Python**

```
if maze.isWallLeft():
    next = nextMove;
return nextMove;
```

**Arduino**

```
if ( maze.isWallLeft() ){
    next = nextMove;
    return nextMove;
}
```

Img 49 Example of translation of the python language to Arduino (C language)

The **nextStep** function (Img 51) can be seen as our **nextMove** function (Img 50). In this way, giving the proper parallelism of the language, it will be easy to transpose the code created here for the MMkit examples, replacing what is necessary in the **nextMove** function.

Our functions for checking walls are **Grigoros.isWallLeft()**, **Grigoros.isWallRight()** or **Grigoros.isWallFront()** (Img 50), only changing the name "Maze" for "Grigoros" (Img 51). These functions return 1, if there is a wall and 0 (zero), if there is no wall.

We can see in the example below, how the **nextMove** function would look like by applying the previous code. The variable **next** (Img 50) is our **nMove** variable (Img 51).

```
1 unsigned char nextMove(unsigned char State)
2 {
3     unsigned char nMove = State; // middle of maze found
4     if (((Grigoros.current_cell.x == 7) || (Grigoros.current_cell.x == 8)) && ((Grigoros.current_cell.y == 7) || (Grigoros.current_cell.y == 8)))
5         if ((Grigoros.current_cell.x == 8) && (Grigoros.current_cell.y == 8))digitalWrite(13, HIGH);
6     return STATE_MOV_STOP;
7 }
8 if (nMove != STATE_MOV_IDLE) {
9     nMove = STATE_MOV_IDLE;
10    return nMove;
11 } else if (Grigoros.isWallLeft() == 0) {
12    nMove = STATE_MOV_FRONT;
13    stateMovement = STATE_MOV_LEFT;
14    return stateMovement;
15 } else if ( Grigoros.isWallFront() == 1) {
16    return stateMovement = STATE_MOV_RIGHT;
17 } else {
18    return stateMovement = STATE_MOV_FRONT;
19 }
20 }
```

Img 50 Example of Arduino language code (C language)

```
1 # Seguir parede Esquerda.
2
3 # guarda passo seguinte logo pudemos tomar 2 decisões seguidas.
4 next = None
5
6 # This function is called by the simulator to get the
7 # next step the mouse should take.
8 def nextStep():
9     global next
10    if next != None:
11        nextMove = next
12        next = None
13        return nextMove;
14    elif not maze.isWallLeft():
15        next = Forward # A seguir a virar manda andar frente
16        return Left # vira esquerda
17    elif maze.isWallFront():
18        return Right
19    else:
20        return Forward
```

Img 51 Example of Python language code

## 8 GLOSSARY

### ARDUINO

ARDUINO is a physical platform for embedded computing. It is an interactive system, which through the use of hardware and software allows to interact with the environment. In a simple way, ARDUINO is a mini computer that can be programmed to process the inputs and outputs of your chip. For more information consult the site <http://www.arduino.cc/>

### Debug

Debug means analyse. In general, we use this term when we want to understand a part of the execution of a code.

### Bootloader

Bootloader is a small program that exists in the microcontrollers memory and is responsible for system boot activities. Parallel to Windows, the bootloader is the boot of the operating system.

### Boolean operator

Boolean operator is a mathematical operation such as sum and subtraction, but is performed with operands True and False.

### Loop

Loop is a statement (function), which is repeated continuously.

### Delay

Delay is an Arduino instruction that blocks the code execution during a period of time. This time is defined between parentheses in milliseconds, that is, 1000 is 1 second.

### IDLE

IDLE is a statement that means doing nothing (waiting state).

### digitalWrite

digitalWrite is a statement of the Arduino that allows you to write in the digital port (digital pin), the value 5 or 0 Volts (in digital, 1 or 0). In general, the statement uses the words HIGH = 5, Volts = 1 digital and LOW = 0, Volts = 0 digital.

### digitalRead

digitalRead is a statement that allows to read the value of the digital port indicated in parentheses.



### pinMode

pinMode identify in which state the Arduino's digital port should work (INPUT or OUTPUT).

### map

The map function allows the user to convert the read values to other values.

### Void

Void word means that the function does not return any value.

### Setup

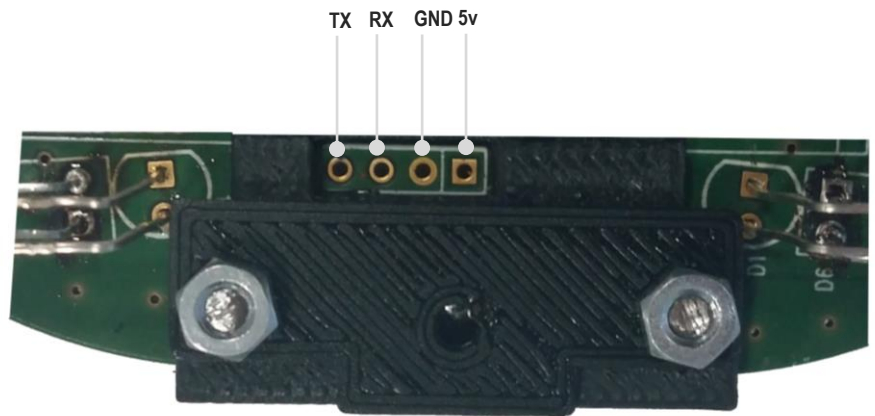
Setup is the function that is only executed once, when the Arduino is rebooted.

### Arduino Leonardo

Arduino Leonardo is the base model for the Micromouse.

## SERIAL COMMUNICATION

Serial communication is a process of exchanging data between digital devices. All ARDUINO boards have at least one serial communication port (which are also known as UART or USART ports). In serial communications, the data bits are transmitted sequentially in a queue, one bit at a time. In ARDUINO the serial communication uses the digital pins 0 (RX) and 1 (TX) as well as the USB interface with the PC, therefore it is not recommended to use the digital pins 0 and 1 for input / output when we want to use serial communication in our sketches. The Arduino Leonardo has a second UART to UART1 that is connected to the Bluetooth pins that to activate must use the statement **Serial1.begin(9600)** in the setup in substitution of **Serial.begin(9600)**.



Img 52 Possible bluetooth connection

### DIODE

A diode (img 53) is a device that allows the passage of current in one direction only. In a hydraulic analogy, it is a valve that allows the passage of water in only one of the directions. The diodes may be useful to prevent that, if someone accidentally inverts the power and ground connections in a circuit, the circuit components are not damaged.



Img 53 Diode by oomlout (1N4001 - DIOD-01) [CC BY-SA 2.0 (<http://creativecommons.org/licenses/by-sa/2.0>)]

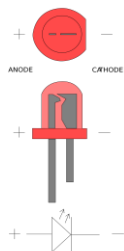


Img 54 Electrical diode symbol (By: Omegatron (Diode\_symbol.svg) [CC BY-SA 3.0 (<http://creativecommons.org/licenses/by-sa/3.0>) or GFDL (<http://www.gnu.org/copyleft/fdl.html>)], via Wikimedia Commons)

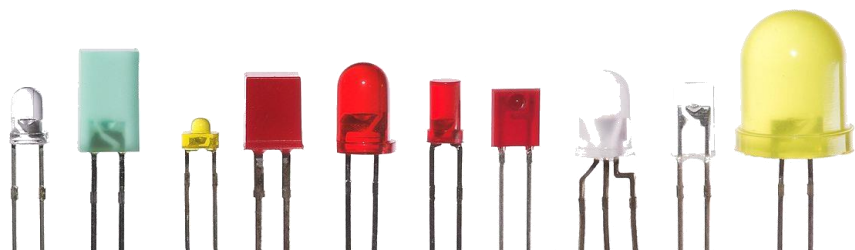
### Light Emitting Diode - LED

LED (img 55) is the acronym of "Light Emitting Diode". It is diode that when crossed by electric current emits light. There are LEDs of the most varied colors and brightness, and can also emit ultraviolet or infrared light (used, for example, in the remote controls of televisions). If we look closely to a LED, we will notice two things: one is that the legs are of different lengths and the other is that one of the sides is not cylindrical but flat. This allows to identify that the longest leg is the anode (positive pole) and that the shorter (or flat side) is the cathode (negative pole). For it to emit light, the longer leg must be attached to the highest potential and the shortest to the lowest potential.

If the LED is turned in the reverse direction it will not be damaged (unless it is subjected to very high current), but since it works as an insulator, there is no current flowing and therefore does not light up. When connected in the right direction, it is essential that a resistance is connected in series with the LED to limit the current flowing through it, otherwise it can burn. In addition to the normal ones that emit light of only one colour, there are also bi-coloured and tri-coloured. There are, for example, so-called RGB LEDs that are made up of 3 encapsulated LEDs, one red, one green and one blue. The RGB LED has 4 legs, one being the common lead (anode or cathode). By adjusting the brightness levels of R, G and B, a light of any colour can be obtained. This is the principle used in each pixel of the monitors or TV.



Img 55 LED scheme



Img 56 Some LEDs

(By A Frank99 (Own work) [CC BY-SA 2.0 (<http://creativecommons.org/licenses/by-sa/2.0>)], via Wikimedia Commons)



## FUNCTION

A function (sometimes referred as a procedure or subroutine) is a piece of code that can be invoked from another location in the sketch. See example:

```

1. void setup()
2. {
3.   pinMode(ledPin, OUTPUT);
4. }

```

In the **line 1** are described details about the function such as “setup”. The text that appears before the function name describes the return type and inside the parentheses, the input parameters. The code written between the symbols { and } correspond to the body of the function (the actions or processing).

Once the function is set, it can be called whenever necessary. The **line 3** corresponds to the call of the pinMode() function with the input parameters “ledPin” and OUTPUT.

To know more details of the ARDUINO language, see also: <http://www.arduino.cc/en/Reference/HomePage>

## loop() AND setup() FUNCTION

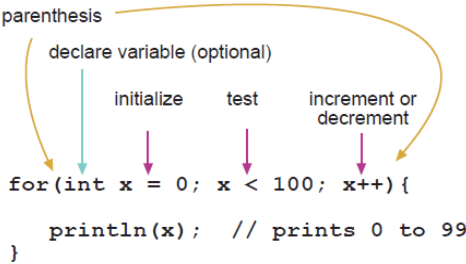
There are two special functions that are part of all ARDUINO sketches: setup() and loop().

The **setup()** function is executed only once, only at the beginning of the program. In it must be the tasks necessary to prepare the program, to enter the main cycle, such as configuring board pin operation modes, set serial baud rates, etc.

The **loop()** function is executed immediately and endlessly, that is, after setup() the ARDUINO sketch will be permanently executing the loop(). These two functions should be included in all ARDUINO sketches even if they are not being used.

## “FOR” STATEMENT

The **for** statement is used to repeat a block of code expressions. Usually, a variable is used to define the increment and stop value. It is often used with arrays to perform repetitive operations on collected data or pins.



## “IF” STATEMENT

The **if** statement is an example of a control structure that aims to check whether a particular condition has been reached (or not), and if so, execute the instructions in its code block (inside {...}). For example, if we want to connect a LED when the variable **x** rises above the value of 500, we can write:

```
if (x>500){  
  digitalWrite(ledPin, HIGH) ;  
}
```

## LOGICAL OPERATORS

The logical or boolean operators are the following:

Símbolo	Operador	
&&	E (conjunção)	AND
	OU (disjunção)	OR
!	NÃO (negação)	NOT

These logical operators can be used to test various conditions.

&& - will be true if both operands are true.

|| - will be true if at least one of the operands is true.

! - only has one operand and inverts its logical value, that is, it will be true if the operand is false.

You can also use expressions that involve several logical operators. For example:

```
if (x==5 && (y==10 || z!=25)) { ... }
```

## LINEAR VOLTAGE REGULATOR

It is an active device (such as a transistor) for voltage regulation. The operating principle resembles a variable resistor which continuously regulates a resistive voltage divider to maintain a constant output voltage. It is considered an inefficient method since it regulates the voltage generating heat, on the other hand it is very cheap.

## SKETCH

In the world of ARDUINOS, sketch is the designation given to programs made by users.

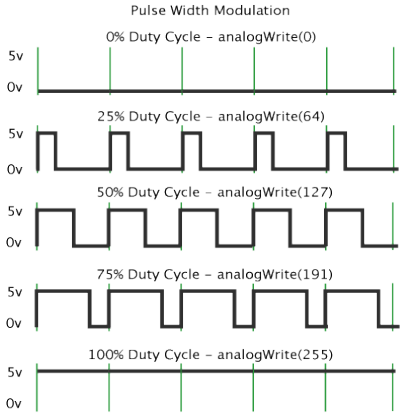
An ARDUINO sketch should always have the `setup()` and `loop()` functions, otherwise it will not work. See the `setup()` and `loop()` functions.





## PWM

PWM (Img 57), acronym of "Pulse-with modulation", is a technique that allows, by digital signals, to emulate an analog result. The digital control inherent in the microcontroller produces a square wave that stays in time in one of two possible states: 5V (ON) or 0V (OFF). By regulating the time in which the signal remains in one state and the other, it is possible to modulate the signal and, for example, control, the brightness of an LED, generate audio signals or control the speed of motors. ARDUINO exposes the pins 3, 5, 6, 9, 10 e 11 with the PWM function. On these pins besides the **digitalWrite()** function that allows you to write the values 0 or 1, the **analogWrite(int i)** function is also available which allows you to set values on a scale from 0 to 255, such that **analogWrite(255)** corresponds to 100% of the time in state 1 (also known as duty cycle 100%), while the **analogWrite(127)** corresponds to a duty cycle of 50% (that is, in a complete wave cycle, half the time is 0V and another half is 5V).



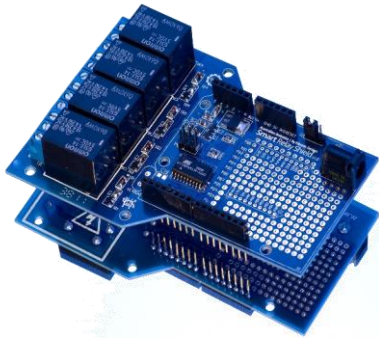
Img 57 PWM

## ARDUINO Shields (Extensions)

ARDUINO can be extended through the interconnection of *shields* (Img 58 and 59) which are circuit boards that contain other devices for specific use (e.g. GPS receivers, LCD monitors, Ethernet connections, etc.). These boards can simply fit onto the top of the ARDUINO to allow additional functionalities. The use of shields is not strictly necessary, since the same circuits can be mounted using a breadboard or even making our own extension boards.



Img 58 Ethernet



Img 59 Relay Shield

## DATA TYPE

The types of data we can use in sketches are following:

Each data type uses a certain amount of ARDUINO memory: some variables use only 1 byte, some 2, and others 4 or more (1 byte is 8 bits). You cannot copy data from one data type to another, for example, if x is an int and y is a string then  $x = y$  does not work because the two types of data are different.

Tipo de dados	RAM	Gama de valores
void	N/A	N/A
boolean	1 byte	0 to 1 (True ou False)
byte	1 byte	0 a 255
char	1 byte	-128 a 127
unsigned char	1 byte	0 a 255
int	2 byte	-32768 a 32767
unsigned int	2 byte	0 a 65535
word	2 byte	0 a 65535
long	4 byte	-2147483.648 a 2.147.483.647
unsigned long	4 byte	0 a 4294967295
float ou double	4 byte	-3.4028235E+38 a 3.4028235E+38
string	1 byte + x	Arrays de chars
array	1 byte+ x	Vector de variáveis

## VARIABLE AND CONSTANT

Variables and constants are the elementary parts manipulated by a program. Both correspond to memory spaces in memory for storing values.

A variable, during the execution of the program, can see the changed values.

Constants, as the name implies, cannot be changed during the execution time of the programs.

Both must be named so that they can be referenced. Some programming languages (ARDUINO language included) require that when declaring a variable as defining the type and sometimes even set an initial value. See also data types.