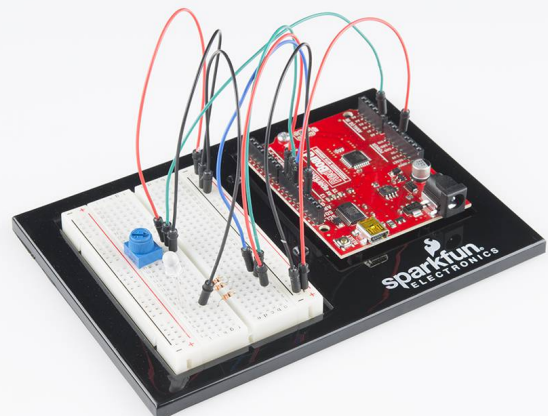
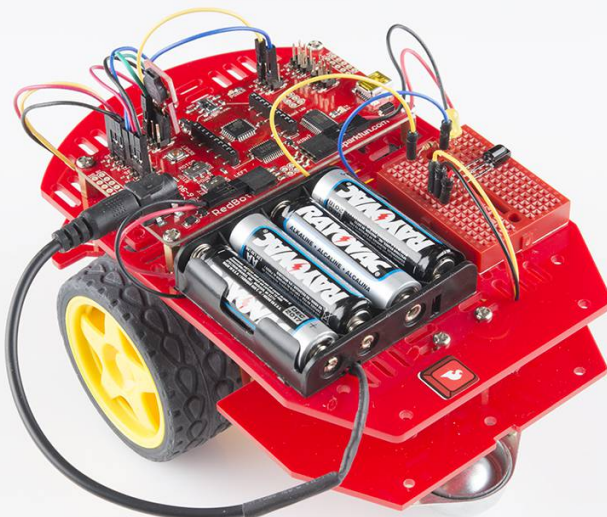
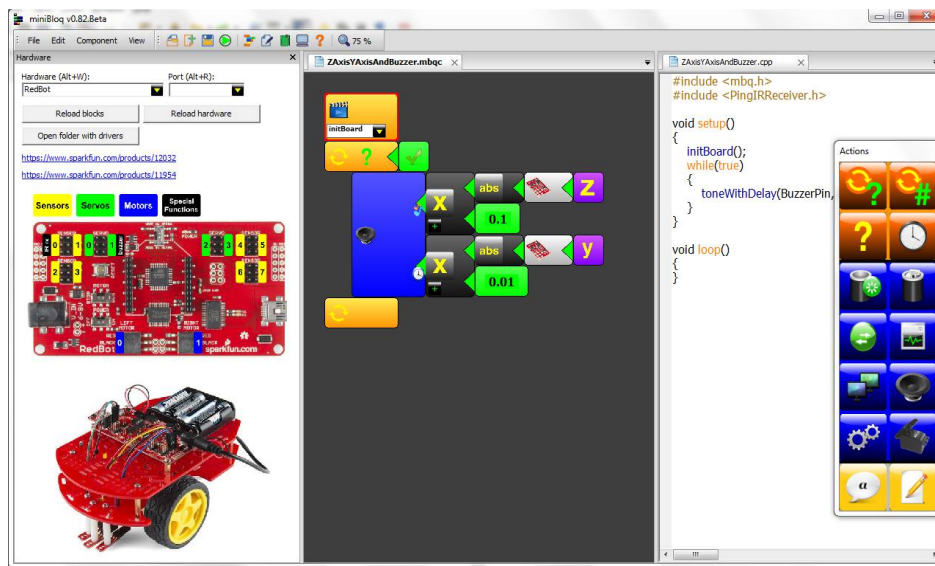


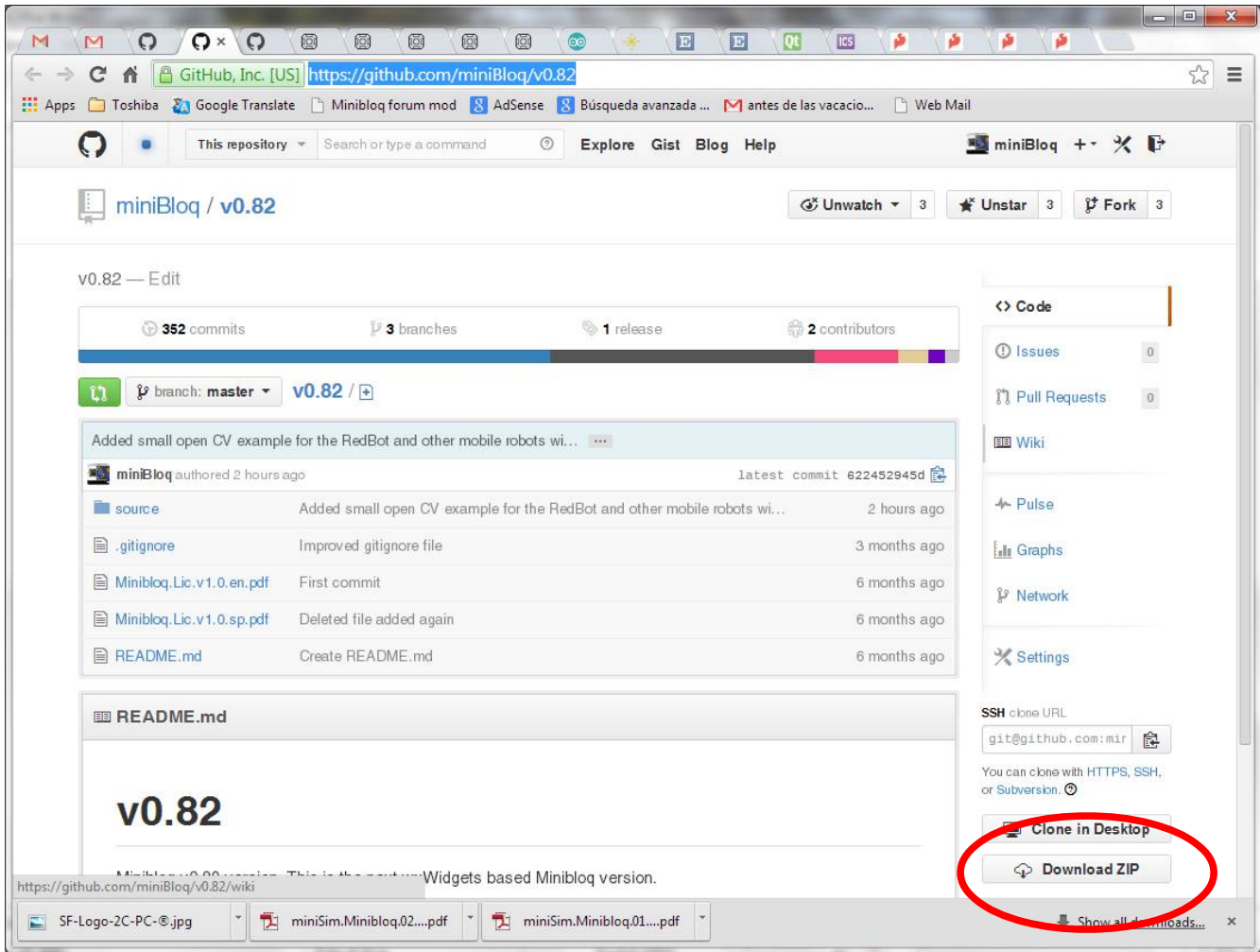
HIR 2014: miniBlox + RedBot + RedBoard

Julián U. da Silva Gillig. Rev. 2014.03.10



1. Getting the new miniBLoq version

Although it has not been officially released yet, the new miniBLoq.v0.82 version can be downloaded from its GitHub repository. Just go to <https://github.com/miniBLoq/v0.82> and press the **"Download ZIP"** button there:



Note: As this is still the development version (with sources), the internal paths are a bit more complex than in the standard distribution. So every relative path in this document will be inside the **source\Bin\Minibloq** folder.

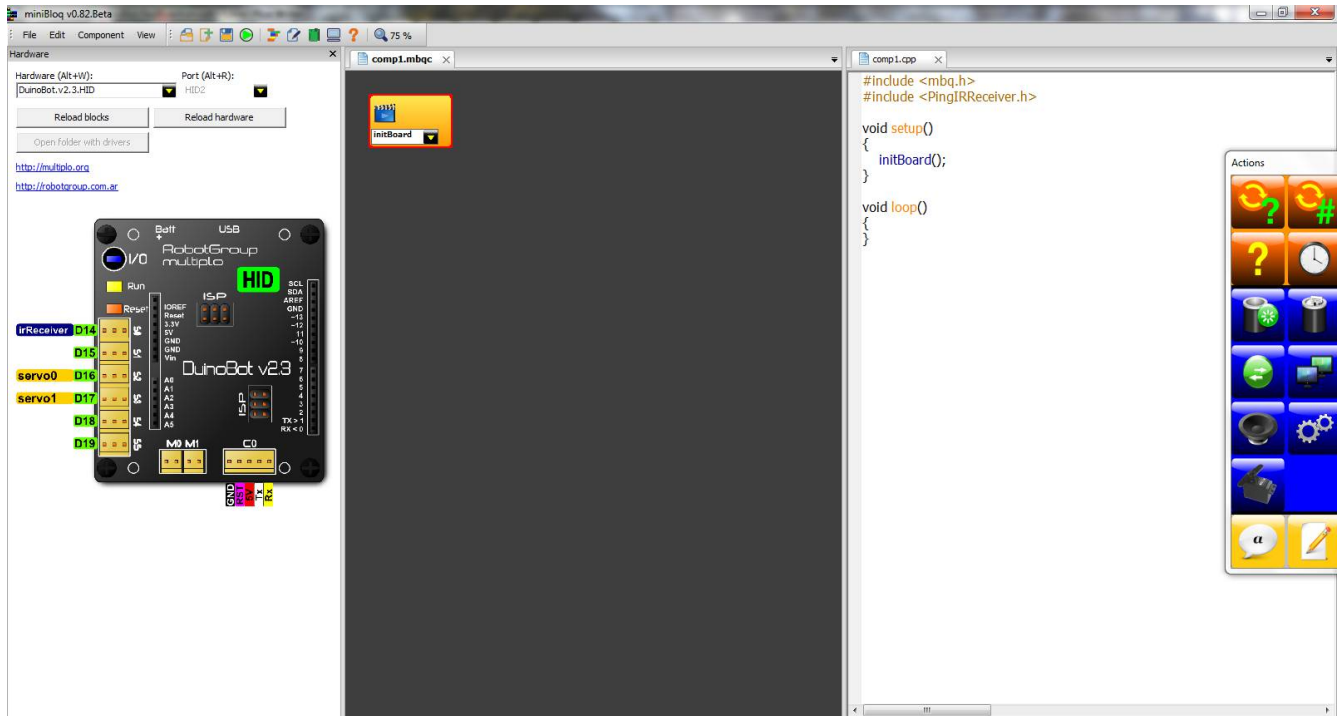
Once you have downloaded it, decompress the zip file and run miniBLoq.exe, from the following relative path in your miniBLoq's installation folder:

mbq\v0.82

For example, if you decompress miniBLoq in **C:\miniBLoq**, you will find the executable in

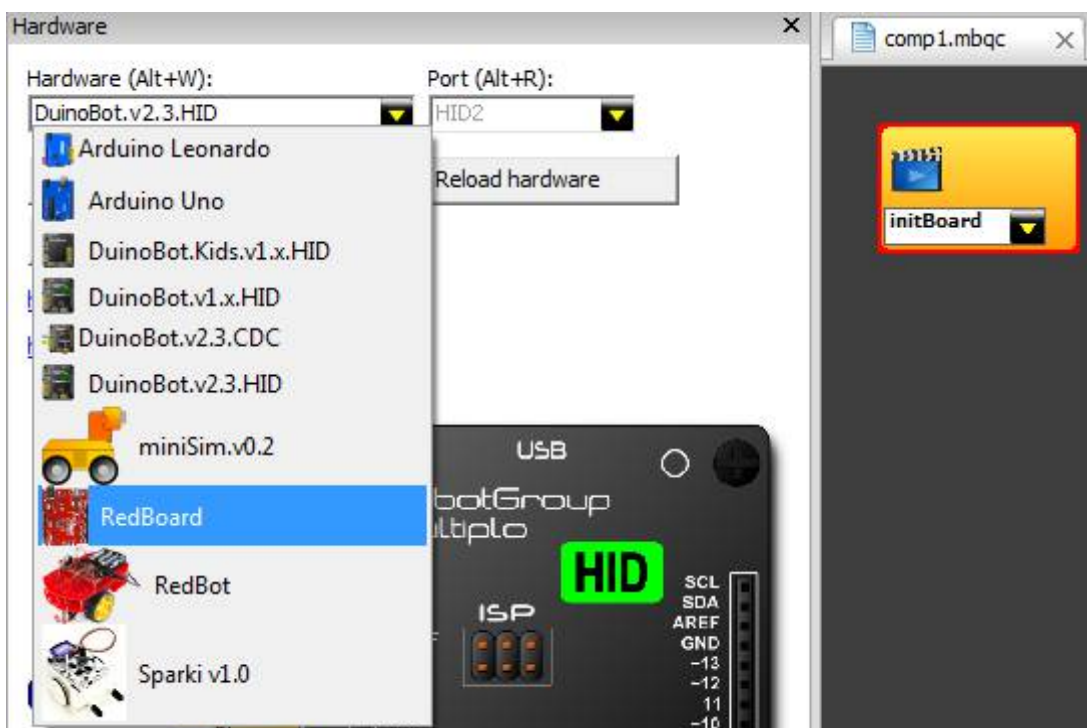
C:\miniBLoq\mbq\v0.82\miniBLoq.exe

The first time you run it, you should see something like this:



2. Getting started with SparkFun's RedBoard

One of the new features on miniBlox.v0.82 is that now, SparkFun's [RedBoard](#) is included in the boards list. This Arduino-compatible board is also the brain of the [SparkFun Inventor's Kit](#) (SIK), so new miniBlox's SIK examples run on the RedBoard. To start working with this board, just select it from the miniBlox's boards list:



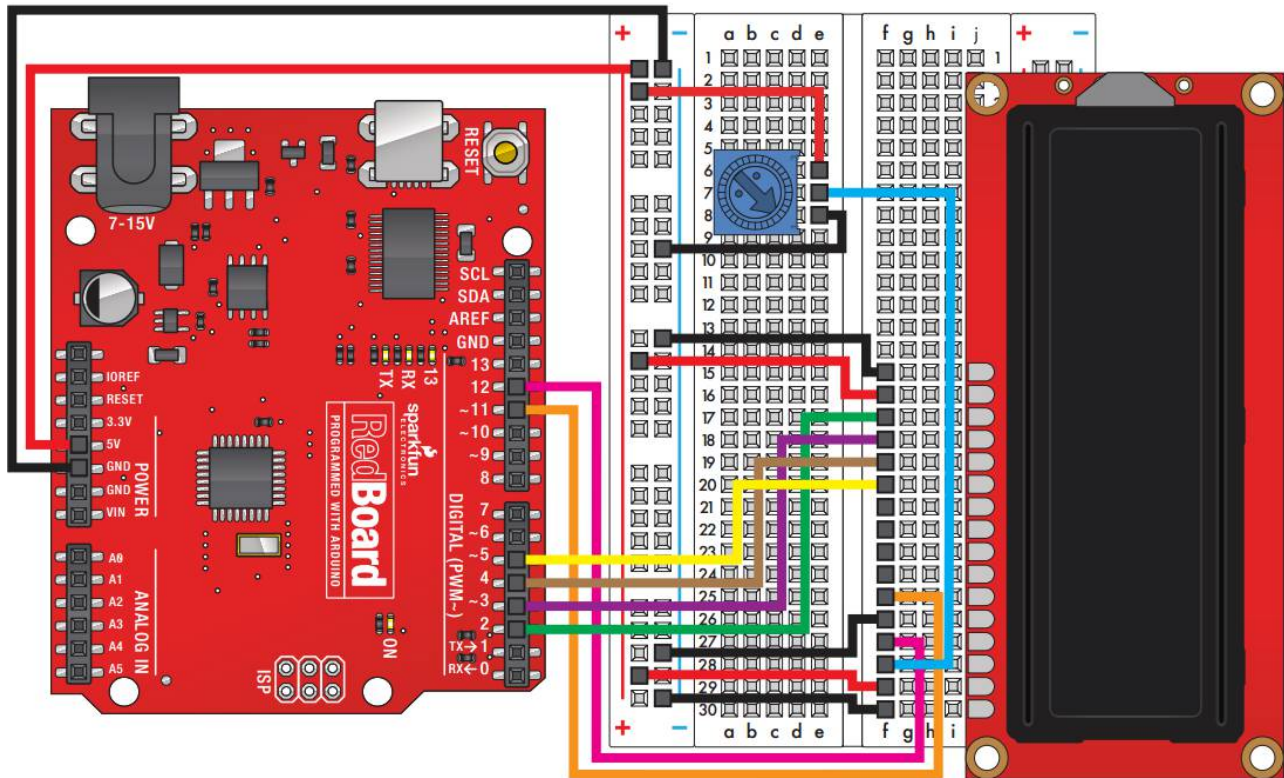
As the RedBoard is compatible with Arduino Uno, all the included examples should work right out of the box. So, to start playing with the board, you just need to go to the **File->Examples** menu and open any example in the following folders:

Arduino
ArduinoTextCoded
SparkfunInventorsKit

Also, most of the examples under the **DuinoBot** folder should work.

3. New blocks for the SparkFun Inventor's Kit display

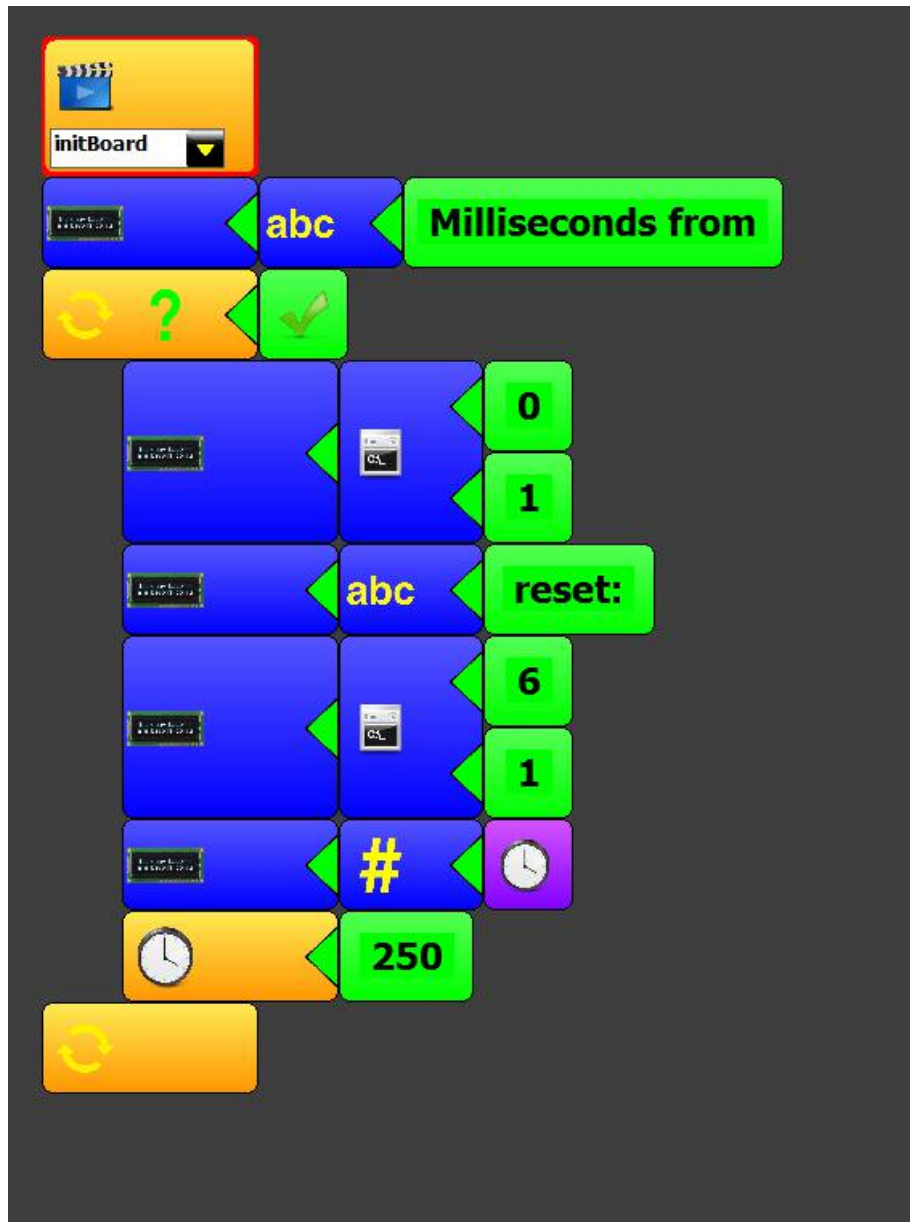
One of the nice devices deployed with the SIK is the [16x2 White on Black LCD](#). miniBloq now includes specific blocks to work with this LCD seamlessly. First, you will need to wire it as described in the [SIK Guide](#) (page 77):



Once everything is wired up, you can open the following miniBloq's example (in the **File->Examples** menu):

SparkfunInventorsKit\5.textLCDMillis

where you will see the following blocks:



and the following code:

```
#include <mbq.h>
#include <PingIRReceiver.h>

void setup()
{
    initBoard();
    textLCD.print("Milliseconds from");
    while(true)
    {
        textLCD.setCursor((int)0, (int)1);
        textLCD.print("reset:");
        textLCD.setCursor((int)6, (int)1);
```

```
        textLCD.print(timestamp());  
        delay(250);  
    }  
}  
  
void loop()  
{  
}
```

As you can see, the new LCD block features two method blocks: one to print (both numbers and text) and another to set the cursor position:

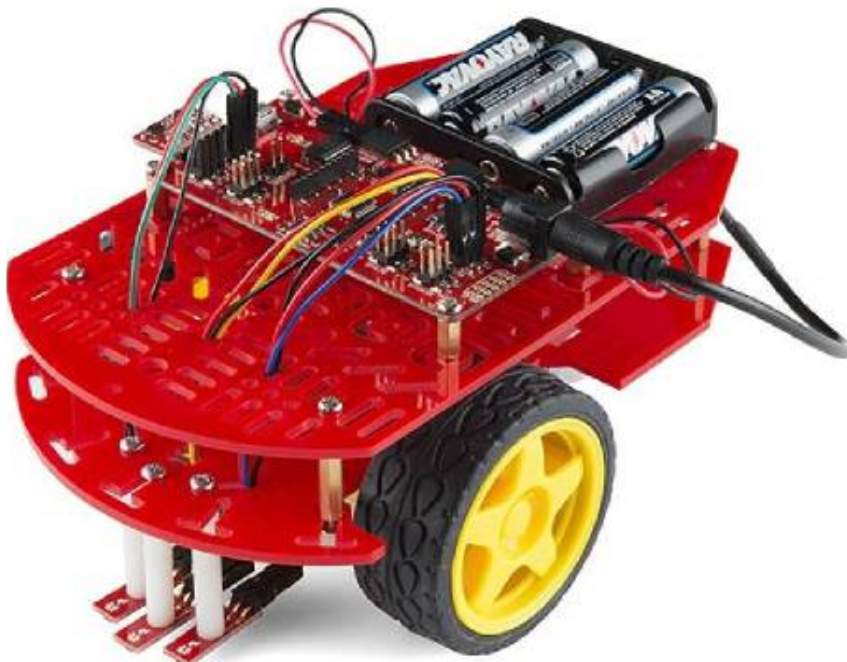
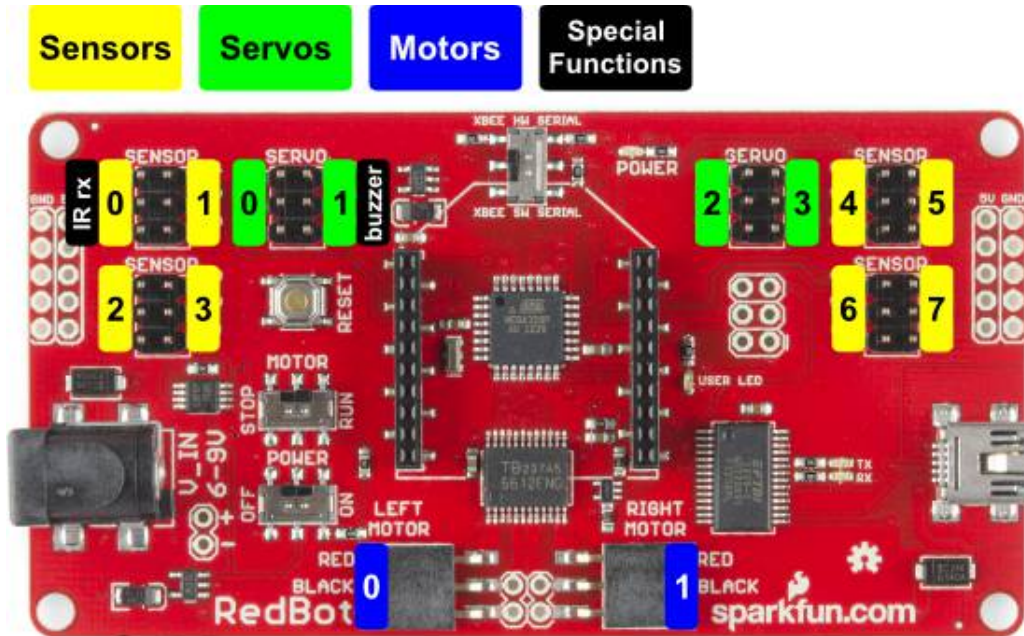


With just these two blocks, you can do nearly anything with your SIK's LCD display.

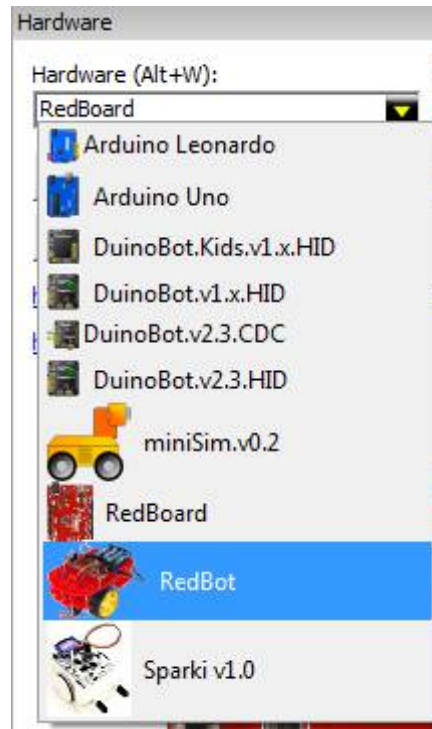
4. Getting started with SparkFun's RedBot

4.1. RedBot + miniBlox

The RedBot is a small, handy and flexible educational robot. It features an Arduino compatible board but with some nice extra features, such as XBee expansion connector, built-in dual H-bridges for DC motor control, on/off switch, R/C servo connectors, and a small form factor:



This new miniBlox version directly works with the RedBot, and also includes specific blocks and examples to start using it easily. Just select the RedBot in miniBlox's hardware list and the environment will automatically load the blocks which work with the robot:



The [RedBot Mainboard](#) has two built in [H-bridges](#) to control the left and right [motors](#). So, when you select the RedBot in the Hardware Manager, it loads the motor block, which will let you control the RedBot's motors without adding any extra initialization code:



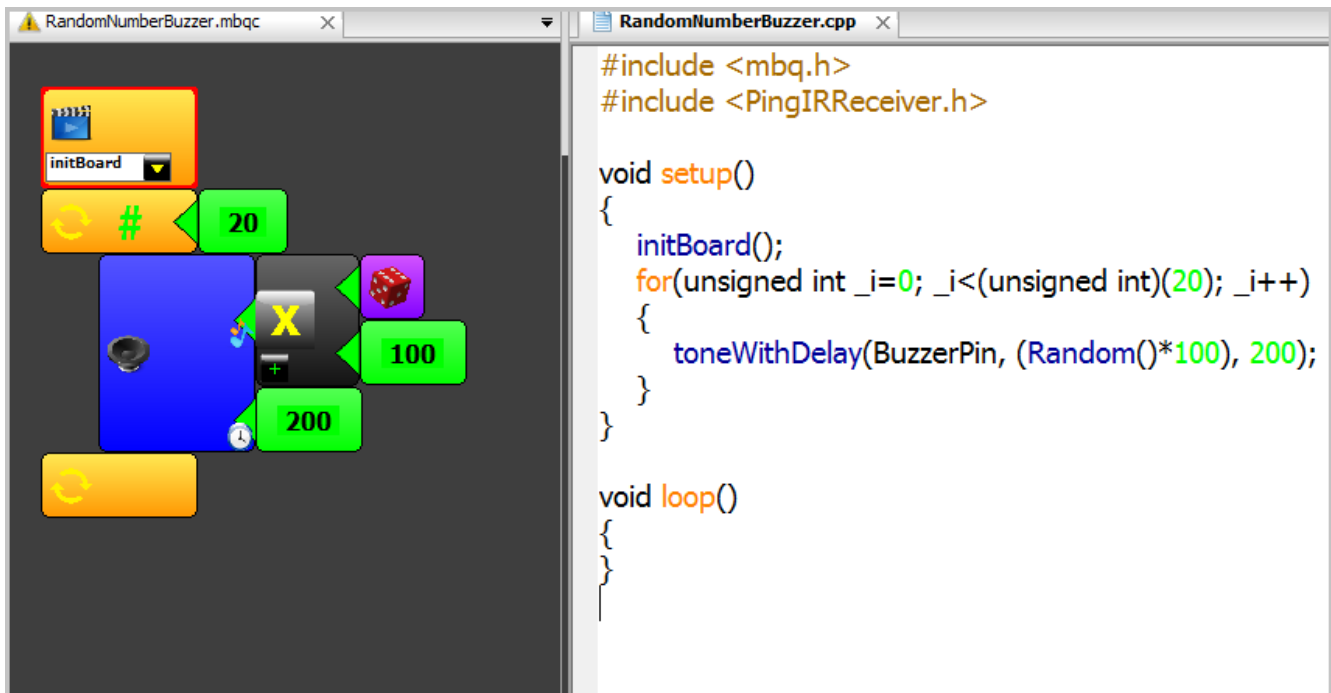
This block receives just one parameter: the power that will be sent to the motor. This parameter is often called just "speed". Although power is more appropriate, since it represents how much time the motor is on in relation to the time that the motor is off. This is because the motors in the RedBot are controlled using a technique called [Pulse width modulation, or PWM](#). In miniBlox, the power value goes from **-100** to **+100**. This means that when you set the motor's power as **100**, it will rotate at its maximum speed in one direction, while a value of **-100** means that the motor will rotate at its maximum speed in the opposite direction. Of course, a value of **0** stops the motor.

A final comment about the motor block: it's important to note that, to keep compatibility with the

existing miniBq's robotics examples, the block motor names for the motors are **motor0** for the left motor, and **motor1** for the right one. This way, you can make use of nearly all of the examples included in the **_examples/DuinoBot** folder (remember that you can always open the **_examples** folder using the **File->Examples** menu) without modifying them. There are more than 30 examples in that folder, so what are you waiting to start playing with the robot?

4.2. Let's start with some music

To make music with the RedBot, you will need to add the [buzzer](#), and connect it as is explained [in this page](#) in the [RedBot's Assembly guide](#) (also, [here](#) are 2 pictures showing the RedBot from the top, with a few of its accessories already connected). Now, you can open any of the buzzer examples in the **_examples\DuinoBot** folder and play with them. For instance, the following code (from the **230.RandomNumberBuzzer** example) will play 20 random notes in the RedBot's buzzer:



```
#include <mbq.h>
#include <PingIRReceiver.h>

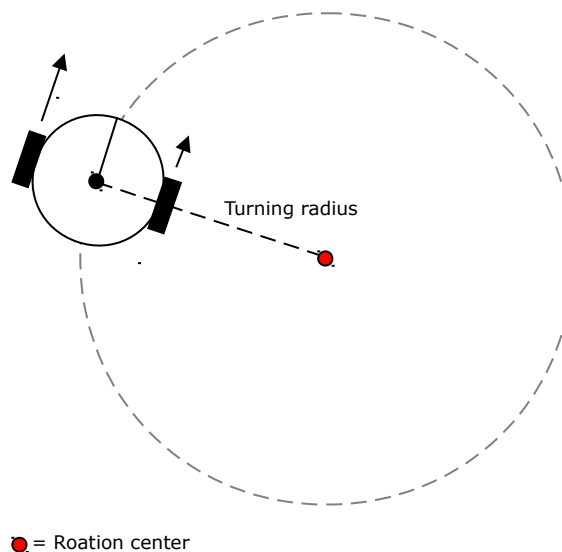
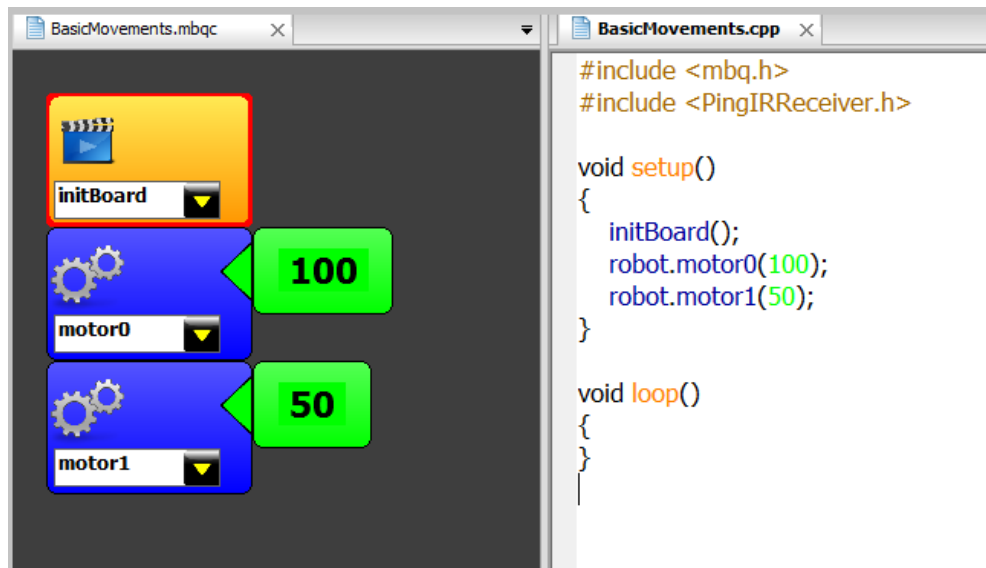
void setup()
{
  initBoard();
  for(unsigned int _i=0; _i<(unsigned int)(20); _i++)
  {
    toneWithDelay(BuzzerPin, (Random()*100), 200);
  }
}

void loop()
{
}
```

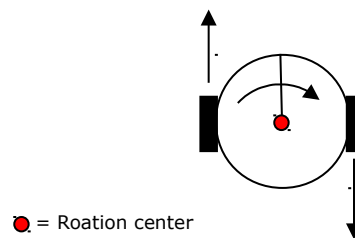
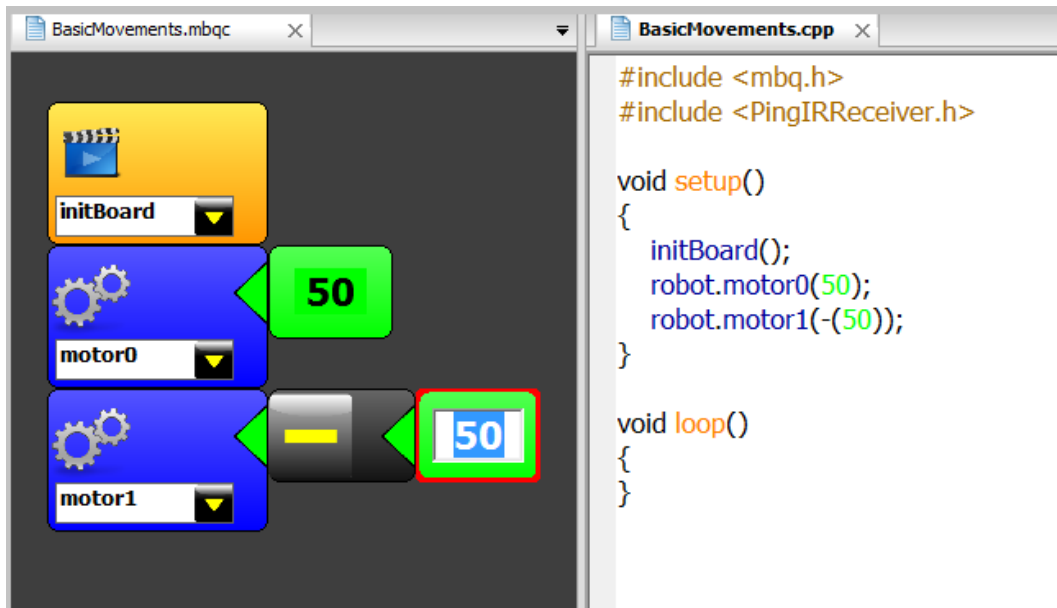
5. Basic movements with the RedBot

Driving the RedBot is easy: just set the speed of each wheel in your program, and the robot will start moving. Here are some examples (you can go to **File->Examples** and open the **RedBot\10.BasicMovements** example there):

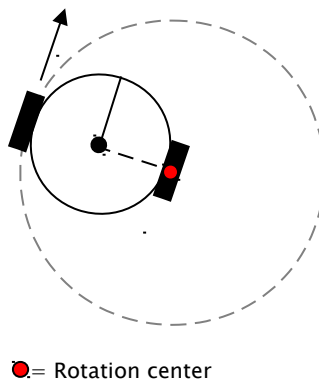
5.1. Make the robot turn with an external rotation center:



5.2. Make the robot rotate around its center:

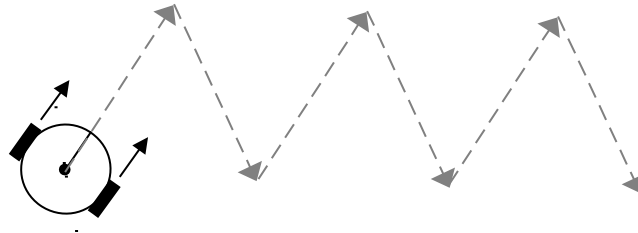


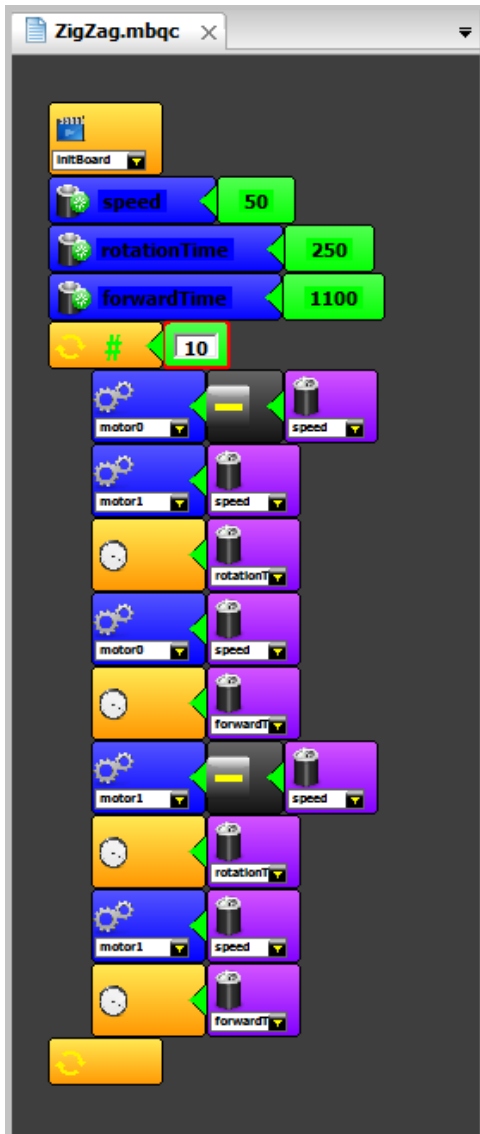
Also, if you set a speed of zero to one of the wheels, that wheel will become the rotation center:



Please note that once you set the speed of one of the robot's motors, it will remain moving with that speed value until another motor block changes it. The following example shows this.

5.3. ZigZag path:





```

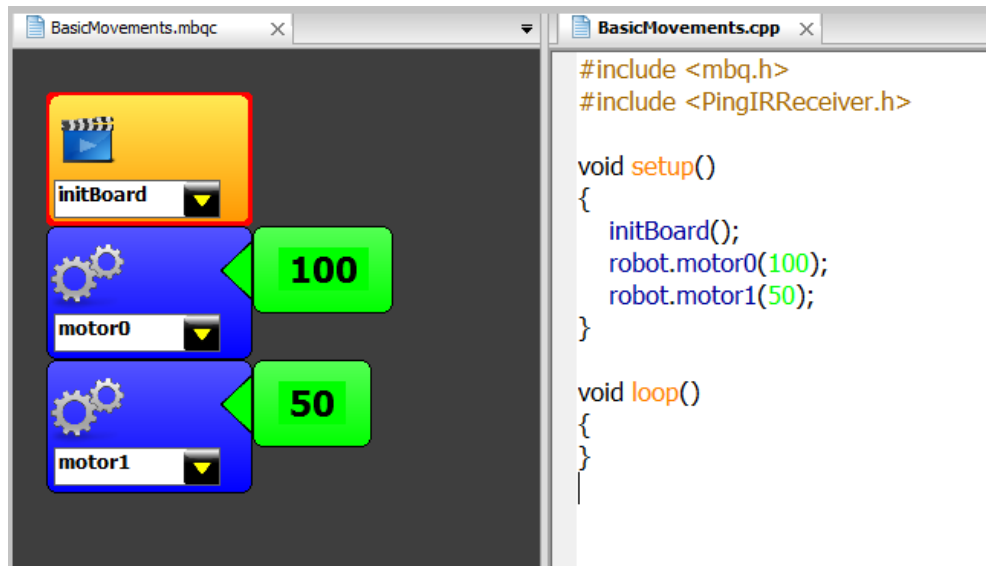
#include <mbq.h>
#include <PingIRReceiver.h>

void setup()
{
    initBoard();
    float speed = 50;
    float rotationTime = 250;
    float forwardTime = 1100;
    for(unsigned int _i=0; _i<(unsigned int)(10); _i++)
    {
        robot.motor0(-(speed));
        robot.motor1(speed);
        delay(rotationTime);
        robot.motor0(speed);
        delay(forwardTime);
        robot.motor1(-(speed));
        delay(rotationTime);
        robot.motor1(speed);
        delay(forwardTime);
    }
}

void loop()
{
}

```


As you can see in the previous example, after setting the motor's speeds, a delay block keeps those speeds until a change is needed again. Also, as you may already have noted, each motor's speed is set with an individual block, so in theory, one motor starts moving before the other:



But in fact, the time between the first motor block and the second one is so small (specially when compared with the time that it takes to a motor to start moving), that for practical purposes, it's nearly zero.

5.4. Accessing the H-bridge

Finally, if you want to access the H-bridge bits directly, without the motor block, you can try the following example:

examples\RedBot\40.LowLevelMotorControl

It basically drives each direction pin in the H-bridge with a digital output block, and each enable (or PWM input) pin with an analog output block:



To better understand it, our advice is to take a look to the [RedBot Mainboard schematic](#). And perhaps you may also find useful the [H-bridge's datasheet](#).

6. Using the RedBot with standard IR remote controllers

6.1. The IR Receiver block

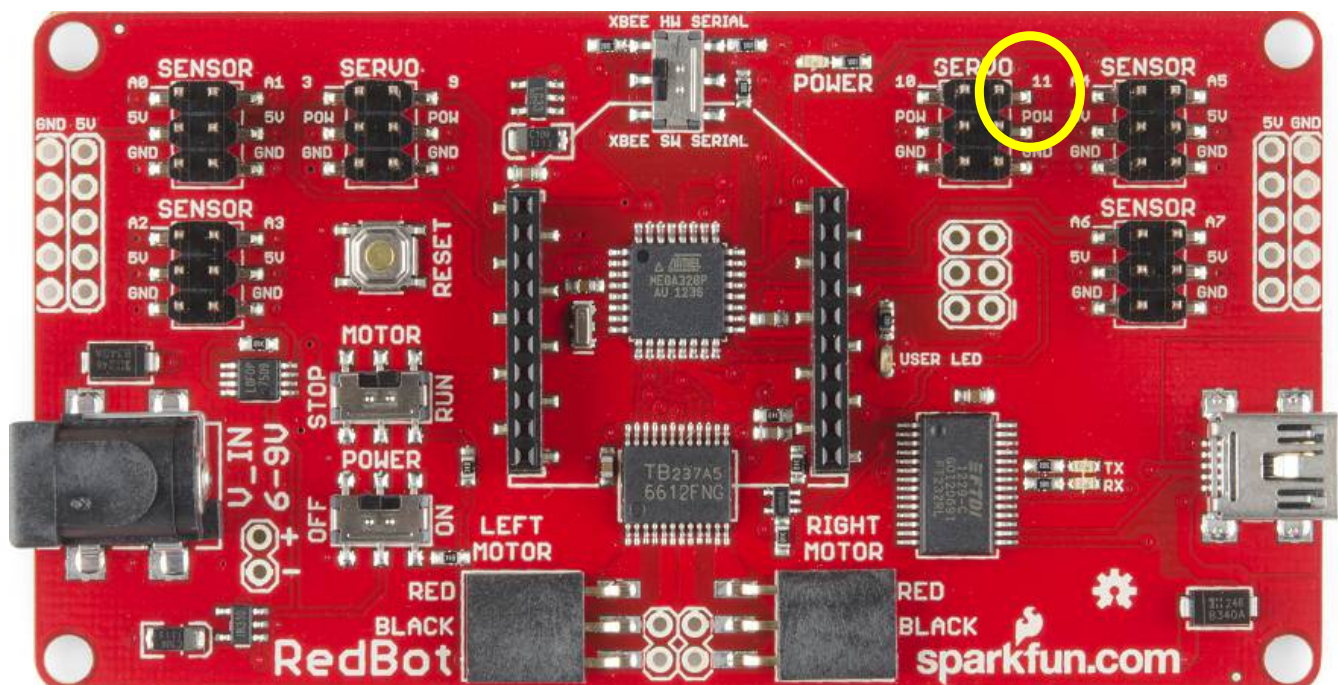
Another standard block that can be useful (and funny) when used with the RedBot is the IR receiver block. It works with standard [RC5 remote controllers](#), which are pretty common and cheap. The library behind this block has been integrated with RedBot, so you don't need to do nothing but opening the included examples and start modifying them. The IR Receiver (or *sensor*) block is in the numerical picker:



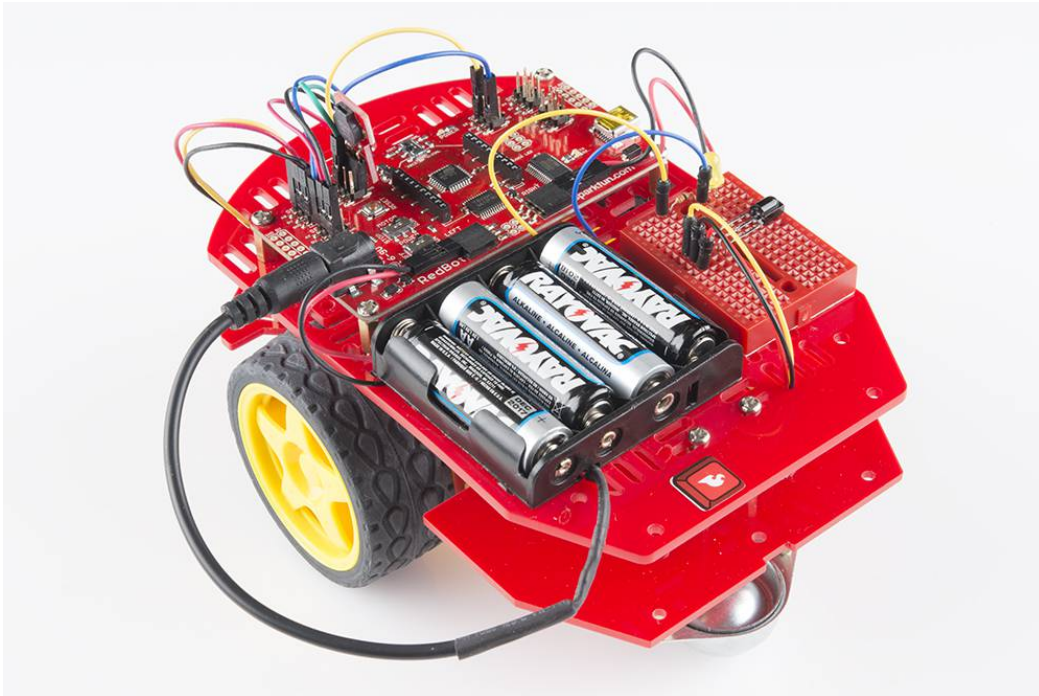
The block does not receive parameters, and it returns a number from 0 to 10, in response to the numerical keys being pressed in the remote controller. For example, if no key is pressed, it returns 0 each time it's read. If you press the **2** key, it will return 2. If you press **0**, it will return 10. Nothing is returned for other keys (such as volume, channel, power, etc.). This works the same way as the standard IR receiver block for all the other supported boards in miniBloq. You can also learn more about IR communications in [this SparkFun's tutorial](#), or in SparkFun's [IR Control Kit Hookup Guide](#).

6.2. Connecting an IR receiver to the RedBot

To enable the RedBot to receive IR RC5 commands, you need to connect an IR receiver, like [this one](#), also from SparkFun. All the included examples in miniBloq work assuming it's connected with its output pin to the **pin 11** in the RedBot main board (which is also the **data** pin in the **Servo 3** connector in miniBloq's map on the Hardware Manager view of the board). The following picture shows it with a yellow circle:



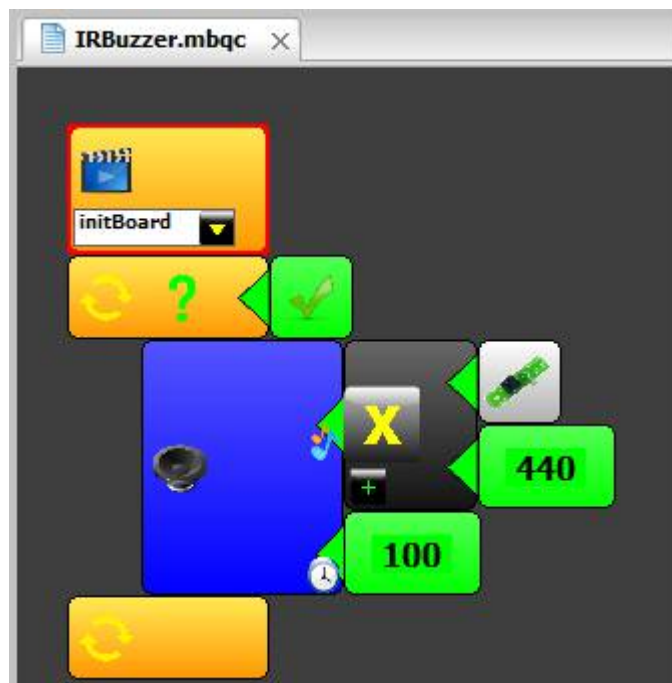
It may be handy to add a small [breadboard](#) to the RedBot, like in the following picture, moving the battery holder to a side:



By adding that breadboard, you will be able not just to connect the IR receiver, but also to make a bunch of experiments with other standard electronic components.

6.3. Playing music with a remote control keyboard

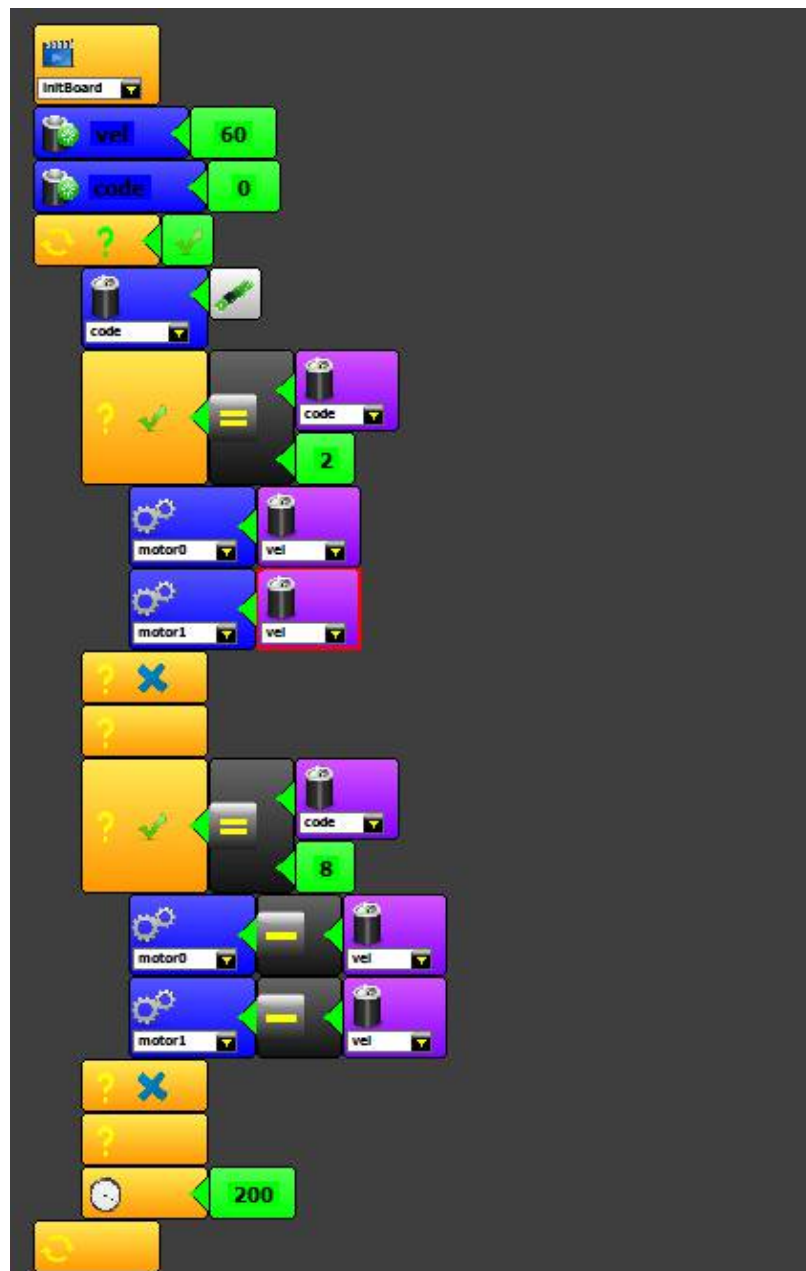
If you still have the buzzer connected to the robot's board (please see section 4.2 on this tutorial), you can try the program in the **_examples\DuinoBot\130.IRBuzzer** folder to use the IR remote controller's numerical keyboard to play notes:



This small program reads an IR code (remember: it returns a number from 0 to 10, using the numerical keys only), multiplies it by 440 (the A note!) and sends the result to the frequency parameter (in buzzer block). This is done every 100 milliseconds (using the duration parameter in buzzer block).

6.4. Controlling robot's movements

As you can see, using the IR receiver block is easy. So, why not control the robot's movements with it? It can be done using a decision (or **if**) block. This way, the program can ask for the number returned by the IR block, and then execute different actions. A complete example can be found in the **_examples\RedBot\60.IRRemoteRC5Robot** folder, but we can start doing something really simple to just make the robot move forward and backward when pressing the **2** and the **8** keys in the controller:



7. The new accelerometer block

There is a new [accelerometer sensor](#) for the RedBot, and thus, there is also a new accelerometer block in miniBloq:



Since it's a block which returns a number, you will find it in the numerical picker:



Accelerometer block

The returned number (which can be huge), represents the acceleration in one of the 3 robot's axes: X, Y and Z (You can learn more about how a 3D accelerometer works, with the [Accelerometer Basics tutorial](#)). And of course, to work properly in the following examples, the

accelerometer has to be connected [as shown here](#). So, let's do something with the accelerometer.

7.1. Making sounds with the accelerometers

If you open the example in the `_examples\RedBot\90.ZAxisAndBuzzer` folder, you will see the following blocks:



This simple program will generate notes with the RedBot's buzzer, just by rotating the robot around its Z axis (if you don't know which is the Z axis, try rotating the robot with this program loaded, until you figure it out).

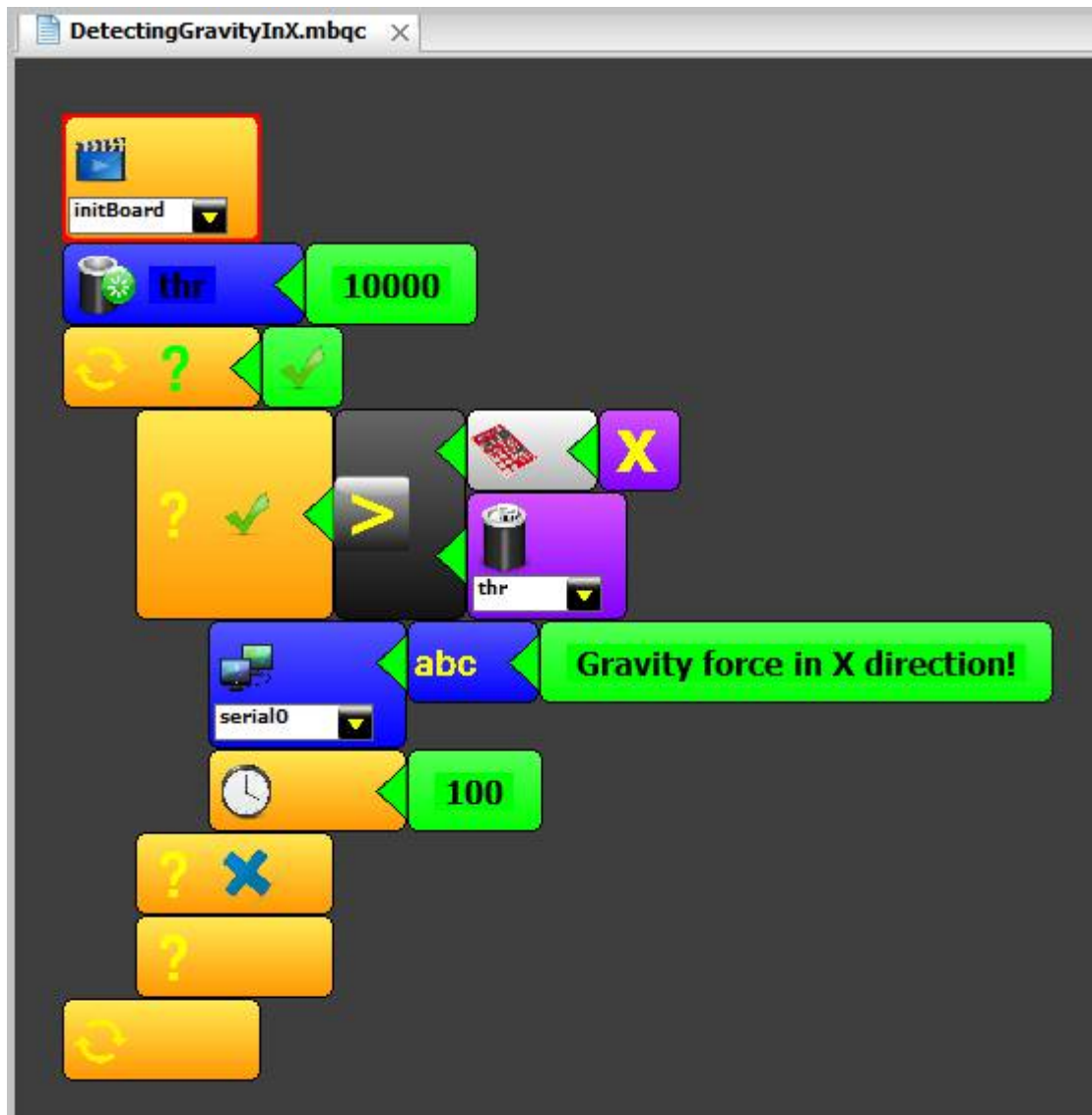
The accelerometer block has 3 methods, to read the acceleration in the X, Y or in the Z axis. In this example, we read the acceleration belonging to the Z axis only. As this value can be negative, which makes no sense for generating frequencies for the buzzer (or *notes*), we use the **ABS** (absolute value) block to convert negative values into positive numbers. After that, the (now positive) value from the Z axis is multiplied by a factor of less than 1, just to reduce the number, in order to generate an audible frequency. If you want to see what kind of huge numbers the accelerometer block is returning, try replacing the buzzer block in this example by a terminal block (and a delay block after it).

7.2. Detecting gravity

Here is another example using the accelerometer:

_examples\RedBot\87.DetectingGravityInX

and here is the code:



To run it, you need to keep the RedBot connected with the the USB cable to the PC, and use miniBlox's terminal (you can go to the menu **View->Terminal**). Now, if you rotate the robot around its axes, it will detect when the gravity force is applied in the X axis direction, sending a string to the terminal window.

7.3 The accelerometer as a bumper

Finally, if you are an advanced user and want to experiment with miniBlox's text programming capabilities (and the RedBot's accelerometer, of course), try the following example, which can be found in the **_examples\RedBot\100.AccelTextCodedReadBumper** folder:

```
#include <RedBot.h>
#include <mbq.h>

int speed = 50;

void go()
{
    serial0.begin(115200);
    accel.enableBump();

    while(true)
    {
        robot.motor0(speed);
        robot.motor1(speed);
        if (accel.checkBump())
        {
            serial0.println("Ouch!");
            robot.motor0(0);
            robot.motor1(0);
            delay(200);
            robot.motor0(-speed);
            robot.motor1(-speed);
            delay(1000);
            robot.motor0(0);
            robot.motor1(0);
            delay(200);
            robot.motor0(speed);
            robot.motor1(-speed);
            delay(1500);
            robot.motor0(0);
            robot.motor1(0);
            delay(200);
        }
    }
}
```

It should make the RedBot to go forward until it hits an obstacle.

8. RedBot + XBee

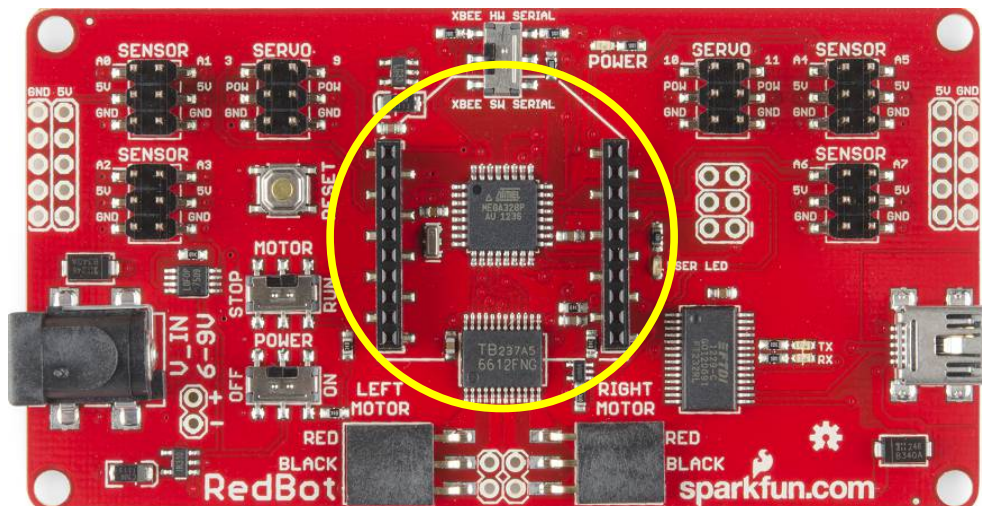
The Redbot can be used both as an autonomous robot (where its own small microcontroller makes the decisions regarding the robot's behavior), or as a remote controlled robot, using wireless communications. In this last configuration, an external computer takes control of the whole robot. This has some advantages, like the extra computing power, or the possibility of using the computer's camera and other useful peripherals.

8.1. Using XBee modules with the RedBot

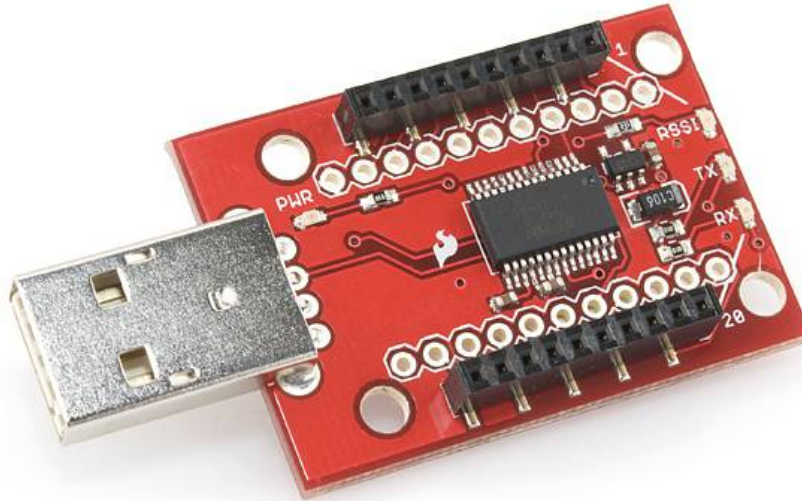
Fortunately, the RedBot includes a socket to plug an XBee wireless communication module into it. There are a lot of XBee modules out there, and you can choose your own, of course. But for this tutorial we have used [XBee 1mW with Wire Antenna, Series 1](#):



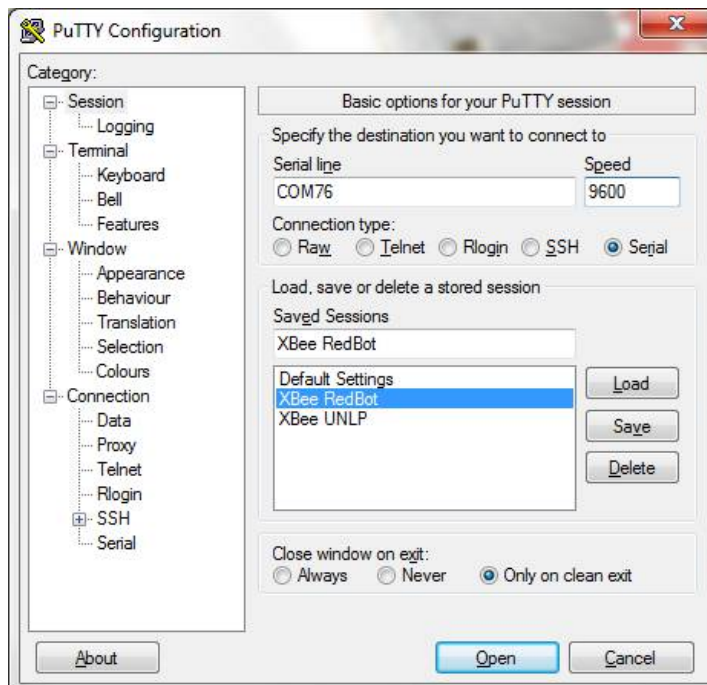
It should be connected here:



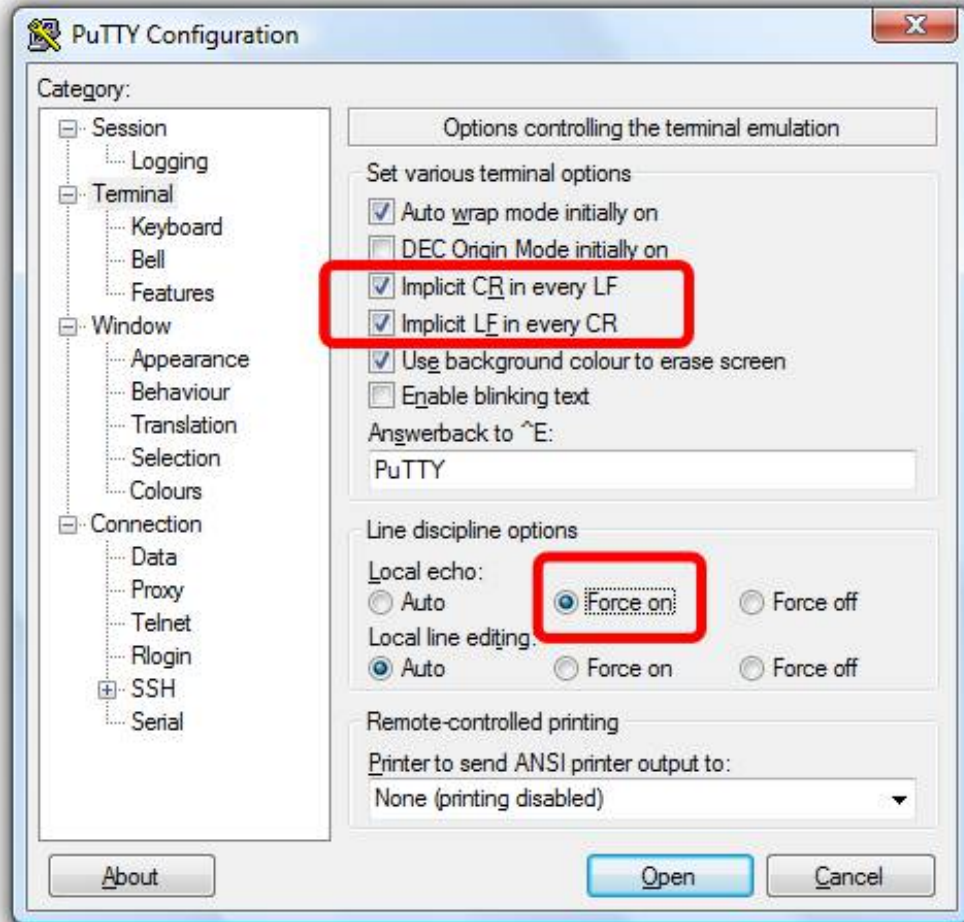
Obviously, we will need two of these modules, since the other needs be connected to the computer, which will become the **external brain** of the robot. Perhaps the simplest way of connecting one of those modules to the USB port of a computer, is using an adaptor like the [XBee Explorer Dongle](#):



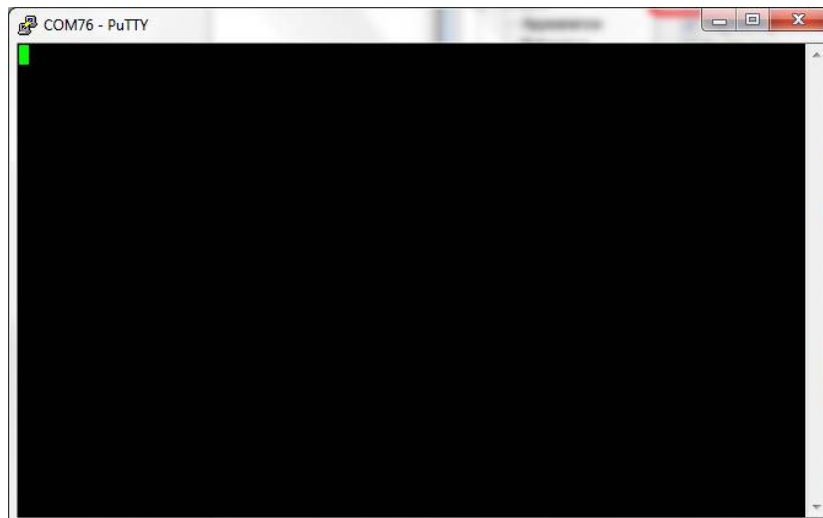
But before starting to enjoy the wireless link between the robot and the computer, it will be necessary to configure some parameters into the XBee modules. To do that, we need a good terminal program. So, the first step is getting one. For this tutorial, we selected [PuTTY](#). We need to configure both the robot and the computer side XBee modules. So, before opening the terminal program it will be necessary to insert the robot's module into the XBee Explorer Dongle. Then plug the dongle board to an USB port on the computer. This way, we can configure that module with some serial commands. Now we can open PuTTY (or the terminal program of your choice). Select in its main window a serial connection, and introduce "9600" in **Speed**, plus the serial port id into the **Serial line**:



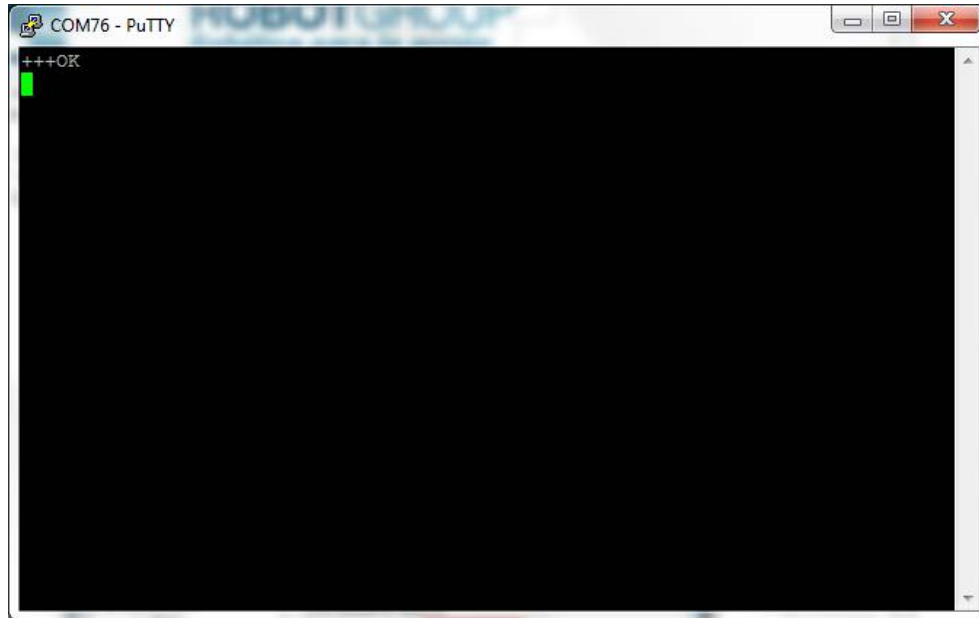
Now select "Terminal" at the left of the PuTTY's main window, and check the following options:



Once you press the **Open** button, you should see something like this:



That's the terminal itself, where you can now input the AT commands, needed to configure the robot's XBee module. But prior to start configuring, please press the "+" key 3 times. After that, you will see an **OK** returned by the XBee module:



Now, try to input the following commands, pressing **ENTER** after each one. You will see an **OK** if the command was successfully executed (please note that the computer's XBee will need a slightly different configuration):

```
ATID1000
ATMY2
ATDH0
ATDL1
ATBD5
ATWR
ATCN
```

After doing this, this module will be configured as the client (robot), and will feature a baudrate of 38400. So if you need to change something in the future, it will be necessary to input 38400 instead of 9600 in the terminal's speed configuration.

Now, you need to close the terminal, unplug the dongle from the USB port, and carefully remove the XBee module from the dongle, to connect it to the robot's XBee socket. After that, connect the remaining module to the dongle and plug it into the USB port, so we can configure it. Press the ++ again, and after seeing the **OK**, type these commands:

```
ATID1000
ATMY1
ATDH0
ATDL2
```

```
ATBD5
ATWR
ATCN
```

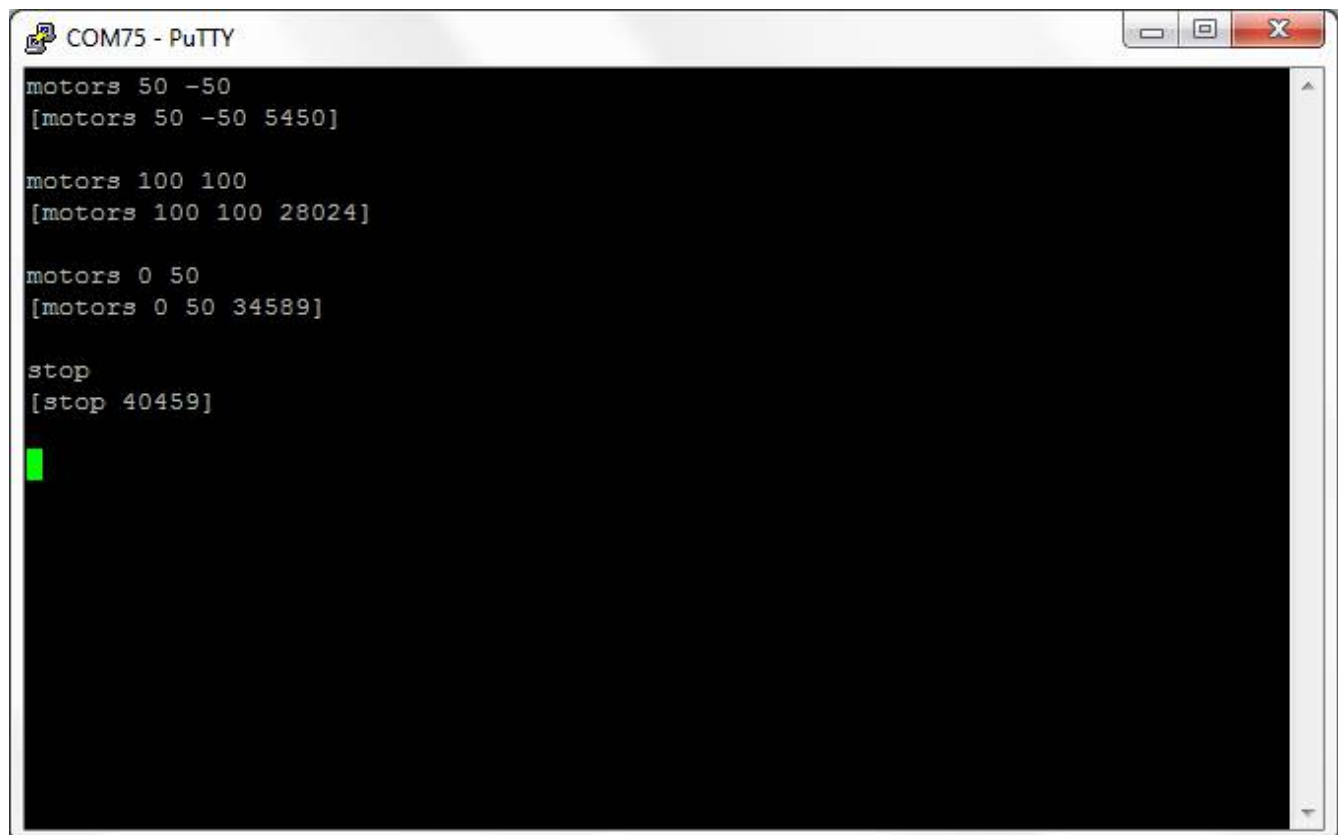
This module must remain in the dongle, since is the one that will be used on the computer side. One more comment: we selected 38400 as the baudrate (**ATBD5** command) because that is the maximum speed that we have found that these XBee modules are really stable. You can find more information about XBee modules [here](#).

8.2. Small command interpreter for wireless robot control

So, what else do we need to control the robot remotely? A command interpreter. Even with a small one, it will be enough to make the robot obey to your computer. The command interpreter is the program that we will upload to the RedBot's flash memory, in order to make the robot work by reading (real time) commands from a serial port or from its XBee module. Since writing a command interpreter can be a little difficult at first, there is one already provided with miniBlox. You can find it in the following examples subfolder:

examples\RedBot\110.smallProtocol

It's a pure text program, so you will not see blocks there. Once you upload it to your robot, you can test if it's working by opening the RedBot's serial port in terminal (PuTTY, for example) at a 38400 baudrate. There, you can enter commands. The interpreter will understand just two commands (if you take a look to the source code, you can figure out how to add more): **motors** and **stop**.

A screenshot of a PuTTY terminal window titled "COM75 - PuTTY". The window has a black background with white text. The text shows a series of commands being entered and responses being received from the robot. The commands are "motors 50 -50", "motors 100 100", "motors 0 50", and "stop". The responses are "[motors 50 -50 5450]", "[motors 100 100 28024]", "[motors 0 50 34589]", and "[stop 40459]". A green cursor is visible on the line following the "stop" command.

```
COM75 - PuTTY
motors 50 -50
[motors 50 -50 5450]

motors 100 100
[motors 100 100 28024]

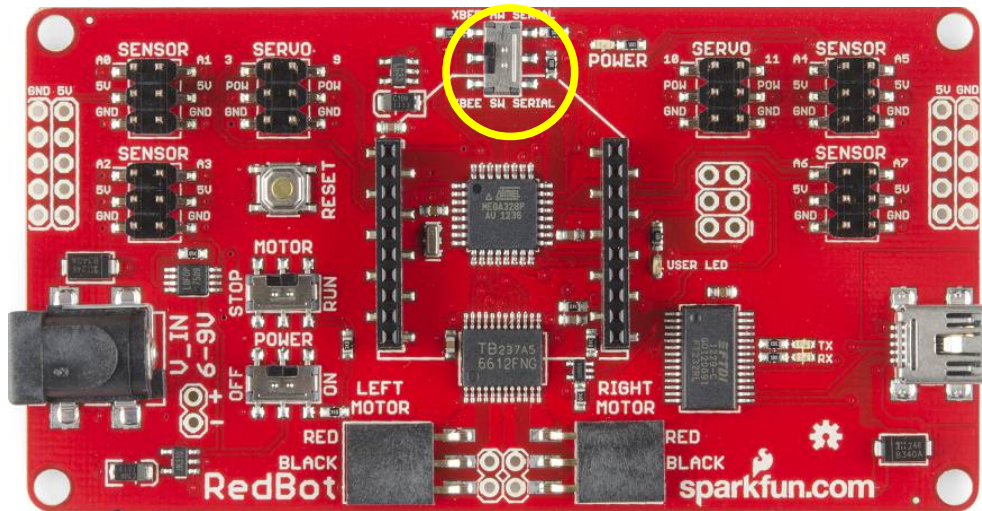
motors 0 50
[motors 0 50 34589]

stop
[stop 40459]
█
```

As you can see in the previous screen capture, the **motors** command takes two arguments: the speed for each motor, which can be any number between -100 and 100. While **stop** does not need any argument.

8.3. Using the interpreter with the XBee link

Now that you have tested the interpreter with the USB cable and the RedBot's serial port, try the following: If you have already connected the robot's XBee into the robot's socket, turn the robot off, unplug the USB cable, and move the switch located near the XBee module to the position labeled "**XBEE HW SERIAL**":



Now turn on the robot and, with the other XBee inserted into the dongle (and with the dongle connected to the USB port), try to open that port with the terminal program, and try input **motors** and **stop** commands there. The robot must obey.

9. Controlling the RedBot with OpenCV and Python

And just to see what can be done with a powerful XBee equipped RedBot, let's control it with the computer's camera. To do that, we will make use of the [Python](#) interpreter also included in the miniBlox's package. You don't need to know Python to try this example (although it may help, of course). First, we need the interpreter uploaded to the RedBot's flash and the XBee wireless link working, as explained in the previous section. Now, go to the following miniBlox's subdirectory:

lang\PPythonWin\v2.7.5.1

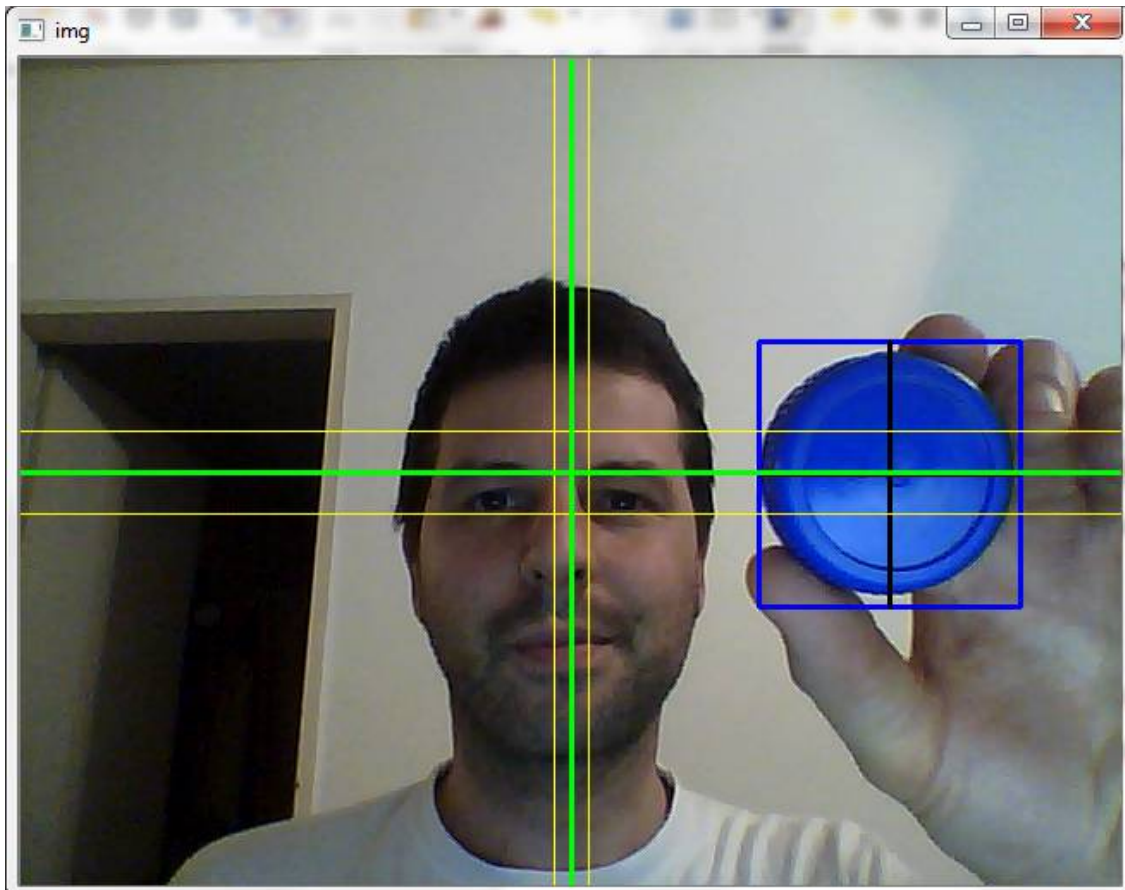
There, you will find the file **IDLE-Portable.exe**. Please run it. Then go to the **File->Open** menu, and find the following python program inside miniBlox's examples:

_examples\RedBot\110.smallProtocol\videoJoystick.py

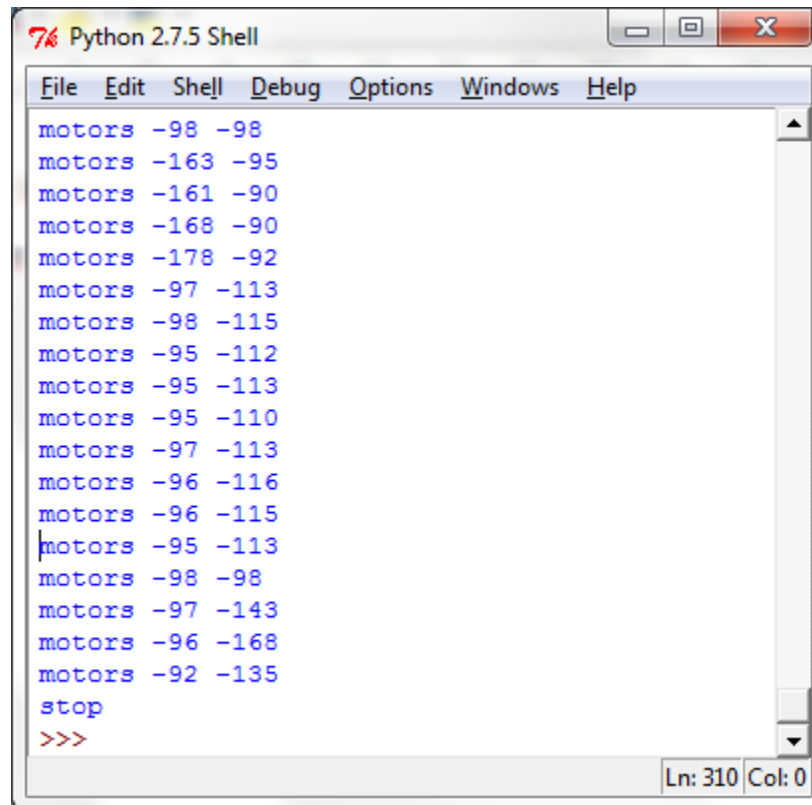
The only change that you will need to do is the serial port number in line 47 (which is the text 'COM76' here, which you have to replace with your own serial port number, for example 'COM9'):

```
sp = serial.Serial('COM76', 38400) ##SparkFun's XBee module (RedBot)
```

Now you can run the module, both by going to the menu **Run->Run Module** or by pressing the **F5** key. You will see the following screen (to stop the program just press the **ESC** key):



There, the Python program is tracking any blue object (you will need to experiment a bit to find the blue that the program wants) and uses it as a virtual joystick to control the robot. The program then transforms the coordinates in the screen to valid motor speeds to be sent to the RedBot. You may note that there is another (text) window there, showing in real time the sent commands:

A screenshot of a Python 2.7.5 Shell window. The window has a title bar that says "Python 2.7.5 Shell" and a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area contains a list of motor speed commands in blue text: "motors -98 -98", "motors -163 -95", "motors -161 -90", "motors -168 -90", "motors -178 -92", "motors -97 -113", "motors -98 -115", "motors -95 -112", "motors -95 -113", "motors -95 -110", "motors -97 -113", "motors -96 -116", "motors -96 -115", "motors -95 -113", "motors -98 -98", "motors -97 -143", "motors -96 -168", "motors -92 -135", "stop", and a red prompt ">>>". The status bar at the bottom right shows "Ln: 310 Col: 0".

```
Python 2.7.5 Shell
File Edit Shell Debug Options Windows Help
motors -98 -98
motors -163 -95
motors -161 -90
motors -168 -90
motors -178 -92
motors -97 -113
motors -98 -115
motors -95 -112
motors -95 -113
motors -95 -110
motors -97 -113
motors -96 -116
motors -96 -115
motors -95 -113
motors -98 -98
motors -97 -143
motors -96 -168
motors -92 -135
stop
>>>
Ln: 310 Col: 0
```

By moving the blue control object a bit, you will soon figure out how the “video virtual joystick” works (tip: the object at the center of the screen means zero speed for each motor).

To do the video capture and processing, we have included the [OpenCV](#) library in miniBloq, so it can be used from any Python program.

There are a lot of possibilities now that you can use your own computer as the RedBot's external brain. So why not try to add other capabilities to your robot, such as *speech recognition*, or even *face tracking*?