4tronix

Bit:Bot – The Integrated Robot for BBC Micro:Bit

A great way to engage young and old kids alike with the BBC micro:bit and all the languages available. Both block-based and text-based languages can support the Bit:Bot

You can also use the Radio or Bluetooth functionality of the Micro:Bit to send and receive commands and date.

NEW There is now a Microsoft PXT package for Bit:Bot (thanks to Sten Roger Sandvik, @stenrs on Twitter).

Warning: The line follower sensors share the same pins as the buttons. Depending what language you are using, when the Micro:Bit is started or reset it will check the 2 buttons and start pairing if they are both pressed. With the Bit:Bot, this translates to both line follower sensors getting reflections. You can stop it happening by lifting it off the surface before switching on.

Features

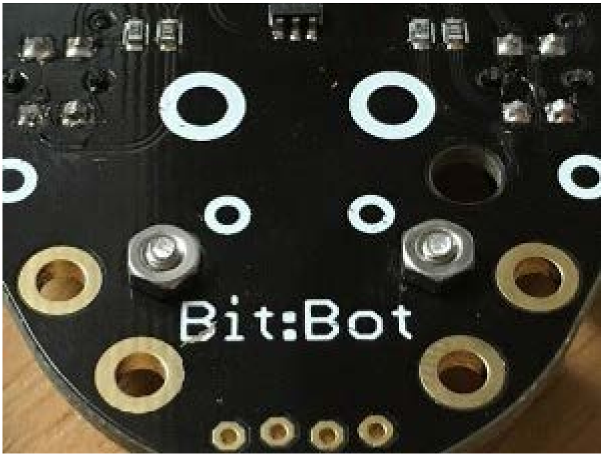
The Bit:Bot gives you all these features:

- 2 micro-metal gear motors. Both fully controllable in software, for both speed (0 to 100%) and direction (forward or reverse)
- Wheels with rubber tyres for maximum grip
- Really smooth metal ball front caster
- 12 mini neopixels in 2 sets of 6 along the arms either side. Select any colour for any pixel, produce stunning lighting effects as your Bit:Bot moves around
- 2 digital line following sensors. Code your own line-following robots and race them to see whose code produces the fastest lap time!
- 2 analog light sensors (front left and front right) so your Bit:Bot can be programmed to follow a light source such as a torch, or you could code it to go and hide in the darkest place it can find
- Buzzer, so you can make beeping sounds whenever you want
- Powered from integrated 3xAA battery holder with on/off switch and blue indicator LED
- Easily plug your BBC micro:bit in and out using the edge connector
- Extension port for additional neopixels (such as McRoboFace)
- Expansion connections at the front for additional sensors (in development)

Assembling

1. Use the M2 6mm (panhead) screws and nuts to attach the front caster housing, then push the caster ball into the housing
2. Use the M2.5 6mm panhead and 8mm countersunk screws to fit the battery pack onto the 2 metal pillars: **ENSURE the on/off switch is at the rear of the Bit:Bot**
3. Push the wheels on with the smooth side outwards. The axle should come flush with the outside of the wheel and not protrude (or the inside can catch on the motor housing)
4. Push your BBC micro:bit into the edge connector with the LEDs and switches on the top

Step 1 – Fit the Front Caster



Step 2 – Fit the Battery Holder

At this point you should have 4 screws left. Either 4 x 8mm countersink, or 2 x 6mm panhead and 2 x 8mm countersink.

If you have the 6mm panhead screws, use these to fit the 12mm brass pillars to the Bit:Bot main PCB.

Always use 8mm countersink screws to fit the battery holder to the brass pillars.

Use either 6mm panhead or 8mm countersink to fit the 12mm brass pillars to the main board (above)

Use the 8mm countersink screws to fit the battery holder to the brass pillars.

Step 3 – Fit the Wheels

Push the wheels on, so that the axle is flush with the outside of the wheel

Step 4 – Attach your BBC micro:bit

Know Your Bit:Bot Above

This shows the neopixels (6 on each arm), the 2 light sensors, on/off switch and indicator LED

The buzzer is below the on/off switch and the edge-connector is below the front of the battery holder

Below

Now you can see the 2 line sensors and the port for neopixel extensions and general purpose expansion connector. Connection labelling is on the underside

Programming

I find Microsoft PXT the best block based language for Bit:Bot as it happily works with the extended pin set used and supports neopixels easily also.

NEW There is now a Microsoft PXT package for Bit:Bot (thanks to Sten Roger Sandvik, @stenrs on Twitter). Go to the Advanced tab or the Tools gear icon and select Add Package, then search for BitBot.

For text-based programming there is micro-python, and I prefer to use this offline using the Mu editor. It provides a very neat and easy way of interfacing to the micro:bit without all the fuss of dragging and dropping. **NOTE:** At the time of writing (December 2016), there are problems with Mu when using PWM with neopixels and other things, so it best to use the online micropython editor for now.

The following examples use both of these languages to show code fragments.

Note on examples: We want to show people how the various features can be used. We don't want to do the coding for you – that is the learning experience you should be taking away from using it. In the following examples, we will give you the tools to use all the features of Bit:Bot but it is up to you to combine them together in your code to make it do something useful. We do give a complete example of line-following – but it is a very basic algorithm and will get confused at T-junctions and crossings; and it doesn't use the neopixels.

Motors

Each motor has two pins connected to it. One determines the speed and the other the direction:

Left motor: Speed Pin 0, Direction, Pin 8

Right motor: Speed Pin 1, Direction Pin 12

The simplest way to make the motors move is to set the Speed pin to HIGH and the Direction pin to LOW (to move full speed forwards)

In Python, move left motor Forwards:

```
pin0.write_digital(1)
pin8.write_digital(0)
```

In PXT, move left motor Forwards:

NB. You can find the output pin commands in the "advanced" tab, under "pins"

To move the motor at full speed in reverse, we change which pin is 0 (Low) and 1 (High)

In Python, move left motor Reverse:

```
pin0.write_digital(0)
pin8.write_digital(1)
```

In PXT, move left motor Reverse:

If we want to change the speed of a motor, so that it is not going at full speed all the time, we need to use PWM (Pulse Width Modulation). This is means of changing the amount of power given to the motor by switching it on and off very fast. The percentage value of PWM determines the amount of each cycle that the output is ON. So a percentage of 100% is the same as being on all the time and thus the same as the examples above. A percentage of 50% would mean that the motor is only energised half the time, so it will go much slower. Note that the actual speed of the motor is not the same as the percentage of PWM – the motor won't turn at all if the PWM value is too low and you will also get some stuttering at certain values. Nevertheless, being able to change the speed makes for a much better robot. For example, you can make a line follower that smoothly follows the line, rather than the normal shaking left and right.

To change the PWM value of a pin, we must use the analog_write commands. These can be set to a value between 0 (always off) to 1023 (always on), so 50% would be 511. Here are the commands to change the speed of the Right motor to approx 75% (value is 770)

In Python, move right motor forwards at 75%

```
pin1.write_analog(770)
pin12.write_digital(0)
```

In PXT, move right motor forwards at 75%

Doing this for the motors moving in reverse is a little confusing. Remember we need to change the direction pin to 1 for reverse. Then we need to set the amount of time in each cycle that the speed pin is LOW. This is the opposite of moving forwards, where we set the time for it to be High. Se we simply take the number (770 in this case) away from 1023, giving 253.

In Python, move right motor Reverse at 75%

```
pin1.write_analog(253)
pin12.write_digital(1)
```

In PXT, move right motor reverse at 75%

Neopixels

In fact, the name "neopixel" is a termed coined by Adafruit, but like "hoover" was a name of a brand of vacuum cleaner and is now a general term for all similar products, whoever makes it. The generic term is "smart RGB pixel" and is usually referenced with the name of the chip WS2812B. However, there are many different chips, all performing in a compatible way. The ones on the Bit:Bot are actually SK6812-3535

These smart RGB pixels are able to display any of 16 million colours by selecting a value of 0 to 255 for each of the Red, Green and Blue LEDs on each chip. The whole thing is controlled by a single pin on the BBC micro:bit (pin 13 for Bit:Bot). It is simple to use the included neopixel libraries to control each pixel individually.

The pixels are labelled on the Bit:Bot. From 0 to 5 on the left arm and from 6 to 11 on the right arm. If you connect any more neopixels into the extension port, then the new ones will start at 12.

In Python, set neopixel 2 to purple (red and blue)

```
import neopixel
np = neopixel.NeoPixel(pin13, 12)
np[2] = (40, 0, 40)
np.show( )
```

The first line imports the neopixel library. We only need to do this once, at the very top of your Python programThe second line creates an Python list with an element for each pixel. As shown, it specifies 12 pixels connected to pin 13. If you added more neopixels then you would increase the number from 12 by the number of pixels you added. eg. if you added a McRoboFace, then the total would be 12 + 17 = 29 so you would change the line to: np = neopixel.NeoPixel(pin13, 29)

The third line sets the pixel we have selected (number 2 in this case) to the colour which is set by three values in the brackets, each value can be from 0 to 255 and covers Red, Green and Blue. In our example we have set Red and Blue to 40.

The fourth line tells the neopixel library to copy all the data to the neopixels, from the Python list that we have been working with. It is only at this point that the LEDs change. In general, you would make all the changes you want and only at the end would you use a np.show( )

In PXT, set neopixel 2 to purple

Just like in Python, we need to add the neopixel library. Do this from the menu. Select add package and then select neopixels

You could replace the item “Purple” with the red/green/blue block shown underneath if you want

Line Follower Sensors

These are digital inputs and connected to Pin 11 (left) and Pin 5 (right). These are the same pins as used by the buttons, so pressing a button will have the same effect as detecting a black line. This may have unexpected side-effects – as switching the micro:bit on when both buttons are pressed can cause it to enter Bluetooth pairing mode (depending what software is installed).

So you can use the normal Button inputs to read the sensors if you want, or you can use digital_read commands (as shown below). If the left sensor detects a line, it means the Bit:Bot is too far to the right, so it should move left. The opposite is the case if the right sensor detects a line. Here is some simple code for line following in Python (the actual motor commands are in separate functions for clarity)

```
while True:
    lline = pin11.read_digital()
    rline = pin5.read_digital()
    if (lline == 1):
        spinLeft( )
    elif (rline == 1):
        spinRight( )
    else:
```

```
forward(speed)
```

In PXT, this looks more complicated than the Python, as all the code is inline. You may need to add pauses during the loop depending on your line following track

Light Sensors

These are analog sensors and will give a value of 0 to 1023, where 0 is fully dark and 1023 is maximum brightness. As there are only 3 analog pins available on the micro:bit (without affecting the LED displays) and we are using 2 of them to control the motors, we only have one left (Pin 2) to read the analog values from 2 line sensors. How can we do this? Well, the Bit:Bot has an analog switch that uses a digital output signal (pin 16) to determine whether the analog input we are reading is for the left sensor or the right sensor.

Therefore, to read the light sensors we need to set the selection output pin first, then read the analog data.

In Python, we can do it like this to read the values into 2 variables called leftVal and rightVal:

```
Pin16.write_digital(0) # select left sensor
leftVal = Pin2.read_analog()
Pin16.write_digital(1) # select right sensor
rightVal = Pin2.read_analog()
```

In PXT, we can do it a very similar way:

Buzzer

The buzzer is a very simply device that outputs a 2.4kHz sound when it is set to ON. It is NOT controlled by the tone signal that can be output by the micro:bit on Pin 0 so you don't need to install any libraries to operate it.

It is connected to Pin14. Setting this to ON (1) will activate the buzzer and setting to OFF (0) will deactivate it.

In Python, a very simple and annoying beep, beep, beep sound can be made as follows:

```
while True:
    pin14.write_digital(1)
    sleep(400)
    pin14.write_digital(0)
    sleep(400)
```

An equally annoying sound can be made in PXT:

Ultrasonic Distance Sensor

This optional HC-SR04 ultrasonic distance sensor addon can be used most easily in Microsoft PXT. In MicroPython we are hampered by the lack of a microsecond resolution timer. (but see the demo "Ultrasonic Obstacle Avoider" below)

Within PXT you need to add the package 'Sonar'. You can do this from the Advanced tab (at the bottom) or using the Tool menu (top right cog gear icon)

Both the Ping and the Echo are on the same pin (Pin 15). So you can make a block like this:

This simply prints every 3 seconds the distance measured by the sensor

Example Micropython Programs

- Simple PWM motor test
- Neopixel colour test
- Light follower
- Larson Scanner example
- Line follower
- Ultrasonic Obstacle Avoider
- POST - Power On Self Test. This test is run on each Bit:Bot before shipping