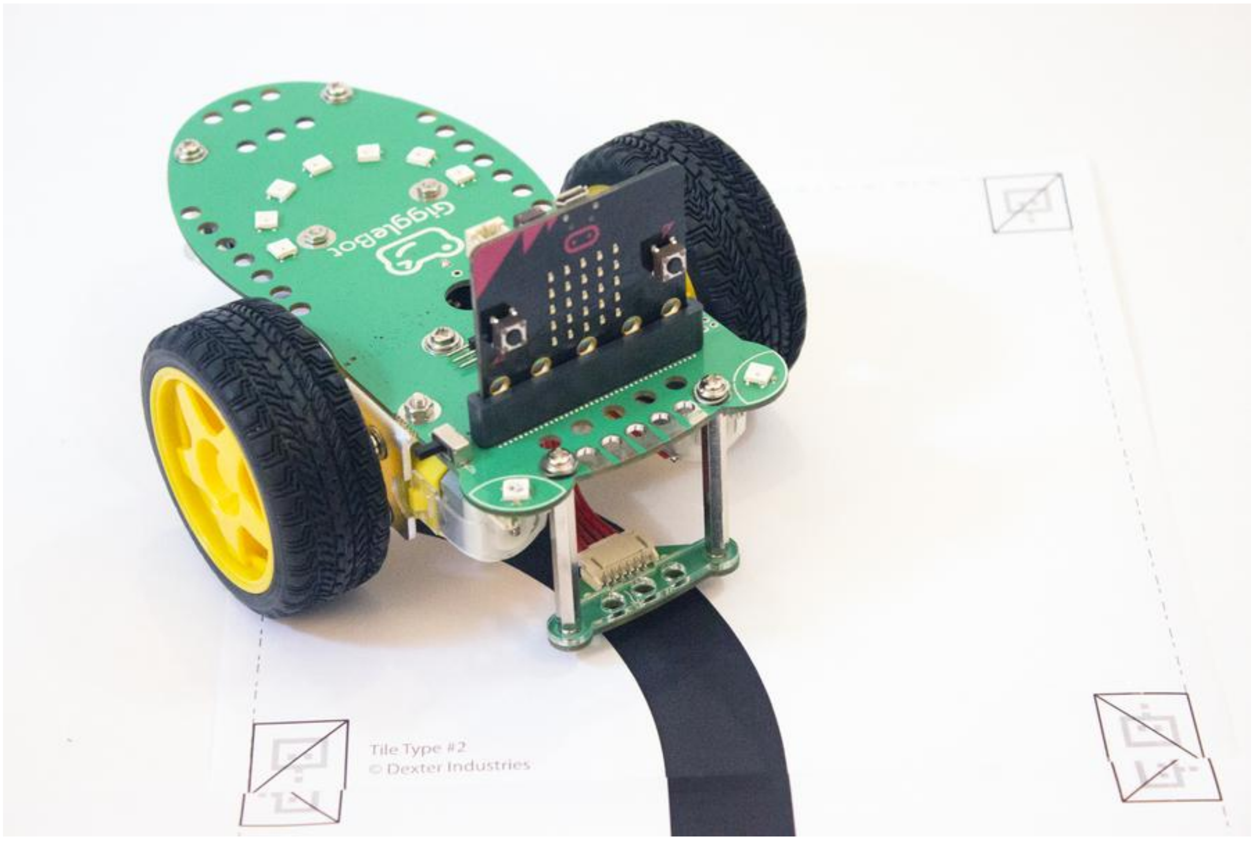
Gigglebot Micropython Library Documentation

Dexter Industries

Oct 11, 2018

Contents:

This library is meant to be used with the Mu editor , a Micro:bit, and GiggleBot.



For step by step installation instructions and how to get started, please see the main GiggleBot website

For an introduction to MicroPython on the Micro:Bit, see the official tutorial.

CHAPTER 1

Tutorial for GiggleBot

This is a tutorial on how to control your gigglebot

1.1 Take the GiggleBot on a stroll

This first tutorial will demonstrate how to control the robot's movements. The GiggleBot will :

1. set its speed to 75% of its maximum power (default is 50%)
2. go forward for a second,
3. go backward for a second,
4. wait for a second,
5. turn left for half a second,
6. turn right for half a second.

```
from gigglebot import *
set_speed(75,75)
drive(FORWARD,1000)
drive(BACKWARD,1000)
microbit.sleep(1000)
turn(LEFT,500)
turn(RIGHT,500)
```

Note: There is no need to make use of the `stop()` method here because each of those timed functions will make sure the robot stops at the end of the delay.

Note: `init()` does not need to be called in this example as we are not making use of the lights on the robot.

1.2 Big Smile

Let's use the Neopixels to turn the smile leds to a nice red, followed by green and then blue.

```
from gigglebot import *
init()
while True:
    set_smile(R=100,G=0,B=0)
    microbit.sleep(500)
    set_smile(R=0,G=100,B=0)
    microbit.sleep(500)
    set_smile(R=0,G=0,B=100)
    microbit.sleep(500)
```

1.3 Rainbow Smile

You are not limited to the basic red,green,blue colors as they can be mixed. Let's create a rainbow of colors! The `init()` method returns a variable that lets you control each neopixel individually. We'll make use of this to create a rainbow.

```
from gigglebot import *
strip=init()
strip[2]=(255,0,0)
strip[2]=(248,12,18)
strip[3]=(255,68,34)
strip[4]=(255,153,51)
strip[5]=(208,195,16)
strip[6]=(34,204,170)
strip[7]=(51,17,187)
strip[8]=(68, 34, 153)
strip.show()
```

1.4 Rainbow Cycle

Here is how you can get the smile to cycle through the colours of the rainbow.

```
from gigglebot import *

# first define the colors of the rainbow in an array
colors = []
colors.append((255,0,0))
colors.append((248,12,18))
colors.append((255,68,34))
colors.append((255,153,51))
colors.append((208,195,16))
colors.append((34,204,170))
colors.append((51,17,187))
colors.append((68, 34, 153))

strip=init()

# offset will let us know which colour is due to be displayed on which LED
```

(continues on next page)

(continued from previous page)

```
offset = 0

# Looping forever
while True:
    offset = offset + 1

    # we might run into an issue of trying to display color 8 - which doesn't exist -␣
↪on LED 7
    # we need to catch that case before it crashes the code.
    if offset > 7:
        offset = 0
    for i in range(7):
        if i+offset > 7:
            colind = i+offset-7
        else:
            colind = i+offset
        strip[i+2]=colors[colind]
    # display the colors
    strip.show()
    # wait a bit for the human eye to catch the colors in question
    microbit.sleep(100)
# colors were taken from http://colrd.com/palette/22198/?download=css
```

CHAPTER 2

Sensors tutorial

The GiggleBot comes with a couple of onboard sensors:

1. Two light sensors, near the front eyes LEDs.
2. Two line sensors, on the underside of the line follower.

2.1 Light Sensors

The GiggleBot comes with two light sensors right in front of the LED on each eye. They are very small and easy to miss. However, they are more versatile than the Micro:Bit's light sensor as they can be read together to detect which side is receiving more light.

The main method to query the light sensors is `read_sensor()`. The same method is used for both light sensors and line sensors.

Here's how to read both light sensors in one call:

```
left, right = read_sensor(LIGHT_SENSOR, BOTH)
```

And here's how to read just one side at a time:

```
right = read_sensor(LIGHT_SENSOR, RIGHT)
```

2.1.1 Chase the Light

This tutorial will turn the GiggleBot into a cat, following a spotlight on the floor. Many cats do that when you shine a flashlight in front of them, they will try to hunt the light spot. When running this code, you will be able to guide your GiggleBot by using a flashlight.

This is how to use this project:

1. Start GiggleBot and wait for sleepy face to appear on the Micro:Bit.

2. Press button A to start the Chase the Light game (heart will replace the sleepy face).
3. Chase the light as long as you want.
4. Press button B to stop the GiggleBot and display sleepy face again.

First, a bit of explanation on the algorithm being used here.

On starting the GiggleBot, the Micro:Bit will:

1. Assign a value to the *diff* variable (here it is using 10).
2. Display a sleepy face image.
3. Resets whatever readings from the buttons it might have had.
4. Start a forever loop.

This forever loop only waits for one thing: for the user to press button A, and that's when the light chasing begins.

As soon as button A is pressed, the Micro:Bit will display a heart as it's quite happy to be active! And then it starts a second forever loop! This second forever loop is the actual Light Chasing game. It will end when button B is pressed by the user.

How to Chase a Light:

1. Take reading from both light sensors.
2. Compare the readings, using *diff* to allow for small variations. You will most likely get absolutely identical readings, even if the light is mostly equal. Using a differential value helps stabilize the behavior. You can adapt to your own lighting conditions by changing this value.
3. If the right sensor reads more than the left sensor plus the *diff* value, then we know it's brighter to the right. Turn right.
4. If the left sensor reads more than the right sensor plus the *diff* value, then it's brighter to the left. Turn left.
5. If there isn't that much of a difference between the two sensors, go straight.
6. If button B gets pressed at any time, stop the robot, change sleepy face, and get out of this internal loop. The code will fall back to the first loop, ready for another game.

```
from gigglebot import *
# value for the differential between the two sensors.
# you can change this value to make it more or less sensitive.
diff = 10
# display sleepy face
microbit.display.show(microbit.Image.ASLEEP)
# the following two lines resets the 'was_pressed' info
# and discards any previous presses
microbit.button_a.was_pressed()
microbit.button_b.was_pressed()

# start first loop, waiting for user input
while True:
    # test for user input
    if microbit.button_a.was_pressed():
        # game got started! Display much love
        microbit.display.show(microbit.Image.HEART)

        # start game loop
        while True:
            # read both sensors
            left, right = read_sensor(LIGHT_SENSOR, BOTH)
```

(continues on next page)

(continued from previous page)

```
            # test if it's brighter to the right
            if right > left+diff:
                turn(RIGHT)

            # test if it's brighter to the left
            elif left > right+diff:
                turn(LEFT)

            # both sides being equal, go straight
            else:
                drive(FORWARD)

            # oh no, the game got interrupted
            if microbit.button_b.is_pressed():
                stop()
                microbit.display.show(microbit.Image.ASLEEP)

                # this line here gets us out of the game loop
                break
```

What else can be done with the light sensors?

You could modify this code to turn the GiggleBot into a night insect? Those would avoid light instead of chasing it.

You could detect when it gets dark or bright. Imagine the GiggleBot inside your closet. When someone opens the door, the sudden light can be detected. The GiggleBot can let you know someone went through your things while you were away.

2.2 Line Sensors

In front of GiggleBot, attached to the body, there is a line follower sensor. It contains two line sensors. You can spot them from the top of the line follower by two white dots. And from the bottom, they are identified as *R* and *L* (for *right* and *left*)

_static/images/GigglebotLineFollowingSensors.jpg

photo courtesy of Les Pounder

The easiest way of reading the sensors is as follow:

```
from gigglebot import *
left, right = read_sensor(LINE_SENSOR, BOTH)
```

The lower the number, the darker it is reading. Values can go from 0 to 1023 and depend a lot on your environment. If you want to write a line follower robot, it is best to take a few readings first, to get a good idea of what numbers will represent a black line, and what numbers represent a white line.

2.2.1 Calibrating the Line Follower

Calibrating the line follower means figuring out which numbers get returned when it's over a black line, so that you can later code an actual line follower robot.

The best approach for this is to get readings in various parts of your line, from both sensors, for both the black line and the background color.

The following code will display the values onto the microbit leds when you press button A, allowing you to manually position your robot around your circuit and take readings.

```
from gigglebot import *
# reset all previous readings of button_a
# strictly speaking this is not necessary, it is just a safety thing
microbit.button_a.was_pressed()
while True:
    if microbit.button_a.is_pressed():
        left, right = read_sensor(LINE_SENSOR, BOTH)
        microbit.display.scroll(left)
        microbit.display.scroll(right)
```

2.2.2 Follow the Line

Once you have gotten readings from the line sensors, you are ready to code a line follower robot.

Here we are coding for a line that is thick enough that both sensors can potentially be over the line. The robot will stop if it loses track of the line, in other words, if both sensors detect they're over the background color.

The logic will be as follow:

1. If both sensors detect a black line, forge straight ahead.
2. If neither sensor detects a black line, give up and stop.
3. If the right sensor detects a black line but not the left sensor, then steer to the right.
4. If the left sensor detects a black line but not the right sensor, then steer to the left.

We are also using the LEDs on the LED smile to indicate what is going on while we follow the line.

```
from gigglebot import *
# reset all previous readings of button_a, and button_b
# strictly speaking this is not necessary, it is just a safety thing
microbit.button_a.was_pressed()
microbit.buttom_b.was_pressed()
microbit.display.show(microbit.Image.YES)
strip=init()
# speed needs to be set according to your line and battery level.
# do not go too fast though.
set_speed(60, 60)
# threshold is a little over the highest number you got that indicates a
# black line.
threshold = 90
while True:
    # if both buttons are pressed, run calibration code
    if microbit.button_a.is_pressed() and microbit.button_b.is_pressed():
        left, right = read_sensor(LINE_SENSOR, BOTH)
        microbit.display.scroll(left)
        microbit.display.scroll(right)
```

(continues on next page)

(continued from previous page)

```
    # if button A is pressed run line following code until button B gets pressed
    # or until we're over white/background
    if microbit.button_a.is_pressed():
        while not microbit.button_b.is_pressed():
            left, right = read_sensor(LINE_SENSOR, BOTH)
            if left < threshold and right < threshold:
                # both sensors detect the line
                strip[2]=(0,255,0)
                strip[8]=(0,255,0)
                strip.show()
                drive(FORWARD)
            elif right > threshold and left > threshold:
                # neither sensor detects the line
                stop()
                strip[2]=(255,0,0)
                strip[8]=(255,0,0)
                strip.show()
                break
            elif left > threshold and right < threshold:
               # only the right sensor detects the line
                strip[2]=(0,255,0)
                strip[8]=(0,0,0)
                strip.show()
                turn(RIGHT)
            elif right > threshold and left < threshold:
                # only the left sensor detects the line
                strip[2]=(0,0,0)
                strip[8]=(0,255,0)
                strip.show()
                turn(LEFT)
        stop()
```

CHAPTER 3

API

`gigglebot.`**`LEFT = 0`**
Left, either a left turn, or the left motor.

`gigglebot.`**`RIGHT = 1`**
Right, either a right turn, or the right motor.

`gigglebot.`**`BOTH = 2`**
indicates both motors.

`gigglebot.`**`FORWARD = 1`**
Forward direction if the motor power is positive. Please note that if the motor power is negative, this forward would become a backward.

`gigglebot.`**`BACKWARD = -1`**
Backward direction if the motor power is positive. Please note that if the motor power is negative, this backward would become a forward.

`gigglebot.`**`motor_power_left = 50`**
Power to the left motor. From -100 to 100. Negative numbers will end up reversing the movement. Default value is 50%.

`gigglebot.`**`motor_power_right = 50`**
Power to the right motor. From -100 to 100. Negative numbers will end up reversing the movement. Default value is 50%.

`gigglebot.`**`neopixelstrip = None`**
neopixel variable to control all neopixels. There are 9 neopixels on the Gigglebot. Pixel 0 is the right eye, pixel 1 is the left eye. Pixel 2 is the first of the rainbow pixel, on the right side. In order to control the neopixels, you must call `init()` beforehand.

`gigglebot.`**`LINE_SENSOR = 5`**
I2C command to read the line sensors.

`gigglebot.`**`LIGHT_SENSOR = 6`**
I2C command to read the light sensors.

`gigglebot.init()`

Loads up the neopixel library and sets up the neopixelstrip variable for later use.

Important: It is possible to use the Gigglebot without calling this method but the leds will not work. Should you need more RAM space, you can choose to ignore the neopixels.

Returns the neopixel strip

`gigglebot.drive(`*dir=1*, *milliseconds=-1*`)`

This results in the Gigglebot driving FORWARD or BACKWARD.

The following snippet of code will see the gigglebot drive forward for a second:

```
from gigglebot import *
drive(FORWARD, 1000)
```

And this snippet of code will do the same thing:

```
import gigglebot
gigglebot.drive(gigglebot.FORWARD, 1000)
```

Parameters

- **dir = FORWARD** (*int*) – Possible values are FORWARD (1) or BACKWARD (-1). Please note there are no tests done on this value. One could theoretically use 2 to double the speed.
- **milliseconds = -1** (*int*) – If this parameter is omitted, or a negative value is supplied, the robot will keep on going until told to do something else, like turning or stopping. If a positive value is supplied, the robot will drive for that quantity of milliseconds.

`gigglebot.turn(`*dir=0*, *milliseconds=-1*`)`

Will get the gigglebot to turn left or right by temporarily removing power to one wheel.

Parameters

- **dir=LEFT** (*int*) – Either LEFT (0) or RIGHT (1) to determine the direction of the turn.
- **milliseconds=-1** (*int*) – If this parameter is omitted, or a negative value is supplied, the robot will keep on going until told to do something else, like turning or stopping. If a positive value is supplied, the robot will drive for that quantity of milliseconds.

`gigglebot.set_speed(`*power_left*, *power_right*`)`

Assigns left and right motor powers. If both are the same speed, the GiggleBot will go mostly straight.

Note: It is possible that the GiggleBot does not go straight by default. If so, you need to adjust the speed of each motor to correct the course of the robot.

`gigglebot.stop()`

Stops the GiggleBot right away.

`gigglebot.set_smile(`*R=25*, *G=0*, *B=0*`)`

Controls the color of the smile neopixels, all together.

Parameters

- **R = 25** (*int*) – Red component of the color, from 0 to 254.

- **G = 0** (*int*) – Green component of the color, from 0 to 254.
- **B = 0** (*int*) – Blue component of the color, from 0 to 254.

gigglebot.**set_eyes**(*which=2*, *R=0*, *G=0*, *B=10*)

Controls the color of the two eyes, each one individually or both together.

Parameters

- **which = BOTH** (*int*) – either LEFT (0), RIGHT (1), or BOTH (2).
- **R = 0** (*int*) – Red component of the color, from 0 to 254.
- **G = 0** (*int*) – Green component of the color, from 0 to 254.
- **B = 10** (*int*) – Blue component of the color, from 0 to 254.

gigglebot.**set_eye_color_on_start**()

Sets the eye color to blue if the batteries are good, to red if the batteries are running low. This is called by the *init()*, usually at the start of the program. You are free to call this method whenever you want if you need to keep a closer watch on the voltage level.

gigglebot.**pixels_off**()

Turns all neopixels off, both eyes and smile.

gigglebot.**set_servo**(*which=0*, *degrees=90*)

Parameters

- **which** (*int*) – Which servo to control: LEFT (0), RIGHT (1), or BOTH (2).
- **degrees** (*int*) – Position of the servo, from 0 to 180.

Note: Moving the servo is not instantaneous. It is possible that you will need to give it time to reach its final position.

The following is an example that will get the servo moving from 0 to 180 degrees every second.

```python
from gigglebot import *
import microbit
while True:
    set_servo(BOTH, 0)
    microbit.sleep(1000) # sleeps for 1000 milliseconds
    set_servo(BOTH, 180)
    microbit.sleep(1000) # sleeps for 1000 milliseconds
```

gigglebot.**servo_off**(*which*)

Removes power from the servo.

Parameters **which** (*int*) – Determines which servo, LEFT (0), RIGHT (1), BOTH (2).

gigglebot.**read_sensor**(*which_sensor*, *which_side*)

Reads the GiggleBot onboard sensors, light or line sensors.

Parameters

- **which_sensor** (*int*) – Reads the light sensors LIGHT_SENSOR (6), or the line sensors LINE_SENSOR (5). Values are from 0 to 1023.
- **which_side** (*int*) – Reads LEFT (0), RIGHT (1), or BOTH (2) sensors. When reading both sensors, an array will be returned.

Returns either an integer or an array of integers (left, then right)

You can read the sensors this way:

```
left, right = read_sensor(LIGHT_SENSOR, BOTH)
```

`gigglebot.`**`volt`**`()`

Returns the voltage level of the batteries

Returns voltage level of the batteries.

CHAPTER 4

Indices and tables

- genindex
- search

Python Module Index

Index