hub2hub

Release 0.1.0

Nard Strijbosch

Jun 01, 2021

GETTING STARTED

CHAPTER

ONE

INSTALLATION

The following steps guide you through the process to install the `hub2hub` library. This installation is only required on the LEGO Eduction SPIKE Prime hub or the LEGO MINDSTORMS Robot Inventor hub. For the PoweredUP hubs used in your project it is sufficient to update them with the latest official firmware via the PoweredUP app.

1.1 Step 1

If you did not update your SPIKE Prime software yet, install one of the latest versions of the SPIKE Prime app (1.3.5, 1.3.4 or 1.3.3) and connect your device. If the app asks for a hub update, perform the hub update.

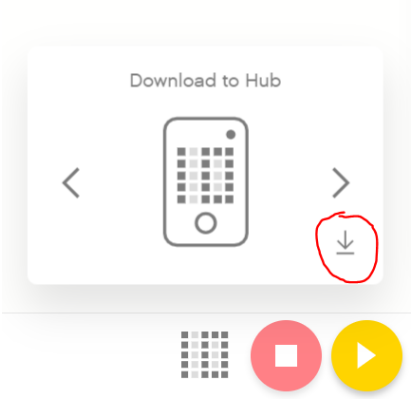
> **Warning:** This library does not work when using the MINDSTORMS Robot Inventor app. Due to a firmware difference, the Bluetooth module is not available when using the MINDSTORMS Robot Inventor App and corresponding firmware. Luckily you can just connect your MINDSTORMS hub to the SPIKE Prime app and update the SPIKE PRIME Firmware on the hub.

1.2 Step 2

Download and open the project: Instal_hub2hub_v011.llsp

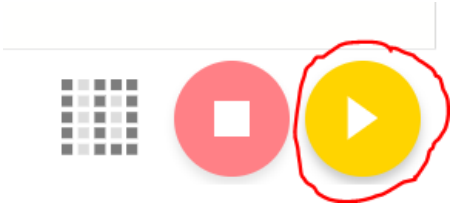
1.3 Step 3

Set the SPIKE Prime execution mode in download mode, and select an unused project slot.



1.4 Step 4

Run the project by pressing the play button and wait until the hub is powered down. This process can take up to a minute. If you use a USB cable to connect the hub, the hub will probably restart automatically.



1.5 Step 5

Installation is successful. The hub2hub library is now installed on your hub. You can now safely use the slot you used for this installation for a new project.

> **Warning:** Firmware updates of the hub are likely to remove the library from the filesystem of the hub. Hence, this procedure should be repeated if a firmware update removed the library from the hub.

CHAPTER

TWO

HUBS

2.1 Technic Hub

class TechnicHub(*bt_handler*)

Class to control a Technic Hub.

Parameters **bt_handler** – The bluetooth handler.

led

Led

motion

Motion

port.A.**device**

port.B.**device**

port.C.**device**

port.D.**device**

Device

port.A.**motor**

port.B.**motor**

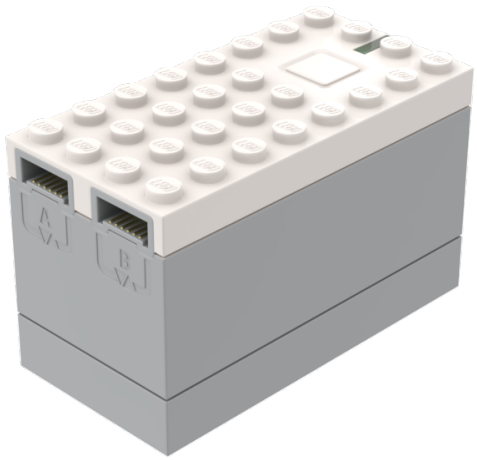
port.C.**motor**

port.D.**motor**
Motor

2.1.1 Example

```
from hub2hub import TechnicHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a technic hub
ble = ble_handler()
Thub = TechnicHub(ble)

# connect to a technic hub: press green button on the technic hub
Thub.connect()

k = 0
while True:
    Thub.led(k%11)

    yaw, pitch, roll = Thub.motion.yaw_pitch_roll()
    shaken = Thub.motion.was_gesture(3)

    print('Roll angle: ', roll, 'Shaken?: ', shaken)

    k+=1
    sleep_ms(1000)
```

2.2 City Hub



class CityHub(*bt_handler*)
Class to control a Technic Hub.

Parameters **bt_handler** – The bluetooth handler.

led
Led

port.A.**device**

port.B.**device**
Device

port.A.**motor**

port.B.**motor**
Motor

2.2.1 Example

```
from hub2hub import CityHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a city hub
ble = ble_handler()
Chub = CityHub(ble)

# connect to a city hub: press green button on the city hub
Chub.connect()

k = 0
while True:
    Chub.led(k%11)

    k+=1
    sleep_ms(1000)
```

2.3 Remote

class Remote(*bt_handler*)
Class to control a PoweredUP remote

Parameters **bt_handler** – The bluetooth handler.

led
Led

left.**plus**

left.**red**

left.**min**

right.**plus**

right.**red**

right.**min**
Button

2.3.1 Example

```
from hub2hub import Remote, ble_handler
from time import sleep_ms

# Initialize ble handler and a remote
ble = ble_handler()
Remote = Remote(ble)

# connect to a remote: press green button on the remote
Remote.connect()

k = 0
while True:
    Remote.led(k%11)
    print('Left plus pressed: ', Remote.left.plus.is_pressed())
    print('Right plus was pressed: ', Remote.right.plus.was_pressed())

    k+=1
    sleep_ms(1000)
```

2.4 Mario

class Mario(*bt_handler*)

Class to control a LEGO Mario

Parameters **bt_handler** – The bluetooth handler.

barcode

Barcode

motion

MotionMario

pants

Pants

2.4.1 Example

```
from hub2hub import ble_handler, Mario
from time import sleep_ms

ble = ble_handler()
mario = Mario(ble)

mario.connect()

while True:
    gesture = mario.motion.was_gesture(1024)
    barcode, color = mario.barcode.get()
    pants = mario.pants.get()
    print('barcode: ', barcode, 'color: ', color, 'gesture: ', gesture, 'pants: ', pants)
    sleep_ms(100)
```

CHAPTER

THREE

SENSORS

3.1 Color Sensor

The following modes are supported by the color sensor.

Mode	Name	Values	Unit	Datasets
0	Color	0-10	Index	1
1	Reflected light	0-100	Percentage	1
2	Ambient light	0-100	Percentage	1
3	Control LEDs	0-100	Percentage	3
4	Raw reflected light	0-1024	RAW	2
5	RGB I	0-1024	RAW	4
6	HSV	0-360	RAW	3
7	SHSV	0-360	RAW	4

3.1.1 Examples

Measure ambient light

```
from hub2hub import TechnicHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a technic hub
ble = ble_handler()
Thub = TechnicHub(ble)

# connect to a technic hub: press green button on the technic hub
Thub.connect()

# Color sensor
ColorSensor = Thub.port.A.device

# Set mode to ambient light
ColorSensor.mode(2)

k = 0
```

(continues on next page)

(continued from previous page)

```
while True:
    Thub.led(k%11)

    ambient, = ColorSensor.get()

    print('Ambient light: ', ambient)

    k+=1
    sleep_ms(1000)
```

Control LEDs

```
from hub2hub import TechnicHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a technic hub
ble = ble_handler()
Thub = TechnicHub(ble)

# connect to a technic hub: press green button on the technic hub
Thub.connect()

# Color sensor connected to port A
ColorSensor = Thub.port.A.device

k = 0
while True:
    Thub.led(k%11)

    Led1 = 9 if k%3 == 0 else 0
    Led2 = 9 if k%3 == 1 else 0
    Led3 = 9 if k%3 == 2 else 0

    ColorSensor.mode(3,[Led1, Led2, Led3])

    k+=1
    sleep_ms(1000)
```

3.2 Ultrasonic Sensor

The following modes are supported by the ultrasonic sensor.

Mode	Name	Value	Unit	Datasets
0	Distance long	0-250	cm	1
1	Distance short	0-32	cm	1
2	SINGL?	0-250	cm	1
3	LISTN?	0-1	ST?	1
4	Time Raw	0-14577	us	1
5	Control LEDs	0-100	Percentage	4
6	PING?	0-100	Percentage	1
7	A/D Raw?	0-1024	Raw	1

3.2.1 Examples

Measure distance

```
from hub2hub import TechnicHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a technic hub
ble = ble_handler()
Thub = TechnicHub(ble)

# connect to a technic hub: press green button on the technic hub
Thub.connect()

# Ultrasonic sensor connected to port A
USSensor = Thub.port.A.device

# Set mode to distance long
USSensor.mode(0)

k = 0
while True:
    Thub.led(k%11)

    distance, = USSensor.get()

    print('distance: ', distance)

    k+=1
    sleep_ms(1000)
```

Control LEDs

```
from hub2hub import TechnicHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a technic hub
ble = ble_handler()
Thub = TechnicHub(ble)

# connect to a technic hub: press green button on the technic hub
Thub.connect()

# Ultrasonic sensor connected to port A
USSensor = Thub.port.A.device

k = 0
while True:
    Thub.led(k%11)

    Led1 = 9 if k%4 == 0 else 0
    Led2 = 9 if k%4 == 1 else 0
    Led3 = 9 if k%4 == 2 else 0
    Led4 = 9 if k%4 == 3 else 0

    USSensor.mode(5,[Led1, Led2, Led3, Led4])

    k+=1
    sleep_ms(1000)
```

3.3 Force Sensor

The following modes are supported by the force sensor.

Mode	Name	Value	Unit	Datasets
0	Force	0-10	N	1
1	Touch	0-1	ST	1
2	Tap	0-3	TEV	1
3	Peak force	0-10	N	1
4	Force Raw	0-1023	RAW	1
5	Peak force Raw	0-1023	RAW	1

3.3.1 Examples

Measure RAW force

```
from hub2hub import TechnicHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a technic hub
ble = ble_handler()
Thub = TechnicHub(ble)

# connect to a technic hub: press green button on the technic hub
Thub.connect()

# Force sensor connected to port A
ForceSensor = Thub.port.A.device

# Set mode to raw force
Force.mode(4)

k = 0
while True:
    Thub.led(k%11)

    raw_force, = ForceSensor.get()

    print('Raw force: ', raw_force)

    k+=1
    sleep_ms(1000)
```

3.4 Color/Distance Sensor

The following modes are supported by the color/distance sensor.

Mode	Name	Value	Unit	Datasets
0	Color	0-10	Index	1
1	Distance	0-10	Distance	1
2	Presses	0-…	Count	1
3	Reflected light	0-100	Percentage	1
4	Ambient Light	0-100	Percentage	1
5	Control LED color	0-10	Index	1
6	RGB I	0-1023	RAW	3
7	Transmit data with infra red	N/A	N/A	N/A

3.4.1 Examples

Measure distance

3.5 WeDo Distance Sensor 

The following modes are supported by the WeDo distance sensor.

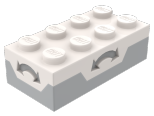
Mode	Name	Value	Unit	Datasets
0	Distance	0-10	Distance	1
1	Count	0-..	Count	1

3.5.1 Examples

Measure count

```
from hub2hub import TechnicHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a technic hub
ble = ble_handler()
Thub = TechnicHub(ble)

# connect to a technic hub: press green button on the technic hub
Thub.connect()

# WeDo distance sensor connected to port A
WeDoDistance = Thub.port.A.device

# Set mode to count of taps on the sensor
WeDoDistance.mode(1)

k = 0
while True:
    Thub.led(k%11)

    count, = WeDoDistance.get()

    print('Count: ', count)

    k+=1
    sleep_ms(1000)
```

3.6 WeDo Tilt Sensor



The following modes are supported by the WeDo tilt sensor.

Mode	Name	Value	Unit	Datasets
0	Angle	-45 - 45	Distance	2
1	Tilt	0-10	Count	1

3.6.1 Examples

Measure tilt angles

```
from hub2hub import TechnicHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a technic hub
ble = ble_handler()
Thub = TechnicHub(ble)

# connect to a technic hub: press green button on the technic hub
Thub.connect()

# WeDo tilt sensor connected to port A
TiltSensor = Thub.port.A.device

# Set mode to angle
TiltSensor.mode(0)

k = 0
while True:
    Thub.led(k%11)

    angle_x, angle_y = TiltSensor.get()

    print('angle x: ', angle_x, 'angle y: ', angle_y)

    k+=1
    sleep_ms(1000)
```

CHAPTER

FOUR

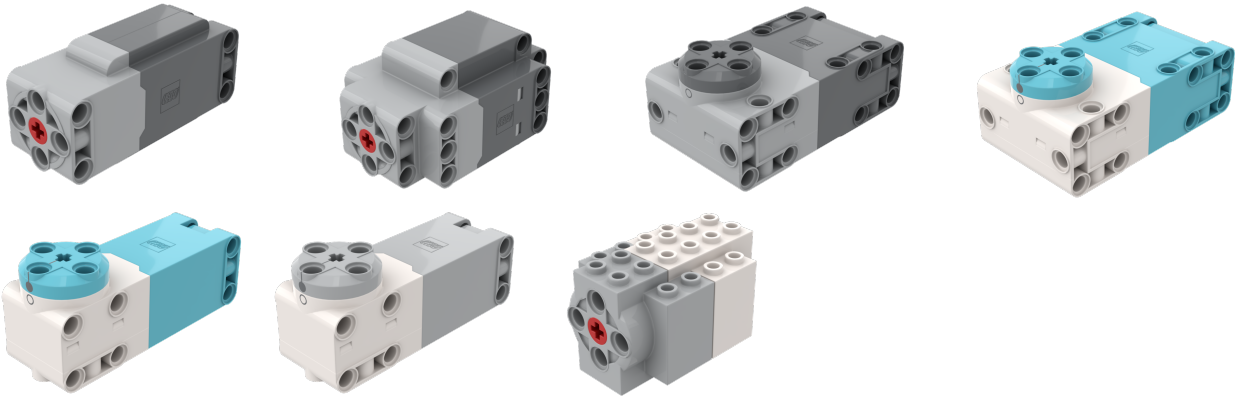
MOTORS

4.1 Basic Motor

All PoweredUP basic motors currently supported are:

4.2 Servomotors

All PoweredUP servo motors currently supported are:



The modes of each of the servo motors are given below.

Mode	Name	Value	Unit	Datasets
0	Power	0-100	Percentage	1
1	Speed	0-100	Percentage	1
2	Relative Position		Degrees	1
3	Absolute Position	-180 - 179	Degrees	1
4	Load	0-100	Percentage	1

4.2.1 Example

Run at constant speed

```
from hub2hub import TechnicHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a technic hub
ble = ble_handler()
Thub = TechnicHub(ble)

# connect to a technic hub: press green button on the technic hub
Thub.connect()

# Servo motor connected to port A
Motor = Thub.port.A.motor

# Set speed to 50
Motor.run_at_speed(50)

# Wait 1 second
sleep_ms(1000)

# Set speed to 0
Motor.run_at_speed(0)
```

CHAPTER

FIVE

PUPHUB

class PUPhub(*bt_handler*)

General LEGO PoweredUP hub class

The methods included in this class are identical for each PoweredUP hub

Replace PUPhub with

- TechnicHub for 
- CityHub for 
- Remote for 
- Mario for 

connect(*timeout=30000*, *address=None*)

Connect to the PoweredUP hub

Parameters

- **timeout** – time of scanning for devices in ms, default is 30000
- **address** – mac address of device (optional), connect to a specific device if set

disconnect()

Disconnect from a PoweredUP hub

is_connected()

Check if hub is connected

Returns True if hub is connected

Return type boolean

5.1 Example

```
from hub2hub import CityHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a city hub
ble = ble_handler()
Chub = CityHub(ble)

# connect to a city hub: press green button on the city hub
Chub.connect()

k = 0
while True:
    Chub.led(k%11)

    k+=1
    sleep_ms(1000)
```

CHAPTER

SIX

LED

Led(*color*)

Set LED color

Supported on: 

Parameters **color** (*integer*) – color index

6.1 Example

```
from hub2hub import CityHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a city hub
ble = ble_handler()
Chub = CityHub(ble)

# connect to a city hub: press green button on the city hub
Chub.connect()

k = 0
while True:
    Chub.led(k%11)

    k+=1
    sleep_ms(1000)
```

CHAPTER

SEVEN

BUTTON

class Button
Class to control a button

is_pressed()
check if button is pressed

Returns True if button is pressed; False if button is not pressed

was_pressed()
check if button was pressed since last call

Returns True if button was pressed since last call; False if button was not pressed since last call

presses()
Number of presses since last call

Returns integer value of number of presses since last call

7.1 Example

```
from hub2hub import Remote, ble_handler
from time import sleep_ms

# Initialize ble handler and a remote
ble = ble_handler()
Remote = Remote(ble)

# connect to a remote: press green button on the remote
Remote.connect()

k = 0
while True:
    Remote.led(k%11)
    print('Left plus pressed: ', Remote.left.plus.is_pressed())
    print('Right plus was pressed: ', Remote.right.plus.was_pressed())

    k+=1
    sleep_ms(1000)
```

CHAPTER

EIGHT

MOTION

class Motion

Class to handle motion sensor in PoweredUP hub

Supported on: 

accelerometer()

Measure acceleration around three axis

Returns accleration around x,y,z axis

Return type tuple

gyroscope()

Measure gyro rates around three axis

Returns gyro rates around x,y,z axis

Return type tuple

yaw_pitch_roll()

Measure yaw pitch roll angles

Returns yaw pitch roll angle

Return type tuple

8.1 Example

```
from hub2hub import TechnicHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a technic hub
ble = ble_handler()
Thub = TechnicHub(ble)

# connect to a technic hub: press green button on the technic hub
Thub.connect()

k = 0
while True:
    Thub.led(k%11)
```

(continues on next page)

(continued from previous page)

```
    yaw, pitch, roll = Thub.motion.yaw_pitch_roll()
    shaken = Thub.motion.was_gesture(3)

    print('Roll angle: ', roll, 'Shaken?: ', shaken)

    k+=1
    sleep_ms(1000)
```

CHAPTER

NINE

DEVICE

class Device

Class to control PoweredUp devices connected to a physical port

Supported on:  

mode(*mode*, **data*)

Set the mode of the sensor

Parameters

- **mode** (*byte*) – new mode
- ***data** – optional data to be send allong with mode, e.g., to turn on LEDs of a sensor

get()

Get measurement of the sensor corresponding to the active mode

Returns measurement

Return type tuple

9.1 Examples

9.1.1 Measure ambient light

```
from hub2hub import TechnicHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a technic hub
ble = ble_handler()
Thub = TechnicHub(ble)

# connect to a technic hub: press green button on the technic hub
Thub.connect()

# Color sensor
ColorSensor = Thub.port.A.device

# Set mode to ambient light
ColorSensor.mode(2)
```

(continues on next page)

(continued from previous page)

```
k = 0
while True:
    Thub.led(k%11)

    ambient, = ColorSensor.get()

    print('Ambient light: ', ambient)

    k+=1
    sleep_ms(1000)
```

9.1.2 Control LEDs

```
from hub2hub import TechnicHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a technic hub
ble = ble_handler()
Thub = TechnicHub(ble)

# connect to a technic hub: press green button on the technic hub
Thub.connect()

# Color sensor connected to port A
ColorSensor = Thub.port.A.device

k = 0
while True:
    Thub.led(k%11)

    Led1 = 9 if k%3 == 0 else 0
    Led2 = 9 if k%3 == 1 else 0
    Led3 = 9 if k%3 == 2 else 0

    ColorSensor.mode(3,[Led1, Led2, Led3])

    k+=1
    sleep_ms(1000)
```

CHAPTER

TEN

MOTOR

class Motor(*hub*, *port*, *device*)

Class to control PoweredUp motors

Supported on:  

mode(*mode*)

Set the mode of the motor, this mainly affects the measurement output

Parameters **mode** (*byte*) – new mode

get()

Get measurement of the motor corresponding to the active mode

Returns measurement

Return type tuple

pwm(*power*)

Set motor power

Parameters **power** (*int*) – in range [-100,..., 100]

run_at_speed(*speed*, *max_power=100*, *acceleration=100*, *deceleration=100*)

Start motor at given speed

Parameters

- **speed** (*int*) – a percentage off the maximum speed of the motor in the range [-100,..., 100]
- **max_power** (*int*) – maximum power that can be used by the motor
- **acceleration** (*int*) – the duration time for an acceleration from 0 to 100%. i.e. a time set to 1000 ms. should give an acceleration time of 300 ms from 40% to 70%
- **deceleration** (*int*) – the duration time for a deceleration from 0 to 100%. i.e. a time set to 1000 ms. should give an deceleration time of 300 ms from 70% to 40%

run_for_time(*time*, *speed=50*, *max_power=100*, *acceleration=100*, *deceleration=100*, *stop_action=0*)

Rotate motor for a given time

Parameters

- **time** – time in milliseconds
- **speed** (*int*) – a percentage off the maximum speed of the motor in the range [-100,..., 100]
- **max_power** (*int*) – maximum power that can be used by the motor

- **acceleration** (*int*) – the duration time for an acceleration from 0 to 100%. i.e. a time set to 1000 ms. should give an acceleration time of 300 ms from 40% to 70%
- **deceleration** (*int*) – the duration time for a deceleration from 0 to 100%. i.e. a time set to 1000 ms. should give an deceleration time of 300 ms from 70% to 40%
- **stop_action** (*int*) – action performed after the given time: `float = 0`, `brake = 1`, `hold = 2`

run_for_degrees(*degrees*, *speed=50*, *max_power=100*, *acceleration=100*, *deceleration=100*, *stop_action=0*)

Rotate motor for a given number of degrees relative to current position

Parameters

- **degrees** – relative degrees
- **speed** (*int*) – a percentage off the maximum speed of the motor in the range [-100,..., 100]
- **max_power** (*int*) – maximum power that can be used by the motor
- **acceleration** (*int*) – the duration time for an acceleration from 0 to 100%. i.e. a time set to 1000 ms. should give an acceleration time of 300 ms from 40% to 70%
- **deceleration** (*int*) – the duration time for a deceleration from 0 to 100%. i.e. a time set to 1000 ms. should give an deceleration time of 300 ms from 70% to 40%
- **stop_action** (*int*) – action performed after the given time: `float = 0`, `brake = 1`, `hold = 2`

run_to_position(*degrees*, *speed=50*, *max_power=100*, *acceleration=100*, *deceleration=100*, *stop_action=0*)

Rotate motor for a given number of degrees relative to current position

Parameters

- **degrees** – relative degrees
- **speed** (*int*) – a percentage off the maximum speed of the motor in the range [-100,..., 100]
- **max_power** (*int*) – maximum power that can be used by the motor
- **acceleration** (*int*) – the duration time for an acceleration from 0 to 100%. i.e. a time set to 1000 ms. should give an acceleration time of 300 ms from 40% to 70%
- **deceleration** (*int*) – the duration time for a deceleration from 0 to 100%. i.e. a time set to 1000 ms. should give an deceleration time of 300 ms from 70% to 40%
- **stop_action** (*int*) – action performed after the given time: `float = 0`, `brake = 1`, `hold = 2`

float()

Float motor from current position

brake()

Brake motor at current position

hold()

Actively hold motor at current position

10.1 Example

10.1.1 Measure absolute position

```
from hub2hub import TechnicHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a technic hub
ble = ble_handler()
Thub = TechnicHub(ble)

# connect to a technic hub: press green button on the technic hub
Thub.connect()

# Servo motor connected to port A
Motor = Thub.port.A.motor

# Set mode to absolute position
Motor.mode(3)

k = 0
while True:
    Thub.led(k%11)

    # Get measurement
    absolute_position, = motor.get()

    print('Absolute position: ', absolute_position)

    k+=1
    sleep_ms(1000)
```

10.1.2 Run at constant speed

```
from hub2hub import TechnicHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a technic hub
ble = ble_handler()
Thub = TechnicHub(ble)

# connect to a technic hub: press green button on the technic hub
Thub.connect()

# Servo motor connected to port A
Motor = Thub.port.A.motor

# Set speed to 50
Motor.run_at_speed(50)
```

(continues on next page)

(continued from previous page)

```
# Wait 1 second
sleep_ms(1000)

# Set speed to 0
Motor.run_at_speed(0)
```

10.1.3 Move to absolute position

```
from hub2hub import TechnicHub, ble_handler
from time import sleep_ms

# Initialize ble handler and a technic hub
ble = ble_handler()
Thub = TechnicHub(ble)

# connect to a technic hub: press green button on the technic hub
Thub.connect()

# Servo motor connected to port A
Motor = Thub.port.A.motor

# move to 180 degrees and hold
Motor.run_to_position(180,stop_action = 2)

sleep_ms(1000)

# move to 0 and float
Motor.run_to_position(0, stop_action = 0)
```

CHAPTER

ELEVEN

BARCODE

class Barcode

Class to handle barcode sensor

Supported on: 

get()

Return current barcode and color

Returns barcode, color

Return type tuple

11.1 Example

```
from hub2hub import ble_handler, Mario
from time import sleep_ms

ble = ble_handler()
mario = Mario(ble)

mario.connect()

while True:
    gesture = mario.motion.was_gesture(1024)
    barcode, color = mario.barcode.get()
    pants = mario.pants.get()
    print('barcode: ', barcode, 'color: ', color, 'gesture: ', gesture, 'pants: ', pants)
    sleep_ms(100)
```

CHAPTER

TWELVE

MOTION MARIO

class MotionMario

Class to handle motion sensor in LEGO Mario

Supported on: 

gesture()

Return active gesture

Returns gesture

Return type int

was_gesture(*gesture*)

Return if gesture was active since last call

Parameters **gesture** (*int*) – gesture to check

Returns True if gesture was active since last call, otherwise False

Return type boolean

12.1 Example

```
from hub2hub import ble_handler, Mario
from time import sleep_ms

ble = ble_handler()
mario = Mario(ble)

mario.connect()

while True:
    gesture = mario.motion.was_gesture(1024)
    barcode, color = mario.barcode.get()
    pants = mario.pants.get()
    print('barcode: ', barcode, 'color: ', color, 'gesture: ', gesture, 'pants: ', pants)
    sleep_ms(100)
```

CHAPTER

THIRTEEN

PANTS (MARIO)

class Pants

Class to detect a LEGO Mario pants

Supported on: 

get()

Get current pants

Returns value corresponding to a pants

Return type int

13.1 Example

```
from hub2hub import ble_handler, Mario
from time import sleep_ms

ble = ble_handler()
mario = Mario(ble)

mario.connect()

while True:
    gesture = mario.motion.was_gesture(1024)
    barcode, color = mario.barcode.get()
    pants = mario.pants.get()
    print('barcode: ', barcode, 'color: ', color, 'gesture: ', gesture, 'pants: ', pants)
    sleep_ms(100)
```

The hub2hub library is not part of the original firmware of either the Robot Inventor hub or the SPIKE Prime hub. In the official SPIKE Prime firmware, a low-level ubluetooth library is available to be able to directly communicate over Bluetooth Low Energy (BLE). The documentation of this module can be found here: ubluetooth documentation

The main goal of the hub2hub library is to simplify the python code required to setup and maintain a BLE connection between LEGO hubs. Installation of the library and writing programs that use it are both possible via the official LEGO Education SPIKE Prime app, as explained *here*.

This library is still actively developed, at the moment two different versions are available that cannot be installed at the same time on a single hub.

- The latest version: 0.1.0 supports communication between a LEGO SPIKE Prime or LEGO MINDSTORMS Robot Inventor hub with a PoweredUP hub. This version is documented on this page.
- The previous version: 0.0.3/0.0.4 supports communication between LEGO SPIKE Prime and LEGO MINDSTORMS Robot Inventor hubs. This version is document here

In a future release functionalities of both versions will be combined.

INDEX

T

W

Y