
Adafruitmotor Library Documentation

Release 1.0

Scott Shawcroft

Jan 20, 2021

Contents

1	Dependencies	3
1.1	Installing from PyPI	3
2	Contributing	5
3	Documentation	7
4	Table of Contents	9
4.1	Simple tests	9
4.2	<code>adafruit_motor.motor</code>	14
4.3	<code>adafruit_motor.servo</code>	15
4.4	<code>adafruit_motor.stepper</code>	16
5	Indices and tables	19
	Python Module Index	21
	Index	23

This helper library provides higher level objects to control motors and servos based on one or more PWM outputs.

The PWM outputs can be any object that have a 16-bit `duty_cycle` attribute. Its assumed that the frequency has already been configured appropriately. (Typically 50hz for servos and 1600hz for motors.)

Dependencies

This driver depends on:

- [Adafruit CircuitPython](#)

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the [Adafruit library and driver bundle](#).

1.1 Installing from PyPI

On supported GNU/Linux systems like the Raspberry Pi, you can install the driver locally [from PyPI](#). To install for current user:

```
pip3 install adafruit-circuitpython-motor
```

To install system-wide (this may be required in some cases):

```
sudo pip3 install adafruit-circuitpython-motor
```

To install in a virtual environment in your current project:

```
mkdir project-name && cd project-name
python3 -m venv .env
source .env/bin/activate
pip3 install adafruit-circuitpython-motor
```


CHAPTER 2

Contributing

Contributions are welcome! Please read our [Code of Conduct](#) before contributing to help this project stay welcoming.

CHAPTER 3

Documentation

For information on building library documentation, please check out [this guide](#).

Table of Contents

4.1 Simple tests

Ensure your device works with this simple test.

Listing 1: examples/motor_servo_sweep_simpletest.py

```
1  # SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
2  # SPDX-License-Identifier: MIT
3
4  import time
5  import board
6  import pulseio
7  from adafruit_motor import servo
8
9  # create a PWMOut object on the control pin.
10 pwm = pulseio.PWMOut(board.D5, duty_cycle=0, frequency=50)
11
12 # To get the full range of the servo you will likely need to adjust the min_pulse and
13 # ↪ max_pulse to
14 # match the stall points of the servo.
15 # This is an example for the Sub-micro servo: https://www.adafruit.com/product/2201
16 # servo = servo.Servo(pwm, min_pulse=580, max_pulse=2350)
17 # This is an example for the Micro Servo - High Powered, High Torque Metal Gear:
18 # https://www.adafruit.com/product/2307
19 # servo = servo.Servo(pwm, min_pulse=500, max_pulse=2600)
20 # This is an example for the Standard servo - TowerPro SG-5010 - 5010:
21 # https://www.adafruit.com/product/155
22 # servo = servo.Servo(pwm, min_pulse=400, max_pulse=2400)
23 # This is an example for the Analog Feedback Servo: https://www.adafruit.com/product/1404
24 # servo = servo.Servo(pwm, min_pulse=600, max_pulse=2500)
25 # This is an example for the Micro servo - TowerPro SG-92R: https://www.adafruit.com/product/169
```

(continues on next page)

(continued from previous page)

```

25 # servo = servo.Servo(pwm, min_pulse=500, max_pulse=2400)
26
27 # The pulse range is 750 - 2250 by default. This range typically gives 135 degrees of
28 # range, but the default is to use 180 degrees. You can specify the expected range if
  →you wish:
29 # servo = servo.Servo(board.D5, actuation_range=135)
30 servo = servo.Servo(pwm)
31
32 # We sleep in the loops to give the servo time to move into position.
33 print("Sweep from 0 to 180")
34 for i in range(180):
35     servo.angle = i
36     time.sleep(0.01)
37 print("Sweep from 180 to 0")
38 for i in range(180):
39     servo.angle = 180 - i
40     time.sleep(0.01)
41
42 print("Move to 90 degrees")
43 servo.angle = 90
44 time.sleep(1)
45 print("Release servo motor for 10 seconds")
46 servo.fraction = None
47 time.sleep(10)
48
49 # You can also specify the movement fractionally.
50 print("Sweep from 0 to 1.0 fractionally")
51 fraction = 0.0
52 while fraction < 1.0:
53     servo.fraction = fraction
54     fraction += 0.01
55     time.sleep(0.01)

```

Listing 2: examples/motor_pca9685_dc_motor.py

```

1 # SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
2 # SPDX-License-Identifier: MIT
3
4 # This example uses an Adafruit Stepper and DC Motor FeatherWing to run a DC Motor.
5 # https://www.adafruit.com/product/2927
6
7 import time
8
9 from board import SCL, SDA
10 import busio
11
12 # Import the PCA9685 module. Available in the bundle and here:
13 # https://github.com/adafruit/Adafruit_CircuitPython_PCA9685
14 from adafruit_pca9685 import PCA9685
15
16 from adafruit_motor import motor
17
18 i2c = busio.I2C(SCL, SDA)
19
20 # Create a simple PCA9685 class instance for the Motor FeatherWing's default address.
21 pca = PCA9685(i2c, address=0x60)

```

(continues on next page)

(continued from previous page)

```

22  pca.frequency = 100
23
24  # Motor 1 is channels 9 and 10 with 8 held high.
25  # Motor 2 is channels 11 and 12 with 13 held high.
26  # Motor 3 is channels 3 and 4 with 2 held high.
27  # Motor 4 is channels 5 and 6 with 7 held high.
28
29  # DC Motors generate electrical noise when running that can reset the microcontroller,
    ↳ in extreme
30  # cases. A capacitor can be used to help prevent this. The demo uses motor 4 because,
    ↳ it worked ok
31  # in testing without a capacitor.
32  # See here for more info: https://learn.adafruit.com/adafruit-motor-shield-v2-for-
    ↳ arduino/faq#faq-13
33  pca.channels[7].duty_cycle = 0xFFFF
34  motor4 = motor.DCMotor(pca.channels[5], pca.channels[6])
35
36  print("Forwards slow")
37  motor4.throttle = 0.5
38  print("throttle:", motor4.throttle)
39  time.sleep(1)
40
41  print("Forwards")
42  motor4.throttle = 1
43  print("throttle:", motor4.throttle)
44  time.sleep(1)
45
46  print("Backwards")
47  motor4.throttle = -1
48  print("throttle:", motor4.throttle)
49  time.sleep(1)
50
51  print("Backwards slow")
52  motor4.throttle = -0.5
53  print("throttle:", motor4.throttle)
54  time.sleep(1)
55
56  print("Stop")
57  motor4.throttle = 0
58  print("throttle:", motor4.throttle)
59  time.sleep(1)
60
61  print("Spin freely")
62  motor4.throttle = None
63  print("throttle:", motor4.throttle)
64
65  pca.deinit()

```

Listing 3: examples/motor_pca9685_stepper_motor.py

```

1  # SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
2  # SPDX-License-Identifier: MIT
3
4  # This example uses an Adafruit Stepper and DC Motor FeatherWing to run a Stepper,
    ↳ Motor.
5  # https://www.adafruit.com/product/2927

```

(continues on next page)

(continued from previous page)

```

6
7 import time
8
9 from board import SCL, SDA
10 import busio
11
12 # Import the PCA9685 module. Available in the bundle and here:
13 # https://github.com/adafruit/Adafruit_CircuitPython_PCA9685
14 from adafruit_pca9685 import PCA9685
15
16 from adafruit_motor import stepper
17
18 i2c = busio.I2C(SCL, SDA)
19
20 # Create a simple PCA9685 class instance for the Motor FeatherWing's default address.
21 pca = PCA9685(i2c, address=0x60)
22 pca.frequency = 1600
23
24 # Motor 1 is channels 9 and 10 with 8 held high.
25 # Motor 2 is channels 11 and 12 with 13 held high.
26 # Motor 3 is channels 3 and 4 with 2 held high.
27 # Motor 4 is channels 5 and 6 with 7 held high.
28
29 pca.channels[7].duty_cycle = 0xFFFF
30 pca.channels[2].duty_cycle = 0xFFFF
31 stepper_motor = stepper.StepperMotor(
32     pca.channels[4], pca.channels[3], pca.channels[5], pca.channels[6]
33 )
34
35 for i in range(100):
36     stepper_motor.onestep()
37     time.sleep(0.01)
38
39 for i in range(100):
40     stepper_motor.onestep(direction=stepper.BACKWARD)
41     time.sleep(0.01)
42
43 pca.deinit()

```

Listing 4: examples/motor_pca9685_servo_sweep.py

```

1 # SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
2 # SPDX-License-Identifier: MIT
3
4 import time
5
6 from board import SCL, SDA
7 import busio
8
9 # Import the PCA9685 module. Available in the bundle and here:
10 # https://github.com/adafruit/Adafruit_CircuitPython_PCA9685
11 from adafruit_pca9685 import PCA9685
12 from adafruit_motor import servo
13
14 i2c = busio.I2C(SCL, SDA)
15

```

(continues on next page)

(continued from previous page)

```

16 # Create a simple PCA9685 class instance.
17 pca = PCA9685(i2c)
18 # You can optionally provide a finer tuned reference clock speed to improve the
19 # ↪accuracy of the
20 # ↪timing pulses. This calibration will be specific to each board and its environment.
21 # ↪See the
22 # ↪calibration.py example in the PCA9685 driver.
23 # ↪pca = PCA9685(i2c, reference_clock_speed=25630710)
24 pca.frequency = 50
25
26 # To get the full range of the servo you will likely need to adjust the min_pulse and
27 # ↪max_pulse to
28 # ↪match the stall points of the servo.
29 # This is an example for the Sub-micro servo: https://www.adafruit.com/product/2201
30 # servo7 = servo.Servo(pca.channels[7], min_pulse=580, max_pulse=2350)
31 # This is an example for the Micro Servo - High Powered, High Torque Metal Gear:
32 # ↪https://www.adafruit.com/product/2307
33 # servo7 = servo.Servo(pca.channels[7], min_pulse=500, max_pulse=2600)
34 # This is an example for the Standard servo - TowerPro SG-5010 - 5010:
35 # ↪https://www.adafruit.com/product/155
36 # servo7 = servo.Servo(pca.channels[7], min_pulse=400, max_pulse=2400)
37 # This is an example for the Analog Feedback Servo: https://www.adafruit.com/product/
38 # ↪1404
39 # servo7 = servo.Servo(pca.channels[7], min_pulse=600, max_pulse=2500)
40 # This is an example for the Micro servo - TowerPro SG-92R: https://www.adafruit.com/
41 # ↪product/169
42 # servo7 = servo.Servo(pca.channels[7], min_pulse=500, max_pulse=2400)
43
44 # The pulse range is 750 - 2250 by default. This range typically gives 135 degrees of
45 # ↪range, but the default is to use 180 degrees. You can specify the expected range if
46 # ↪you wish:
47 # servo7 = servo.Servo(pca.channels[7], actuation_range=135)
48 servo7 = servo.Servo(pca.channels[7])
49
50 # We sleep in the loops to give the servo time to move into position.
51 for i in range(180):
52     servo7.angle = i
53     time.sleep(0.03)
54 for i in range(180):
55     servo7.angle = 180 - i
56     time.sleep(0.03)
57
58 # You can also specify the movement fractionally.
59 fraction = 0.0
60 while fraction < 1.0:
61     servo7.fraction = fraction
62     fraction += 0.01
63     time.sleep(0.03)
64
65 pca.deinit()

```

Listing 5: examples/motor_pca9685_continuous_servo.py

```

1 # SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
2 # SPDX-License-Identifier: MIT
3

```

(continues on next page)

(continued from previous page)

```
4 import time
5
6 from board import SCL, SDA
7 import busio
8
9 # Import the PCA9685 module. Available in the bundle and here:
10 # https://github.com/adafruit/Adafruit_CircuitPython_PCA9685
11 from adafruit_pca9685 import PCA9685
12
13 from adafruit_motor import servo
14
15 i2c = busio.I2C(SCL, SDA)
16
17 # Create a simple PCA9685 class instance.
18 pca = PCA9685(i2c)
19 # You can optionally provide a finer tuned reference clock speed to improve the
20 # accuracy of the
21 # timing pulses. This calibration will be specific to each board and its environment.
22 # See the
23 # calibration.py example in the PCA9685 driver.
24 # pca = PCA9685(i2c, reference_clock_speed=25630710)
25 pca.frequency = 50
26
27 # The pulse range is 750 - 2250 by default.
28 servo7 = servo.ContinuousServo(pca.channels[7])
29 # If your servo doesn't stop once the script is finished you may need to tune the
30 # reference_clock_speed above or the min_pulse and max_pulse timings below.
31 # servo7 = servo.ContinuousServo(pca.channels[7], min_pulse=750, max_pulse=2250)
32
33 print("Forwards")
34 servo7.throttle = 1
35 time.sleep(1)
36
37 print("Backwards")
38 servo7.throttle = -1
39 time.sleep(1)
40
41 print("Stop")
42 servo7.throttle = 0
43
44 pca.deinit()
```

4.2 adafruit_motor.motor

Simple control of a DC motor. DC motors have two wires and should not be connected directly to the PWM connections. Instead use intermediate circuitry to control a much stronger power source with the PWM. The [Adafruit Stepper + DC Motor FeatherWing](#), [Adafruit TB6612 1.2A DC/Stepper Motor Driver Breakout Board](#) and [Adafruit Motor/Stepper/Servo Shield for Arduino v2 Kit - v2.3](#) do this for popular form factors already.

Note: The TB6612 boards feature three inputs XIN1, XIN2 and PWMX. Since we PWM the INs directly its expected that the PWM pin is consistently high.

- Author(s): Scott Shawcroft

class adafruit_motor.motor.DCMotor(*positive_pwm, negative_pwm*)

DC motor driver. *positive_pwm* and *negative_pwm* can be swapped if the motor runs in the opposite direction from what was expected for “forwards”.

Parameters

- **positive_pwm** (*PWMOut*) – The motor input that causes the motor to spin forwards when high and the other is low.
- **negative_pwm** (*PWMOut*) – The motor input that causes the motor to spin backwards when high and the other is low.

throttle

Motor speed, ranging from -1.0 (full speed reverse) to 1.0 (full speed forward), or None. If None, both PWMs are turned full off. If 0.0, both PWMs are turned full on.

4.3 adafruit_motor.servo

Servos are motor based actuators that incorporate a feedback loop into the design. These feedback loops enable pulse width modulated control to determine position or rotational speed.

- Author(s): Scott Shawcroft

class adafruit_motor.servo.ContinuousServo(*pwm_out*, *, *min_pulse=750*,
max_pulse=2250)

Control a continuous rotation servo.

Parameters

- **min_pulse** (*int*) – The minimum pulse width of the servo in microseconds.
- **max_pulse** (*int*) – The maximum pulse width of the servo in microseconds.

throttle

How much power is being delivered to the motor. Values range from -1.0 (full throttle reverse) to 1.0 (full throttle forwards.) 0 will stop the motor from spinning.

class adafruit_motor.servo.Servo(*pwm_out*, *, *actuation_range=180*, *min_pulse=750*,
max_pulse=2250)

Control the position of a servo.

Parameters

- **pwm_out** (*PWMOut*) – PWM output object.
- **actuation_range** (*int*) – The physical range of motion of the servo in degrees, for the given *min_pulse* and *max_pulse* values.
- **min_pulse** (*int*) – The minimum pulse width of the servo in microseconds.
- **max_pulse** (*int*) – The maximum pulse width of the servo in microseconds.

actuation_range is an exposed property and can be changed at any time:

```
servo = Servo(pwm)
servo.actuation_range = 135
```

The specified pulse width range of a servo has historically been 1000-2000us, for a 90 degree range of motion. But nearly all modern servos have a 170-180 degree range, and the pulse widths can go well out of the range to achieve this extended motion. The default values here of 750 and 2250 typically give 135 degrees of motion.

You can set `actuation_range` to correspond to the actual range of motion you observe with your given `min_pulse` and `max_pulse` values.

Warning: You can extend the pulse width above and below these limits to get a wider range of movement. But if you go too low or too high, the servo mechanism may hit the end stops, buzz, and draw extra current as it stalls. Test carefully to find the safe minimum and maximum.

actuation_range

The physical range of motion of the servo in degrees.

angle

The servo angle in degrees. Must be in the range 0 to `actuation_range`. Is `None` when servo is disabled.

4.4 adafruit_motor.stepper

Stepper motors feature multiple wire coils that are used to rotate the magnets connected to the motor shaft in a precise way. Each increment of the motor is called a step. Stepper motors have a varying number of steps per rotation so check the motor's documentation to determine exactly how precise each step is.

- Author(s): Tony DiCola, Scott Shawcroft

`adafruit_motor.stepper.BACKWARD = 2`

“Step backward

`adafruit_motor.stepper.DOUBLE = 2`

Step so that each step only activates two coils to produce more torque.

`adafruit_motor.stepper.FORWARD = 1`

Step forward

`adafruit_motor.stepper.INTERLEAVE = 3`

Step half a step to alternate between single coil and double coil steps.

`adafruit_motor.stepper.MICROSTEP = 4`

Step a fraction of a step by partially activating two neighboring coils. Step size is determined by `microsteps` constructor argument.

`adafruit_motor.stepper.SINGLE = 1`

Step so that each step only activates a single coil

class `adafruit_motor.stepper.StepperMotor` (*ain1, ain2, bin1, bin2, *, microsteps=16*)

A bipolar stepper motor or four coil unipolar motor. The use of microstepping requires pins that can output PWM. For non-microstepping, can set `microsteps` to `None` and use digital out pins.

PWM**Parameters**

- **ain1** (*PWMOut*) – `pwmio.PWMOut`-compatible output connected to the driver for the first coil (unipolar) or first input to first coil (bipolar).
- **ain2** (*PWMOut*) – `pwmio.PWMOut`-compatible output connected to the driver for the third coil (unipolar) or second input to first coil (bipolar).
- **bin1** (*PWMOut*) – `pwmio.PWMOut`-compatible output connected to the driver for the second coil (unipolar) or second input to second coil (bipolar).

- **bin2** (*PWMOut*) – `pwmio.PWMOut`-compatible output connected to the driver for the fourth coil (unipolar) or second input to second coil (bipolar).
- **microsteps** (*int*) – Number of microsteps between full steps. Must be at least 2 and even.

Digital Out

Parameters

- **ain1** (*DigitalInOut*) – `digitalio.DigitalInOut`-compatible output connected to the driver for the first coil (unipolar) or first input to first coil (bipolar).
- **ain2** (*DigitalInOut*) – `digitalio.DigitalInOut`-compatible output connected to the driver for the third coil (unipolar) or second input to first coil (bipolar).
- **bin1** (*DigitalInOut*) – `digitalio.DigitalInOut`-compatible output connected to the driver for the second coil (unipolar) or first input to second coil (bipolar).
- **bin2** (*DigitalInOut*) – `digitalio.DigitalInOut`-compatible output connected to the driver for the fourth coil (unipolar) or second input to second coil (bipolar).
- **microsteps** – set to `None`

onestep (*, *direction=1*, *style=1*)

Performs one step of a particular style. The actual rotation amount will vary by style. *SINGLE* and *DOUBLE* will normal cause a full step rotation. *INTERLEAVE* will normally do a half step rotation. *MICROSTEP* will perform the smallest configured step.

When step styles are mixed, subsequent *SINGLE*, *DOUBLE* or *INTERLEAVE* steps may be less than normal in order to align to the desired style's pattern.

Parameters

- **direction** (*int*) – Either *FORWARD* or *BACKWARD*
- **style** (*int*) – *SINGLE*, *DOUBLE*, *INTERLEAVE*

release ()

Releases all the coils so the motor can free spin, also won't use any power

CHAPTER 5

Indices and tables

- `genindex`
- `modindex`
- `search`

a

`adafruit_motor.motor`, [14](#)
`adafruit_motor.servo`, [15](#)
`adafruit_motor.stepper`, [16](#)

A

`actuation_range` (*adafruit_motor.servo.Servo attribute*), 16
`adafruit_motor.motor` (*module*), 14
`adafruit_motor.servo` (*module*), 15
`adafruit_motor.stepper` (*module*), 16
`angle` (*adafruit_motor.servo.Servo attribute*), 16

B

`BACKWARD` (*in module adafruit_motor.stepper*), 16

C

`ContinuousServo` (*class in adafruit_motor.servo*), 15

D

`DCMotor` (*class in adafruit_motor.motor*), 14
`DOUBLE` (*in module adafruit_motor.stepper*), 16

F

`FORWARD` (*in module adafruit_motor.stepper*), 16

I

`INTERLEAVE` (*in module adafruit_motor.stepper*), 16

M

`MICROSTEP` (*in module adafruit_motor.stepper*), 16

O

`onestep()` (*adafruit_motor.stepper.StepperMotor method*), 17

R

`release()` (*adafruit_motor.stepper.StepperMotor method*), 17

S

`Servo` (*class in adafruit_motor.servo*), 15
`SINGLE` (*in module adafruit_motor.stepper*), 16

`StepperMotor` (*class in adafruit_motor.stepper*), 16

T

`throttle` (*adafruit_motor.motor.DCMotor attribute*), 15
`throttle` (*adafruit_motor.servo.ContinuousServo attribute*), 15