Adafruitmotor Library Documentation

Release 1.0

Scott Shawcroft

Jul 09, 2020

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 `adafruit_motor.motor` . . . 14
- 4.3 `adafruit_motor.servo` . . . 15
- 4.4 `adafruit_motor.stepper` . . . 16

5 Indices and tables **17**

Python Module Index **19**

Index **21**

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.)

CHAPTER 1

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.

CHAPTER 4

Table of Contents

4.1 Simple tests

Ensure your device works with this simple test.

Listing 1: examples/motor_servo_sweep_simpletest.py

```
import time
import board
import pulseio
from adafruit_motor import servo

# create a PWMOut object on the control pin.
pwm = pulseio.PWMOut(board.D5, duty_cycle=0, frequency=50)

# To get the full range of the servo you will likely need to adjust the min_pulse and␣
↪max_pulse to
# match the stall points of the servo.
# This is an example for the Sub-micro servo: https://www.adafruit.com/product/2201
# servo = servo.Servo(pwm, min_pulse=580, max_pulse=2350)
# This is an example for the Micro Servo - High Powered, High Torque Metal Gear:
#   https://www.adafruit.com/product/2307
# servo = servo.Servo(pwm, min_pulse=500, max_pulse=2600)
# This is an example for the Standard servo - TowerPro SG-5010 - 5010:
#   https://www.adafruit.com/product/155
# servo = servo.Servo(pwm, min_pulse=400, max_pulse=2400)
# This is an example for the Analog Feedback Servo: https://www.adafruit.com/product/
↪1404
# servo = servo.Servo(pwm, min_pulse=600, max_pulse=2500)
# This is an example for the Micro servo - TowerPro SG-92R: https://www.adafruit.com/
↪product/169
# servo = servo.Servo(pwm, min_pulse=500, max_pulse=2400)

# The pulse range is 750 - 2250 by default. This range typically gives 135 degrees of
```

(continues on next page)

(continued from previous page)

```
# range, but the default is to use 180 degrees. You can specify the expected range if you wish:
# servo = servo.Servo(board.D5, actuation_range=135)
servo = servo.Servo(pwm)

# We sleep in the loops to give the servo time to move into position.
print("Sweep from 0 to 180")
for i in range(180):
    servo.angle = i
    time.sleep(0.01)
print("Sweep from 180 to 0")
for i in range(180):
    servo.angle = 180 - i
    time.sleep(0.01)

print("Move to 90 degrees")
servo.angle = 90
time.sleep(1)
print("Release servo motor for 10 seconds")
servo.fraction = None
time.sleep(10)

# You can also specify the movement fractionally.
print("Sweep from 0 to 1.0 fractionally")
fraction = 0.0
while fraction < 1.0:
    servo.fraction = fraction
    fraction += 0.01
    time.sleep(0.01)
```

Listing 2: examples/motor_pca9685_dc_motor.py

```
# This example uses an Adafruit Stepper and DC Motor FeatherWing to run a DC Motor.
#   https://www.adafruit.com/product/2927

import time

from board import SCL, SDA
import busio

# Import the PCA9685 module. Available in the bundle and here:
#   https://github.com/adafruit/Adafruit_CircuitPython_PCA9685
from adafruit_pca9685 import PCA9685

from adafruit_motor import motor

i2c = busio.I2C(SCL, SDA)

# Create a simple PCA9685 class instance for the Motor FeatherWing's default address.
pca = PCA9685(i2c, address=0x60)
pca.frequency = 100

# Motor 1 is channels 9 and 10 with 8 held high.
# Motor 2 is channels 11 and 12 with 13 held high.
# Motor 3 is channels 3 and 4 with 2 held high.
# Motor 4 is channels 5 and 6 with 7 held high.
```

(continues on next page)

(continued from previous page)

```

# DC Motors generate electrical noise when running that can reset the microcontroller in extreme
# cases. A capacitor can be used to help prevent this. The demo uses motor 4 because it worked ok
# in testing without a capacitor.
# See here for more info: https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/faq#faq-13
pca.channels[7].duty_cycle = 0xFFFF
motor4 = motor.DCMotor(pca.channels[5], pca.channels[6])

print("Forwards slow")
motor4.throttle = 0.5
print("throttle:", motor4.throttle)
time.sleep(1)

print("Forwards")
motor4.throttle = 1
print("throttle:", motor4.throttle)
time.sleep(1)

print("Backwards")
motor4.throttle = -1
print("throttle:", motor4.throttle)
time.sleep(1)

print("Backwards slow")
motor4.throttle = -0.5
print("throttle:", motor4.throttle)
time.sleep(1)

print("Stop")
motor4.throttle = 0
print("throttle:", motor4.throttle)
time.sleep(1)

print("Spin freely")
motor4.throttle = None
print("throttle:", motor4.throttle)

pca.deinit()
```

Listing 3: examples/motor_pca9685_stepper_motor.py

```
# This example uses an Adafruit Stepper and DC Motor FeatherWing to run a Stepper Motor.
#   https://www.adafruit.com/product/2927

import time

from board import SCL, SDA
import busio

# Import the PCA9685 module. Available in the bundle and here:
#   https://github.com/adafruit/Adafruit_CircuitPython_PCA9685
from adafruit_pca9685 import PCA9685
```

(continues on next page)

(continued from previous page)

```

from adafruit_motor import stepper

i2c = busio.I2C(SCL, SDA)

# Create a simple PCA9685 class instance for the Motor FeatherWing's default address.
pca = PCA9685(i2c, address=0x60)
pca.frequency = 1600

# Motor 1 is channels 9 and 10 with 8 held high.
# Motor 2 is channels 11 and 12 with 13 held high.
# Motor 3 is channels 3 and 4 with 2 held high.
# Motor 4 is channels 5 and 6 with 7 held high.

pca.channels[7].duty_cycle = 0xFFFF
pca.channels[2].duty_cycle = 0xFFFF
stepper_motor = stepper.StepperMotor(
    pca.channels[4], pca.channels[3], pca.channels[5], pca.channels[6]
)

for i in range(100):
    stepper_motor.onestep()
    time.sleep(0.01)

for i in range(100):
    stepper_motor.onestep(direction=stepper.BACKWARD)
    time.sleep(0.01)

pca.deinit()
```

Listing 4: examples/motor_pca9685_servo_sweep.py

```
import time

from board import SCL, SDA
import busio

# Import the PCA9685 module. Available in the bundle and here:
#   https://github.com/adafruit/Adafruit_CircuitPython_PCA9685
from adafruit_pca9685 import PCA9685
from adafruit_motor import servo

i2c = busio.I2C(SCL, SDA)

# Create a simple PCA9685 class instance.
pca = PCA9685(i2c)
# You can optionally provide a finer tuned reference clock speed to improve the␣
↪accuracy of the
# timing pulses. This calibration will be specific to each board and its environment.␣
↪See the
# calibration.py example in the PCA9685 driver.
# pca = PCA9685(i2c, reference_clock_speed=25630710)
pca.frequency = 50

# To get the full range of the servo you will likely need to adjust the min_pulse and␣
↪max_pulse to
```

(continues on next page)

(continued from previous page)

```
# match the stall points of the servo.
# This is an example for the Sub-micro servo: https://www.adafruit.com/product/2201
# servo7 = servo.Servo(pca.channels[7], min_pulse=580, max_pulse=2350)
# This is an example for the Micro Servo - High Powered, High Torque Metal Gear:
#   https://www.adafruit.com/product/2307
# servo7 = servo.Servo(pca.channels[7], min_pulse=500, max_pulse=2600)
# This is an example for the Standard servo - TowerPro SG-5010 - 5010:
#   https://www.adafruit.com/product/155
# servo7 = servo.Servo(pca.channels[7], min_pulse=400, max_pulse=2400)
# This is an example for the Analog Feedback Servo: https://www.adafruit.com/product/
↪1404
# servo7 = servo.Servo(pca.channels[7], min_pulse=600, max_pulse=2500)
# This is an example for the Micro servo - TowerPro SG-92R: https://www.adafruit.com/
↪product/169
# servo7 = servo.Servo(pca.channels[7], min_pulse=500, max_pulse=2400)

# The pulse range is 750 - 2250 by default. This range typically gives 135 degrees of
# range, but the default is to use 180 degrees. You can specify the expected range if␣
↪you wish:
# servo7 = servo.Servo(pca.channels[7], actuation_range=135)
servo7 = servo.Servo(pca.channels[7])

# We sleep in the loops to give the servo time to move into position.
for i in range(180):
    servo7.angle = i
    time.sleep(0.03)
for i in range(180):
    servo7.angle = 180 - i
    time.sleep(0.03)

# You can also specify the movement fractionally.
fraction = 0.0
while fraction < 1.0:
    servo7.fraction = fraction
    fraction += 0.01
    time.sleep(0.03)

pca.deinit()
```

Listing 5: examples/motor_pca9685_continuous_servo.py

```
import time

from board import SCL, SDA
import busio

# Import the PCA9685 module. Available in the bundle and here:
#   https://github.com/adafruit/Adafruit_CircuitPython_PCA9685
from adafruit_pca9685 import PCA9685

from adafruit_motor import servo

i2c = busio.I2C(SCL, SDA)

# Create a simple PCA9685 class instance.
pca = PCA9685(i2c)
```

(continues on next page)

(continued from previous page)

```
# You can optionally provide a finer tuned reference clock speed to improve the accuracy of the
# timing pulses. This calibration will be specific to each board and its environment. See the
# calibration.py example in the PCA9685 driver.
# pca = PCA9685(i2c, reference_clock_speed=25630710)
pca.frequency = 50

# The pulse range is 750 - 2250 by default.
servo7 = servo.ContinuousServo(pca.channels[7])
# If your servo doesn't stop once the script is finished you may need to tune the
# reference_clock_speed above or the min_pulse and max_pulse timings below.
# servo7 = servo.ContinuousServo(pca.channels[7], min_pulse=750, max_pulse=2250)

print("Forwards")
servo7.throttle = 1
time.sleep(1)

print("Backwards")
servo7.throttle = -1
time.sleep(1)

print("Stop")
servo7.throttle = 0

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.

Parameters

- **ain1** (*PWMOut*) – `pulseio.PWMOut`-compatible output connected to the driver for the first coil (unipolar) or first input to first coil (bipolar).
- **ain2** (*PWMOut*) – `pulseio.PWMOut`-compatible output connected to the driver for the third coil (unipolar) or second input to first coil (bipolar).
- **bin1** (*PWMOut*) – `pulseio.PWMOut`-compatible output connected to the driver for the second coil (unipolar) or second input to second coil (bipolar).
- **bin2** (*PWMOut*) – `pulseio.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.

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

Python Module Index

a

adafruit_motor.motor, 14
adafruit_motor.servo, 14
adafruit_motor.stepper, 15

Index

A

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

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*), 16

R

release() (*adafruit_motor.stepper.StepperMotor method*), 16

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*), 14
throttle (*adafruit_motor.servo.ContinuousServo attribute*), 15