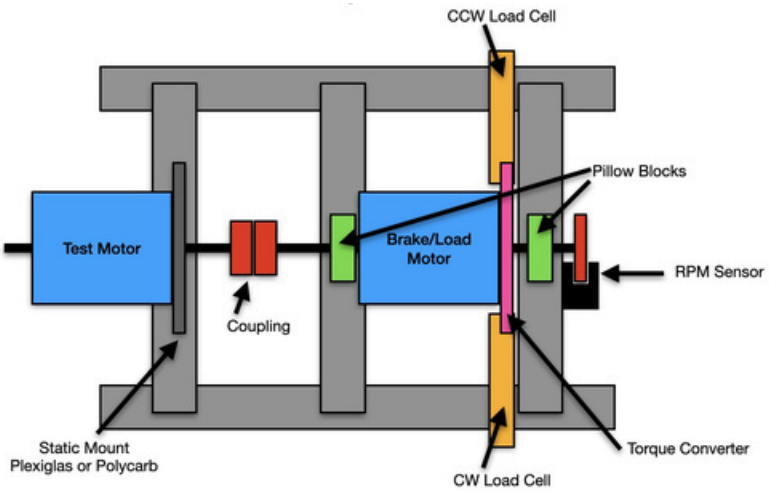
adafruit learning system

Improve Brushed DC Motor Performance

Created by Jan Goolsbey

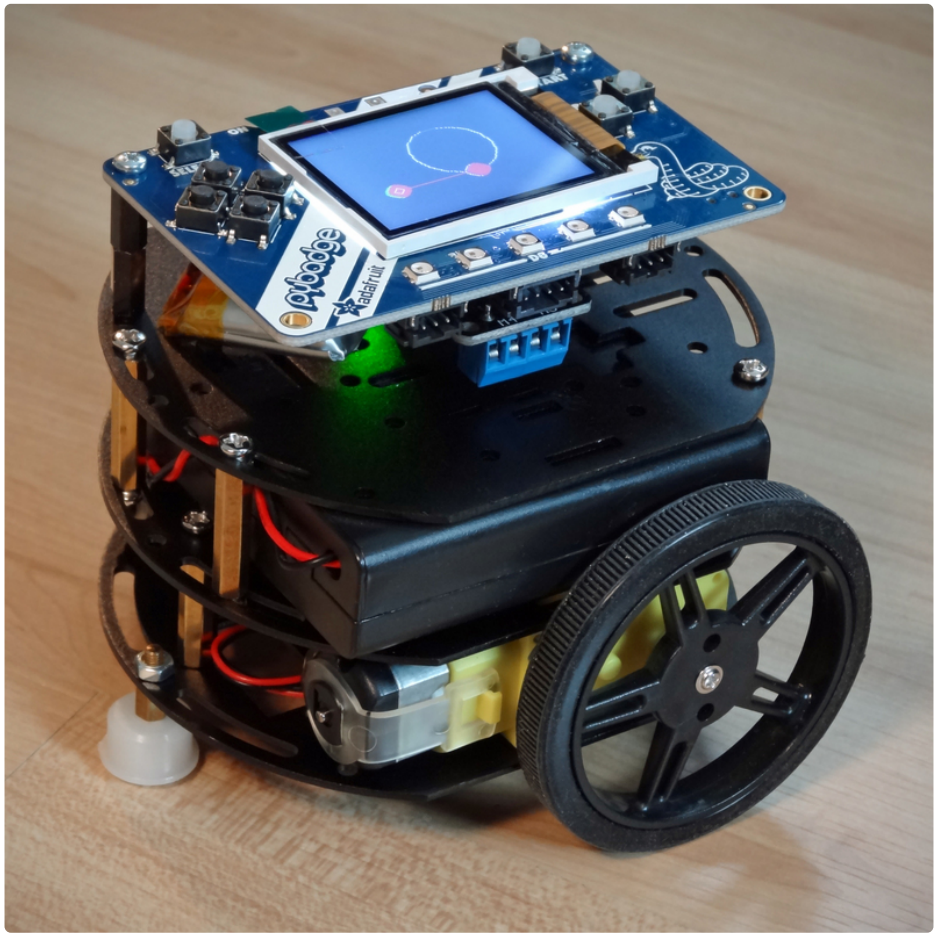


Last updated on 2021-10-22 11:46:29 AM EDT

Guide Contents

Overview

Changing the motor controller's mode of operation and finding an optimal PWM frequency can dramatically improve the control and performance of brushed DC motors. Using CircuitPython, set your motor controller's decay mode from fast to slow and select a PWM frequency below 100Hz for enhanced results.



Jump to the **Code Examples** section to see how it's done.

Works with:

- CRICKIT FeatherWing (https://adafru.it/Cxy)
- CRICKIT for Circuit Playground Express (https://adafru.it/Biy)
- DC Motor + Stepper FeatherWing (https://adafru.it/QAt)
- Motor/Stepper/Servo Shield (https://adafru.it/QAu)
- DRV8871 DC Motor Driver Breakout (https://adafru.it/QAv)
- DRV8833 DC/Stepper Motor Driver Breakout (https://adafru.it/QAw)
- TB6612 1.2A DC/Stepper Motor Driver Breakout (https://adafru.it/sdc)
- L9110H H-Bridge Motor Driver (https://adafru.it/QBK)

- L293D Dual H-Bridge Motor Driver (https://adafru.it/QBL)

Supporting code libraries:

- Adafruit Crickit for CircuitPython (https://adafru.it/QCk)
- Adafruit Motor for CircuitPython (https://adafru.it/BNE)
- Adafruit MotorKit for CircuitPython (https://adafru.it/QBM)
- Adafruit MotorShield for Arduino (https://adafru.it/QBN)

Parts

Assembled DC Motor + Stepper FeatherWing Add-on

A Feather board without ambition is a Feather board without FeatherWings! This is the Fully assembled (with headers) DC Motor + Stepper FeatherWing which will let...

$21.50

In Stock

Add to Cart

Adafruit Motor/Stepper/Servo Shield for Arduino v2 Kit

The original Adafruit Motorshield kit is one of our most beloved kits, which is why we decided to make something even better. We have upgraded the shield kit to make the bestest,...

$19.95

In Stock

Add to Cart

Adafruit CRICKIT FeatherWing for any Feather

Sometimes we wonder if robotics engineers ever watch movies. If they did, they'd know that making robots into slaves always ends up in a robot rebellion. Why even go down that...

$29.95

In Stock

Add to Cart

Adafruit CRICKIT for Circuit Playground Express

Sometimes we wonder if robotics engineers ever watch movies. If they did, they'd know that making robots into slaves always ends up in a robot rebellion. Why even go down that...

$29.95

In Stock

Add to Cart

Your browser does not support the video tag.

Adafruit DRV8833 DC/Stepper Motor Driver Breakout Board

Spin two DC motors or step one bi-polar or uni-polar stepper with up to 1.2A per channel using the DRV8833. This motor driver chip is a

nice alternative to the TB6612 driver. Like that...

$4.95

In Stock

Add to Cart

Adafruit DRV8871 DC Motor Driver Breakout Board - 3.6A Max

Crank up your robotics with powerful Adafruit DRV8871 motor driver breakout board. This motor driver has a lot of great specs that make it useful for a wide variety of...

Out of Stock

Out of Stock

Your browser does not support the video tag.

Adafruit TB6612 1.2A DC/Stepper Motor Driver Breakout Board

Spin two DC motors, step one bi-polar or uni-polar stepper, or fire off two solenoids with 1.2A per channel using the TB6612. These are perhaps better known as "

$4.95

In Stock

Add to Cart

Dual H-Bridge Motor Driver for DC or Steppers - 600mA - L293D

Run four solenoids, two DC motors, or one bi-polar or uni-polar stepper with up to 600mA per channel using the L293D. These are perhaps better known as "the drivers in our...

$4.50

In Stock

Add to Cart

Your browser does not support the video tag.

L9110H H-Bridge Motor Driver for DC Motors - 8 DIP

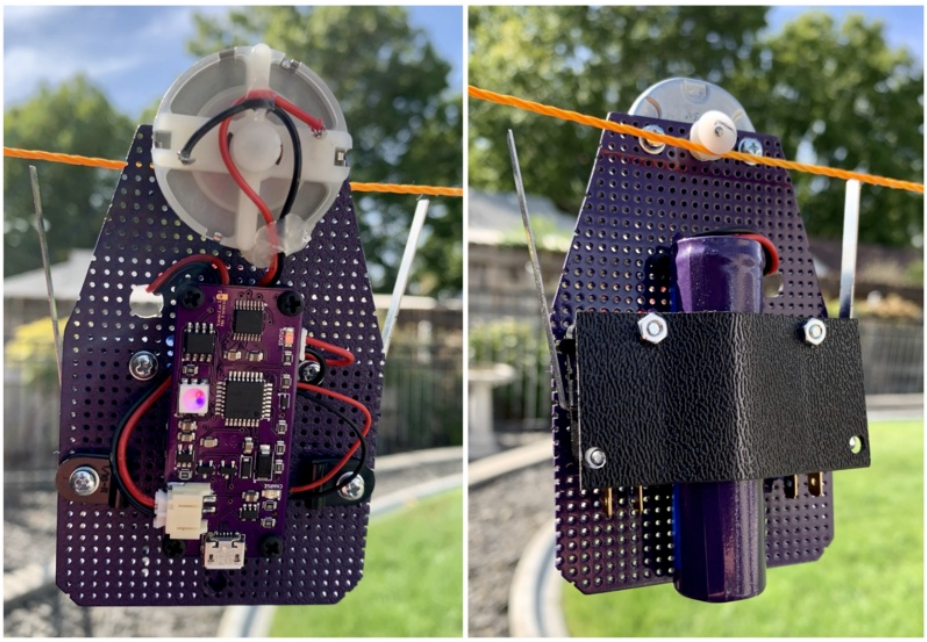
Run two solenoids or a single DC motor with up to 800mA per channel using the super-simple L9110H H-bridge driver. This bridge chip is an 8 DIP package so it's easy to fit onto any...

$1.50

In Stock

Add to Cart

Introduction



The StringCar racer robot was bouncing off the string when accelerating and braking. At lower speeds, the precariously balanced car's pulley dramatically changed speed and disengaged from the string without warning. To avoid sudden starts and stops, the car's CircuitPython code gradually ramped the motor voltage up and down using PWM (Pulse-Width Modulation) signals, so why was the car so jumpy?

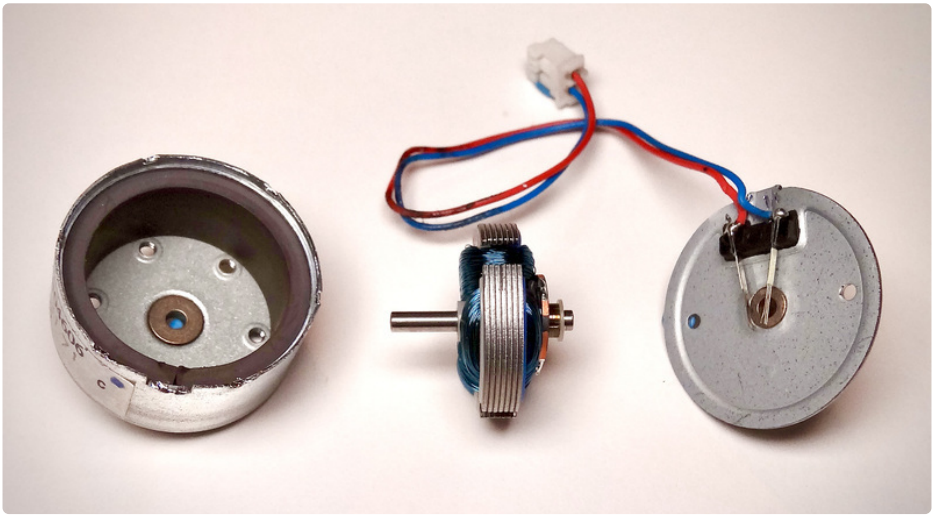
It became obvious that the car's brushed direct-current (DC) motor didn't have enough torque to run reliably at speeds less than 200RPM without some sort of gearing arrangement. A geared motor for the StringCar was too heavy and difficult to balance, so another solution was needed. And if that wasn't enough, the speed control throttle setting wasn't linearly related to actual motor speed making it difficult to measure string length based solely on motor run time.

After some experimentation, the solution was to adjust two PWM signal parameters. No need to add a gearbox or change hardware; we'll just apply some simple changes to the CircuitPython code.

This guide will elaborate on the solution, examine some commonly available motors, and provide programming examples that will improve the performance of your robot's brushed DC motors. First, we'll look at the motors' love/hate relationship with PWM and how to make them friends again.

A Special Acknowledgment: Thank you to Chris Parrott (@ZodiusInfuser) for insight and subsequent discussions leading to including current decay mode as an important factor to consider for brushed DC motor control.

PWM and Brushed DC Motors



Brushed DC motors have an affinity for direct current, engineered to spin in proportion to the level and polarity of the applied DC voltage. For example, a miniature 6-volt DC motor runs at its full rated speed when supplied with power from four AA batteries (four times 1.5 volts equals 6 volts). The motor will run slower with three AA batteries (4.5 volts) or even slower with two AA batteries (3 volts). When the speed of a robot's DC motor needs to be controlled by software, swapping batteries in and out just won't do.

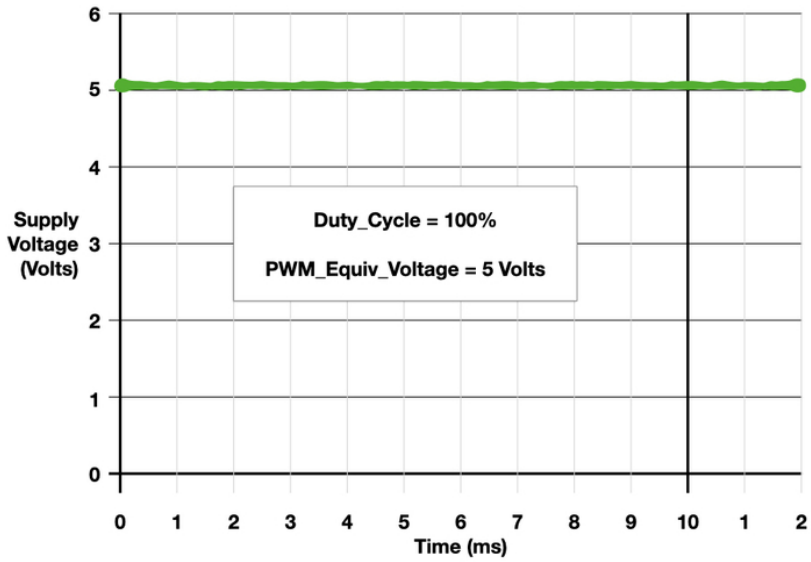
To control the motor with software like CircuitPython, a special signal called Pulse Width Modulation (PWM) is used. Microcontrollers such as the Feather M4 Express send software-controlled PWM signals to an external breakout board in order to regulate motor speed. To cause the motor to spin, the microcontroller furnishes the external motor controller board a pulsing signal which in turn sends a high-power pulse to the attached motor. The PWM signals control the motor’s direction, speed, and available torque.

A typical PWM signal used for a brushed DC motor consists of three primary characteristics, duty cycle, decay mode, and frequency. Let’s get to know each aspect of the PWM signal better.

Duty Cycle

Duty Cycle is the ratio of the full-power pulse's duration to the entire PWM interval period, usually expressed as a percentage. PWM Equivalent Voltage is the product of the power supply voltage times the Duty Cycle divided by 100.

The width of each PWM pulse is set by the program code to adjust the amount of energy for the controller board to send to the attached motor. A pulse with a long duration imparts more energy to the motor, increasing the speed of the motor. A short duration pulse reduces the available energy and the motor spins more slowly. The motor sees changes in pulse energy just like when batteries are added or removed. With a little math, the pulse energy can be expressed as an “equivalent voltage” similar to battery voltage.



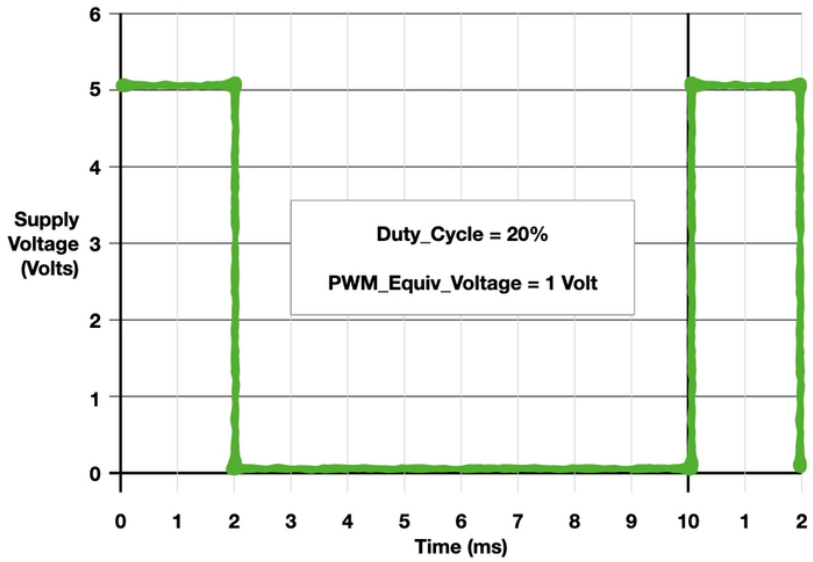
When a controller is sending the full voltage of the power source to the motor, the motor sees a PWM signal with a *duty cycle* of 100%. For example, if the controller output is always at the power supply voltage level, the duty cycle is 100%; if at full voltage for 5 milliseconds (ms) during a 10ms interval period , the duty cycle calculates to 50%. A full voltage pulse for 2ms during the 10ms period has a duty cycle of 20%.

PWM equivalent voltage is equal to the power supply voltage times the duty cycle. If the power source is 5 volts, a duty cycle of 100% has an equivalent voltage of:

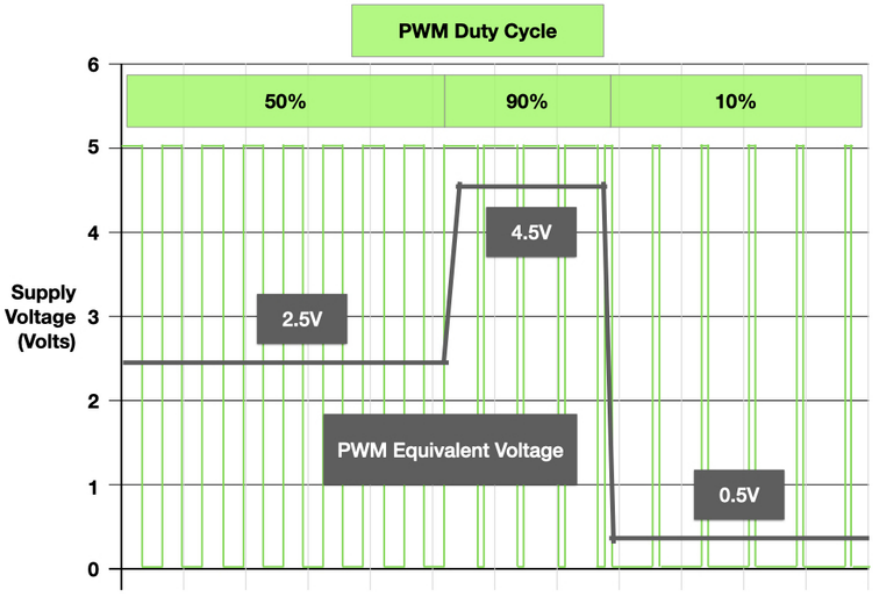
```
5v * (100% / 100) = 5 volts
```

A duty cycle of 20% produces the equivalent voltage:

```
5v * (20% / 100) = 1.0 volts
```



As the PWM duty cycle changes, the motor reacts to the equivalent voltage and spins the motor at a speed that is proportional to that value. A lower duty cycle slows the motor; a higher duty cycle increases motor speed.

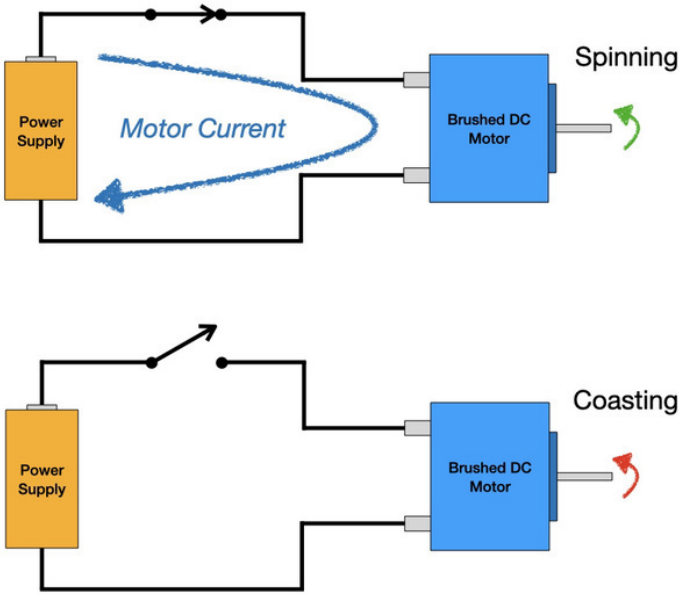


In CircuitPython's motor control libraries, the motor's PWM duty cycle ratio is usually called the motor's *throttle*, expressed as a numeric value between 0 and 1.0 where a value of 0 stops the motor and 1.0 runs the motor at full speed. Forward motor direction is a positive throttle value (0 to +1.0). Reverse direction is a negative value (0 to -1.0).

While duty cycle controls the motor's speed, the controller's motor current decay mode and the PWM signal's frequency can dramatically effect the efficiency of a brushed DC motor, particularly when the PWM duty cycle is less than 30%. Why does the decay mode and PWM frequency play such an important role? Next we'll look at the how motor recirculation current decay mode can help control motor performance.

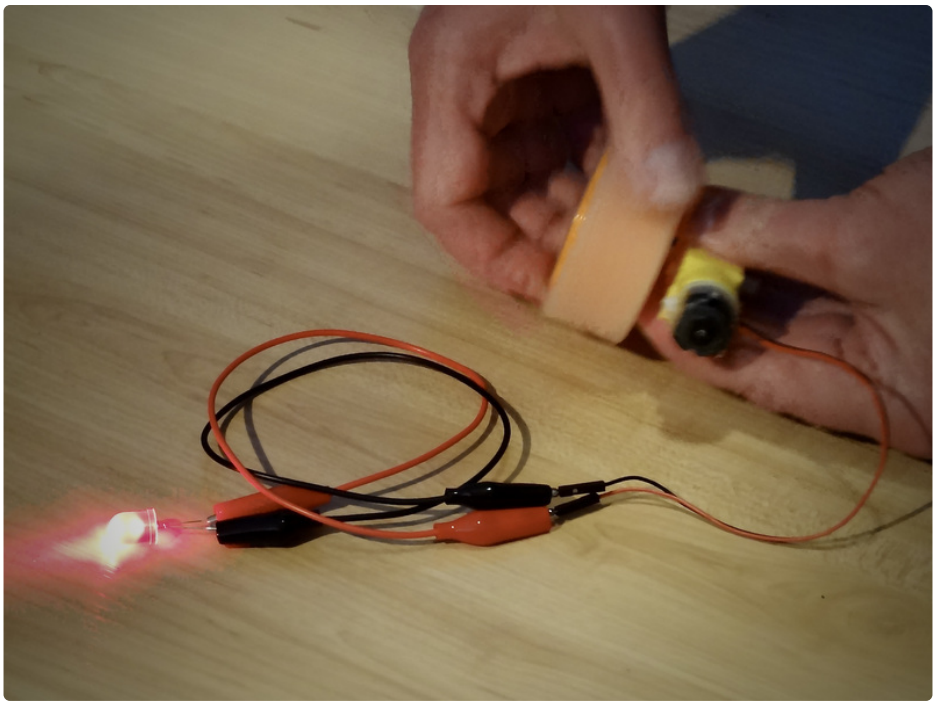
Decay Mode

Motor Recirculation Current Decay Mode is a special behavior of modern motor controller circuitry. Decay mode can be specified in CircuitPython code as either Slow Decay or Fast Decay.

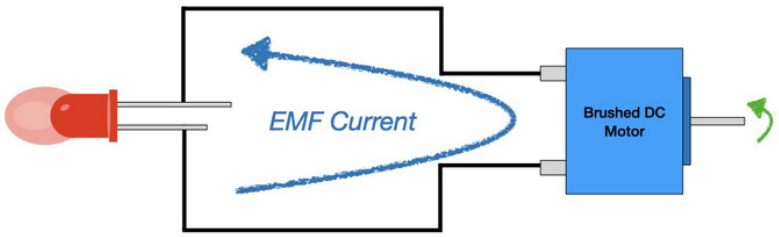


At a minimum, a simple on-off switch can be used to control a motor. When the switch is on, the power supply voltage is applied to the motor causing it to spin. Turn the switch off and the motor abruptly loses power and coasts to a stop. Current flow within the motor rises to the maximum when the switch is on and suddenly drops to zero when the switch is turned off. When a motor controller circuit acts like this switch, it is said to be operating the motor in **Fast Decay Mode** because current to the motor quickly decays. This mode may also be called **Coasting Mode** since the motor is allowed to coast when power is removed.

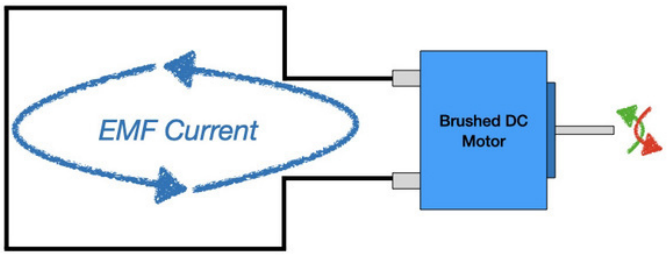
In many cases, such as when our robot nears the edge of a cliff, we'd like more control over the motor than just allowing it to coast to a stop; we'd like to apply the brakes. Another control mode, **Slow Decay Mode**, can be used to quickly stop a spinning motor and provide more precision of the motor's movement. Slow decay mode improves control and braking by taking advantage of a secondary characteristic of a brushed DC motor — its ability to act as a generator. Let's try an experiment to understand how this works.



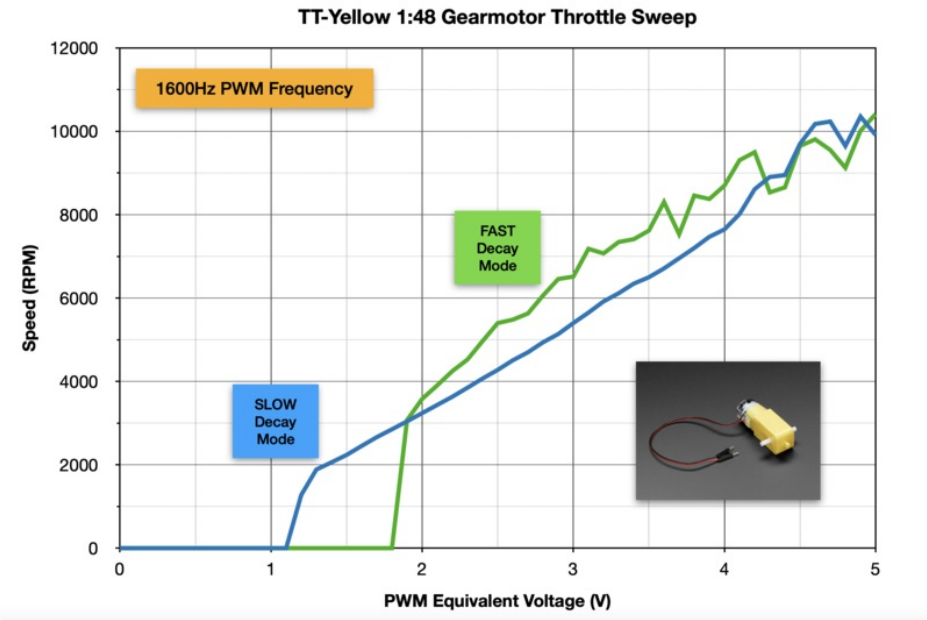
Choose one of your favorite DC motors and an LED from the parts bin. A Yellow-TT motor (https://adafru.it/BzE)and a red or green LED will work nicely. Connect the LED to the motor terminals and give the shaft a quick spin then again in the opposite direction. What did you see? A spinning motor creates electricity in proportion to its speed and direction. This phenomenon is called EMF (electro-motive force) and can be measured in volts.



A DC motor creates electricity even when coasting to a stop. If the motor terminals of a coasting motor are connected to each other, the generated EMF will return the power to the motor causing it to try to spin in the opposite direction (back EMF). The result is a rapid slow down of the motor speed, akin to brakes. Very handy for stopping on a dime.



Motor controller breakouts such as the DRV8833 apply active braking when operating in slow decay mode. It's called slow decay since the motor continues to operate using the current supplied by the motor itself; the motor current doesn't suddenly disappear. Some controller chip vendors also call this **Braking Mode**.



The Yellow-TT motor (https://adafru.it/BzE)'s spin threshold decreases to 1200 RPM when operating in slow decay mode compared to 3000 RPM for fast decay mode. That means that the output shaft of the 1:48 gearbox turns the attached wheel at 25 RPM versus 63 RPM; forward speed drops to 8.5 cm/sec from 21.4 cm/sec.

Also note that the speed versus motor voltage curve for slow decay (blue line) is more linear than fast decay (green line). The linear relationship between speed and voltage simplifies calculating motor speed from the throttle value.

```
equivalent_voltage = power_supply * throttle
```

`motor_speed = (2500 * equivalent_voltage) - 2000`

`gearbox_speed = motor_speed / 48`

With a 5-volt power supply, the motor and gearbox output shaft speeds for a throttle setting of 0.5 are 4250 RPM and 88.5 RPM.

`equivalent_voltage = 5 * 0.5 = 2.5`

`motor_speed = (2500 * 2.5) - 2000 = 4250`

`gearbox_speed = 4250 / 48 = 88.5`

Decay mode terminology is confusing. Remember that the decay mode describes how quickly the motor recirculation current dissipates, not its effect on motor speed. A motor's rotational speed drops more quickly when using slow decay mode (braking) as compared to fast decay (coasting).

Selecting the proper current decay mode for your project will go a long way to fine-tuning required brushed DC motor performance. One other PWM parameter, frequency, is useful for increasing low-speed torque and lowering the throttle value needed to start the motor spinning.

PWM Frequency

PWM Frequency is the count of PWM interval periods per second, expressed in Hertz (Hz). Mathematically, the frequency is equal to the inverse of the interval period's length (PWM_Frequency = 1 / PWM_Interval_Period).

When calculating the PWM Equivalent Voltage, we generally assume that the motor will operate ideally and respond as if it was connected to a non-PWM power source providing the voltage. But that's not the case. For example, a Yellow-TT motor (https://adafru.it/BzE) will easily spin if a single 1.5-volt battery is connected, but will not turn until the PWM Equivalent Voltage coming from a Motor FeatherWing reaches 2.0 volts when operating in fast decay mode. And when it does start, it suddenly rotates at 4000 RPM. Why is that?

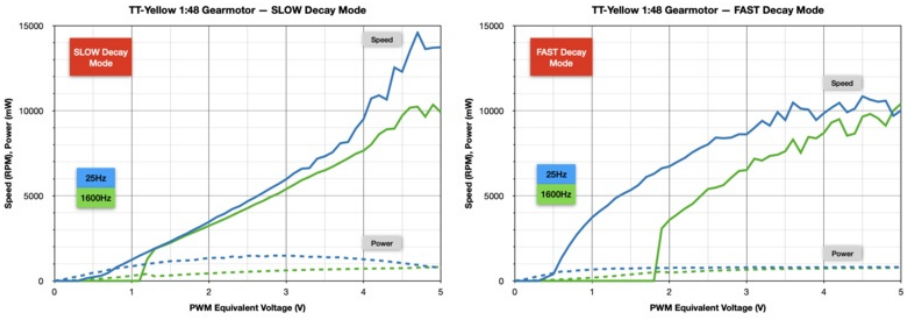
Since a brushed DC motor's internal rotor consists of two or more coils of wire wound around laminated magnetic core material, the motor acts like an inductor. Depending on size of the rotor coil, it may take a few milliseconds for the energy to build sufficiently to turn the shaft.

Inductors are electromagnetic components that capture energy from the incremental buildup of the magnetic field created by an electrical current passing through a wire coil.

Rotor coil inductance becomes an important factor to consider when using PWM for motor speed control. The motor coil works best when the applied voltage is relatively steady since it needs time for its

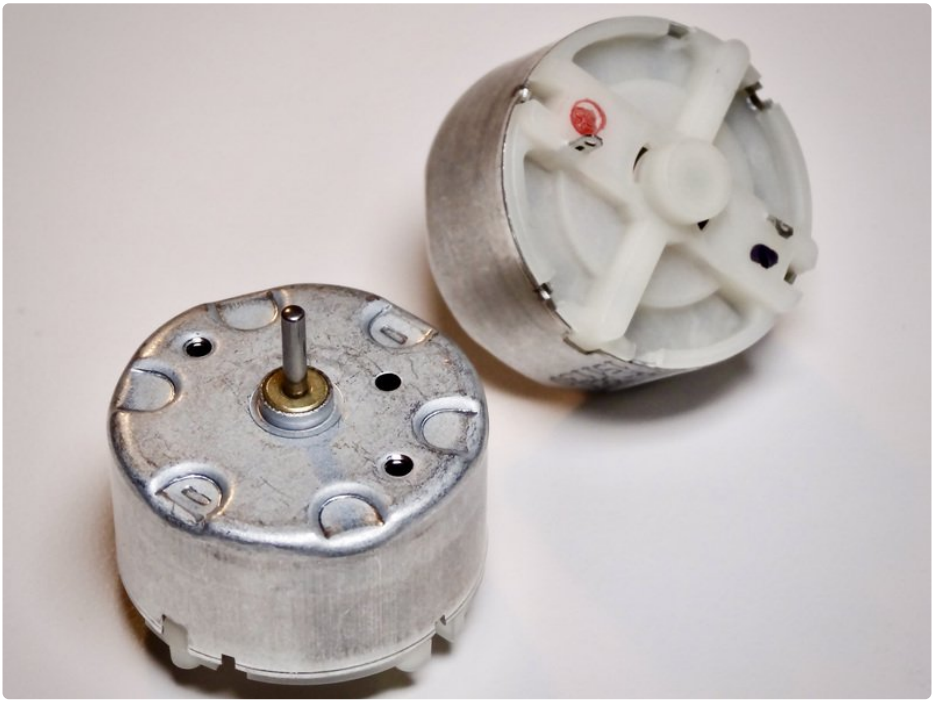
magnetic field to reach the needed strength. At higher PWM frequencies, the pulses from the motor controller board change too quickly to provide enough energy to spin the motor until the equivalent voltage reaches 2.0 volts, although switching to using slow decay mode can help.

When the PWM frequency is lowered, the motor's coils extract more energy from the pulsed PWM signal. That means that the motor will start spinning at a lower equivalent voltage and will operate with improved torque at low speeds. The following graphs compare the Yellow-TT motor's speed response when the default PWM frequency of 1600Hz is changed to 25Hz.



The spin threshold at 25Hz starts at 0.5 volts or less depending on the decay mode selection, increasing the useable motor speed range to as low as 100 RPM. The Yellow-TT gearbox reduces the motor's RPM by a factor of 48, so the attached wheel will be turning at 2 RPM or about 0.7 cm/sec. A velocity like that will make it much easier for your robot to sneak up on the cat.

So now that we know about current decay mode and PWM frequency, how do we go about choosing the best configuration of the two parameters for our robot's motors?



Slow decay mode is usually selected for controlling brushed DC motors because of its ability to dynamically apply braking. It also helps to linearize the relationship between duty cycle and motor speed. Let's compare the two modes.

Motor Recirculation Current Decay Mode	Low RPM Spin Threshold	Throttle Linearity	Dynamic Braking	PWM Frequency Tolerance	Relative Power Consumption
Fast (Coasting)	Poor	Poor	Poor	Poor	Low
Slow (Braking)	Good	Good	Good	Good	High

If low-speed operation is required, slow decay mode increases available torque and significantly improves the overall speed range of the motor. Choosing slow decay also provides a linear mapping of the software throttle setting to motor speed which is helpful when measuring velocity and distance traveled.

Slow decay mode appears to be superior in most categories, but isn't the best choice if power consumption is your primary concern. For some battery-powered robots, the power saved using fast decay mode outweighs the more precise control offered by slow current decay.

Fast decay mode is the default mode in CircuitPython motor control libraries. Changing the operational

mode parameter to slow decay can be accomplished with a simple, one-line statement. See the **Code Examples** section for the details.

After changing the mode, test the operation to confirm that the lowest required motor speed is working as needed. If a lower spin threshold is required, try reducing the PWM frequency to the lowest possible value (usually about 25Hz) and work up towards the maximum of the motor controller. Choose the frequency that provides the best balance of torque throughout the desired speed range while balancing the motor chatter that can happen at lower frequencies.

As a rule of thumb, most small brushed DC motors will operate nicely with a PWM frequency of 50Hz to 100Hz and slow decay mode. Projects like the StringCar Racer that don't use gearbox motors seem to work best at 25Hz.

Motor Controller Comparison

Motor Controller	Adafruit PID	Decay Mode		PWM Frequency		
		SLOW Decay	FAST Decay	Lowest	Highest	CircuitPython Library Default
Crickit FeatherWing		X	X	3 Hz	720 Hz	50 Hz
Crickit for Circuit Playground Express		X	X	3 Hz	720 Hz	50 Hz
DC Motor + Stepper FeatherWing		X	X	24 Hz	2100 Hz	1600 Hz
Motor/Stepper/Servo Shield V2		X	X	24 Hz	2100 Hz	1600 Hz
DRV8833 DC/Stepper Motor Driver Breakout		X	X	1 Hz	50 kHz	500 Hz
DRV8871 DC Motor Driver Breakout		X	X	1 Hz	50 kHz	500 Hz
TB6612 DC/Stepper Motor Driver Breakout		X	X	1 Hz	50 kHz	500 Hz
L293D Dual H-Bridge Motor Driver		X		1 Hz	50 kHz	500 Hz
L9110H H-Bridge Motor Driver		X		1 Hz	50 kHz	500 Hz

In combination with its CircuitPython library, a motor controller board will offer a range of selectable PWM frequencies as well as decay mode. The table above shows the available frequency range and mode support for a variety of Adafruit motor controllers. Refer to the **Code Examples** section for how to select a specific decay mode and PWM frequency for your motor controller board.

CircuitPython Code Examples

The following CircuitPython code examples will sweep an attached motor's duty cycle from 0% to 100% in 2% increments in one direction. Each step of the duty cycle will hold for 1 second. The motor stops at the end of the code. During execution, the code prints the duty cycle as a throttle value for display in the REPL's Serial and Plotter windows.

The first portion of each example imports the needed libraries and instantiates the motor controller with a new PWM frequency. Change the frequency value to select the one that works best for the attached motor.

Each motor controller is implemented differently, so refer to the example's code description to change the controller's frequency. The examples describe the frequency setting techniques when using:

- Crickit FeatherWing and Crickit for Circuit Playground Express
- Motor FeatherWing and Motor Shield
- Breakout Boards and H-Bridge Chips

Crickit FeatherWing and Crickit for Circuit Playground Express

Upon initiation, the Crickit library sets the DC motor PWM frequency to 50Hz, useful for most brushed DC motors like the Yellow-TT motor (https://adafru.it/BzE). It is possible to override the default if a custom PWM frequency is needed while still preserving the ability to use the Crickit library's motor statements such as `motor.throttle`.

The Crickit library also initially sets the controller's decay mode to FAST_DECAY. We'll change the decay mode to SLOW_DECAY to take advantage of the resultant performance improvement.

The Crickit uses two internal GPIO pins for each of the two DC motor controllers. The internal pins reserved for the motor controllers are 18, 19, 22, and 23. This is the section of the example code that changes the PWM to a custom value:

```
PWM_FREQ = 25  # Custom PWM frequency; Crickit min/max 3Hz/720Hz, default is 50Hz
DECAY_MODE = motor.SLOW_DECAY  # Set controller to Slow Decay (braking) mode
THROTTLE_HOLD = 1  # Hold the throttle (seconds)

motor1 = crickit.dc_motor_1
motor2 = crickit.dc_motor_2

motor1.decay_mode = DECAY_MODE
motor2.decay_mode = DECAY_MODE

ss = crickit.seesaw  # To access Seesaw motor pins
ss.set_pwm_freq(22, PWM_FREQ)  # Set motor1A pin to custom PWM frequency
ss.set_pwm_freq(23, PWM_FREQ)  # Set motor1B pin to custom PWM frequency
ss.set_pwm_freq(19, PWM_FREQ)  # Set motor2A pin to custom PWM frequency
ss.set_pwm_freq(18, PWM_FREQ)  # Set motor2B pin to custom PWM frequency
```

Line 1 defines the constant used for the custom PWM frequency value, followed by a constant that defines the controller's decay mode. A throttle holding constant (`THROTTLE_HOLD`) of 1 second is used later in the example code. The next lines define names for the two Crickit motor controllers. Defining the names also instantiates the motor pin definitions that will be changed.

In lines 8 and 9, the decay mode is set for each motor instance using the decay mode constant defined earlier. Next we'll access each motor controller pin individually and reset the default PWM frequency. For example, the first pin of DC_Motor_1's controller is updated in line 12:

`ss.set_pwm_freq(22, PWM_FREQ)`

The PWM frequency of each pin is then set to the custom value stored in the `PWM_FREQ` constant.

Example code for the Crickit FeatherWing and Circuit Playground Express Crickit:

```
# SPDX-FileCopyrightText: 2021 Jan Goolsbey for Adafruit Industries
# SPDX-License-Identifier: MIT

# Crickit PWM Frequency Example
# for Adafruit Crickit FeatherWing (#3343)
# and Circuit Playground Express Crickit(#3093)

import time
from adafruit_motor import motor
from adafruit_crickit import crickit

PWM_FREQ = 25  # Custom PWM frequency; Crickit min/max 3Hz/720Hz, default is 50Hz
DECAY_MODE = motor.SLOW_DECAY  # Set controller to Slow Decay (braking) mode
THROTTLE_HOLD = 1  # Hold the throttle (seconds)

motor1 = crickit.dc_motor_1
motor2 = crickit.dc_motor_2

motor1.decay_mode = DECAY_MODE
motor2.decay_mode = DECAY_MODE

ss = crickit.seesaw  # To access Seesaw motor pins
ss.set_pwm_freq(22, PWM_FREQ)  # Set motor1A pin to custom PWM frequency
ss.set_pwm_freq(23, PWM_FREQ)  # Set motor1B pin to custom PWM frequency
ss.set_pwm_freq(19, PWM_FREQ)  # Set motor2A pin to custom PWM frequency
ss.set_pwm_freq(18, PWM_FREQ)  # Set motor2B pin to custom PWM frequency
print("PWM frequency:", PWM_FREQ)  # Display internal PWM frequency

motor1.throttle = 0  # Stop motor1
motor2.throttle = 0  # Stop motor2
print((0,))  # Plot/print current throttle value
time.sleep(THROTTLE_HOLD)  # Hold at current throttle value

# Sweep up through 50 duty cycle values
for duty_cycle in range(0, 101, 2):
    throttle = duty_cycle / 100  # Convert to throttle value (0 to 1.0)
    motor1.throttle = throttle
    motor2.throttle = throttle
    print((throttle,))  # Plot/print current throttle value
    time.sleep(THROTTLE_HOLD)  # Hold at current throttle value

motor1.throttle = 0  # Stop motor1
motor2.throttle = 0  # Stop motor2
print((0,))  # Plot/print current throttle value
time.sleep(THROTTLE_HOLD)  # Hold at current throttle value
```

Motor FeatherWing and MotorShield

Upon instantiation, the `adafruit_motorkit` `MotorKit` class sets the DC motor PWM frequency to a default of 1600Hz and the controller's decay mode to FAST_DECAY. Changing the frequency and decay mode to a custom value is accomplished using a `MotorKit` parameter after instantiation.

```
PWM_FREQ = 25  # Custom PWM frequency; MotorKit min/maz 24Hz/2100Hz, default is 1600Hz
DECAY_MODE = motor.SLOW_DECAY  # Set controller to Slow Decay (braking) mode
THROTTLE_HOLD = 1  # Hold the throttle (seconds)

# Instantiate motor controller wing
motorwing = MotorKit(i2c=board.I2C(), address=0x60)
motorwing.frequency = PWM_FREQ  # Set custom PWM frequency
print("PWM frequency:", motorwing.frequency)  # Verify internal PWM frequency
motor1 = motorwing.motor1
motor1.decay_mode = DECAY_MODE
```

The first line defines the constant used for the custom PWM frequency value. A constant value for the decay mode is defined as SLOW_DECAY, followed by a throttle holding constant (`THROTTLE_HOLD`) that's used later in the example code. Line 6 is the typical statement used to instantiate the Motor FeatherWing or Shield. Line 7 updates the `MotorKit` PWM frequency parameter with the custom value stored in the `PWM_FREQ` constant.

After the motor is instantiated in line 9, the controller's decay mode is set to SLOW_DECAY.

Example code for Motor FeatherWing and MotorShield:

```
# SPDX-FileCopyrightText: 2021 Jan Goolsbey for Adafruit Industries
# SPDX-License-Identifier: MIT

# Motor FeatherWing PWM Frequency Example
# for Adafruit Motor FeatherWing (#3243) and Motor Shield (#1438)

import time
import board
from adafruit_motor import motor
from adafruit_motorkit import MotorKit

PWM_FREQ = 25  # Custom PWM frequency; MotorKit min/maz 24Hz/2100Hz, default is 1600Hz
DECAY_MODE = motor.SLOW_DECAY  # Set controller to Slow Decay (braking) mode
THROTTLE_HOLD = 1  # Hold the throttle (seconds)

# Instantiate motor controller wing
motorwing = MotorKit(i2c=board.I2C(), address=0x60)
motorwing.frequency = PWM_FREQ  # Set custom PWM frequency
print("PWM frequency:", motorwing.frequency)  # Verify internal PWM frequency
motor1 = motorwing.motor1
motor1.decay_mode = DECAY_MODE

motor1.throttle = 0  # Stop motor1
print((0,))  # Plot/print current throttle value
time.sleep(THROTTLE_HOLD)  # Hold at current throttle value

# Sweep up through 50 duty cycle values
for duty_cycle in range(0, 101, 2):
    throttle = duty_cycle / 100  # Convert to throttle value (0 to 1.0)
    motor1.throttle = throttle
    print((throttle,))  # Plot/print current throttle value
    time.sleep(THROTTLE_HOLD)  # Hold at current throttle value

motor1.throttle = 0  # Stop motor1
print((0,))  # Plot/print current throttle value
time.sleep(THROTTLE_HOLD)  # Hold at current throttle value
```

Breakout Boards and H-Bridge Chips

The `adafruit_motor` library's `DCMotor` class is the one to use with Adafruit motor controller breakout boards and H-bridge integrated circuits. Upon instantiation, the library works with the `pwmio.PWMOut` class to define which GPIO pins to use to drive the motor, set the 500Hz default PWM frequency and FAST_DECAY controller mode. We'll change the default frequency when the GPIO pins are defined and will change the controller's decay mode to SLOW_DECAY.

Refer to your selected breakout board's or H-bridge chip's learning guide for how to properly connect it to a microcontroller.

```
PWM_FREQ = 25  # Custom PWM frequency in Hz; PWMOut min/max 1Hz/50kHz, default is 500Hz
DECAY_MODE = motor.SLOW_DECAY  # Set controller to Slow Decay (braking) mode
THROTTLE_HOLD = 1  # Hold the throttle (seconds)

# DC motor setup; Set pins to custom PWM frequency
pwm_a = pwmio.PWMOut(PWM_PIN_A, frequency=PWM_FREQ)
pwm_b = pwmio.PWMOut(PWM_PIN_B, frequency=PWM_FREQ)
motor1 = motor.DCMotor(pwm_a, pwm_b)
motor1.decay_mode = DECAY_MODE
```

The first line defines the constant used for the custom PWM frequency value. Next, a constant value for the controller's mode is defined as SLOW_DECAY, followed by a throttle holding constant (`THROTTLE_HOLD`) that's used later in the example code. The two GPIO pins needed for the motor controller are defined next. Choose two pins that have their own unused PWM channel.

Lines 6 and 7 use `pwmio.PWMOut` to establish the pins and set the custom PWM frequency. After the motor is instantiated as `motor1` in line 8, the controller's decay mode is set to SLOW_DECAY.

Example code for breakout motor controller boards:

```
# SPDX-FileCopyrightText: 2021 Jan Goolsbey for Adafruit Industries
# SPDX-License-Identifier: MIT

# Breakout PWM Frequency Example
# for Adafruit motor controller breakout boards and H-bridge drivers
# TB6612 (#2448), DRV8833 (#3297), DRV8871 (#3190), L9110 (#4489), L293D (#807)

import time
import board
import pwmio
from adafruit_motor import motor

PWM_PIN_A = board.D5  # Pick two PWM pins on their own channels
PWM_PIN_B = board.D6
PWM_FREQ = 25  # Custom PWM frequency in Hz; PWMOut min/max 1Hz/50kHz, default is 500Hz
DECAY_MODE = motor.SLOW_DECAY  # Set controller to Slow Decay (braking) mode
THROTTLE_HOLD = 1  # Hold the throttle (seconds)

# DC motor setup; Set pins to custom PWM frequency
pwm_a = pwmio.PWMOut(PWM_PIN_A, frequency=PWM_FREQ)
pwm_b = pwmio.PWMOut(PWM_PIN_B, frequency=PWM_FREQ)
motor1 = motor.DCMotor(pwm_a, pwm_b)
motor1.decay_mode = DECAY_MODE

motor1.throttle = 0  # Stop motor1
print((0,))  # Plot/print current throttle value

# Sweep up through 50 duty cycle values
for duty_cycle in range(0, 101, 2):
    throttle = duty_cycle / 100  # Convert to throttle value (0 to 1.0)
    motor1.throttle = throttle
    print((throttle,))  # Plot/print current throttle value
    time.sleep(1)  # Hold at current throttle value

motor1.throttle = 0  # Stop motor1
print((0,))  # Plot/print current throttle value
```

Arduino Code Example

Using the Adafruit_Motor_Shield_V2_Library

To adjust the PWM frequency of the Motor Shield or Motor FeatherWing, refer to the Arduino code example for the Adafruit_Motor_Shield_V2_Library:

DC Motor Test Example (https://adafru.it/QCm)

Using this library, the frequency can be changed from the default 1600Hz by including the new value in the `begin` statement:

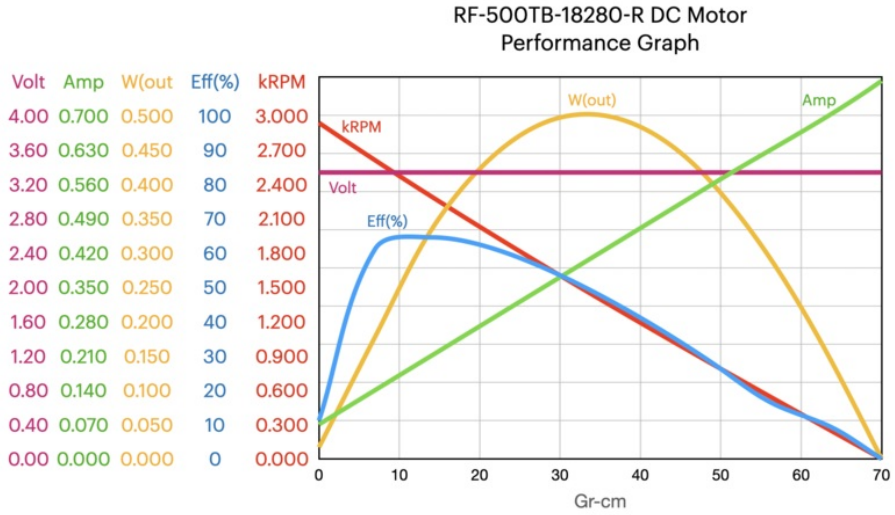
```
AFMS.begin(100); // Set the PWM frequency to 100Hz
```

Note that at this time, the library does not provide a parameter for changing the default controller decay mode from FAST_DECAY to SLOW_DECAY.

Using the `analogWrite()` Function

The PWM frequency of microcontroller GPIO pins using the `analogWrite()` function is fixed and cannot be changed. PWM signals sent directly to a motor controller breakout from GPIO pins will default to a frequency of 490Hz to 1000Hz depending on the microcontroller board used. See Arduino Reference: analogWrite() (https://adafru.it/QCn) for more information.

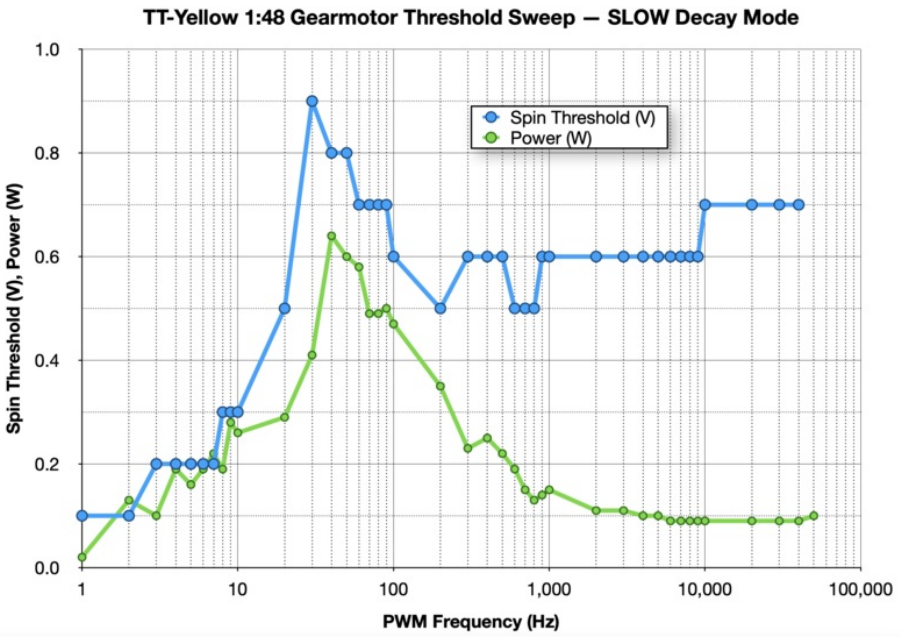
Measuring Motor Performance



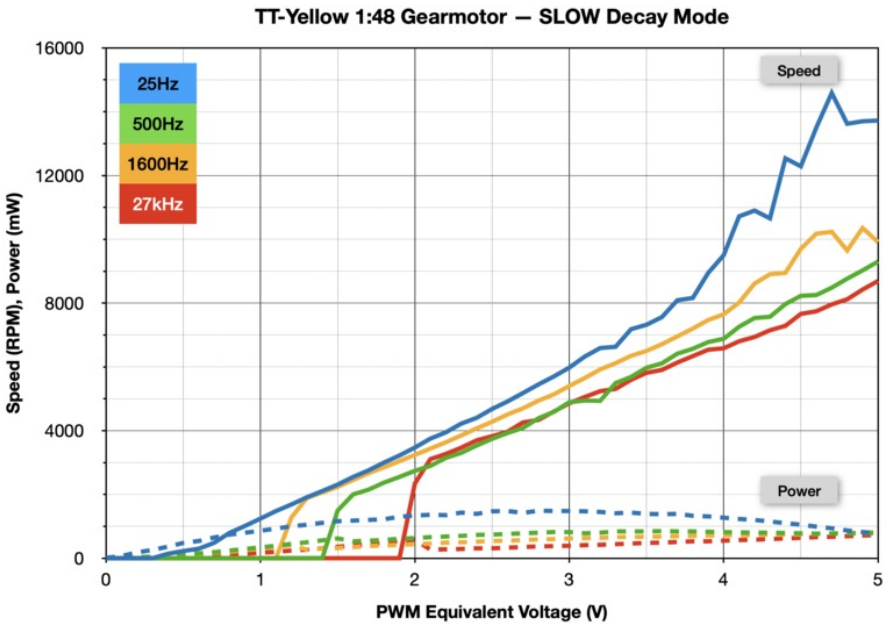
Motor manufactures' data sheets provide excellent performance information about a motor's optimum operating parameters such as voltage and current at the motor's rated speed. Most manufacturers also provide tables and graphs that show how the motor responds to different voltages and the corresponding changes to RPM, torque, and power. However, very little information is available about how a particular motor responds to PWM signals.

We already know that decay mode and PWM duty cycle can control a motor's speed, but to understand what happens when the mode is changed or frequency is varied, we need a way to measure the motor's response to these parameters. If we measure the motor's performance, picking the best-performing decay mode and PWM frequency is straightforward.

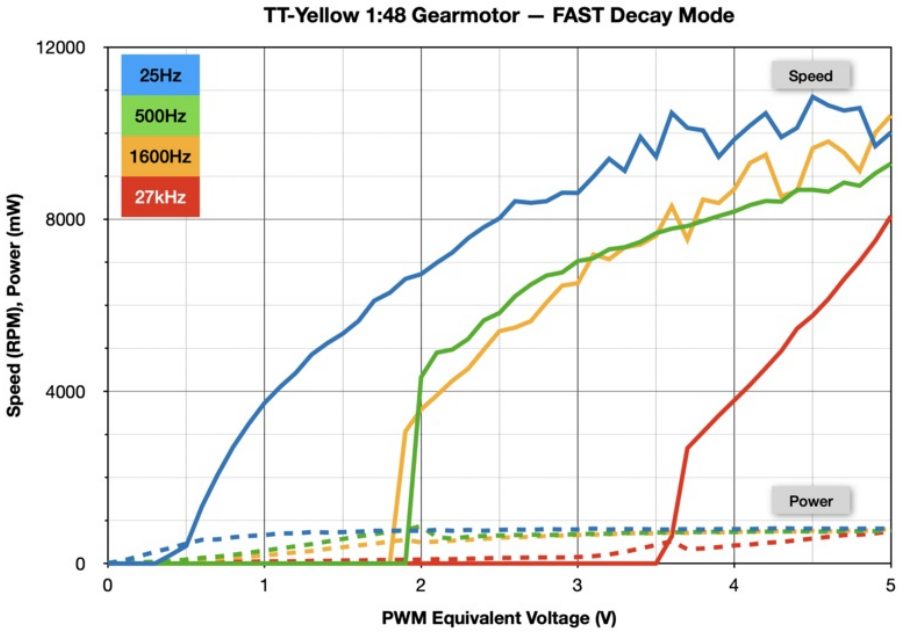
For example, the spin threshold frequency response spectrum of the Yellow-TT motor shows the motor voltage when spin threshold is reached (the blue line) as the PWM frequency is swept from 1Hz to 50kHz. At a PWM frequency of 20Hz, the motor begins to spin at 0.5 volts. As the frequency increases, the spin threshold changes eventually settling at approximately 0.6 volts. If low speed is your robot's thing, a PWM Frequency of 20Hz may be ideal -- as long as some motor chatter is tolerable. If smoother operation is needed, consider trying 400Hz. The chatter will decrease while maintaining a fairly low spin threshold. Selecting a lower frequency usually means that the motor's spin threshold torque will be relatively higher, as well.



The PWM frequency also determines the motor's speed at the spin threshold. The following chart compares a few PWM frequencies used to control a Yellow-TT motor. At 25Hz, the motor starts spinning at 200 RPM when the motor voltage is 0.5 volts. When the frequency is 500Hz, the motor spins at 1500 RPM when the spin threshold voltage of 1.5 volts is applied. Keep this in mind when choosing a frequency, especially if a low motor speeds is needed.



The motor speed curve noticeably changes when fast current decay mode is chosen instead of slow decay.



Test the decay mode and frequency you selected with the motor attached to your robot and operating with its normal power supply source. Remember that PWM frequencies of 60Hz or lower can cause mechanical vibration depending on the motor and the quality of its gearbox. Adjust the PWM frequency for the best balance of vibration and low-speed operation.

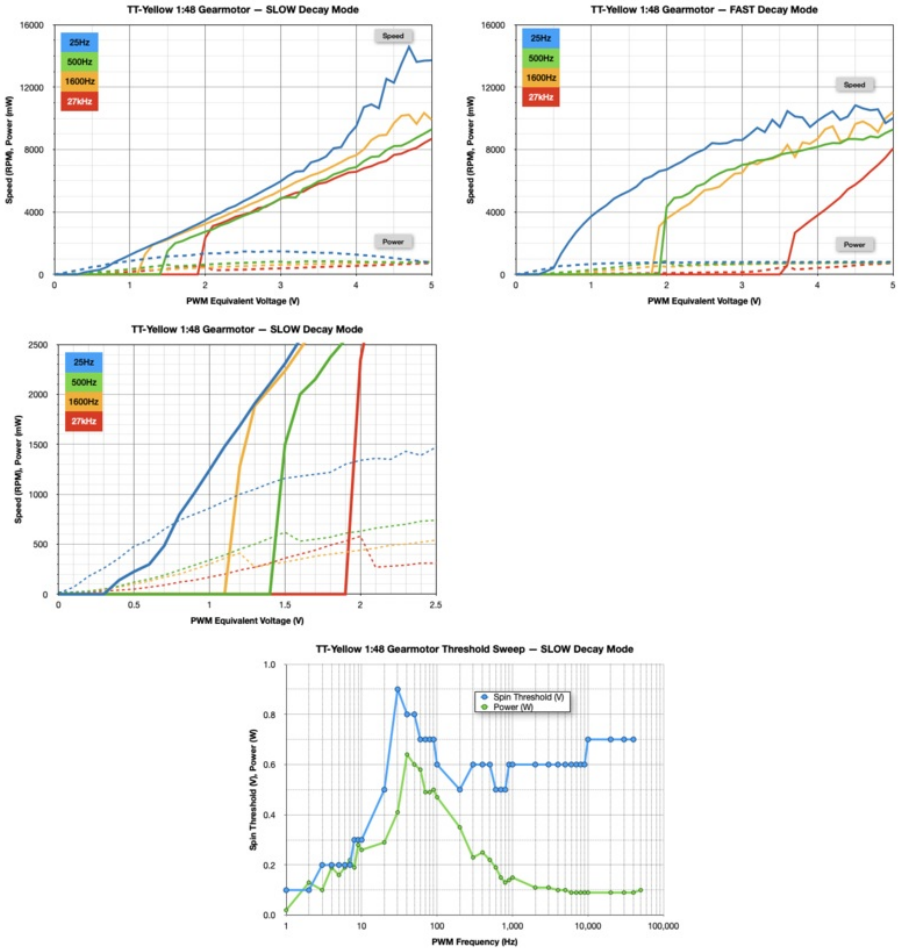
Motor performance charts for a variety of popular brushed DC motors can be found in the **Motor Performance Charts** section.

Motor Performance Charts

After characterizing a few brushed DC motors from the workshop inventory, the results concluded that a slow decay mode in combination with a low PWM frequency will improve most motors' low-speed performance and overall speed range. All motors saw a significant reduction of spin threshold voltage when changing the PWM frequency from the MotorKit library's default of 1600Hz to a more compatible 25Hz.

The following charts show each motor's speed, power, and spin threshold characteristics measured by the automated motor tester as it swept through decay modes, PWM Equivalent Voltage values, and PWM frequencies. Click on the chart images for an enlarged view.

Yellow TT Motor; 1:48 Gear Ratio



N20 DC Motor with Encoder; 1:50 Gear Ratio

Your browser does not support the video tag.

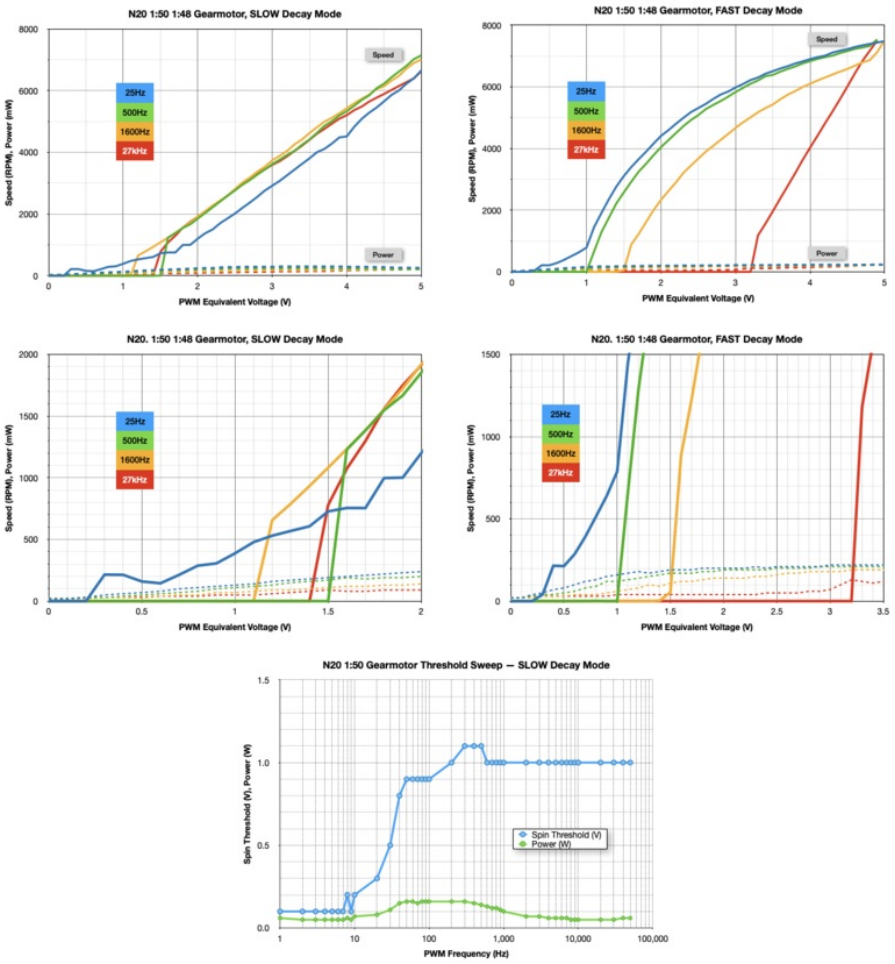
N20 DC Motor with Magnetic Encoder - 6V with 1:50 Gear Ratio

The first step in a robotics project is to get a motor spinning. Once you've done that you quickly learn that not all motors go the same speed, even if they are the same part...

$12.50

In Stock

Add to Cart



N20 DC Motor with Encoder; 1:298 Gear Ratio

Your browser does not support the video tag.

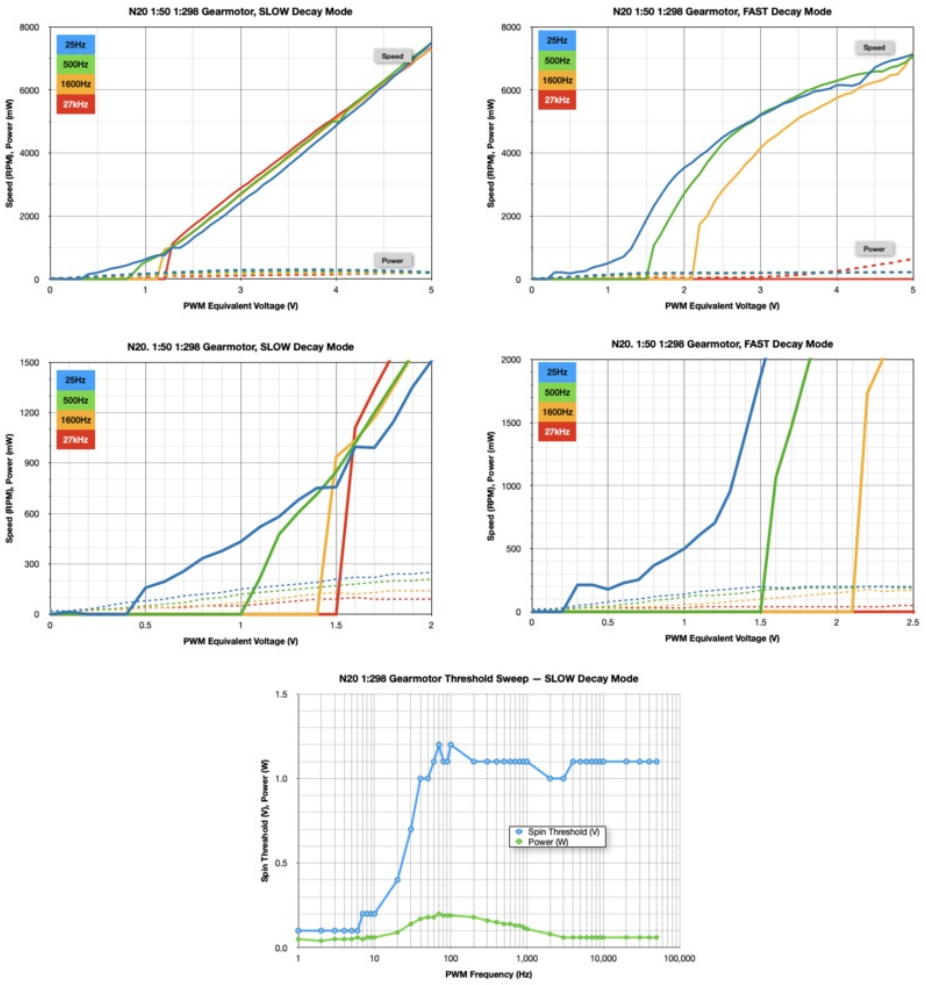
N20 DC Motor with Magnetic Encoder - 6V with 1:298 Gear Ratio

The first step in a robotics project is to get a motor spinning. Once you've done that you quickly learn that not all motors go the same speed, even if they are the same part...

$12.50

In Stock

Add to Cart



Geared DC Motor with Encoder; 1:20 Gear Ratio

Your browser does not support the video tag.

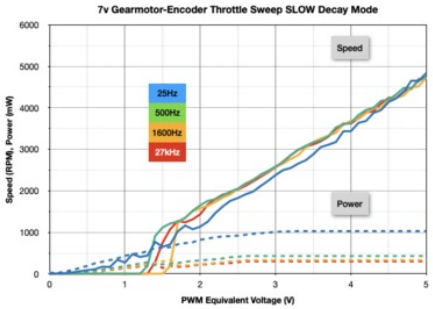
Geared DC Motor with Magnetic Encoder Outputs - 7 VDC 1:20 Ratio

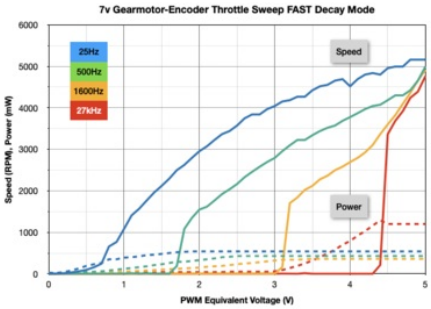
The first step in a robotics project is to get a motor spinning. Once you've done that you quickly learn that not all motors go the same speed, even if they are the same part...

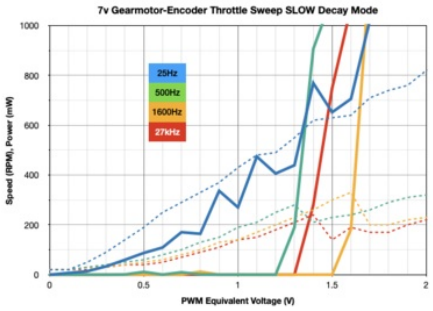
$13.50

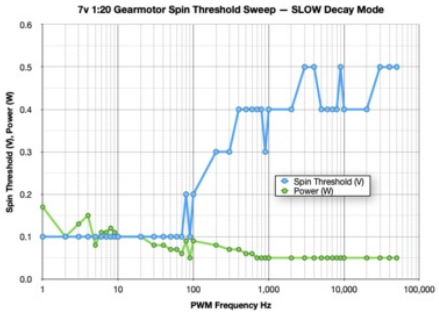
In Stock

Add to Cart









CD/DVD Spindle Motor; RF-300FA-12350

Your browser does not support the video tag.

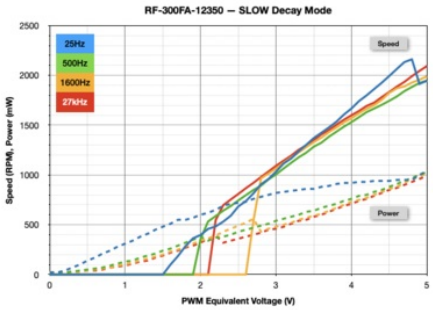
CD DVD Spindle Motor

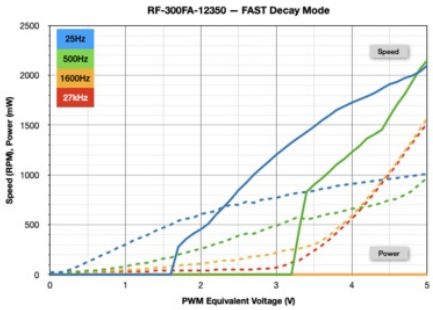
What's this? A record player for ants?? Not at all! This is a DVD/CD Spindle Motor, that thing that's inside a CD or DVD player, that turns the disc...

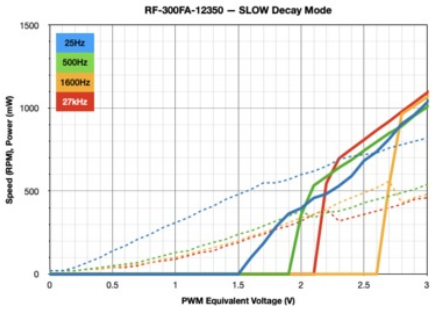
$1.95

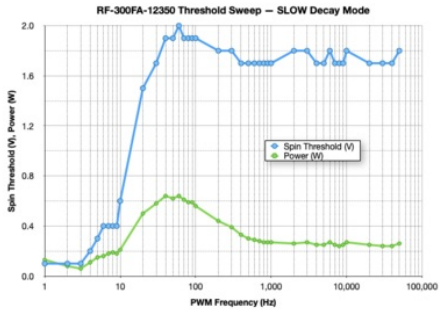
In Stock

Add to Cart


RF-300FA-12350 — SLOW Decay Mode
25Hz
500Hz
1600Hz
27kHz
Speed
Power
Speed (RPM), Power (mW)
PWM Equivalent Voltage (V)

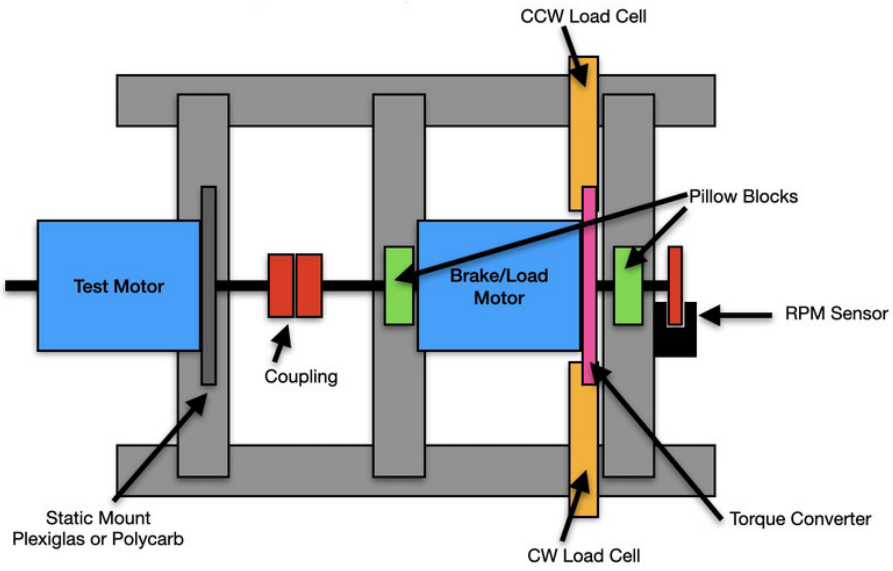

RF-300FA-12350 — FAST Decay Mode
25Hz
500Hz
1600Hz
27kHz
Speed
Power
Speed (RPM), Power (mW)
PWM Equivalent Voltage (V)


RF-300FA-12350 — SLOW Decay Mode
25Hz
500Hz
1600Hz
27kHz
Speed (RPM), Power (mW)
PWM Equivalent Voltage (V)


RF-300FA-12350 Threshold Sweep — SLOW Decay Mode
Spin Threshold (V)
Power (W)
Spin Threshold (V), Power (W)
PWM Frequency (Hz)

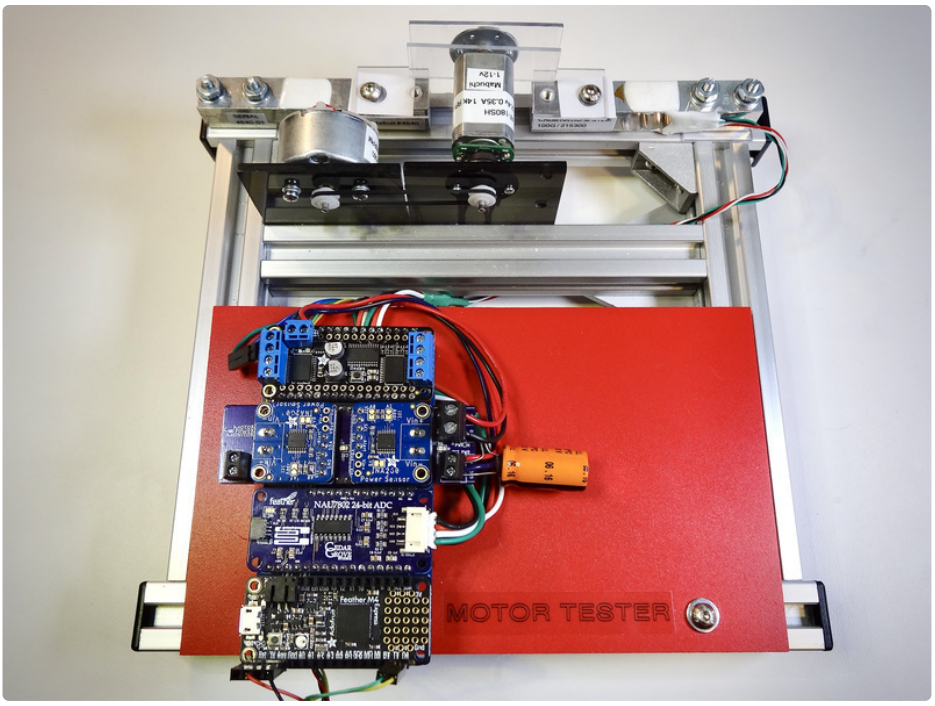
A Motor Testing Appliance

To fully characterize a motor's frequency response curve scientifically, an appliance for measuring motor performance is essential. At a minimum, measuring motor RPM under load while varying the PWM frequency will yield the frequencies where motor speed begins to drop and stall. A more sophisticated testing appliance adds sensors for measuring voltage, current, power, and torque.



The motor tester used for this Learning Guide consisted of six fundamental components:

- Test Motor
- Brake/Load Motor and Torque Converter
    - Custom FeatherWing 24-bit ADC (https://adafru.it/QD8) for load cells (https://adafru.it/QD9)
- RPM Sensors
    - Magnetic RPM Sensor (Pololu #1523) (https://adafru.it/QDa)
    - Optical Sensor (Adafruit #3986 with #3782) (https://adafru.it/QDb)
- INA260 Voltage and Current Sensor (https://adafru.it/EGR)
    - Custom FeatherWing for INA260 sensor (https://adafru.it/QDc)
- Motor FeatherWing (https://adafru.it/QAt)
- Feather M4 Express (https://adafru.it/Cmy)



The motor tester's code, written in CircuitPython, exercises the test motor, applies a braking current to the brake motor to simulate a load, and measures the current drawn by the test motor. In addition, motor speed and torque are recorded throughout the test.

The data used to create the collection of performance charts for this learning guide were captured by this apparatus.

References

Mabuchi Motor Model Designation Reference (https://adafru.it/QCo)

Wikipedia: DC Electromagnetic Motor (https://adafru.it/QCp)

Wikipedia: Electrical Impedance (https://adafru.it/QCr)Wikipedia: Resistance Inductance (RL) Circuit (https://adafru.it/L8C)

Texas Instruments: Current Recirculation and Decay Modes (https://adafru.it/RAX)

Coil Inductance Calculator (https://adafru.it/QCq)