DC, Servo, Stepper Motors and Solenoids with the Pico

Created by John Park


Last updated on 2023-11-08 04:19:16 PM EST
# Table of Contents

- **Overview** 3
  - Parts
  - Power
  - Tools

- **Installing CircuitPython** 8
  - CircuitPython Quickstart
  - Flash Resetting UF2

- **Installing the Mu Editor** 11
  - Download and Install Mu
  - Starting Up Mu
  - Using Mu

- **Installing Libraries** 13
  - Installing the Adafruit CircuitPython Motor Library

- **DC Motors** 14
  - Wiring the Circuit
  - Code
  - How It Works

- **Servo Motors** 19
  - Code
  - How It Works

- **Stepper Motor** 24
  - Code
  - How It Works

- **Solenoids** 30
  - Code
  - How It Works

- **Full Motor Party** 35
  - Code
Overview

You've got an RP2040 Pico and some motors and a burning desire to get them working together. This guide is here for you!

In it, you can pick among different types of motors/drives and learn how to wire them up and control them using CircuitPython!

With lots of pins, and a "it's OK if it breaks" price, the Pico is great for robotics projects, and thanks to CircuitPython its also incredibly easy to use DC motors, Servo motors, Stepper motors and Solenoids.

Let's get this motor party started!

Parts

Raspberry Pi Pico RP2040 with Loose Unsoldered Headers
The Raspberry Pi foundation changed single-board computing when they released the Raspberry Pi computer, now they're...
https://www.adafruit.com/product/4883

Raspberry Pi Pico RP2040
The Raspberry Pi foundation changed single-board computing when they released the Raspberry Pi computer, now they're ready to...
https://www.adafruit.com/product/4864
Standard servo - TowerPro SG-5010
This high-torque standard servo can rotate approximately 180 degrees (90 in each direction). You can use any servo code, hardware, or library to control these servos. Good for...
https://www.adafruit.com/product/155

DC Gearbox Motor - "TT Motor" - 200RPM - 3 to 6VDC
Perhaps you've been assembling a new robot friend, adding a computer for a brain and other fun personality touches. Now the time has come to let it leave the nest and fly on...
https://www.adafruit.com/product/3777

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...
https://www.adafruit.com/product/4489

Orange and Clear TT Motor Wheel for TT DC Gearbox Motor
Usually when one needs an orange wheel it's a garnish for a cocktail, like a tasty Sidecar. And speaking of cars, this wheel is for driving, not...
https://www.adafruit.com/product/3766
Small Reduction Stepper Motor - 5VDC
32-Step 1/16 Gearing
This is a great first stepper motor, good for small projects and experimenting with steppers. This uni-polar motor has a built-in mounting plate with two mounting holes. There are only...
https://www.adafruit.com/product/858

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...
https://www.adafruit.com/product/3297

Mini Push-Pull Solenoid - 5V
Solenoids are basically electromagnets: they are made of a coil of copper wire with an armature (a slug of metal) in the middle. When the coil is energized, the slug is pulled into the...
https://www.adafruit.com/product/2776

47uF 25V Electrolytic Capacitors - Pack of 10
We like capacitors so much we made a kids' show about them. ...
https://www.adafruit.com/product/2194
N-channel power MOSFET
When you need to switch a lot of power, N-channel MOSFETs are best for the job. These FETs can switch over 60A and 30V and are TO-220 packages so they fit nicely into any breadboard or...
https://www.adafruit.com/product/355

1N4001 Diode - 10 pack
This here is a 10 pack of the classic 1N4001 power blocking diode. These are good for reverse polarity protection (put it between your DC power jack and circuitry to avoid a...

Colorful Round Tactile Button Switch Assortment - 15 pack
Little clicky switches are standard input "buttons" on electronic projects. These work best in a PCB but can be...
https://www.adafruit.com/product/1009

Full Sized Premium Breadboard - 830 Tie Points
This is a 'full-size' premium quality breadboard, 830 tie points. Good for small and medium projects. It's 2.2" x 7" (5.5 cm x 17 cm) with a standard double-strip...
https://www.adafruit.com/product/239
Hook-up Wire Spool Set - 22AWG Solid Core - 6 x 25 ft
Perfect for bread-boarding, free wiring, etc. This box contains 6 spools of solid-core wire. The wire is easy to solder to and when bent it keeps its shape pretty well. We like to have...
https://www.adafruit.com/product/1311

Power

4 x AA Battery Holder with 2.1mm Plug and On/Off Switch
Here's another addition to our growing family of AA battery holders. Behold! A holder for four (4) AA batteries!...
https://www.adafruit.com/product/3788

Female DC Power adapter - 2.1mm jack to screw terminal block
If you need to connect a DC power wall wart to a board that doesn't have a DC jack - this adapter will come in very handy! There is a 2.1mm DC jack on one end, and a screw terminal...
https://www.adafruit.com/product/368
Alkaline AA batteries (LR6) - 4 pack
Battery power for your portable project!
These batteries are good quality at a good price, and work fantastic with any of the kits or projects in the shop that use AAs. This is a...
https://www.adafruit.com/product/3349

Tools

Automatic Self-Adjusting Wire Strippers and Cutter
If you've got a lot of wire-stripping to do, you may want to try out an automatic stripper/cutter! They are fantastic for stripping and cutting stranded and solid-core 24 or...
https://www.adafruit.com/product/4747

Installing CircuitPython

CircuitPython () is a derivative of MicroPython () designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the CIRCUITPY drive to iterate.

CircuitPython Quickstart

Follow this step-by-step to quickly get CircuitPython working on your board.

Download the latest version of CircuitPython for the Raspberry Pi Pico from circuitpython.org
Click the link above and download the latest UF2 file.

Download and save it to your desktop (or wherever is handy).

Start with your Pico unplugged from USB. Hold down the BOOTSEL button, and while continuing to hold it (don't let go!), plug the Pico into USB. Continue to hold the BOOTSEL button until the RPI-RP2 drive appears!

If the drive does not appear, unplug your Pico and go through the above process again.

A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.
You will see a new disk drive appear called RPI-RP2.

Drag the adafruit_circuitpython_etc.uf2 file to RPI-RP2.

The RPI-RP2 drive will disappear and a new disk drive called CIRCUITPY will appear.

That's it, you're done! :)

Flash Resetting UF2

If your Pico ever gets into a really weird state and doesn't even show up as a disk drive when installing CircuitPython, try installing this 'nuke' UF2 which will do a 'deep clean' on your Flash Memory. You will lose all the files on the board, but at least you'll be able to revive it! After nuking, re-install CircuitPython

flash_nuke.uf2
Installing the Mu Editor

Mu is a simple code editor that works with the Adafruit CircuitPython boards. It’s written in Python and works on Windows, MacOS, Linux and Raspberry Pi. The serial console is built right in so you get immediate feedback from your board's serial output!

Mu is our recommended editor - please use it (unless you are an experienced coder with a favorite editor already!).

Download and Install Mu

Download Mu from https://codewith.mu().

Click the Download link for downloads and installation instructions.

Click Start Here to find a wealth of other information, including extensive tutorials and how-to's.

Windows users: due to the nature of MSI installers, please remove old versions of Mu before installing the latest version.
Starting Up Mu

The first time you start Mu, you will be prompted to select your 'mode' - you can always change your mind later. For now please select CircuitPython!

The current mode is displayed in the lower right corner of the window, next to the "gear" icon. If the mode says "Microbit" or something else, click the Mode button in the upper left, and then choose "CircuitPython" in the dialog box that appears.

Mu attempts to auto-detect your board on startup, so if you do not have a CircuitPython board plugged in with a CIRCUITPY drive available, Mu will inform you where it will store any code you save until you plug in a board.

To avoid this warning, plug in a board and ensure that the CIRCUITPY drive is mounted before starting Mu.

Using Mu

You can now explore Mu! The three main sections of the window are labeled below; the button bar, the text editor, and the serial console / REPL.
Now you're ready to code! Let's keep going...

## Installing Libraries

Alongside the core CircuitPython libraries (which are baked into CircuitPython), you'll also add the [Adafruit Motor library](https:// adafruit.com/products/motor-library) to add some helpful control over your motors.

### Installing the Adafruit CircuitPython Motor Library

Download the [library bundle](https:// adafruit.com/products/motor-library) here.

Copy the adafruit_motor folder from the bundle to the lib folder on your CIRCUITPY drive.

Before continuing make sure your board's lib folder has the adafruit_motor library folder copied over.
DC Motors

Continuous DC motors can turn all the way around like a wheel. You can get them ungeared where they turn at a stunning 2000 to 6000 times a minute (RPM) or geared where they turn at only about 250 RPM.

Check this page (1) for more info on DC motor types!

You will also need an H-Bridge chip:

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...
https://www.adafruit.com/product/4489

With an H-bridge driver, you can control the direction the motor turns (clockwise or counter-clockwise) and the speed, from stopped to full speed.

All DC motors have two wires that are used to power and control them.
Wiring the Circuit

Follow the diagram above to breadboard the circuit. These are the connections you'll make:

**Pico**

- Pico pin 34 (PWM6A) to H-bridge pin 7 (Input B)
- Pico pin 32 (PWM5B) to H-bridge pin 6 (Input A)
- Pico GND to H-bridge pin 8 (GND)
- Pico GND to H-bridge pin 5 (GND)
- Pico pin 30 (RESET) to reset button to GND

**Power**

- Power supply 5V+ to 47uF capacitor to GND
- 5V rail to H-bridge pin 2 (VCC)
- 5V rail to H-bridge pin 3 (VCC)

**Motor**

- either motor wire to H-bridge pin 1 (Output A)
- other motor wire to H-bridge pin4 (Output B)

The Pico is not connected to the +5V power supply, and will be powered over USB.
Copy the code from the element below, and paste it into a fresh file in Mu. Save the file to your Pico's CIRCUITPY drive as code.py It will automatically run the code!

This is the basic motor test code that will run the motor at two different speeds in each direction and then stop. It's a great way to check that everything's working before moving on to something more complex.

```python
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# This example uses an L9110 H-bridge driver to run a DC Motor using two PWM pins.
# https://www.adafruit.com/product/4489

# Hardware setup:
# DC motor via L9110 H-bridge driver on two PWM pins that are on their own channels
# e.g., RP2040 Pico pins GP28, GP27

import time
import import board
import pwmio
from adafruit_motor import motor

PWM_PIN_A = board.GP28  # pick any pwm pins on their own channels
PWM_PIN_B = board.GP27

# DC motor setup
# DC Motors generate electrical noise when running that can reset the microcontroller in extreme cases. A capacitor can be used to help prevent this.
pwm_a = pwmio.PWMOut(PWM_PIN_A, frequency=50)
pwm_b = pwmio.PWMOut(PWM_PIN_B, frequency=50)
motor1 = motor.DCMotor(pwm_a, pwm_b)

print("***DC motor test***")

print("Forwards slow")
motor1.throttle = 0.5
print("  throttle: ", motor1.throttle)
time.sleep(1)

print("Stop")
motor1.throttle = 0
print("  throttle: ", motor1.throttle)
time.sleep(1)

print("Forwards")
motor1.throttle = 1.0
print("  throttle: ", motor1.throttle)
time.sleep(1)

print("Stop")
motor1.throttle = 0
print("  throttle: ", motor1.throttle)
time.sleep(1)

print("Backwards")
motor1.throttle = -1.0
print("  throttle: ", motor1.throttle)
time.sleep(1)```

©Adafruit Industries
print("\nStop")
motor1.throttle = 0
print("throttle: ", motor1.throttle)
time.sleep(1)

print("\nBackwards slow")
motor1.throttle = -0.5
print("  throttle: ", motor1.throttle)
time.sleep(1)

print("\nStop")
motor1.throttle = 0
print("  throttle: ", motor1.throttle)
time.sleep(1)

print("\nSpin freely")
motor1.throttle = None
print("  throttle: ", motor1.throttle)

print("\n***Motor test is complete***")

How It Works

Libraries

First, the libraries are imported for time so it can pause, board for pin definitions, pwmio for Pulse Width Modulation (PWM), and the motor module from adafruit_motor for easy motor control.

```python
import time
import board
import pwmio
from adafruit_motor import motor
```

Setup

Two PWM pins are used to control the two H-bridge drivers in the L9110. PWM is a digital approximation of an analog signal, allowing a smooth-ish range of varying control to be specified for the motor speed. By using two PWM pins and the dual H-bridge driver, the motor can be set to move both forward and backward.

The Pico has eight channels of PWM, with two pins per channel. In order to clock them separately, the two pins chosen must be on different PWM channels.

The PWM outputs will be set to a particular pin and PWM frequency (the duty cycle is what will be varied by the motor.throttle() control).

Finally, the motor object is created with motor.DCMotor(pwm_a, pwm_b).
PWM_PIN_A = board.GP28
PWM_PIN_B = board.GP27

pwm_a = pwmio.PWMOut(PWM_PIN_A, frequency=50)
pwm_b = pwmio.PWMOut(PWM_PIN_B, frequency=50)
motor1 = motor.DCMotor(pwm_a, pwm_b)

Control

Once the motor has been set up, it is controlled with the motor.throttle command. It can be set to anywhere from \(-1.0\) (full speed backward) to \(1.0\) (full speed forward). Practically speaking, a throttle value between -0.3 to 0.3 won't turn the motor, so keep that in mind.

A throttle of \(0\) stops the motor, while a value of None releases it to rotate freely.

```python
motor1.throttle = 0.5
time.sleep(1)
motor1.throttle = 0
time.sleep(1)
motor1.throttle = 1.0
time.sleep(1)
motor1.throttle = 0
time.sleep(1)
motor1.throttle = -1.0
time.sleep(1)
motor1.throttle = 0
time.sleep(1)
motor1.throttle = -0.5
time.sleep(1)
motor1.throttle = 0
time.sleep(1)
motor1.throttle = None
```
Servo Motors

Hobby Servos, also sometimes called Standard or Micro servos, are small boxy motors.

The size of the 'box' may vary but they always have three wires for power and control and the wires are connected together into one 3-pin plug.

You might find a servo extender or 'extra long headers' helpful for plugging servos into breadboards:

**Servo Extension Cable - 30cm / 12" long**
Stretch out your servo connections with this flexible servo extension cord. It has a 3 pin shrouded "male" connection to plug your servo into and then, 30cm later, a 3 pin...

**Extra-long break-away 0.1" 16-pin strip male header (5 pieces)**
Breakaway header is like the duct tape of electronics, and this header is one better with extra long pins on both sides. This makes it great for connecting things together that...
[https://www.adafruit.com/product/400](https://www.adafruit.com/product/400)

Servos are a lot different than DC motors because while DC motors turn all the way around, standard servos only move back and forth about 180 degrees. But they can be controlled to rotate to a specific angle.
For more info on servos, check out this page.

Follow the diagram above to breadboard the circuit. These are the connections you'll make:

**Pico**

- Pico pin 1 (GP0) to servo yellow data wire
- Pico GND to servo black/brown GND wire
- Pico pin 5 (GP3) to mode button to GND
- Pico pin 30 (RESET) to reset button to GND

**Power**

- Power supply 5V+ to 47uF capacitor to GND
- 5V rail to servo red/orange +V wire

**Code**

Copy the code from the element below, and paste it into a fresh file in Mu. Save the file to your Pico's CIRCUITPY drive as code.py. It will automatically run the code!

Press the mode button to have the servo go directly to 0°, 90°, and 180° positions. Press the button a second time to have it smoothly sweep through its range.
# Pico servo demo
# Hardware setup:
#   Servo on GP0 with external 5V power supply
#   Button on GP3 and ground

import time
import board
from digitalio import DigitalInOut, Direction, Pull
import pwmio
from adafruit_motor import servo

print("Servo test")

led = DigitalInOut(board.LED)
led.direction = Direction.OUTPUT
led.value = True

def blink(times):
    for _ in range(times):
        led.value = False
        time.sleep(0.1)
        led.value = True
        time.sleep(0.1)

# Mode button setup
button = DigitalInOut(board.GP3)
button.direction = Direction.INPUT
button.pull = Pull.UP
mode = -1  # track state of button mode

# Servo setup
pwm_servo = pwmio.PWMOut(board.GP0, duty_cycle=2 ** 15, frequency=50)
servo1 = servo.Servo(
    pwm_servo, min_pulse=500, max_pulse=2200)

# Servo test

def servo_direct_test():
    print("servo test: 90")
    servo1.angle = 90
    time.sleep(2)
    print("servo test: 0")
    servo1.angle = 0
    time.sleep(2)
    print("servo test: 90")
    servo1.angle = 90
    time.sleep(2)
    print("servo test: 180")
    servo1.angle = 180
    time.sleep(2)

# Servo smooth test

def servo_smooth_test():
    print("servo smooth test: 180 - 0, -1º steps")
    for angle in range(180, 0, -1):  # 180 - 0 degrees, -1º at a time.
        servo1.angle = angle
        time.sleep(0.01)
        time.sleep(1)
    print("servo smooth test: 0 - 180, 1º steps")
    for angle in range(0, 180, 1):  # 0 - 180 degrees, 1º at a time.
        servo1.angle = angle
        time.sleep(0.01)
        time.sleep(1)
def run_test(testnum):
    if testnum is 0:
        servo_direct_test()
    elif testnum is 1:
        servo_smooth_test()

while True:
    if not button.value:
        blink(2)
        mode = (mode + 1) % 2
        print("switch to mode %d" % (mode))
        time.sleep(0.8)  # big debounce
        run_test(mode)

How It Works

Libraries

First, the libraries are imported for time so it can pause, board for pin definitions, digitalio to use buttons, pwmio for Pulse Width Modulation (PWM), and the servo module from adafruit_motor for easy servo motor control.

import time
import board
from digitalio import DigitalInOut, Direction, Pull
import pwmio
from adafruit_motor import servo

LED

The on-board LED is set up and turned on to show that the code is running, and will blink when the button is pressed.

led = DigitalInOut(board.LED)
led.direction = Direction.OUTPUT
led.value = True

def blink(times):
    for _ in range(times):
        led.value = False
time.sleep(0.1)
        led.value = True
time.sleep(0.1)

Button

The button is set up on pin GP3 as an input with internal pull up resistor. When the button is pressed the mode variable will increment to set the servo test state.
button = DigitalInOut(board.GP3)
button.direction = Direction.INPUT
button.pull = Pull.UP
mode = -1  # track state of button mode

PWM & Servo

The PWM pin is used to control the servo angle. PWM is a digital approximation of an analog signal, allowing a smooth-ish range of varying control to be specified for the motor speed.

The PWM output will be set to pin GP0, duty cycle to half of the full 65,535, and PWM frequency of 50Hz.

The servo object is then created, specifying the PWM pin and the minimum and maximum pulse values for the servo - in microseconds. (This can vary from servo to servo, see datasheets or ask your vendor for details.)

```python
pwm_servo = pwmio.PWMOut(board.GP0, duty_cycle=2 ** 15, frequency=50)
servo1 = servo.Servo(
    pwm_servo, min_pulse=500, max_pulse=2200)
```

Servo Test Functions

The servo can be controlled to go directly to a particular angle by sending `servo.angle` commands.

This first function allows us to easily test sending the servo directly to 90º, 0º, and 180º.

The second function steps smoothly to an angle in 1º increments.

```python
# Servo test
def servo_direct_test():
    print("servo test: 90")
    servo1.angle = 90
    time.sleep(2)
    print("servo test: 0")
    servo1.angle = 0
    time.sleep(2)
    print("servo test: 90")
    servo1.angle = 90
    time.sleep(2)
    print("servo test: 180")
    servo1.angle = 180
    time.sleep(2)

# Servo smooth test
```
def servo_smooth_test():
    print("servo smooth test: 180 - 0, -1º steps")
    for angle in range(180, 0, -1):  # 180 - 0 degrees, -1º at a time.
        servo1.angle = angle
        time.sleep(0.01)
    time.sleep(1)
    print("servo smooth test: 0 - 180, 1º steps")
    for angle in range(0, 180, 1):  # 0 - 180 degrees, 1º at a time.
        servo1.angle = angle
        time.sleep(0.01)
    time.sleep(1)

Stepper Motor

Stepper motors -- much like their name implies, they "step" along.

These come with four, five or six wires.

Like DC motors, they rotate all the way around. But they do so very slowly, because of the little steps they have to take.

Like Servos, they have precision motion. But not the way servos do, where you can set a specific angle. Instead you can rotate forward and back by little steps.

Learn more about steppers [here](#).

You'll need a stepper motor driver (a.k.a a dual H-Bridge):

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...

[https://www.adafruit.com/product/3297](https://www.adafruit.com/product/3297)
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 "https://www.adafruit.com/product/2448

The DRV8833 motor controller can control the two motor coils inside the stepper motor (think of it like a circular bucket brigade passing the power around A-B-A-B-A-B and so on) by receiving signals from four digital output pins on the Pico.

Wire it up as shown in the diagram here.

Pico

- Pico pin 27 (GP21) to driver pin BIN1
- Pico pin 26 (GP20) to driver pin BIN2
- Pico pin 25 (GP19) to driver pin AIN2
- Pico pin 24 (GP18) to driver pin AIN1
- Pico GND to driver GND
- Pico 3.3V to driver VM
- Pico pin 30 (RESET) to reset button to GND
- Pico pin 5 (GP3) to mode button to GND
Power

- Power supply 5V+ to driver SLP
- Power supply GND to common ground

Motor (wiring color varies by motor)

- Orange motor wire to driver AOUT1
- Pink motor wire to driver AOUT2
- Blue motor wire to driver BOUT1
- Yellow motor wire to driver BOUT2

Code

Copy the code from the element below, and paste it into a fresh file in Mu. Save the file to your Pico's CIRCUITPY drive as code.py. It will automatically run the code!

Press the mode button to have the stepper run through a single revolution forward and then a single revolution backward.

```python
# SPDX-FileCopyrightText: 2021 jedgarpark for Adafruit Industries
# SPDX-License-Identifier: MIT

# Pico stepper demo
# Hardware setup:
#   Stepper motor via DRV8833 driver breakout on GP21, GP20, GP19, GP18
#   external power supply
#   Button on GP3 and ground

import time
import board
from digitalio import DigitalInOut, Direction, Pull
from adafruit_motor import stepper

print("Stepper test")

led = DigitalInOut(board.LED)
led.direction = Direction.OUTPUT
led.value = True

def blink(times):
    for _ in range(times):
        led.value = False
time.sleep(0.1)
        led.value = True
time.sleep(0.1)

# Mode button setup
button = DigitalInOut(board.GP3)
button.direction = Direction.INPUT
button.pull = Pull.UP
```
mode = -1  # track state of button mode

# Stepper motor setup
DELAY = 0.006  # fastest is ~ 0.004, 0.01 is still very smooth, gets steppy after that
STEPS = 513  # this is a full 360º
coils = (  
    DigitalInOut(board.GP21),  # A1  
    DigitalInOut(board.GP20),  # A2  
    DigitalInOut(board.GP19),  # B1  
    DigitalInOut(board.GP18),  # B2  
)
for coil in coils:
    coil.direction = Direction.OUTPUT
stepper_motor = stepper.StepperMotor(  
    coils[0], coils[1], coils[2], coils[3], microsteps=None  
)

def stepper_fwd():
    print("stepper forward")
    for _ in range(STEPS):
        stepper_motor.onestep(direction=stepper.FORWARD)
        time.sleep(DELAY)
    stepper_motor.release()

def stepper_back():
    print("stepper backward")
    for _ in range(STEPS):
        stepper_motor.onestep(direction=stepper.BACKWARD)
        time.sleep(DELAY)
    stepper_motor.release()

def run_test(testnum):
    if testnum is 0:
        stepper_fwd()
    elif testnum is 1:
        stepper_back()

while True:
    if not button.value:
        blink(2)
        mode = (mode + 1) % 2
        print("switch to mode %d" % (mode))
        time.sleep(0.8)  # big debounce
        run_test(mode)

How It Works

Libraries

First, the libraries are imported for time, board for pin definitions, digitalio to use buttons as input and control pins as outputs to the driver board, and the stepper module from adafruit_motor for easy servo motor control.

```python
import time
import board
```
from digitalio import DigitalInOut, Direction, Pull
from adafruit_motor import stepper

LED

The on-board LED is set up and turned on to show that the code is running, and will blink when the button is pressed.

```python
led = DigitalInOut(board.LED)
led.direction = Direction.OUTPUT
led.value = True

def blink(times):
    for _ in range(times):
        led.value = False
time.sleep(0.1)
        led.value = True
time.sleep(0.1)
```

Button

The button is set up on pin GP3 as an input with internal pull up resistor. When the button is pressed the `mode` variable will increment to set the servo test state.

```python
button = DigitalInOut(board.GP3)
button.direction = Direction.INPUT
button.pull = Pull.UP
mode = -1  # track state of button mode
```

Stepper Setup

The stepper is set up next. `DELAY` sets the speed of the stepper, and the `STEPS` value is the number of steps in the motor.

The coils to be controlled by the driver board are set up on four of the Pico's digital output pins, next, and then the stepper object is created with these pins.

```python
DELAY = 0.006  # fastest is ~ 0.004, 0.01 is still very smooth, gets steppy after that
STEPS = 513  # this is a full 360°
coils = (
    DigitalInOut(board.GP21),  # A1
    DigitalInOut(board.GP20),  # A2
    DigitalInOut(board.GP19),  # B1
    DigitalInOut(board.GP18),  # B2
)
for coil in coils:
    coil.direction = Direction.OUTPUT

stepper_motor = stepper.StepperMotor(
```
Stepper Functions

Two functions are made to control the stepper in a full revolution -- one forward, one backward -- and then a helper function is made to call those.

```python
def stepper_fwd():
    print("stepper forward")
    for _ in range(STEPS):
        stepper_motor.onestep(direction=stepper.FORWARD)
        time.sleep(DELAY)
    stepper_motor.release()

def stepper_back():
    print("stepper backward")
    for _ in range(STEPS):
        stepper_motor.onestep(direction=stepper.BACKWARD)
        time.sleep(DELAY)
    stepper_motor.release()

def run_test(testnum):
    if testnum is 0:
        stepper_fwd()
    elif testnum is 1:
        stepper_back()
```

Main Loop

The main loop of the program watches for the button to be pressed and then runs the `run_test()` function alternating forward or backward with each button press.

```python
while True:
    if not button.value:
        blink(2)
        mode = (mode + 1) % 2
        print("switch to mode %d" % (mode))
        time.sleep(0.8)  # big debounce
        run_test(mode)
```
Solenoids

Solenoids are excellent for pushing and pulling small things. They are electromagnets that, when on, will push or pull a small metal slug a short distance. A built in spring is used to return the slug to its resting position when off.

Solenoids use greater amount of current than a microcontroller can supply, so a MOSFET transistor will handle the power duties.

When the MOSFET gate pin receives signal from the Pico GP14 output, the MOSFET will allow the 5V to head through the solenoid to ground, thus energizing the solenoid's electromagnet and moving its slug.

When the Pico GP14 output goes low, the MOSFET gate will close and the solenoid will become de-energized, returning the slug to its resting position.

Follow the diagram above to breadboard the circuit. These are the connections you'll make:

- Pico pin 19 (GP14) to MOSFET gate
- Pico GND to common ground rail
- Pico pin 5 (GP3) to mode button to GND
- Pico pin 30 (RESET) to reset button to GND
- Power supply 5V+ to 47uF capacitor to GND
- 5V rail to solenoid pin 2
- MOSFET source to GND
- MOSFET drain to solenoid pin 1
- Diode to shunt 5V from drain and solenoid pin 1

Code

Copy the code from the element below, and paste it into a fresh file in Mu. Save the file to your Pico's CIRCUITPY drive as code.py It will automatically run the code!

In this program, all you need to do is press the mode button, and then the solenoid will strike four times.

```python
# This example uses an MOSFET transistor circuit to drive a solenoid from a Pico RP2040 digitalio pin
# Hardware setup:
# - Button on GP3 to gnd (uses internal pull up)
# - MOSFET driving solenoid on GP14 w protection diode and 47uF capacitor across power rails
# - external power source should be proper voltage and current for solenoid, not USB power

import time
import board
from digitalio import DigitalInOut, Direction, Pull

# SPDX-FileCopyrightText: 2021 jedgarpark for Adafruit Industries
# SPDX-License-Identifier: MIT
```
print("*** Solenoid Demo ***")

led = DigitalInOut(board.LED)  # onboard LED setup
led.direction = Direction.OUTPUT
led.value = True

def blink(times):
    for _ in range(times):
        led.value = False
time.sleep(0.1)
        led.value = True
time.sleep(0.1)

# Mode button setup
button = DigitalInOut(board.GP3)
button.direction = Direction.INPUT
button.pull = Pull.UP

# Solenoid setup
solenoid = DigitalInOut(board.GP14)  # pin drives a MOSFET
solenoid.direction = Direction.OUTPUT
solenoid.value = False

strike_time = 0.05  # coil on time range ~0.05 - ? beware heat/power drain beyond a few seconds
recover_time = 0.20  # adjust for coil off time/pause between strikes

def solenoid_strike(loops):  # solenoid strike function
    print("solenoid test")
    for _ in range(loops):
        solenoid.value = True
time.sleep(strike_time)
        solenoid.value = False
time.sleep(recover_time)
time.sleep(0.1)

while True:
    if not button.value:
        blink(1)
        solenoid_strike(4)

How It Works

Libraries

First, the libraries are imported for time so it can pause, board for pin definitions, and digitalio for lighting the onboard LED, using the mode button, and sending signals to the MOSFET to fire the solenoid.

import time
import board
from digitalio import DigitalInOut, Direction, Pull
LED

The on-board LED is set up and turned on to show that the code is running, and will blink when the button is pressed.

```python
led = DigitalInOut(board.LED)  # onboard LED setup
led.direction = Direction.OUTPUT
led.value = True

def blink(times):
    for _ in range(times):
        led.value = False
        time.sleep(0.1)
        led.value = True
        time.sleep(0.1)
```

Button

The button is set up on pin GP3 as an input with internal pull up resistor. When the button is pressed the solenoid will strike four times (you can change this number later).

```python
button = DigitalInOut(board.GP3)
button.direction = Direction.INPUT
button.pull = Pull.UP
```

Solenoid Setup

The solenoid will be controlled by setting a GPIO output pin to high. Pin GP14 on the Pico will be used for this -- since it can be anything from a PWM pin, to an I2C pin, to a digital input or output and more, we must first set the pin as a digital IO output pin.

Its `value` will be set low or `False` by default.

```python
solenoid = DigitalInOut(board.GP14)  # pin drives a MOSFET
solenoid.direction = Direction.OUTPUT
solenoid.value = False
```

Time Variables

It takes a fraction of a second to energize and then release the solenoid, as low as ~0.05 seconds depending on the power supply. A variable called `strike_time` is set to define this value. This determines if it'll be a very quick strike, or if the slug will get thrown and stay in that position for a while. You can hold it out for a second or more,
but beware the power drain and possible over-heating of the solenoid coil if it is energized for too long without release.

The amount of time spent recovering in a non-energized state will be defined with the `recover_time` variable. This will determine the tempo of consecutive strikes.

```python
strike_time = 0.05
recover_time = 0.20
```

Note how this adds up to a quarter of a second -- a pretty nice tempo for musical quarter notes in a 60 BPM (beats per minute) pattern. Just in case you’re thinking of drumming with your solenoids.

**Solenoid Strike Function**

The `solenoid_strike(loops)` function is set up to make it easy to loop through a number of solenoid strikes.

It takes an `loops` input value and then uses the `strike_time` and `recover_time` to repeat a pattern of strikes.

```python
def solenoid_strike(loops):  # solenoid strike function
    print("solenoid test")
    for _ in range(loops):
        solenoid.value = True
        time.sleep(strike_time)
        solenoid.value = False
        time.sleep(recover_time)
        time.sleep(0.1)
```

**Main Loop**

The main loop waits for a button press, then blinks the onboard LED and strikes the solenoid four times (this number can be changed to whatever you like).

Press the button again and it repeats the process. Happy Pico-driven solenoiding!

```python
while True:
    if not button.value:
        blink(1)
        solenoid_strike(4)
```
Full Motor Party

If you'd like, you can combine all of the previous steps into one big, popular, well attended motor party!

Mount it all on some cardboard or foam core and you've got a super fun demo board.

The wiring is essentially the same as each discrete motor demo from the previous pages, just spread out a bit for clarity.
Copy the code from the element below, and paste it into a fresh file in Mu. Save the file to your Pico's CIRCUITPY drive as code.py. It will automatically run the code!

Press the mode button to check out each test on the board. Each time you press the mode button it'll advance to the next motor test!

```python

# SPDX-FileCopyrightText: 2021 jedgarpark for Adafruit Industries
# SPDX-License-Identifier: MIT

# Pico motor party demo board
# Hardware setup:
#   Servo on GP0
#   DC motor via L9110 H-bridge driver on GP28, GP27 (they're on different PWM channels)
#   Stepper motor via DRV8833 driver breakout on GP21, GP20, GP19, GP18
#   Button on GP3
#   MOSFET driving solenoid on GP14

import time
import board
from digitalio import DigitalInOut, Direction, Pull
import pwmio
from adafruit_motor import stepper, motor, servo

print("*** Welcome to the Raspberry Pi RP2040 Pico Motor Party Demo ***")

led = DigitalInOut(board.LED)
led.direction = Direction.OUTPUT
led.value = True

def blink(times):
    for _ in range(times):
        led.value = False
time.sleep(0.1)
        led.value = True
time.sleep(0.1)

# Mode button setup
mode_button = DigitalInOut(board.GP3)
mode_button.direction = Direction.INPUT
mode_button.pull = Pull.UP
mode = -1  # track different demo modes

# Solenoid setup
solenoid = DigitalInOut(board.GP14)  # pin drives a MOSFET
solenoid.direction = Direction.OUTPUT
solenoid.value = False

strike_time = 0.04
recover_time = 0.04

# Solenoid test
def solenoid_test(loops):
    print("solenoid test")
    for _ in range(loops):
        solenoid.value = True
time.sleep(strike_time)
        solenoid.value = False
time.sleep(recover_time)
```

©Adafruit Industries
time.sleep(0.1)

# Servo setup
pwm_servo = pwmio.PWMOut(board.GP0, duty_cycle=2 ** 15, frequency=50)
servo1 = servo.Servo(
    pwm_servo, min_pulse=500, max_pulse=2200
)  # tune pulse for specific servo

# Servo test
def servo_direct_test():
    print("servo test: 90")
servo1.angle = 90
time.sleep(2)
    print("servo test: 0")
servo1.angle = 0
time.sleep(2)
    print("servo test: 90")
servo1.angle = 90
time.sleep(2)
    print("servo test: 180")
servo1.angle = 180
time.sleep(2)

# Servo smooth test
def servo_smooth_test():
    print("servo smooth test: 180 - 0, -1º steps")
    for angle in range(180, 0, -1):  # 180 - 0 degrees, -1º at a time.
        servo1.angle = angle
time.sleep(0.01)
time.sleep(1)
    print("servo smooth test: 0 - 180, 1º steps")
    for angle in range(0, 180, 1):  # 0 - 180 degrees, 1º at a time.
        servo1.angle = angle
time.sleep(0.01)
time.sleep(1)

# DC motor setup
pwm_a = pwmio.PWMOut(board.GP28, frequency=50)
pwm_b = pwmio.PWMOut(board.GP27, frequency=50)
motor1 = motor.DCMotor(pwm_a, pwm_b)

# DC motor test
def dc_motor_test():
    print("DC motor test: 0.5")
motor1.throttle = 0.5
time.sleep(0.5)
    print("DC motor test: None")
motor1.throttle = None
time.sleep(0.5)
    print("DC motor test: -0.5")
motor1.throttle = -0.5
time.sleep(0.5)
    print("DC motor test: None")
motor1.throttle = None
time.sleep(0.5)
    print("DC motor test: 0.9")
motor1.throttle = 0.9
time.sleep(0.5)
    print("DC motor test: None")
motor1.throttle = None
time.sleep(0.5)
    print("DC motor test: -0.9")
motor1.throttle = -0.9
time.sleep(0.5)
    print("DC motor test: None")
motor1.throttle = None
# Stepper motor setup
DELAY = 0.006  # fastest is ~ 0.004, 0.01 is still very smooth, gets steppy after that
STEPS = 513  # this is a full 360º
coils = (  
    DigitalInOut(board.GP21),  # A1  
    DigitalInOut(board.GP20),  # A2  
    DigitalInOut(board.GP19),  # B1  
    DigitalInOut(board.GP18),  # B2  
)
for coil in coils:
    coil.direction = Direction.OUTPUT
stepper_motor = stepper.StepperMotor(  
    coils[0], coils[1], coils[2], coils[3], microsteps=None  
)

def stepper_fwd():
    for _ in range(STEPS):
        stepper_motor.onestep(direction=stepper.FORWARD)
        time.sleep(DELAY)
stepper_motor.release()

def stepper_back():
    for _ in range(STEPS):
        stepper_motor.onestep(direction=stepper.BACKWARD)
        time.sleep(DELAY)
stepper_motor.release()

# Stepper test
def stepper_test():
    print("stepper test: forward")
    for _ in range(2):
        stepper_fwd()
    time.sleep(1)
    print("stepper test: backward")
    for _ in range(2):
        stepper_back()
    time.sleep(1)

def run_test(testnum):
    if testnum == 0:
        solenoid_test(4)
    elif testnum == 1:
        servo_direct_test()
    elif testnum == 2:
        servo_smooth_test()
    elif testnum == 3:
        dc_motor_test()
    elif testnum == 4:
        stepper_test()

while True:
    if not mode_button.value:
        blink(2)
        mode = (mode + 1) % 5
        print("switch to mode %d" % (mode))
        time.sleep(0.8)  # big debounce
    run_test(mode)