Motors with Binary Inputs

ECE 476 Advanced Embedded Systems

Jake Glower - Lecture #10

Please visit Bison Academy for corresponding
lecture notes, homework sets, and solutions

Introduction:

Part of the fun of being an engineer is you build things.

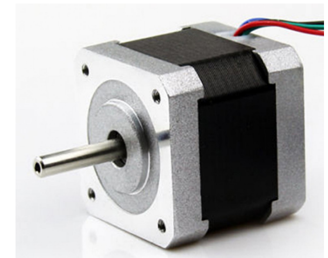
- Motors let you build things that move
- Examples: valves, robotic arms, cars, etc.

Several type of motors exist

- Smaller motors often have digital inputs (this lecture)
- Larger motors often have analog inputs (next lecture)

This lecture looks at driving with a Pi-Pico

- Stepper Motors,
- Solenoids,
- Brushless Servo-Motors (using pulse-width), and
- Digital Servo Motors (using pulse width)



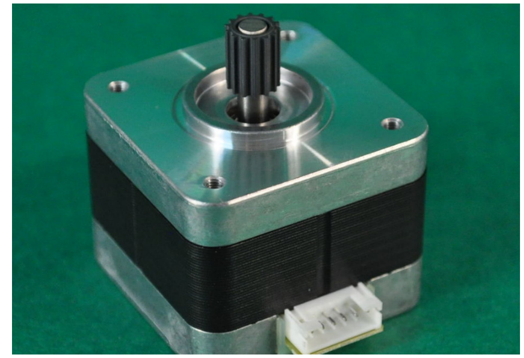
Stepper Motors

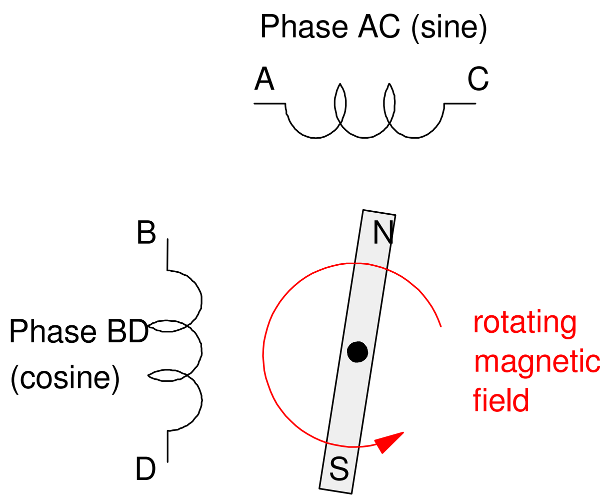
Stepper motors are a common type of motor

- These interface well with microcontrollers.

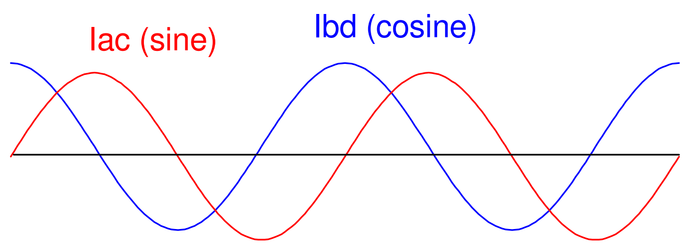
Actually 2-phase AC synchronous motors

- Input: a 2-phase AC sine wave (sine & cosine)
- The frequency sets the motor speed





Stepper Motor



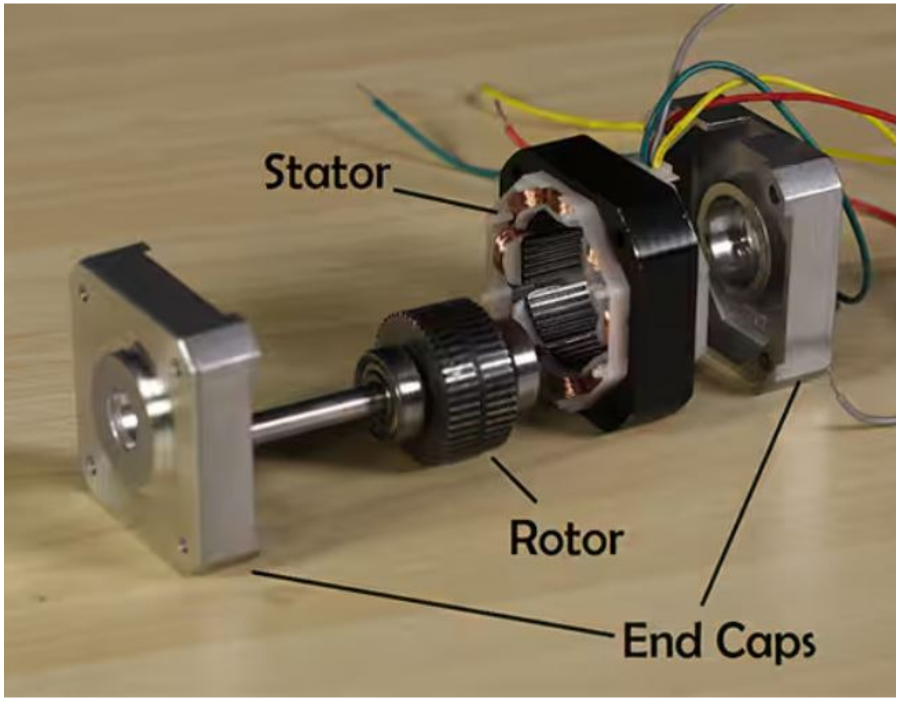
What's inside a stepper motor?

Rotor:

- The thing that spins
- Permanent magnets
    - number varies
- North & South poles
    - two poles per magnet

Stator

- Attached to the case
    - (doesn't spin)
- Can attract the N or S pole
    - depends on the direction of current flow



Stepper Motors: Hardware

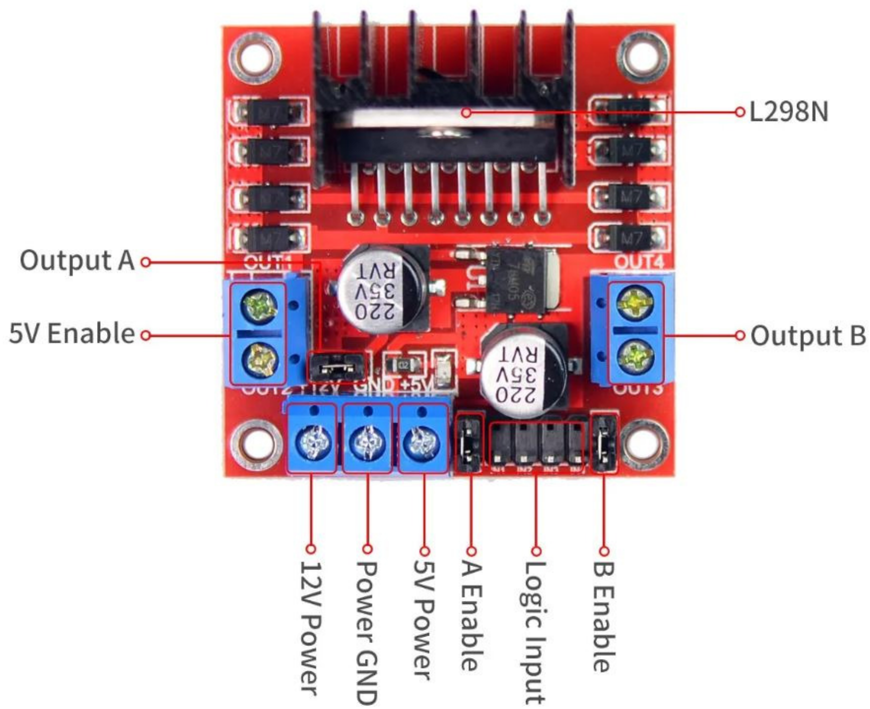
The hardware must allow current to flow both ways in each winding

- Sine waves are positive and negative

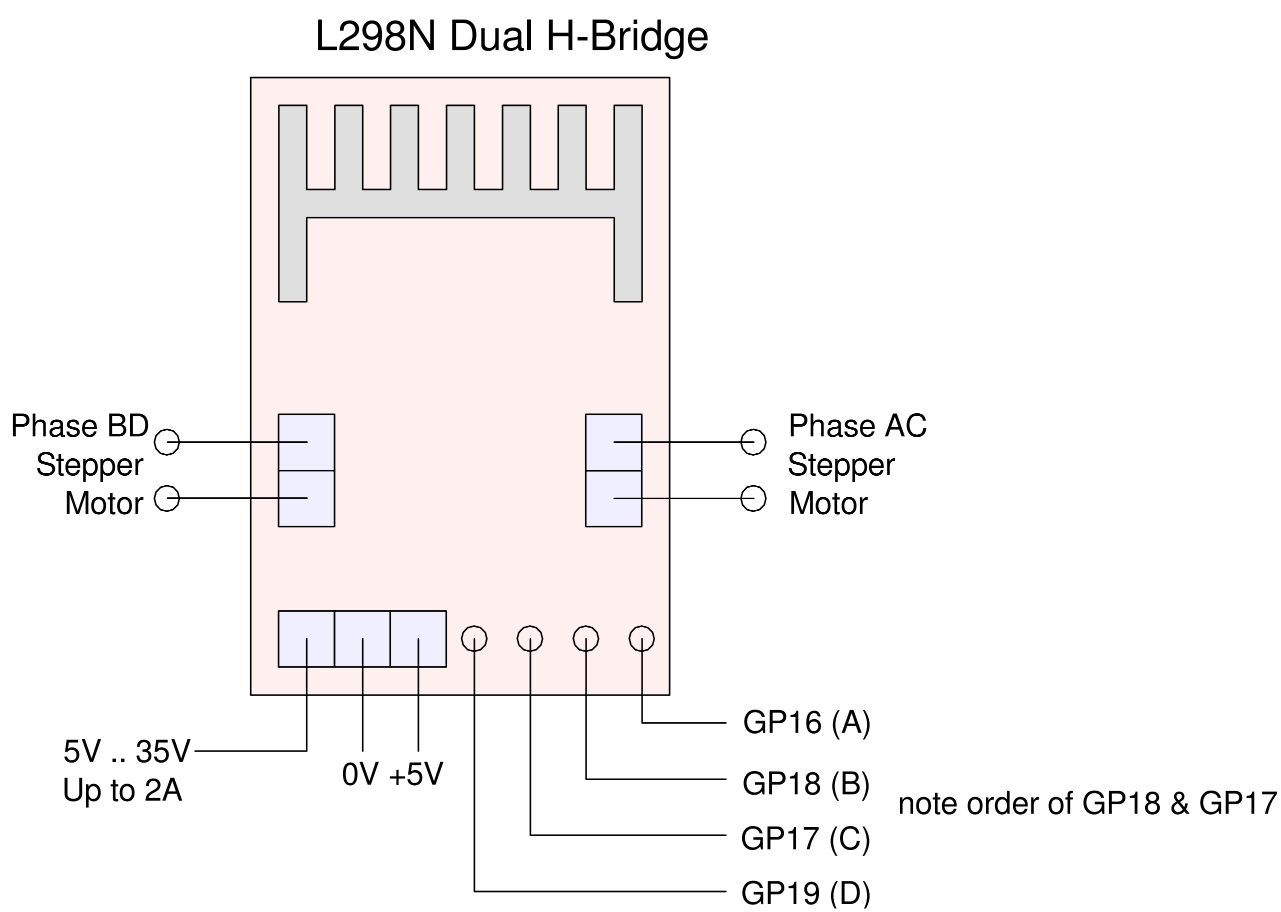
H-bridges are usually used to drive a stepper motor

- L298N from ebay and Amazon
- 5V to 35V operation
- Up to 2A per phase
- Max power = 25W
- About $3 each (2024 prices)

Higher power H-bridges are also available



Connections to your Pico board are:



Four wires from the Pico needed to drive the stepper motor

Software & Stepping

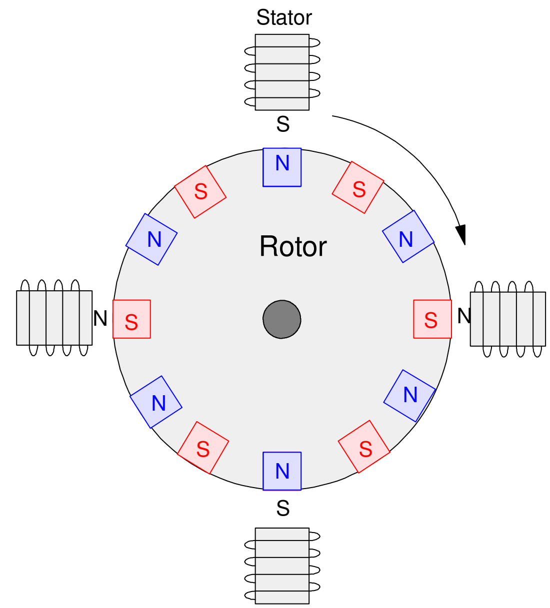
If you approximate a sine wave with a square wave, the motor steps.

The number of steps per rotation depends upon the motor

- How many magnetic poles per rotation
- The ones in lab have 50 poles
  - 50 sine waves equals one rotation

It also depends upon how you approximate a sine wave:

- Full-Stepping
  - four steps per cycle
  - 200 steps per rotation,
- Half-Stepping
  - eight steps per cycle
  - 400 steps per rotation
- Micro-Stepping
  - more than eight steps per cycle



Software - Full Stepping

Approximate sine waves with square waves.

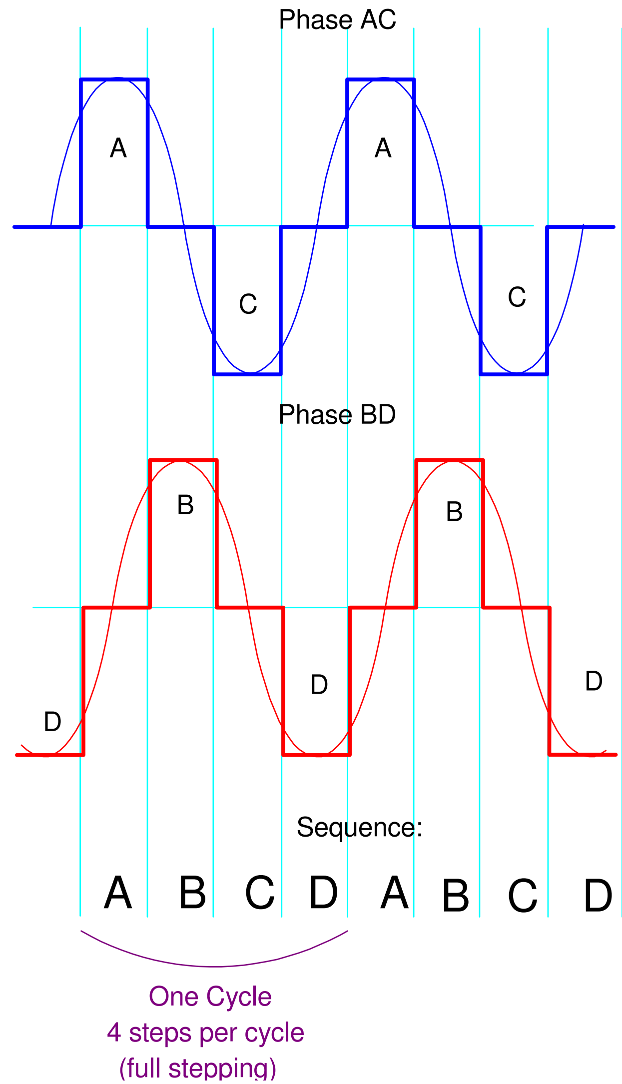
- On for one clock
- Off for one clock

Output two square waves

- sine & cosine

This results in 4 steps per cycle

- A - B - C - D - repeat
- reverse the order to spin the other way
- 200 steps per roation



Full Stepping: Code

Output the sequence

- A - B - C - D
- repeat

Use a table with 8 entries

- One for each step

```
A   =  0001  =  1
B   =  0010  =  2
C   =  0100  =  4
D   =  1000  =  8
```

```
# Stepper Motor - Full Stepping

from time import sleep_ms
from machine import Pin

PA = Pin(16,Pin.OUT)
PB = Pin(17,Pin.OUT)
PC = Pin(18,Pin.OUT)
PD = Pin(19,Pin.OUT)

TABLE = [1, 2, 4, 8]

def Step(X):
    Y = TABLE[X % 4]
    PA.value( Y & 8 )
    PB.value( Y & 4 )
    PC.value( Y & 2 )
    PD.value( Y & 1 )

x = 0
for i in range(0,100):
    x += 1
    Step(x)
    sleep_ms(10)
```

Software - Half-Stepping

Approximate sine waves with square waves

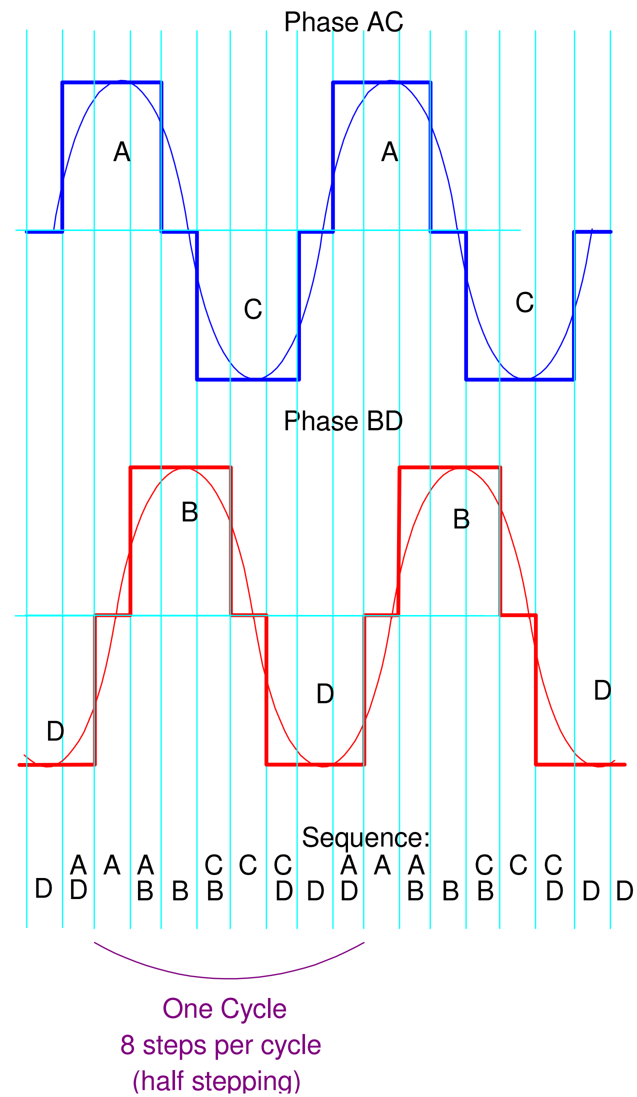
- on for 3 clocks
- off for 1

Output two square waves

- Sine & Cosine

This results in 8 steps per cycle

- A, AB, B, BC, C, CD, D, DA
- repeat
- 400 steps per rotation



Half-Stepping Code

Use a table with 8 entries

- One for each step

```
A   =  0001  =  1
AB  =  0011  =  3
B   =  0010  =  2
BC  =  0110  =  6
C   =  0100  =  4
CD  =  1100  =  12
D   =  1000  =  8
DA  =  1001  =  9
```

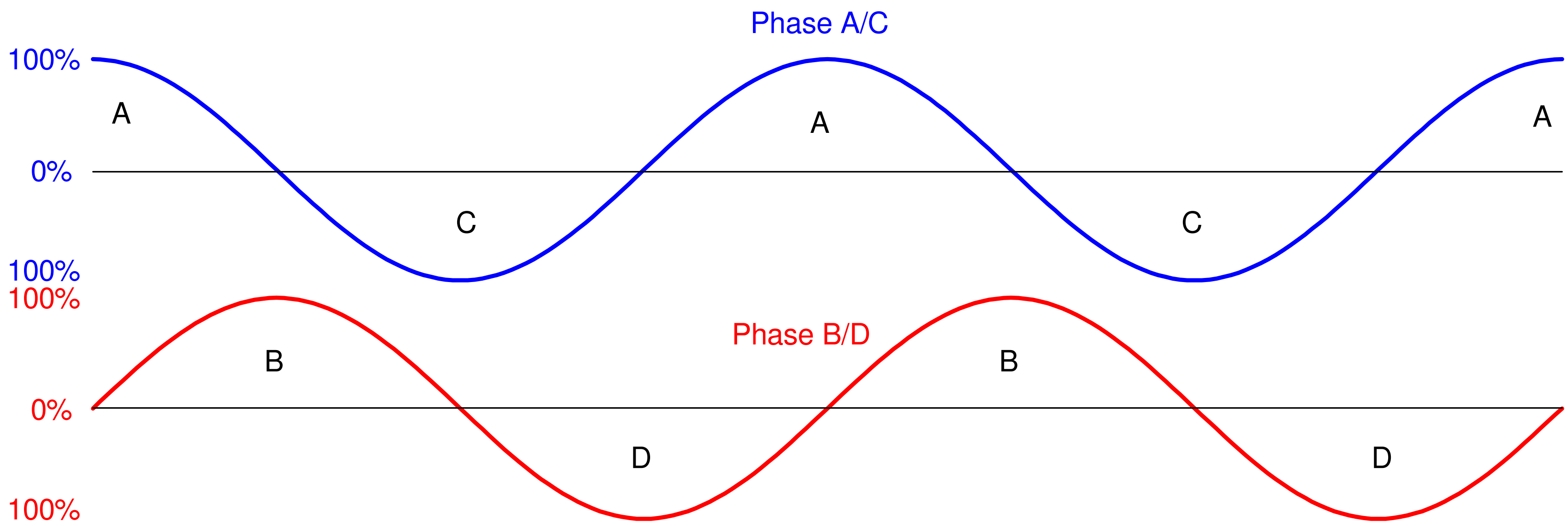
```
# Stepper Motor - Half Stepping

from time import sleep_ms
from machine import Pin

PA = Pin(16,Pin.OUT)
PB = Pin(17,Pin.OUT)
PC = Pin(18,Pin.OUT)
PD = Pin(19,Pin.OUT)

TABLE = [1, 3, 2, 6, 4, 12, 8, 9]

def Step(X):
    Y = TABLE[X % 8]
    PA.value( Y & 8 )
    PB.value( Y & 4 )
    PC.value( Y & 2 )
    PD.value( Y & 1 )

x = 0
for i in range(0,200):
    x += 1
    Step(x)
    sleep_ms(10)
```

Micro-Stepping:

A third option is to use PWM to approximate a sine and cosine wave.

- This is termed *micro-stepping*.
- The number of levels per cycle is arbitrary

PWM allows any voltage from 0% to 100%



MicroStepping: 16 Steps per Cycle

Initialization of the three outputs

- Phase A
- Phase B
- Phase C

Each is a PWM output

- 100Hz
- Duty cycle will vary

Result is 800 steps per rotation

- 50 x 16

```
from machine import Pin, PWM
from time import sleep_ms

PA = Pin(16,Pin.OUT)
PA = PWM(Pin(16))
PA.freq(100)
PA.duty_u16(0)

PB = Pin(17,Pin.OUT)
PB = PWM(Pin(17))
PB.freq(100)
PB.duty_u16(0)

PC = Pin(18,Pin.OUT)
PC = PWM(Pin(18))
PC.freq(100)
PC.duty_u16(0)

PD = Pin(19,Pin.OUT)
PD = PWM(Pin(19))
PD.freq(100)
PD.duty_u16(0)
```

MicroStepping (cont'd)

A table specifies the PWM signal

- 1/2 wave rectified sine wave
- 0 = 0%
- 65,535 = 100%

Each phase is offset

- A = 0 degree delay
- B = 90 degree delay
- C = 180 degree delay
- D = 270 degree delay

```
TABLE16 = [0, 24874, 45962, 60052, 65000,
60052, 45962, 24874,  0,   0,   0, 0, 0, 0,
0, 0]

def Step16(X):
    A = TABLE16[X % 16]
    PA.duty_u16(A)
    B = TABLE16[(X+4) % 16]
    PB.duty_u16(B)
    C = TABLE16[(X+8) % 16]
    PC.duty_u16(C)
    D = TABLE16[(X+12) % 16]
    PD.duty_u16(D)

x = 0
for i in range(0,800):
    x += 1
    Step16(x)
    sleep_ms(5)

PA.duty_u16(0)
PB.duty_u16(0)
PC.duty_u16(0)
PD.duty_u16(0)
```

MicroStepping with 32 steps per cycle

Use a bigger table and you get finer resolution

- pre-compute sin(x)
- faster program execution

Result is 1600 steps / rotation

- 50 x 32

No limit on the number of steps per cycle

```
TABLE32 = [0, 12681, 24874, 36112,
45962, 54046, 60052, 63751, 65000,
63751, 60052, 54046, 45962, 36112,
24874, 12681, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0]

def Step32(X):
    A = TABLE32[X % 32]
    PA.duty_u16(A)
    B = TABLE32[(X+8) % 32]
    PB.duty_u16(B)
    C = TABLE32[(X+16) % 32]
    PC.duty_u16(C)
    D = TABLE32[(X+24) % 32]
    PD.duty_u16(D)

x = 0
for i in range(0,1600):
    x += 1
    Step32(x)
    sleep_ms(5)

PA.duty_u16(0)
PB.duty_u16(0)
PC.duty_u16(0)
PD.duty_u16(0)
```

Solenoids

A solenoid is an electromagnet which can either pull or push a rod back and forth. Think of it as an electronic deadbolt:

- When de-energized, the deadbolt locks the door.
- When energized, the deadbolt is pulled back, allowing the door to open.

Since this is an of/off device, a simple binary output from the Pico can be used.

Sample Solenoid: Applying 12V to the leads draws 1A and applies 60N of force

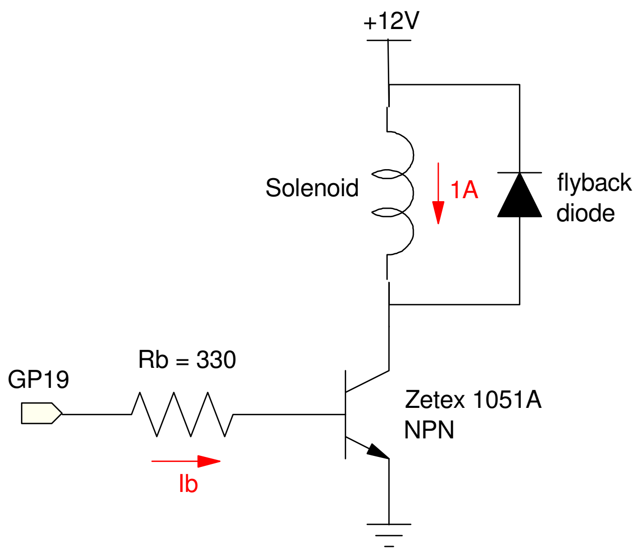
Assume for example a uxcell 12V solenoid is to be driven by a Pi-Pico.

To turn on this solenoid, you need:

- V = 12V
- I = 1A @ 12V

Since a Pi-Pico can't output 12V or 1A directly, add a transistor switch (assume a ZTX1051A NPN transistor).

- Digikey Part: ZTX1051A
- Ic(max) = 4A
- DC Current Gain (min): 300 @ 1A, 2V
- Vce(sat) = 210mV @ 1000mA
- $0.68 (qty 100)



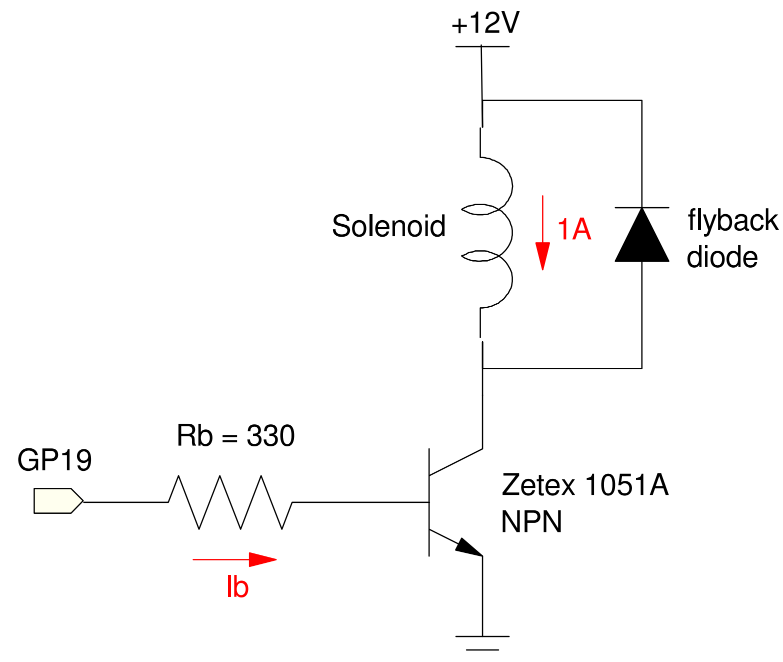
To saturate the transistor, you need

$$h_{fe} \cdot I_b > I_c$$

$$300 \cdot I_b > 1A$$

$$12mA > I_b > 3.33mA$$

(12mA is the max output of a PiPico)



Rb is then

$$R_b = \left(\frac{3.3V - 0.7V}{I_b}\right)$$

$$217\Omega < R_b < 780\Omega$$

Anything in this range should work.

- Let Rb = 330 Ohms.
- Add a flyback diode to save the transistor
  - inductive load

Solenoid Code:

A simple 1/0 on the output turns the solenoid on and off

Output a 1

- Solenoid turns on
- Lock open

Output a 0

- Solenoid turns off
- Lock closed

```
# Turning a solenoid on and off
from machine import Pin
from time import sleep

GP19 = Pin(19,Pin.OUT)

while(1):
    print('Solenoid On')
    GP19.value(1)
    sleep(1)
    print('Solenoid Off')
    GP19.value(0)
    sleep(1)
```

Brushless DC Motors

Brushless DC motors have become all the rage since about 2010

- quadcopters
- RC cars
- RC aircraft

They're small, inexpensive, and powerful

- $20 for the one shown
    - includes ESC controller
- 120W (10V @ 12A)

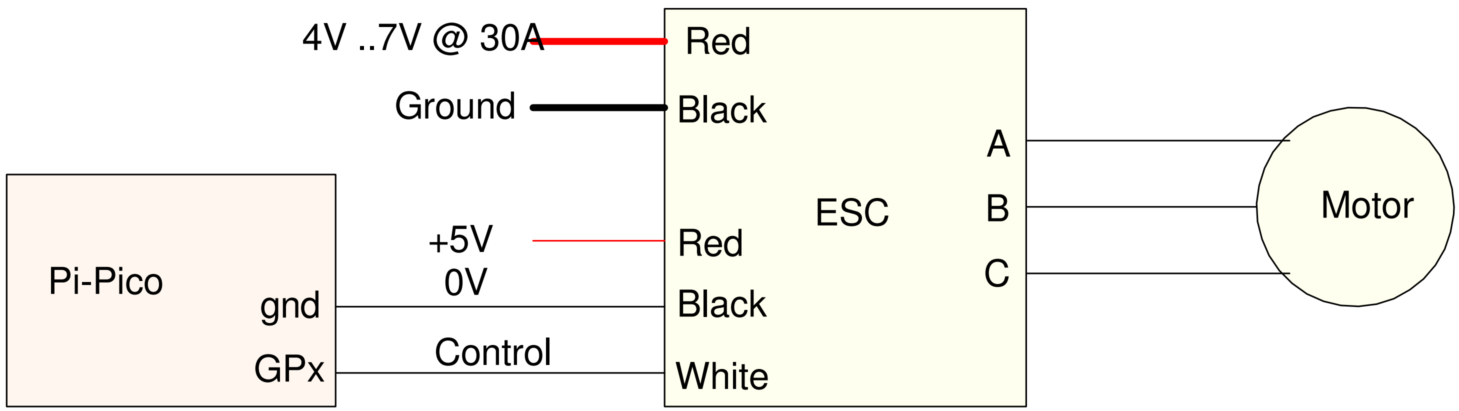
Larger ones are available

- $29 for 1000W (1.5hp)
- 46 grams

An ESC controller acts as the interface

Inputs:

- Red: +3.3V
- Black: Ground
- White: Control Signal from Pi-Pico

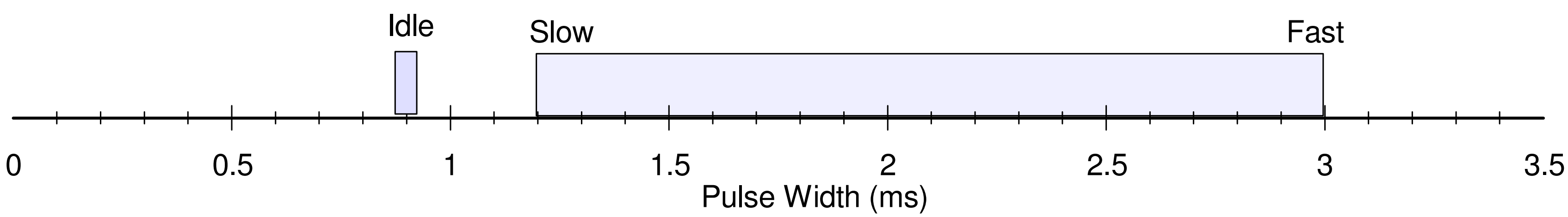


Connections from a Pi-Pico to a BLDC motor.

Controlling the Motor Speed

The control input is a square wave:

- Frequency = 50Hz to 330Hz
- Stop (power on): 0.9ms pulse
- Slow: 1.2ms pulse
- Fast: 3.0ms pulse



The speed of the BLDC motor is set by the pulse width on the Control line

Sample Code

Frequency = 50Hz

- T = 20ms

Startup:

- Pulse-Width = 0.9ms
- Press GP15 button to start

Running:

- slow = 1.2ms
- fast = 3.0ms
- proportional inbetween

```
from machine import Pin
from time import sleep

# GP15 = push button
# GP16 = control input to BLDC

Button = Pin(15, Pin.IN, Pin.PULL_UP)

Control = Pin(16, Pin.OUT)
Control = PWM(Pin(16))
Control.freq(50)

Control.duty_ns(900_000)
while(Button.value() == 1):
    pass

while(1):
    print('slow')
    Control.duty_ns(1_200_000)
    sleep(1)
    print('fast')
    Control.duty_ns(3_000_000)
    sleep(1)
```

Digital Servo Motor

Similar to a BLDC

- Motor is geared down
- Slower output speed
- Higher torque

Two Types:

- Output = Angle (shown)
  - Open / close a valve
  - Turn steering wheels
  - Pan & tilt camera
  - Specify the position of a robotic arm
- Output = Speed
  - 360 degree servo motor
  - Set the speed of an RC car

Digital Servo Motor: Control Input

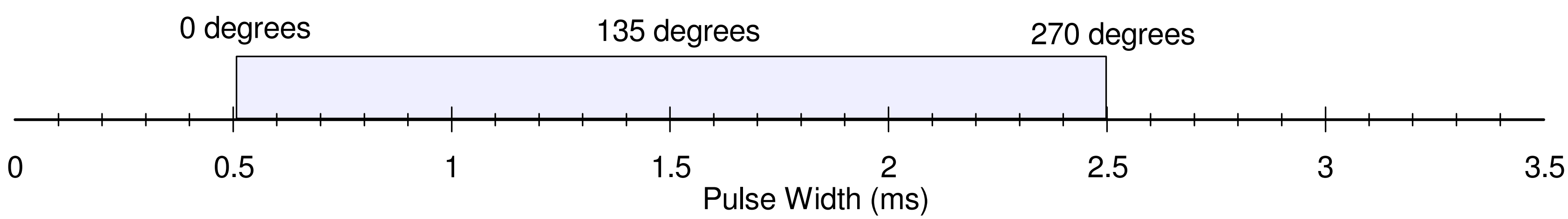
Similar to a BLDC: 3-wires

- Red: 5.0V to 6.8V, up to 3.0A (varies with the motor)
- Black: Ground
- White: Control Input

The control input controls the output

Example: 270 degree digital servo motor

- Frequency = 50 - 330Hz
- Pulse Width = 500 - 2500us

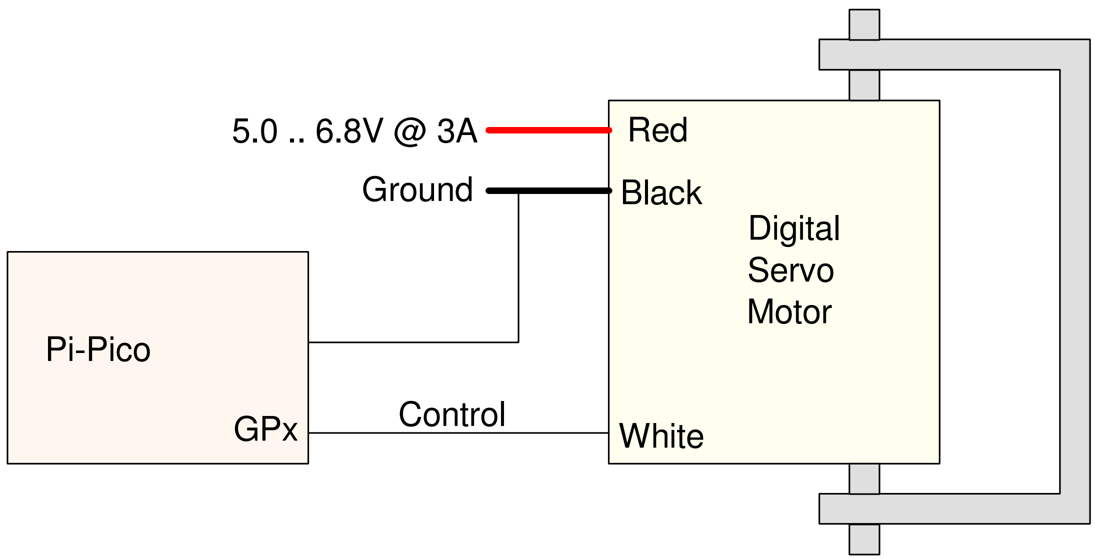


The pulse width sets the angle of the motor

Digital Servo Motor: Hardware

Wiring a Digital Servo Motor is fairly simple

- You need a common ground
- The Pi-Pico provides the control signal
  - PWM singal
- External power is applied to the red wire



Digital Servo Motor: Code

The code is almost the same as before.

This code has two input buttons

- GP14 decreases the angle
  - pulse width gets smaller
- GP15 increases the angle
  - pulse width gets larger

GP16 is the control input

```
from machine import Pin
from time import sleep_ms

# GP14 = push button (decrease angle)
# GP15 = push button (increae angle)
# GP16 = control input to digital motor

Up = Pin(15, Pin.IN, Pin.PULL_UP)
Down = Pin(14. Pin.IN, Pin.PULL_UP)

Control = Pin(16, Pin.OUT)
Control = PWM(Pin(16))
Control.freq(50)

x = 1_500_000

while(1):
    if(Up.value() == 0):
        x += 1000
    if(Down.value() == 0):
        x -= 1000
    if(x < 500_000):
        x = 500_000
    if(x > 2_500_000):
        x = 2_500_000)
    Control.duty_ns(x)
    sleep_ms(1)
```

Continuous Rotation Servo Motor

Used for driving a car, etc

For the example given here:

- Red = 4.8V - 6.0V
- No-Load Current: 100mA
- Stall Current: 550mA
- Pulse Width: 700 - 2300 us
- No-Load Speed: 110 rpm

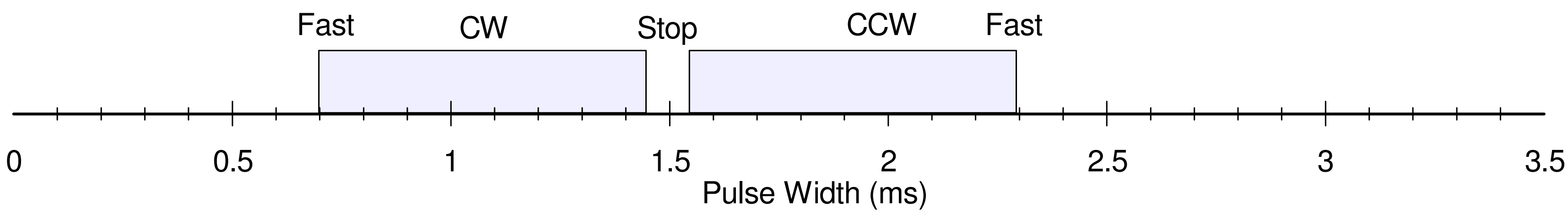
Price:

- $15 for two (2024 prices)

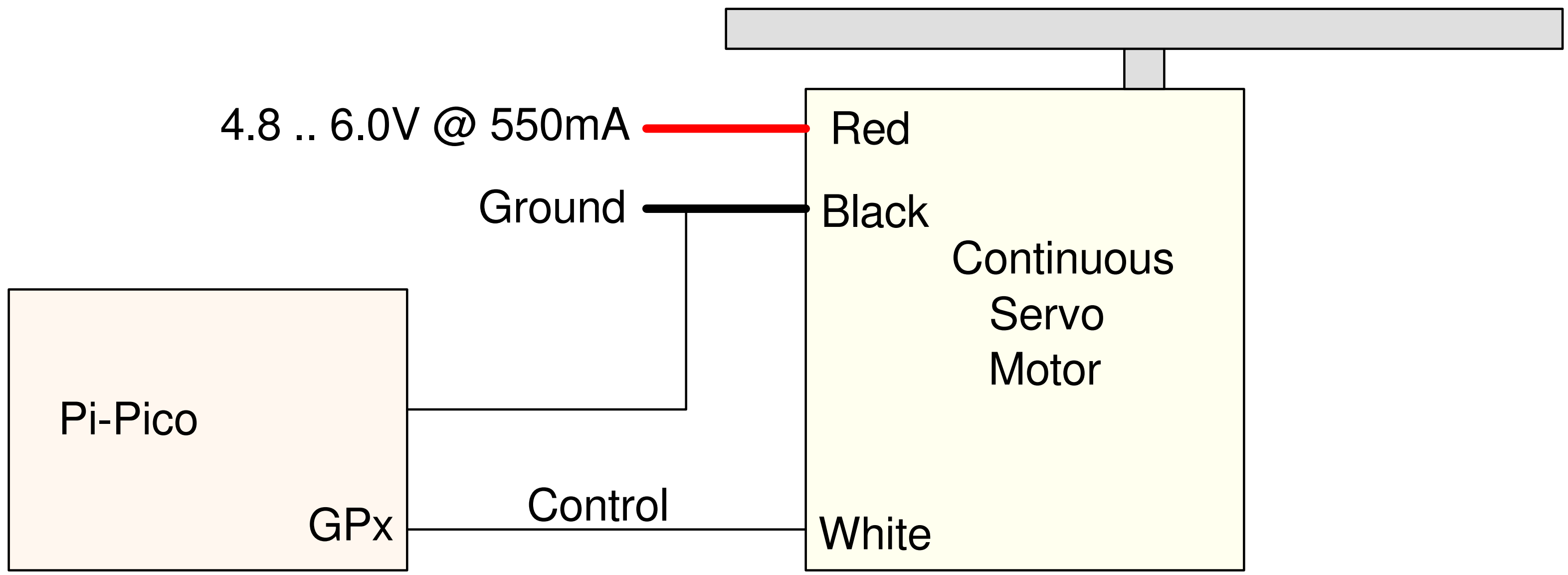
Control Signals

The pulse width sets the speed:

- CW: 1500us - 700us
- CCW: 1500us - 2300us
- Stop: 1500us +/- 45us



Wiring to a Pi-Pico is similar to before:



Wiring from a Pi-Pico to a Continuous Servo Motor

Code:

Coding is almost the same

- Button 15: Increase the speed
- Button 14: Decrease the speed
- GP16: Control input

Just set the limits to

- min = 700us
  - reverse fast
- max = 2300us
  - forward fast

```
from machine import Pin
from time import sleep_ms

# GP14 = push button (decrease angle)
# GP15 = push button (increae angle)
# GP16 = control input to motor

Up = Pin(15, Pin.IN, Pin.PULL_UP)
Down = Pin(14. Pin.IN, Pin.PULL_UP)

Control = Pin(16, Pin.OUT)
Control = PWM(Pin(16))
Control.freq(50)

x = 1_500_000

while(1):
    if(Up.value() == 0):
        x += 1000
    if(Down.value() == 0):
        x -= 1000
    if(x < 700_000):
        x = 700_000
    if(x > 2_300_000):
        x = 2_300_000)

    Control.duty_ns(x)

    sleep_ms(1)
```

Summary:

Digital motors are pretty easy to interface with a Pi-Pico:

- With a stepper motor, you mimic a 2-phase sine wave with four wires from the Pi-Pico
- With digital servo motors, you control the speed with a pulse width.

Note that these motors are low-power:

- The stepper motor draws 3A @ 5V, meaning 15W
- The digital servo motor draws 2A @ 5V, meaning 10W
- The continuous servo motor draws 550mA @ 5V, meaning 2.7W

Subtract losses in the motors and the power these can deliver is fairly small. If that's all you need, however, these are easy ways to interface motors to a Pi-Pico.