11: Motors with Analog Inputs

Introduction:

In the previous lecture, different types of motors with digital interfaces to a Pi-Pico were presented. Other than the BLDC motor, these were fairly low power:

- BLDC: 6.0V @ 50A = 300W
- Stepper Motor: 5V @ 3A = 15W
- Digital Servo Motor: 5V @ 2A = 10W

If you need motor power, larger motors are needed. These typically have analog inputs.

This lecture looks at two types of motors and ways to interface these to a Pi-Pico:

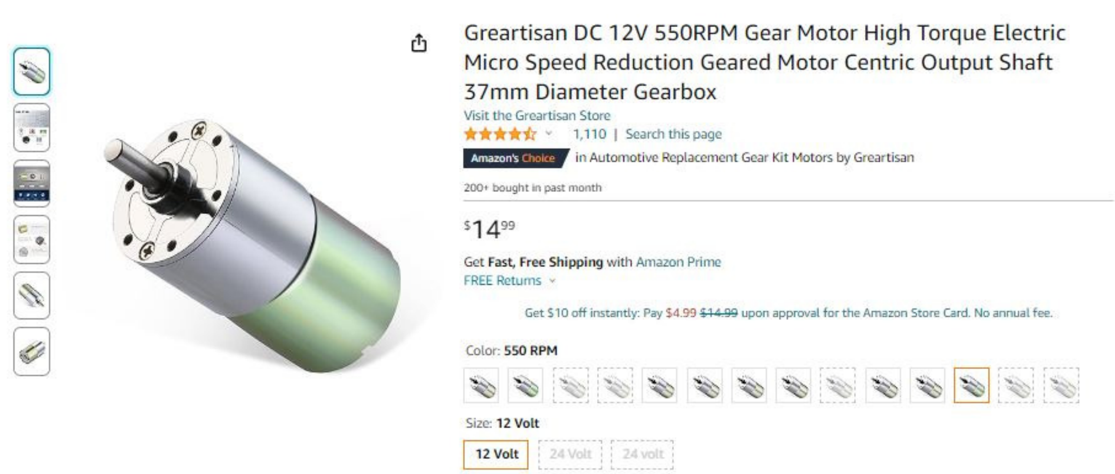
- DC Servo Motors
- 3-Phase AC Synchronous Motors

DC Servo Motors (6W)

A DC servo motor is the motor you're probably familiar with:

- If you apply a DC voltage to the motor, it spins.
- If you increase the DC voltage, it spins faster.

These are also the oldest type of electrical motor, dating back to 1900 when they were called *dynamos.*



12V, 6 Watt (500mA) DC Servo Motor

In terms of hardware, there are several options. If you need less than 6 Watts of power, a good choice is a Greartisan DC Gearhead Motor (below). This is a DC servo motor with a gear attached - allowing the no-load speed to vary from 5rpm to 600rpm.

In terms of the motor driver, what you need depends upon whether the motor just goes one direction or whether it can go forwards and in reverse.

Uni-Directional Hardware: If the motor just spins in one direction, a simple BJT transistor switch can be used to connect the Pi-Pico to the motor. Assuming a Zetex 1051A again, the base current needs to be:

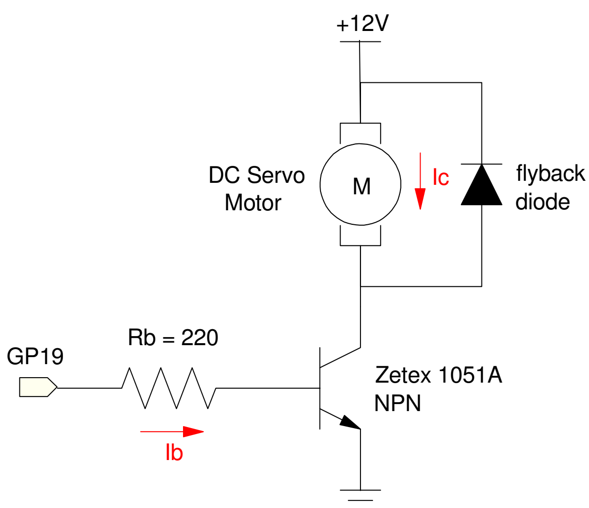
$$I_c(\max) = 500mA$$

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

$$12mA > I_b > \left(\frac{500mA}{300}\right) = 2.67mA$$

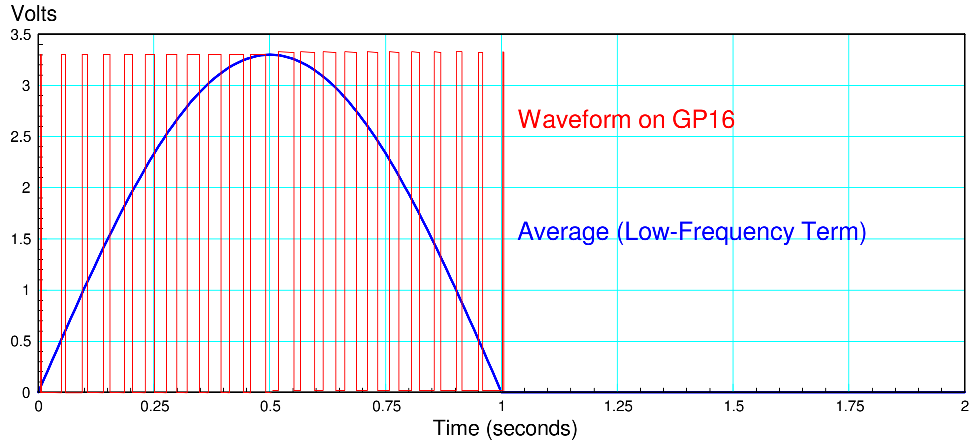
$$R_b = \left(\frac{3.3V - 0.7V}{I_b}\right)$$

$$216\Omega < R_b < 975\Omega$$



Hardware Connection for driving a DC motor in one direction
This configuration can be used for loads up to 3.5A

Pulse-width modulation can then be used to control the speed of the motor. For example, to make the motor speed up and slow down as follows:



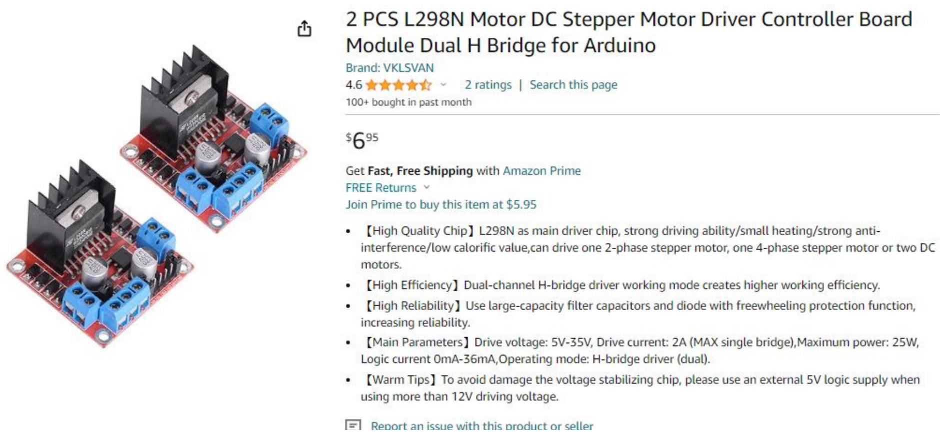
PWM used on GP16 to vary the speed of the DC motor

The following code can be used.

- Note: this is almost the same code used before. The only difference is increasing the PWM frequency to 20kHz. The frequency of the PWM produces an audible sound at the motor. Keeping this above 20kHz keeps this out of the audible range.

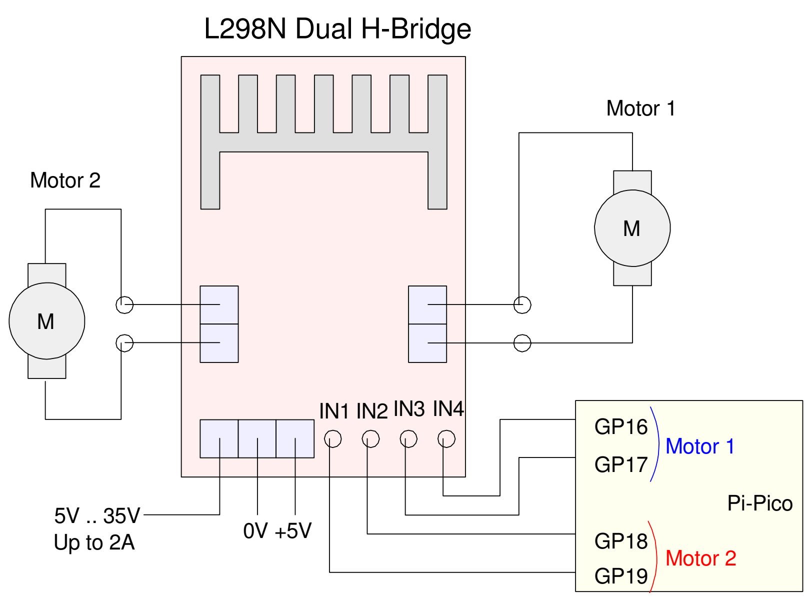
```
from machine import Pin, PWM
from time import sleep_ms
from math import sin, pi

Aout = Pin(16, Pin.OUT)
Aout = PWM(Pin(16))
Aout.freq (20_000)

Table = []
for i in range(0,100):
    Table.append(int(65535*sin(i*pi/100)))
for i in range(0,100):
    Table.append(0)

i = 0
while(1):
    i = (i + 1) % 200
    Aout.duty_u16(Table[i])
    sleep_ms(10)
```

Bidirectional Motion: If you need the motor to be able to spin both directions, then an H-bridge can be used along with PWM.



L298N dual H-bridge drivers can be used to drive the motor forward and reverse

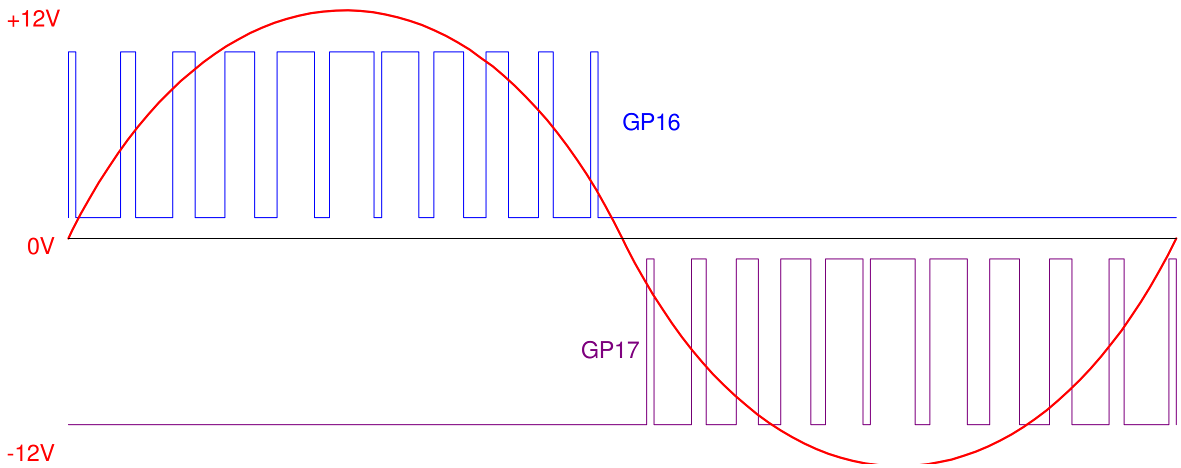
The hardware connection is similar to what was used for a stepper motor:



Two DC motors can be driven with a dual H-bridge driver

In terms of software, PWM again sets the speed of the motor with the direction set by which pin gets the PWM signal

	GP16	GP17
Forward	PWM	0V
Reverse	0V	PWM



The DC motor can be driven at 0..+12V using PWM on GP16 and 0V..-12V using PWM on GP17

The code is almost the same as before:

```
from machine import Pin, PWM
from time import sleep_ms
from math import sin, pi

Aout = Pin(16, Pin.OUT)
Aout = PWM(Pin(16))
Aout.freq (20_000)

Aout = Pin(17, Pin.OUT)
Aout = PWM(Pin(16))
Aout.freq (20_000

Table = []
for i in range(0,100):
    Table.append(int(65535*sin(i*pi/100)))
for i in range(0,100):
    Table.append(0)

i = 0
while(1):
    i = (i + 1) % 200
    if(Table[i] > 0):
        Aout.duty_u16(Table[i])
        Bout.duty_u16(0)
    else:
        Aout.duty_u16(0)
        Bout.duty_u16(-Table[i])
    sleep_ms(10)
```

DC Servo Motors (100W - 1000W)

If you need more power, use a different DC servo motor. The following for example is a motor for an

24V, 300 Watt DC motor (20A). Motor weight = 4 Lbs

In terms of hardware, everything remains the same except you need to increase the current capability to 20A.

For unidirectional motion, a power MOSFET can be used. Searching Digikey for

- MOSFET
- N-Channel
- 30A+
- Through Hole

results in 1,252 options. Selecting an IRFB7545PBF Mosfet

- Vds(max) = 60V
- Rds(on) = 5.9mOhm @ 57A @ 10V
- 4010pF @ 25V
- Vgs(th) (max) = 3.7V
- Ids(max) = 95A

What this tells you is:

- If you set Vgs < 3.7V, the MOSFET turns off (good)
- If you set Vgs = 10V, the on resistance is 5.9mOhm (good)
- At max current (20A), the Mosfet drops V = IR = 11.8mV and dissipates 236mW (good)
- This Mosfet can handle up to 95A (overkill)

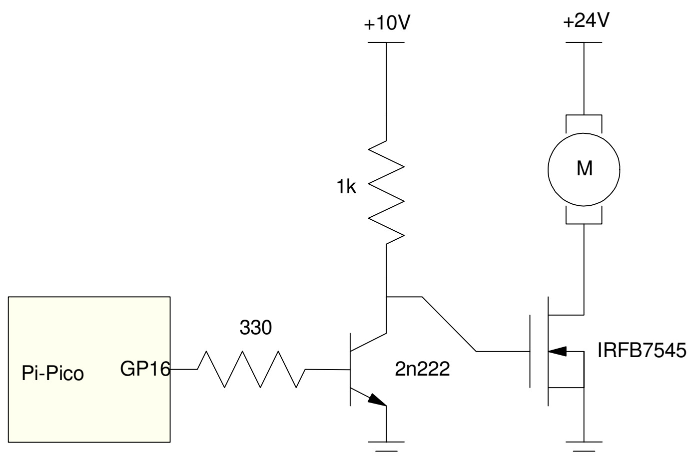
Mosfets will almost always out-perform BJT transistors.

IRFB7545PBF

DigiKey Part Number	448-IRFB7545PBF-ND
Manufacturer	Infineon Technologies
Manufacturer Product Number	IRFB7545PBF
Description	MOSFET N-CH 60V 95A TO220
Manufacturer Standard Lead Time	12 Weeks
Customer Reference	
Detailed Description	N-Channel 60 V 95A (Tc) 125W (Tc) Through Hole TO-220
Datasheet	Datasheet
EDA/CAD Models	IRFB7545PBF Models

One option for a MOSFET: An IRFB7545 is capable of 95A and 60V

A PIC can only output 3.3V. To bring this up to 10V, an NPN transistor can be used:

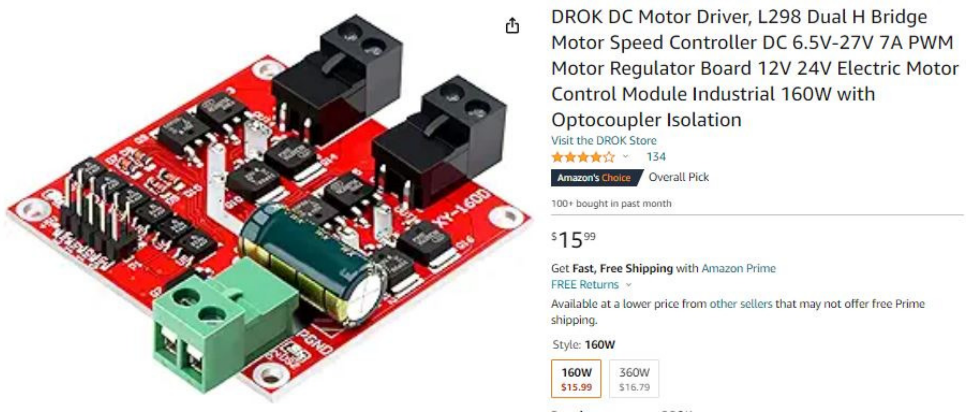


The capacitance of the Mosfet set the maximum switching frequency. The RC time constant is

$$t = RC = (1k\Omega)(4010pF) = 4.01\mu s$$

3 time constants is roughly how long it takes the Mosfet to turn on or off (121us). Inverting this circuit can run at up to 80kHz. Using 20kHz PWM should likewise be fine. (Same code as before).

For bi-directional motion, use an H-bridge capable of more current such as a DROK DC Motor Driver. Replace the previous H-bridge with this one and you're good to go for voltages up to 27V and currents up to 7A.

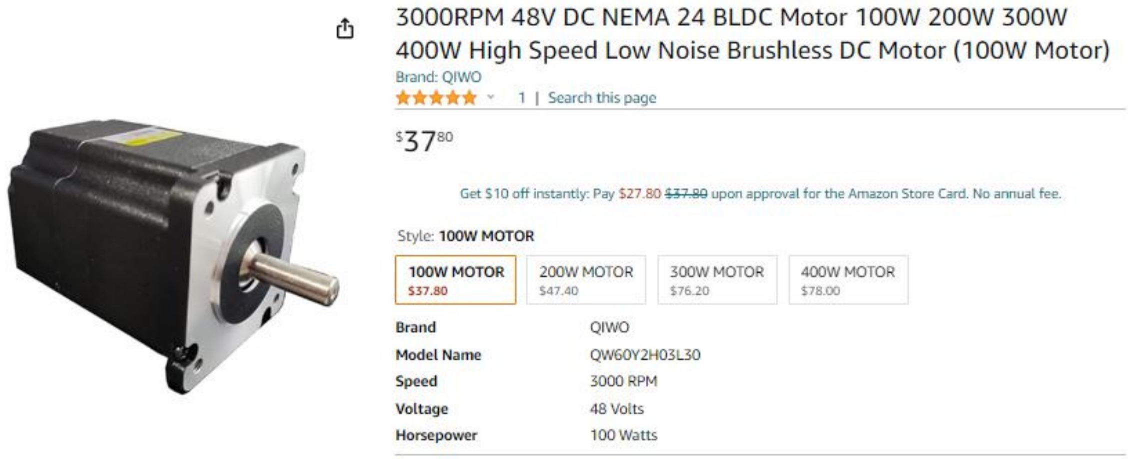


For even more power (up to 1200W)

- replace potentiometer with Pico output (voltage from PWM or D/A)
- replace SPDT switch with Pico output (voltage low or high for direction)

3-Phase AC Synchronous Motors (100W - 400W)

For larger motors, 3-Phase AC synchronous motors are usually the better option than DC motors. On Amazon, for example, you can buy a 400W motors for $78. To put this in perspective, a Tour-De-France athlete can output about 300W of power over long stretches and 1000W for a short burst. Put one of these motors on your bicycle and you can compete with world-class athletes.



3-Phase AC Synchronous Motor (BLDC motor)

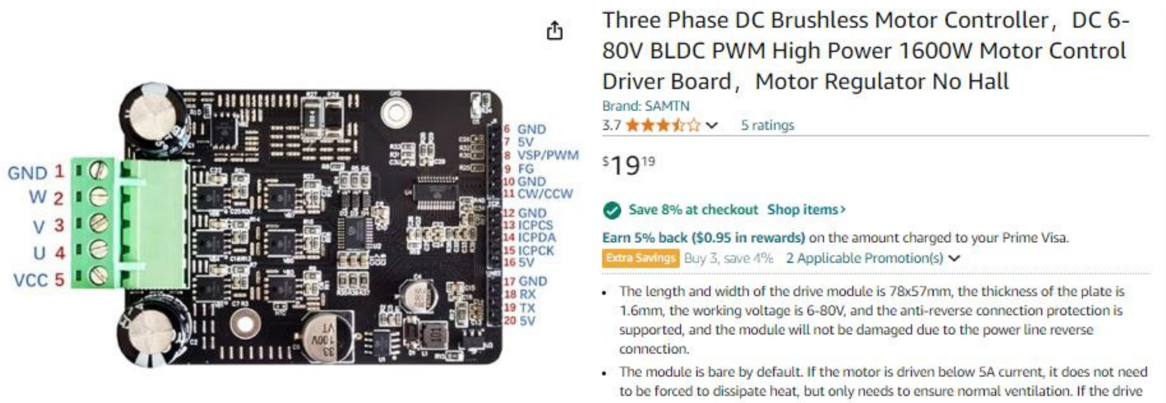
Ever since about the year 2000, DC motors have been replaced by 3-phase AC synchronous motors, also known as *brushless DC motors* (BLDC). The reason is size, cost, efficiency and life.

- AC motors only have one set of coils for the rotor. DC motors, in contrast, have six or more sets of coils in the rotor. This reduces the ripple in the torque and voltage, but it also creates a motor that's 6x heavier than its AC counterpart.
- AC motors do not need the commutatiors that DC motors need - so there's less parts to wear out
- AC motors are inherently more efficient. With DC motors, when you switch out a coil in the rotor, all of the energy stored in the rotor's inductance $\left(\frac{1}{2}LI^2\right)$ is lost.
- AC motors produce less RF interference. When the coils of a DC motor are switched out, a spark is created as the magnetic field collapses and the energy stored is dissipated. These sparks create RF interference, requiring greater shielding for DC motors.

The problem with 3-phase AC motors is the motor's input needs to be a 3-phase sine wave where the frequency sets the speed. Fortunately, driver boards can be purchases on Amazon for about $20. Several types are available. For the first board, the spped can be set with a

- PWM input,
- I2C input, or
- A 0-5V input

For the latter board, only the analog input is used. In addition, direction is set with an input pin.



BLDC Motor Driver with PWM, I2C, and SCI input. (Amazon)

BLDC Motor Driver with 0-5V Analog Input (Amazon)

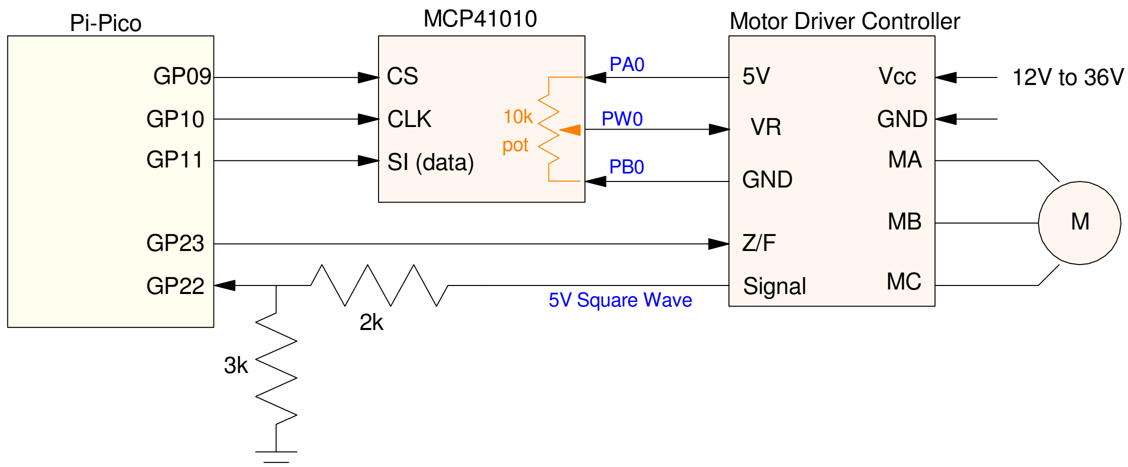
With the latter board, there are five inputs on the right side:

- 5V: 5V output
- Signal: Square wave output. Frequency indicates the speed of the BLDC motor
- Z/F: Direction control. 0V = CW, 5V = CCW (3.3V logic compatible)
- VR: 0V to 5V speed control (analog)
- GND: Common ground for all components

Normally, speed is set by connecting a potentiometer between 5V and ground, with the wiper going to VR. You can make this microprocessor-controlled by outputting an analog signal from the Pi-Pico. There are several ways to do this:

Option 1: Digital Potentiometer

One way to output 0-5V is to use a digital potentiometer. To manually control the speed of the motor, a potentiometer is connected between 5V and ground on the motor driver controller, with the wiper going to the speed control input (Z/F). This can be replaced with a digital potentiometer, such as a MCP41010.



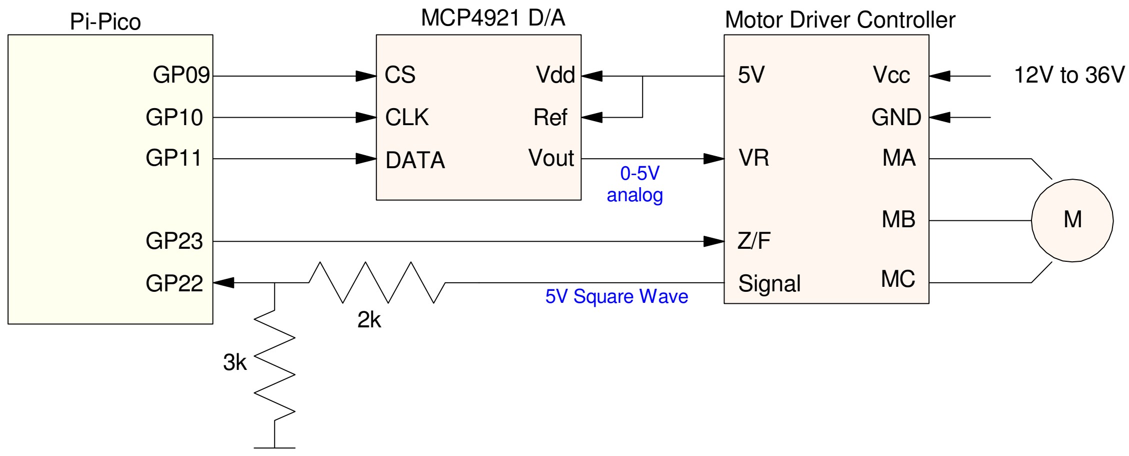
Driving a 3-phase motor using a digirtal potentiometer to output 0-5V to control the motor's speed

Option 2: D/A

Another way to output 0-5V analog is to use a D/A chip such as the MCP4921 (lecture #9). This uses

- Three output pins to drive the D/A using a SPI interface,
- A fourth output pin (GP23) to drive the motor direction (Z/F), and
- One input pin (GP22) to read the motor speed.

Note that the output of the motor driver is a 5V square wave on signals. To prevent damage to a Pi-Pico, this votlage needs to be dropped down to 3.3V.



Driving a 3-Phase Motor with a 12-bit D/A to set the motor speed

Option #3: PWM Output

A third option is to use the PWM output on the Pi-Pico. This output has two problems, however:

- The output chatters between 0V and 3.3V rather than being a constant output (what the ESC controller wants), and
- The maximum the output can be is 3.3V, whereas 5.0V is full speed.

These problems can be overcome using a 2nd-order low-pass filter - similar to what was done in lecture #9.

- If you set the DC gain of the filter to 1.56, 3.3V is converted to 5.15V
- If you use a 2nd-order filter with a corner at 100 rad/sec, the ripple should be decreased to 1.3mVpp

$$k = 1.56 = 1 + \frac{R_9}{R_8}$$

$$3 - k - 2\cos\theta$$

$$\theta = 43.9^0$$

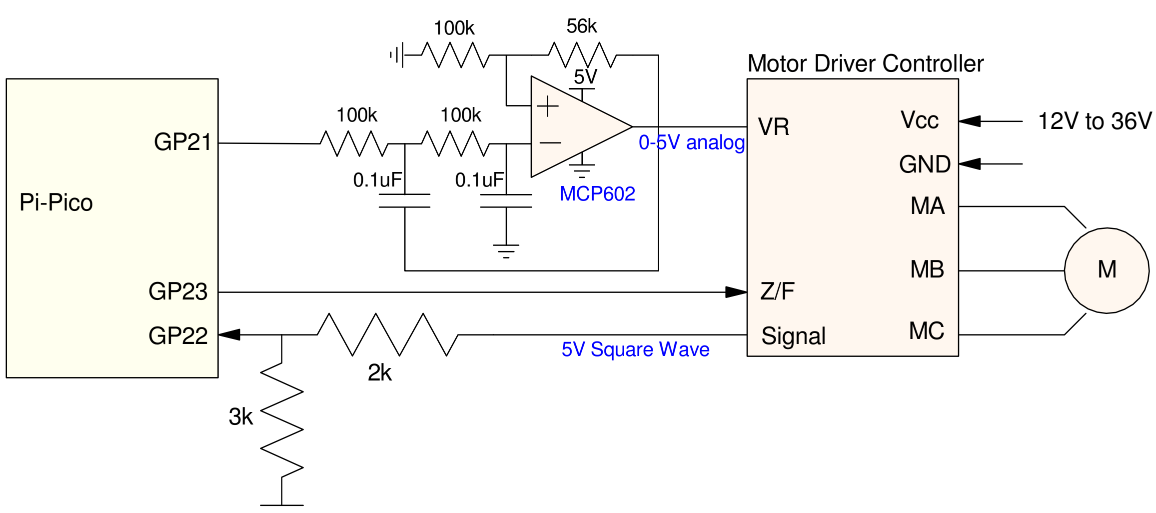
$$\frac{1}{RC} = 100$$

$$V_2 = \left(\frac{k \cdot 100^2}{s^2 + 144s + 100^2}\right) V_0$$

Assuming V0 is a 3.3Vpp sine wave at 1kHz (2000π - the ripple we're trying to remove)

$$V_2 \approx \left(\frac{1.56 \cdot 100^2}{s^2 + 144s + 100^2}\right)_{s = j2000\pi} \cdot 3.3V_{pp}$$

$$V_2 \approx 1.3mV_{pp}$$



Driving the ESC controller with PWM output along with a low-pass filter

Option #4: PWM Input

Eventually, someone will come up with an ESC controller with a PWM input. This will simplify the hardware somewhat. Until then, you can get by with any of the other three options to output a 0-5V analog signals.

For the rest of this lecture, we'll go with option #3 (PWM and low-pass filter),

Speed vs. Voltage Test

To see how this motor controller works, output a 0V to 5V signal and record the motor's speed as measured by the frequency on the *Signal* line.

- VR sets the direction (+3.3V for CW, 0V for CCW)
- A PWM signal sets the voltage on Signal (from 0% to 100% of 5V)
- The speed of the motor is monitored by measuring the frequency on the *Signal* line (GP22)

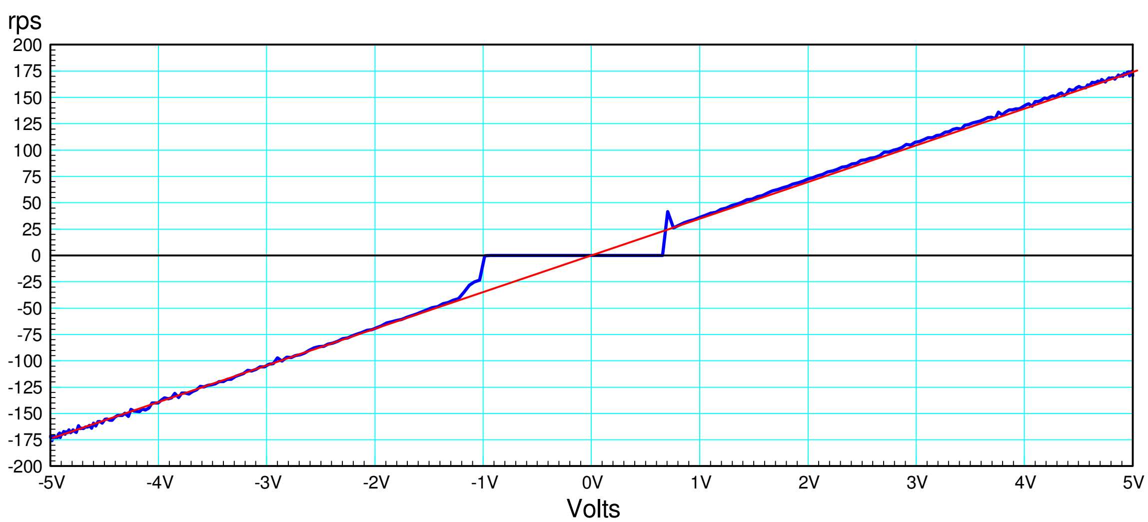
As can be seen in the figure below, the speed is almost a constant times voltage

$$\omega \approx \left(\frac{175}{5}\right) V \text{ rps}$$

There is a dead-zone however: you need at least 1V to turn on the motor. This dead-zone isn't too surprising and may actually be a good thing.

- When the motor is spining, it generates back-emf, limiting the input current.
- When the motor is stationary (or the speed is low), the back-emf will be almost zero.

In this case, only the armature resistance limits current to the motor. This may exceed the motor's rating and burn out the armature if you try to operate the motor a too low of a speed. So, having a dead-zone isn't all bad.



Experimental motor speed vs. input votlage (speed as measured by the frequency on the *Signal* line of the ESC controller)

Code: Only the main routine is shown below:

- The voltage is varied from -5V to +5V using a sine wave with a period of 62.83 seconds
- Edge interrupts count rising edges on GP22 (*Signal* from the ESC controller.)
- The interrupt records the period in the variable, dT
- The frequency in Hz (proportional to motor speed) is then found as

$$freq = \left(\frac{10^6}{period(us)}\right) Hz$$

```
t = 0
dt = 0.1
kv = 65535 / 5
Hz = 0

while(t < 64):

    V = 5*sin(t/10)

    if(V > 0):
       Vout.duty_u16(int(V*kv))
       Dir.value(0)
    else:
        Vout.duty_u16(int(-V*kv))
        Dir.value(1)

    Hz = 1_000_000 / dT

    print('{: 7.2f}'.format(V), '{: 7.2f}'.format(Hz))
    t += dt
    sleep(dt)
```

Code for recording motor speed vs. voltage

Step Response:

A second test you can run on this motor is the step response: apply a step input and look at the resulting speeed. In the figure below, two step inputs are applied:

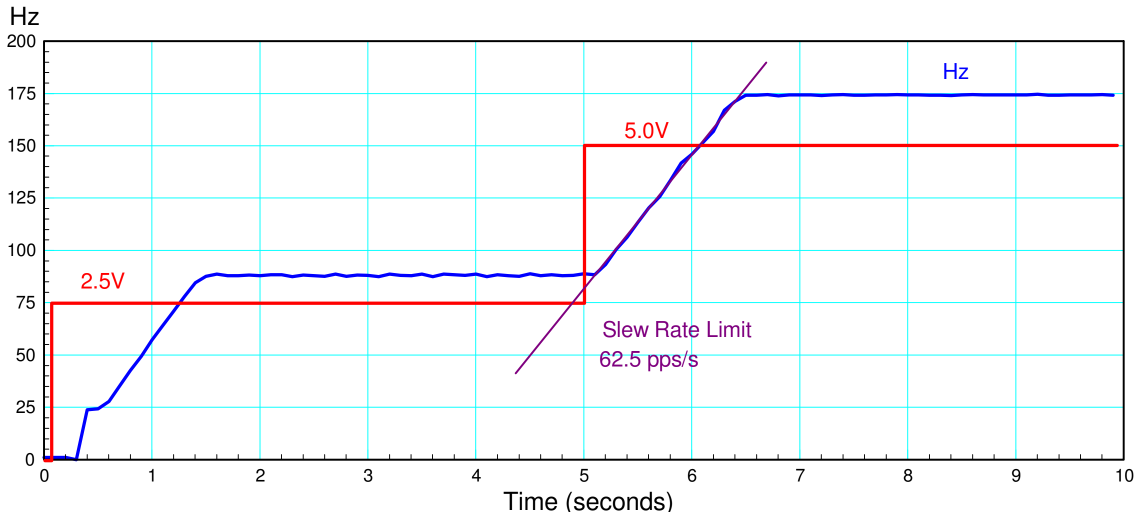
- A step input from 0V to 2.5V, and
- A step input from 2.5V to 5.0V

In the figure below, a couple of things are evident:

- The controller has a dead-zone for speed less than 25Hz, and
- The controller applies a slew-rate limit of 62.5Hz / sec

The slew-rate limit is also probably a good thing. AC synchronous motors only have torque when running at synchronous speed. If you stall the motor (or the motor's speed does not match the controller's output frequency), all torque is lost and hte motor stalls.

If you actually applied a step input to the motor (the input frequency suddenly changes), the motor would probably stall and lose all torque. This slew-rate limit allows the motor to follow the input frequency and keep operating when the desired speed changes.



Step Response of the AC Motor & ESC Controller

Code: This is the same code as before only with the input votlage changed to a step change at t=0 and t=5 seconds

```
t = 0
dt = 0.1
kv = 65535 / 5
Hz = 0

while(t < 10):
    if(t < 5):
        V = 2.5
    else:
        V = 5

    if(V > 0):
       Vout.duty_u16(int(V*kv))
       Dir.value(0)
    else:
        Vout.duty_u16(int(-V*kv))
        Dir.value(1)

    Hz = 1_000_000 / dT

    print('{: 7.2f}'.format(V), '{: 7.2f}'.format(Hz))
    t += dt
    sleep(dt)
```

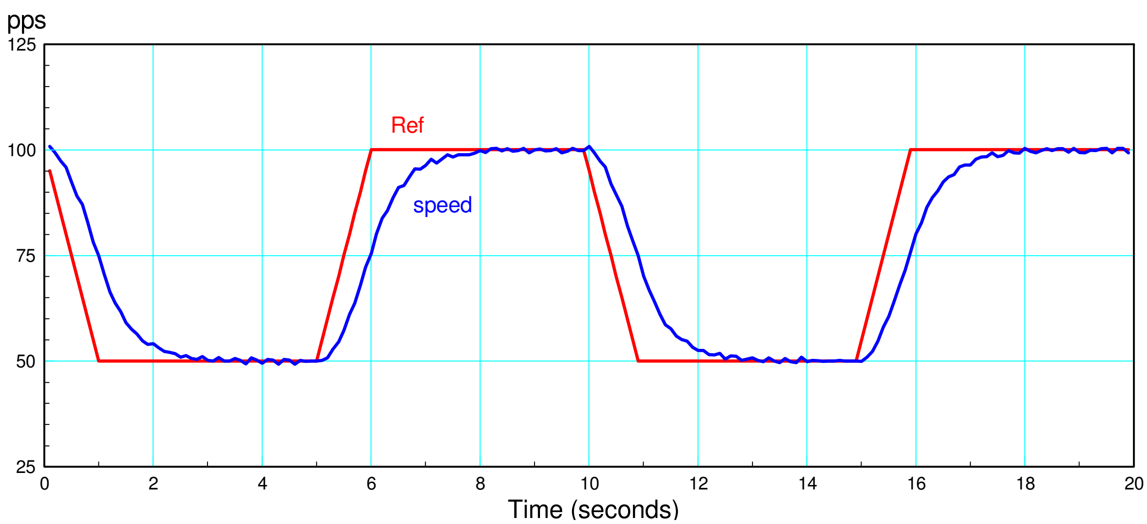
Code for a step response

Motor Speed Control

Finally, the speed of the motor can be controlled by using feedback and an integrator. The integrator serves as a search function: it integrates up and down, trying to find the votlage that makes the output track the set point. By trial and error, a decent response is obtained using

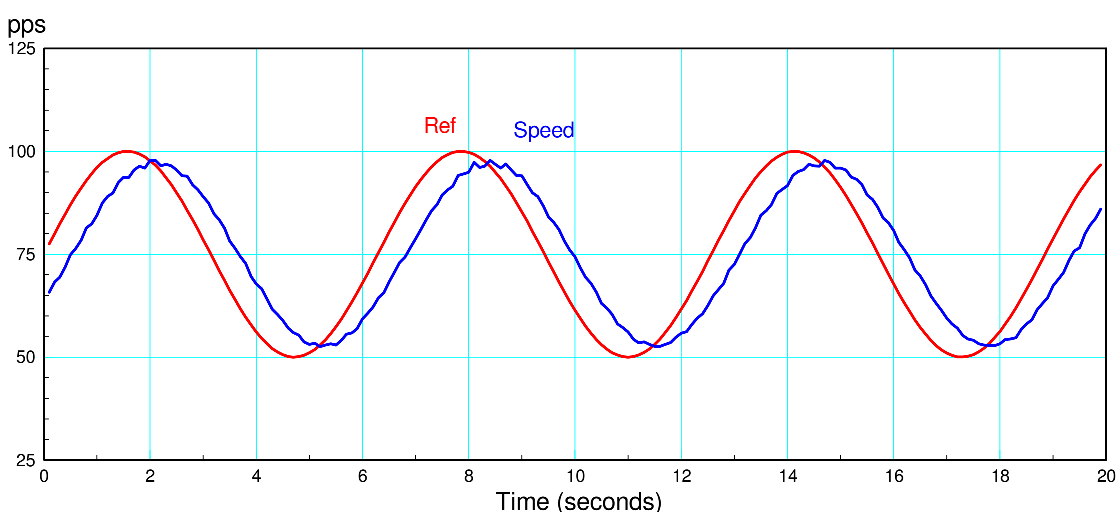
$$V = 0.05 \int (Ref - \omega) dt$$

The response for the set point (Ref) switching between 50 and 100Hz with a slew-rate limit of 50 rps/s for the set point is shown below. Note that right after the step change, the speed is changing at a constant rate. This is the slew-rate limit found before.



Step Response with I Control. The set point (Ref) is slew-rate limited to 50 rps/s

When tracking a 1 rad/sec sine wave, there is a slight time lag as well.



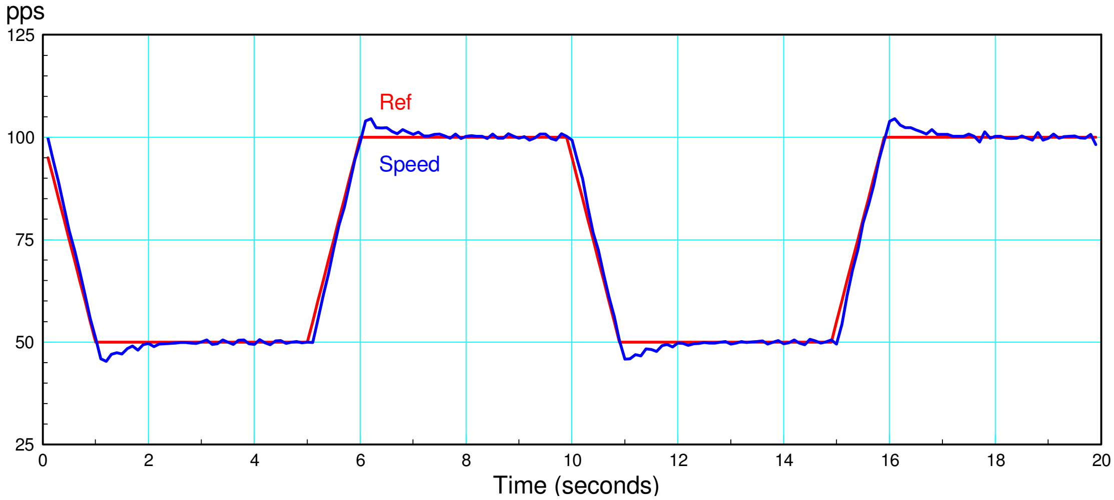
Tracking a 1 rad/sec sine wave with I control (I = 0.05).

A slightly better response can be obtained using a feed-forward term. From the static test, we know the relationship between speed and voltages

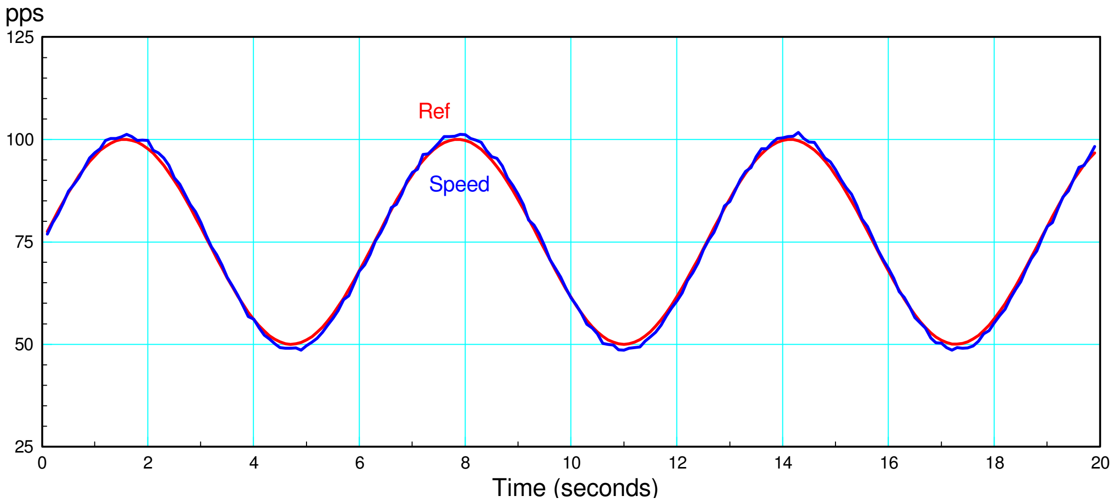
$$\omega \approx \left(\frac{175}{5}\right) V$$

Knowing this, the voltage to the motor can be improved by adding this as your initial guess. The feedback then adjusts the actual voltage to maintain speed:

$$V = \left(\frac{5}{175}\right) Ref + 0.05 \int (Ref - \omega) dt$$



Tracking a slew-rate limited step input with I + Feedforward Control



Tracking a 1 rad/sec sine wave with I + Feedforward Control

```
while(t < 20):

    if(sin(pi*t/5) > 0 ):
        Ref = min(50, Ref - 50*dt)
    else:
        Ref = max(100, Ref + 50*dt)

    Hz = 1000000/dT

    I += 0.05*(Ref - Hz)*dt
    V = I + Ref*5/175

    if(V > 0):
       Vout.duty_u16(int(V*kv))
       Dir.value(0)
    else:
        Vout.duty_u16(int(-V*kv))
        Dir.value(1)

    print('{: 7.2f}'.format(t), '{: 7.2f}'.format(Ref), '{:
7.2f}'.format(Hz), '{: 7.2f}'.format(V))
    t += dt
    sleep(dt)

print('Stop')
Vout.duty_u16(0)
```

I Control with a feed-forward term for controlling the motor's speed

Summary

Motors with analog inputs can be driven fairly easily with a Pi-Pico and a PWM output. The main thing you need is an H-bridge for DC motors or an ESC controller for an AC motor.

References

Pi-Pico and MicroPython

- https://github.com/geeekpi/pico_breakboard_kit
- https://micropython.org/download/RPI_PICO/
- https://learn.pimoroni.com/article/getting-started-with-pico
- https://www.w3schools.com/python/default.asp
- https://docs.micropython.org/en/latest/pyboard/tutorial/index.html
- https://docs.micropython.org/en/latest/library/index.html
- https://www.fredscave.com/02-about.html

Pi-Pico Breadboard Kit

- https://wiki.52pi.com/index.php?title=EP-0172

Other

- https://docs.sunfounder.com/projects/sensorkit-v2-pi/en/latest/
- https://electrocredible.com/raspberry-pi-pico-external-interrupts-button-micropython/
- https://peppe8o.com/adding-external-modules-to-micropython-with-raspberry-pi-pico/
- https://randomnerdtutorials.com/projects-raspberry-pi-pico/
- https://randomnerdtutorials.com/projects-esp32-esp8266-micropython/