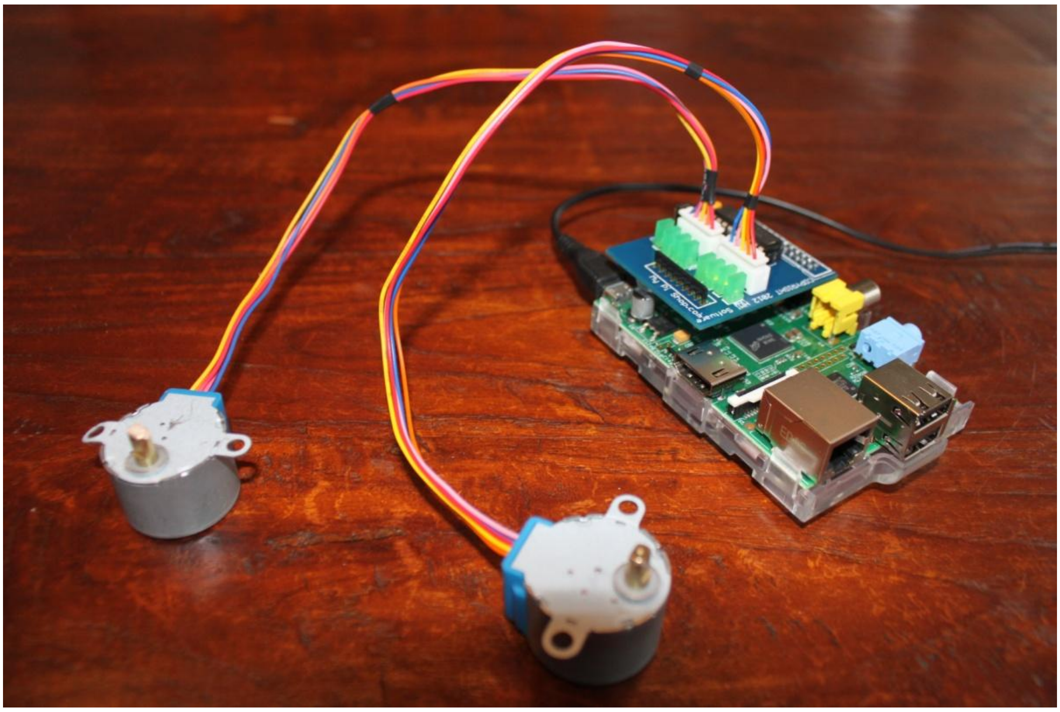
Raspberry Pi Stepper Motor

Constructors Manual



Bob Rathbone Computer Consultancy

www.bobrathbone.com

17th of September 2025

Version 2.5

Contents

Figures

Tables

Introduction

This project has been designed to help students or hobbyists get started with driving stepper motors on the Raspberry Pi. It covers two types of stepper motor namely unipolar and bipolar. The difference between these two is explained in the next section. The principal hardware required to run stepper motors on a Raspberry Pi consists of the following components:

- A Raspberry Pi computer (all models)
- A single or dual stepper motor driver board
- As alternative to the above an I2C interface can be used (28BYJ-48 stepper motors only)
- Object orientated Python3 driver code

Either

- One or two x 5-Wire "28BYJ-48" stepper motor (bipolar motor) with ULN2803A driver

Or

- A 12-volt #324 (Nema17) high torque stepper motor (unipolar motor) with H-Driver

Raspberry Pi computer

The **Raspberry Pi** is a credit-card-sized single-board computer developed in the United Kingdom by the Raspberry Pi Foundation with the intention of promoting the teaching of basic computer science in schools.

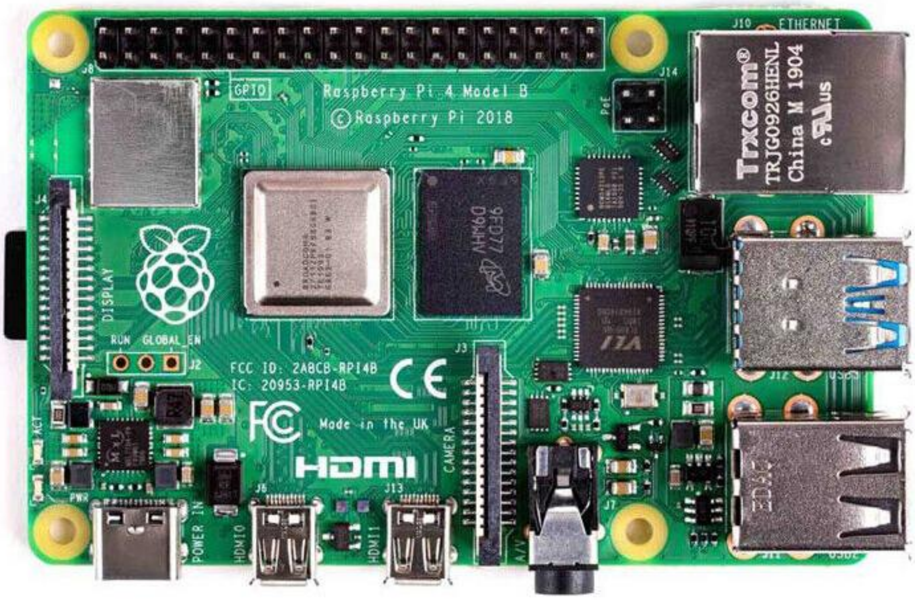


Figure 1 Raspberry Pi model 4B Computer

More information on the Raspberry Pi computer may be found here:
http://en.wikipedia.org/wiki/Raspberry_Pi

If you are new to the Raspberry Pi, try the following beginners guide.
http://elinux.org/RPi_Beginners

Stepper Motor Theory

Types of stepper motor

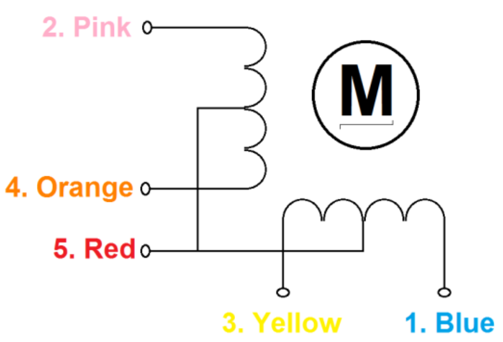
A good place to start is the following Wikipedia Article:
http://en.wikipedia.org/wiki/Stepper_motor

There are two types of stepper motor in popular use. These are:

1. Unipolar stepper motors typically driven using single transistors or Darlington pairs
2. Bipolar motors typically driven using an H-Bridge circuit

Unipolar stepper motors

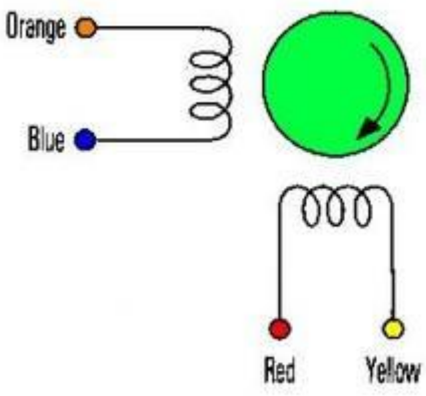
Figure 2 A 5-Wire 28BYJ-48 Stepper Motor Wiring



The 28BYJ-48 stepper motor is a so-called unipolar motor. A unipolar stepper motor has two or more windings, each with centre tap. Each section of windings is switched on for each direction of magnetic field. Since in this arrangement a magnetic pole can be reversed without switching the direction of current, the circuit can be made very simple (e.g., a single transistor) for each winding. In this project the ULN2803A Integrated Circuit is used. This is an eight Darlington pair driver circuit. These motors can normally be driven from 5-volt logic circuits.

Bipolar stepper motors

Figure 3 A 4-Wire Bipolar Stepper Motor Wiring



Bipolar stepper motors have a single winding per phase. The current in each winding needs to be reversed in order to reverse the magnetic pole, so the driving circuit is more complicated, typically with an **H-Bridge** arrangement, however there are several off-the-shelf driver chips available to make this a simple affair. This project is using the **A4988 H-Bridge** circuit driver board. There are two leads per phase, none are common. Bipolar motors are more efficient than unipolar motors as both phases are used at once. They can deliver higher torque and speed than a unipolar motor of the same weight. These motors usually require much higher currents than can be obtained from 5V logic (typically 10 times greater) and will require 8 to 12 volts or higher.

Unipolar driver waveforms

There are a number of ways to drive a unipolar stepper motor as shown below

Figure 4 Unipolar stepper motor drive methods

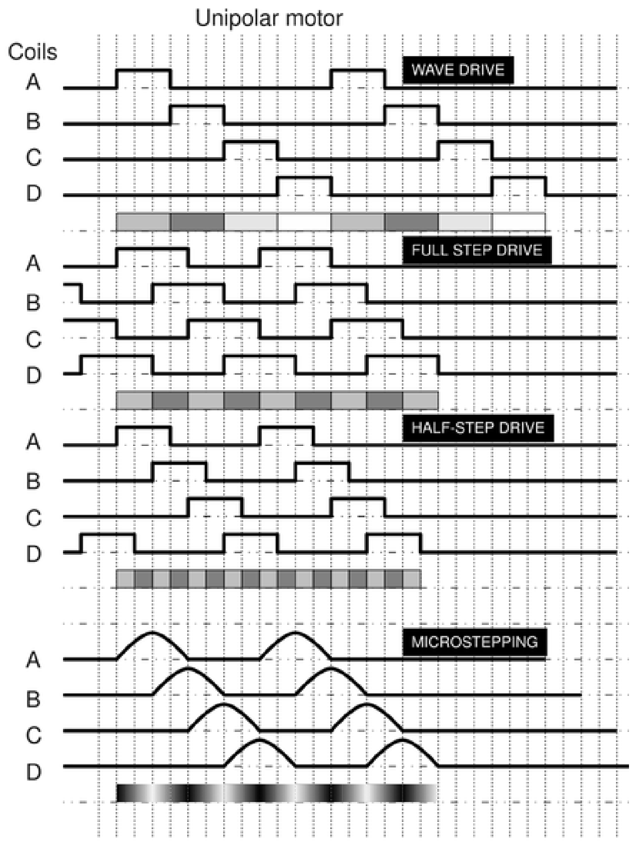


Figure 5 Unipolar driver waveforms

Wave drive

In this drive method, only a single phase is activated at a time. It is the fastest drive method but is rarely used.

Full step drive

This is the usual method for full step driving the motor. Two phases are always on so the motor will provide its maximum rated torque.

Half stepping

When half stepping, the drive alternates between two phases on and a single phase on. This increases the angular resolution, but the motor also has less torque.

Microstepping

Here the windings are driven with sinusoidal AC waveform to give smoother operation. This requires different hardware and isn't used in this project.

Bipolar driver waveforms

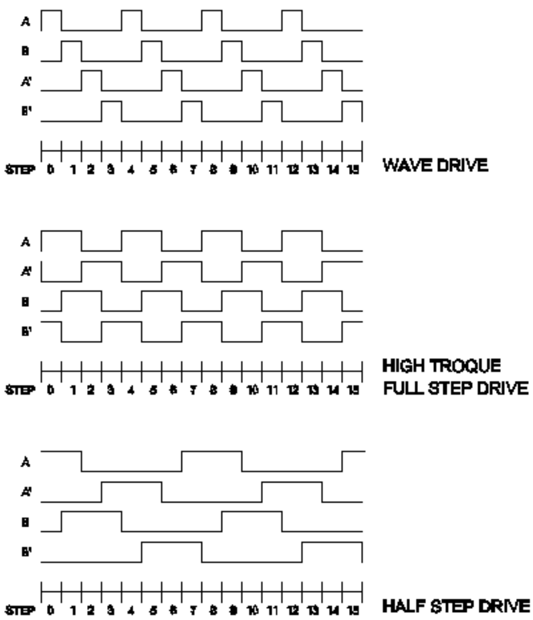


Figure 6 Bipolar driver waveforms

Wave drive

The wave drive is the simplest method where a pulse is applied to only one winding at a time.

Full step

The full step driving requires pulses are applied to two windings at a time which will provide higher torque.

Half step

The half step drive is alternately applying pulses to one and two windings. The stepper motor can move at a finer pitch and has twice the number of steps per revolution. However, the torque varies for each step which can cause more vibration.

Wiring and construction

Raspberry Pi 40-pin GPIO header

The following shows the pin outs for the GPIO for models 2B, 3B and 4B
See: http://elinux.org/RPi_Low-level_peripherals. For more details.

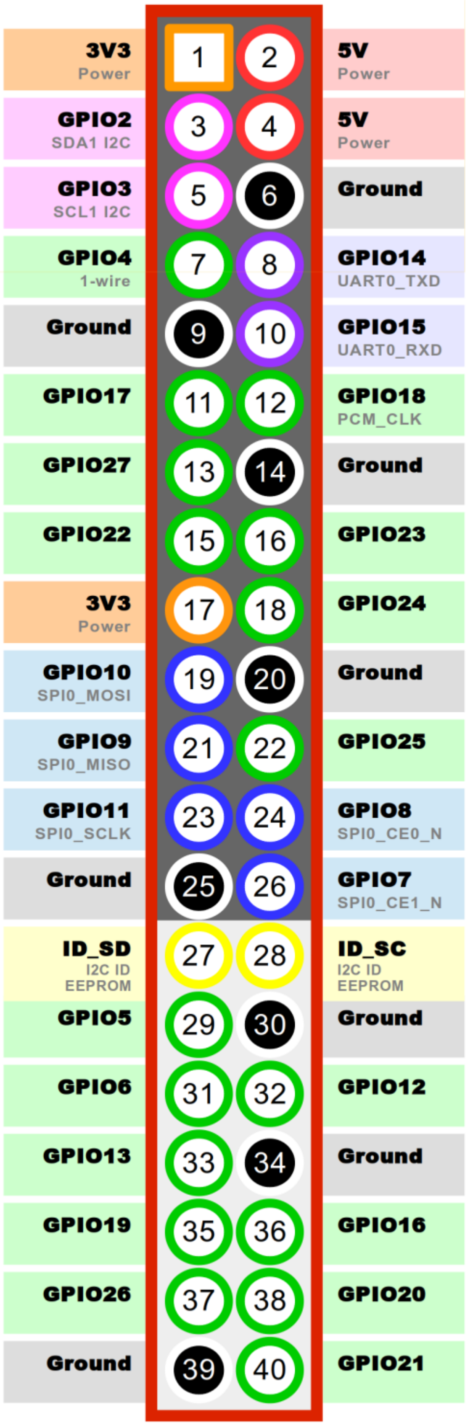


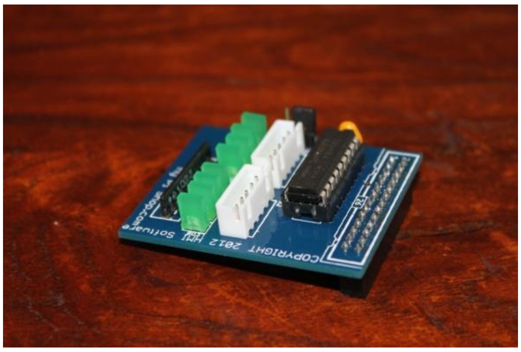
Figure 7 GPIO Numbers

The above diagram shows the 40 pin GPIO header viewed from above.

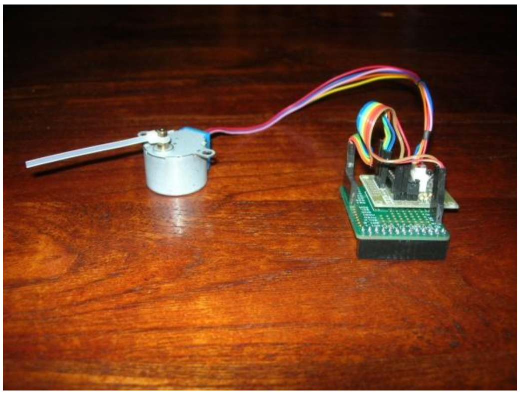
Unipolar Motor driver boards

ULN2803A Darlington pair driver boards

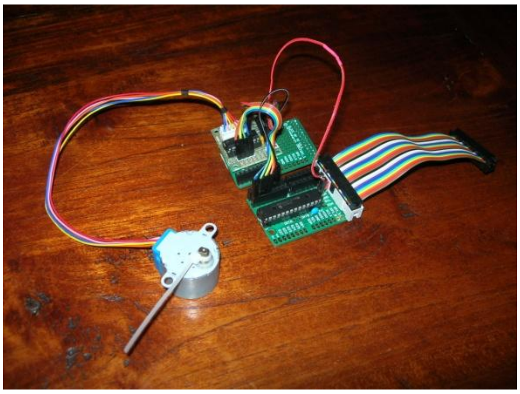
A number of stepper motor driver boards are available.



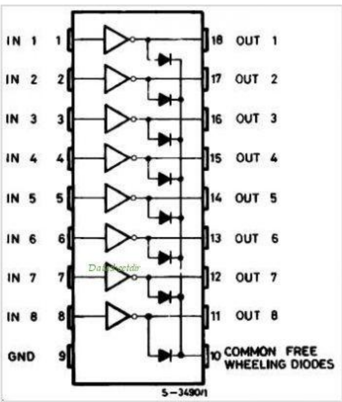
This board is available from ModMyPi and uses the ULN2803A Eight Darlington outputs Driver Chip. It can drive two stepper motors. This board can drive unipolar devices up to 50 volts.



This is an example of a single stepper motor driver board. It uses four outputs of a ULN2003A Seven Darlington outputs Driver Chip. It is piggy-backed on top of a slice of ModPi prototype board using a glue gun. This allows it to be plugged into the GPIO header of the Raspberry Pi.



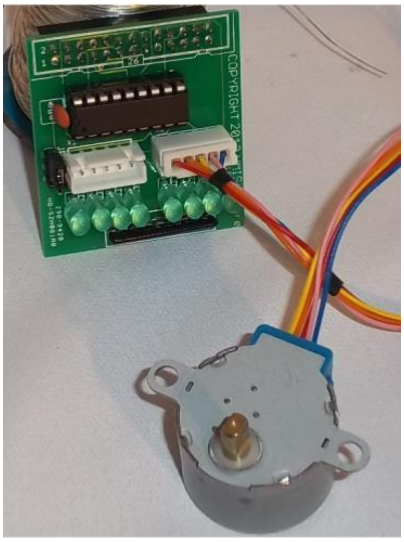
This example shows the I2C interface using a 16 port MCP23017 I/O expander available from mostRaspberry Pi suppliers. It is connecting to the above single stepper motor driver board. Sixteen I/O ports mean that this board can drive up to four motors using only two pins (I2C) on the Raspberry Pi. Since I2C can support up to eight devices many more motors can be driven. Unfortunately, this hardware appears no longer to be available but may be in the future.



The diagram on the left shows the ULN2803A Eight Darlington pair outputs Driver Chip. This chip can drive two bipolar stepper motors (four outputs are used for each motor.

Construction

Figure 8 Unipolar motor and driver board



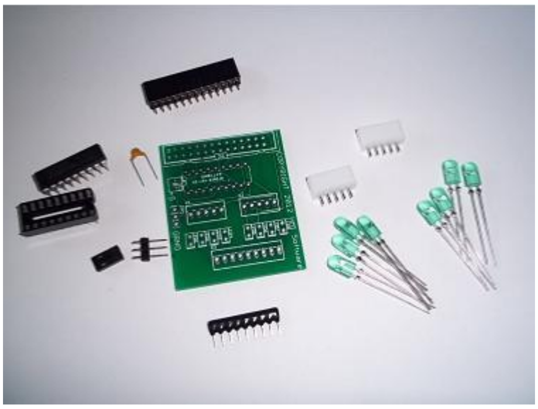
The unipolar stepper motor board uses 8 I/O pins to drive up to two stepper motors.

The jumper shipped with the board allows the stepper motor to use the +5V from the Raspberry Pi. If you want to use a different stepper you can remove the jumper and supply up to 12 volts to the centre pin and connect ground to the pin that had no connection.

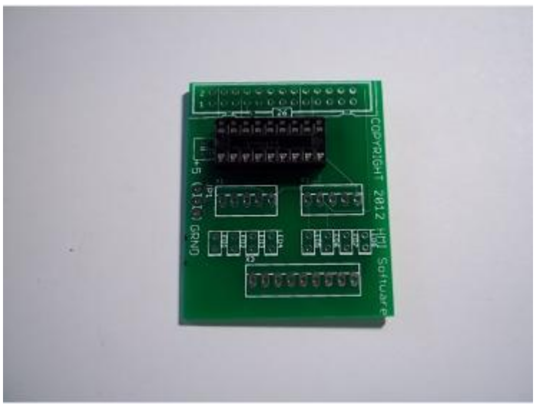
The left white five pin connector is for the first motor and the right connector is for the second motor.

Table 1 GPIO interface wiring

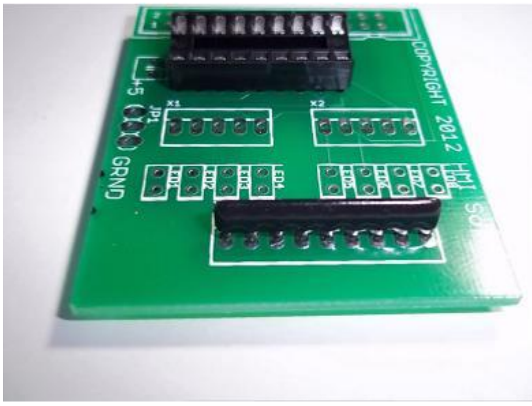
GPIO pin	Connector	Description
17	1	Motor 1 output 1
18	1	Motor 1 output 2
27	1	Motor 1 output 3
22	1	Motor 1 output 4
23	2	Motor 2 output 1
24	2	Motor 2 output 2
25	2	Motor 2 output 3
4	2	Motor 2 output 4



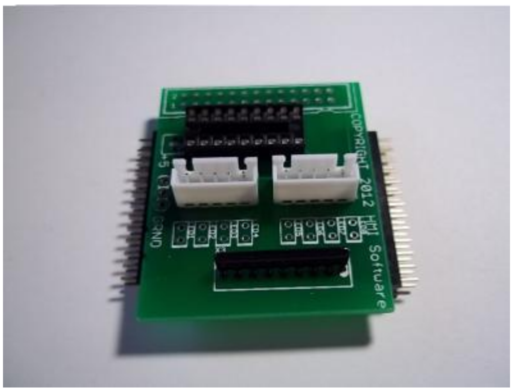
The driver board comes as a kit. Not shown is a small stick-on plastic pad to prevent the card shorting on the Raspberry Pi board.



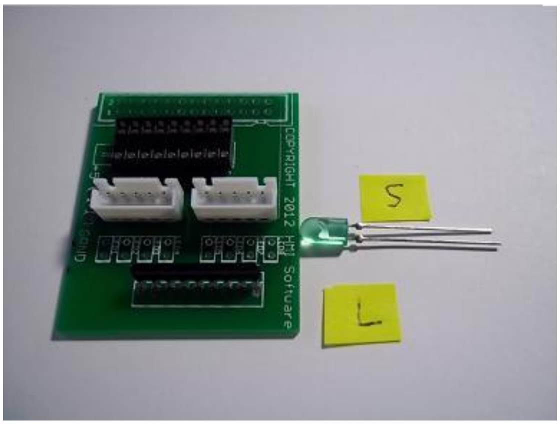
Install and solder first. Orientate the notch at one end of the socket to the left side of the board.



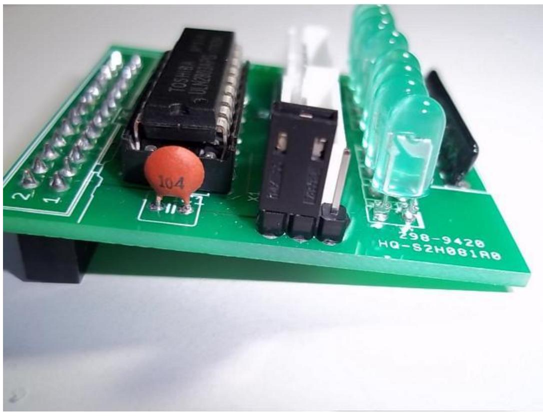
Install resistor pack. Be sure to orient it as shown. I.E. the text on the resistor pack cannot be seen from this side.



Install white connectors. Be sure to orient them as shown.



Install LED's. Be sure to orient them as shown. Long Leg towards resistor pack.



Insert and solder the capacitor. Solder the 26-pin female socket to the underside of the board as shown above. Insert the ULN2803A Motor Driver Chip with the notch towards the capacitor. Insert and solder the power supply pins and insert the jumper as shown. Finally stick the plastic pad underneath the resistor block in such a way that it rests on the power supply capacitor on the Raspberry Pi board.

Bipolar stepper motor A4988 driver

This project uses the **A4988** or **DRV8825** H-Circuit chip to drive the Nema17 stepper motor.

Note: Note that the motor requires a between 8 and 12 volts + connected to VMOT and GND. Take care not to accidentally connect it to VDD as this will destroy the Raspberry Pi.

Figure 9 A4988 driver circuit connection to Raspberry Pi

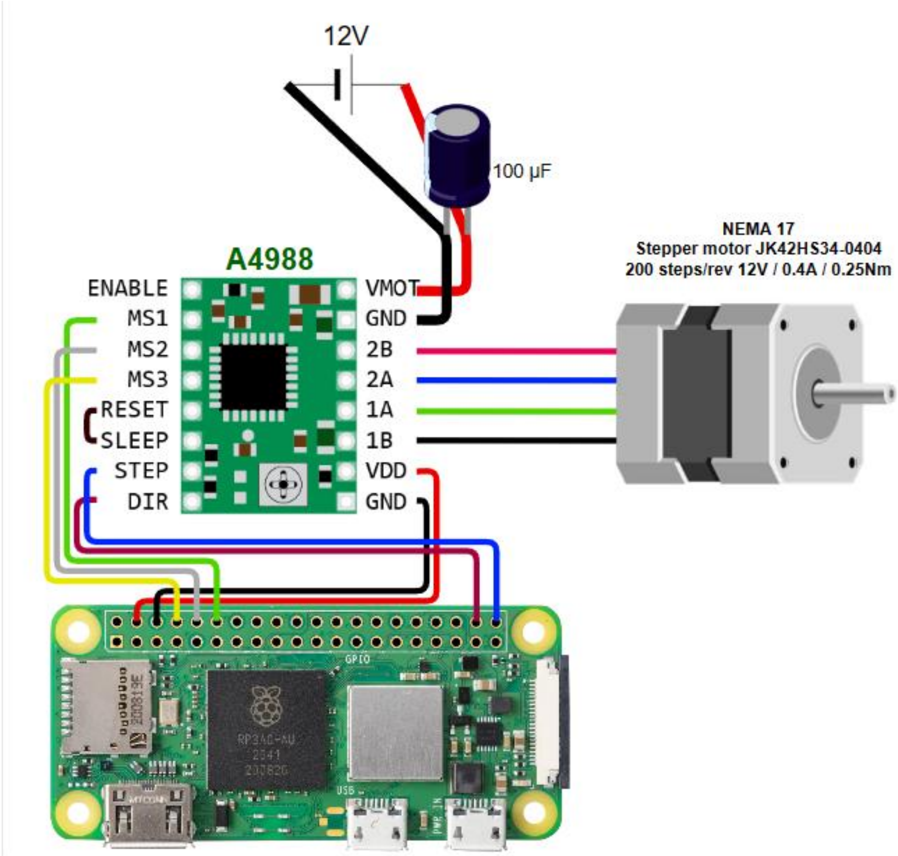


Table 2 A4988 to Raspberry Pi Zero 40-pin header wiring

GPIO/SUPPLY	Physical pin	A4988 Signal	Description
21	40	STEP	Step motor control
20	38	DIR	Direction control
18	12	MS1	Driver signal 1
15 (RXD)	10	MS2	Driver Signal 2
14 (TXD)	8	MS3	Driver signal 3
5V	4	VDD	5V supply
GND	6	GND	Ground 0V
n/a	n/a	! SLEEP	Wire to RESET
n/a	n/a	! RESET	Wire to SLEEP
25	n/a	! ENABLE	Enable is LOW
n/a	n/a	VMOT	Motor Voltage 8-24 Volts +
n/a	n/a	GND	Motor voltage GND (0v)

Early versions of the Raspberry Pi only had a 26-pin header. Below is the original wiring for the GPIO inputs for Raspberry Pi’s with a 26-pin header.

The bipolar A4988 H-Bridge driver board

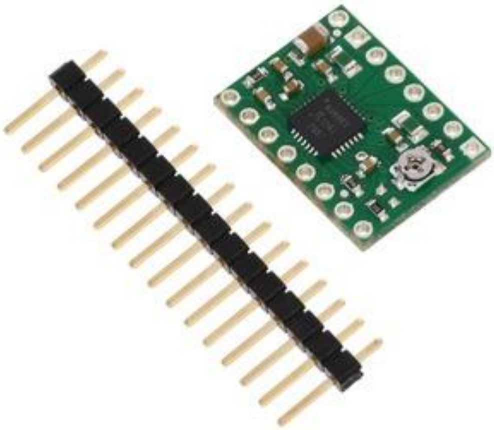


Figure 10 A4988 H-Bridge driver board kit

The A4988 H-Bridge driver board usually comes as a simple kit including 2.54mm SIL connectors or as a ready assembled unit. Break the in-line connector in half and solder the short pins into the board. The long pins can then connect directly into a breadboard or can be connected via matching female connectors into an interface PCB such as the ModMyPi Humble PI.

Solder the pins as shown to the underside of the board. The current adjustment potentiometer at the bottom of the board is used to adjust the current through the motor windings to increase or decrease torque and heat output. It is advised not to solder the A49988 driver board directly into the prototype but to plug the board into two 8-pin 2.54mm SIL sockets. This has two advantages a) easy replacement if the chip fails or removal so that you can read the pin name on the underside of the board.

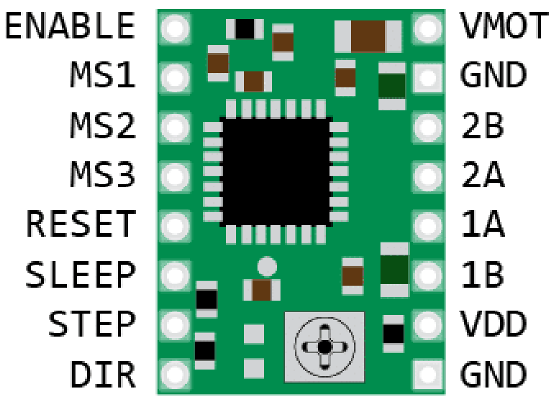


Figure 11 A4988 H-Circuit pin assignments

The diagram on the left shows the pin names when viewed from the component side. However, the pin names are not shown on the board in this view.

The small potentiometer (pot) at the bottom of the view of the driver board is used to adjust the driver current. This can be used to reduce the current so that the motor gets less hot but this also reduces the motor torque proportionately. Turn the pot clockwise for maximum torque.

A4988 H-Bridge circuit

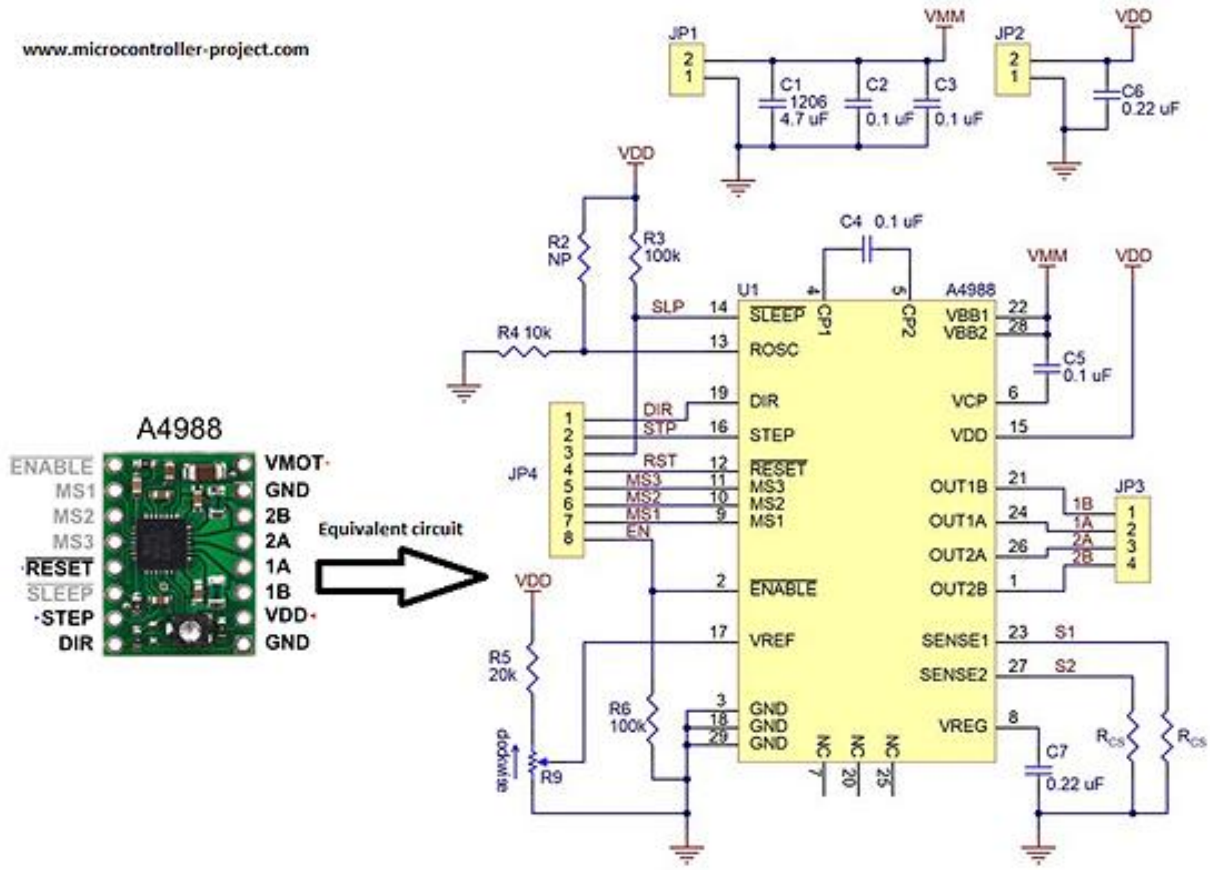


Figure 12 A4988 Circuit Diagram – Courtesy MicroController-project.com

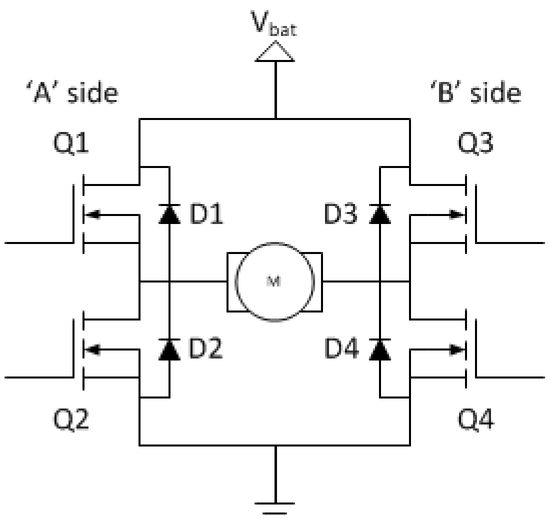


Figure 13 H-Bridge circuit

In general, an H-Bridge is a rather simple circuit, containing four switching elements, with the load at the centre, in an H-like configuration.

The switching elements (Q1..Q4) are usually bi-polar or FET transistors, in some high-voltage applications IGBTs (insulated-gate bipolar transistor) are used.

Integrated solutions also exist but whether the switching elements are integrated with their control circuits or not is not relevant for the most part for this discussion. The diodes (D1..D4) are called catch diodes and are usually of a Schottky type.

Driver boards using the **DRV8825** chip can be used in place of the **A4988** chip. The **DRV8825** has a higher maximum supply voltage than the A4988 (45 V vs 35 V), which means the DRV8825 can be used more safely at higher voltages and is less susceptible to damage from LC (Inductive/Capacitive) voltage spikes. See *Table 5 A4988 and DRV8825 differences* on page 14 for further information.

The illustration below shows a four-wire Nema17 motor. There are in two pairs of wires as shown in Table 3 below.

Figure 14 Nema17 6-pin stepper motor wiring pairs

Table 3 Nema17 4-pin wiring connections to the A4988 driver board

Wire	Alternative	Pair	A4988 Signal	Description
Red	Yellow	2	2B	Coil 2 B connection
Yellow	Blue	2	2A	Coil 2 A connection
Green	Red	1	1A	Coil 1 A connection
Grey	Green	1	1B	Coil 1 B connection

Note: Two colour schemes are shown for the motor connections in the above table. Other wiring schemes are possible, for example, the pairs could be swapped or the wires for both pairs could be reversed. Either of these options will reverse the direction of the motor.

NEMA stands for National Electrical Manufacturers Association. It's essentially a standard that defines the physical dimensions of the motor's faceplate. In the case of a NEMA 17 motor, the faceplate is 1.7 inches square (approximately 43.2 mm x 43.2 mm).

6-Wire Stepper Motors

A 6-wire stepper motor is similar to a 4-wire configuration with the added feature of a common tap placed between either end of each phase as shown in Figure 15 below.

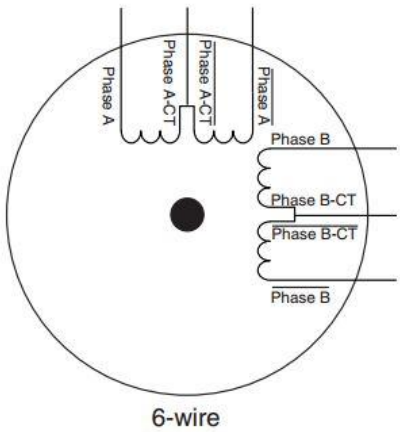


Figure 15 Six-wire stepper motor

Stepper motors with these centre taps are often referred to as unipolar motors. This wiring configuration is best suited for applications requiring high torque at relatively low speeds. Most National Instruments stepper motor interfaces do not support 6-Wire stepper motors, although some motors do not require the centre taps to be used and can be connected normally as a 4-wire motor.

Figure 16 RS PRO Hybrid Stepper Motor, 12 V, 1.8°

Although all stepper motor drives currently distributed by National Instruments are designed for bipolar motors, many 6-wire stepper motors can be operated in either unipolar or bipolar modes. Be sure to check with your motor's manufacturer to make sure the motor is capable of bipolar operation. This will usually be shown in the motor's documentation.

In the illustration on the left is an example of a stepper motor from RS that can be used in either a unipolar or bipolar configuration. Such motors are known as “Hybrid”.

Note that if using 12 Volt Hybrid motors in unipolar mode these will require 12-volt driver circuitry (not covered in this project).

When using these motors in a bipolar configuration with the 12 Volt H4988 H-Bridge driver, do not connect the centre taps.

A4988 H-Bridge mounted on a 40-pin prototyping board

The A4988 H-Bridge driver board can be easily mounted on any suitable expander board such as the Protomate prototyping board as shown below.

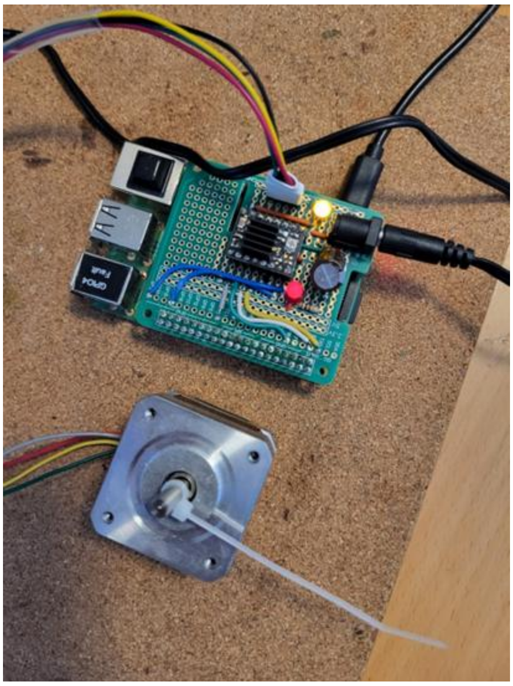


Figure 17 A4988 driver mounted on a 40-pin prototyping board

The +12 Volt motor supply connects to the power socket shown on the right. There are two LEDs connected via 100 Ohm resistors; the red one is for the **+5 Volt** power to the driver board and the orange one is for the **+12 Volt** stepper motor supply.

Note: The 100uF capacitor next to the motor power connector is rated at 16 volts. If you wish to use a higher voltage supply e.g. 24 volts you will need to replace this capacitor with one a with a higher voltage rating.

WARNING: The motor interface board is supplied with two separate power supplies. These are **5V** to the **VDD** pin on the **A4899** driver board and **12V** or greater for the motor connected to the **VMOT** pin. Take great care to make sure that the **12V** motor supply is only connected to the **VMOT** pin and does not get connected by accident to any of the Raspberry Pi connections. Such a mistake will irreparably damage the Raspberry Pi making it unusable.

Nema17 A4988 driver board kit parts list

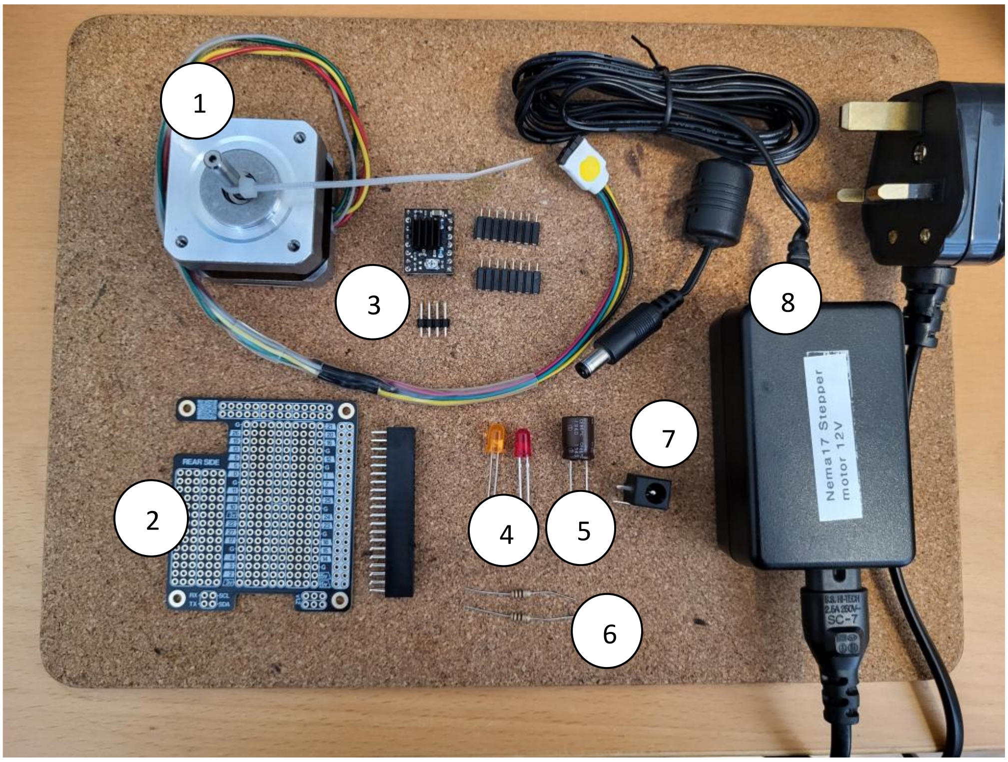


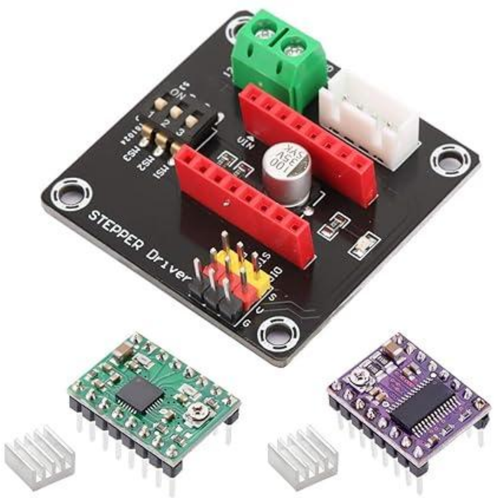
Figure 18 – Nema17 Stepper Motor driver board parts

Table 4 Nema17 A4988 driver board parts list

Item(s)	Description
1.	Nema17 Stepper motor 200 steps, 1.8° per step, 1.7A 12-37 Volts DC
2.	Raspberry Pi 40-pin prototyping board with 40-pin female connector.
3.	A4988 bipolar motor H-Circuit driver board and connectors
4.	Orange and red LEDs for 12V and 5V power on indicators
5.	100uF 16V capacitor for 12V supply smoothing
6.	2 x 100 Ohm resistors to connect in-line with LEDs
7.	PCB mount DC Socket, Dual, 2.1 / 2.5mm for 12V power supply
8.	12+ Volt power supply

Using the Stepper Driver Expansion Board

These driver boards are marketed under many different names by retailers such as **Zoazoqa**, **Dollatech**, **Youmil** and many others. They are readily available from suppliers such as **Amazon** an **Ali-express** They can use either the or **A4988** or **DVR8825 H-Circuit** driver boards. The expansion board can be bought with or without the H-Circuit driver boards.



A4988 DRV8825

Figure 19 42-Stepper Motor driver board

This type of board has the advantage that it does not require any soldering skills. However, it has two characteristics that severely limit its usefulness. These are:

A) Speed/Step setting (ms1, ms2 and ms3) can only be set using the three DIP switch settings. These signals could have so easily been brought out to three pins which would have allowed software control of speed/step-size.

B) Connection to the Raspberry Pi or any other device is by jumper wires which are notoriously un-reliable as they keep falling off.

Despite these disadvantages the board is a great introduction into using Nema17 high-torque motors.

The Amazon link is shown below:
https://www.amazon.co.uk/Zodazoqa-Stepper-Expansion-Ramps1-4-Accessories/dp/B0FBG46YR6/ref=sr_1_4

The following table shows the differences between the **A4999** and **DRV8825 H-Circuit** driver plug-in boards. The A4999 board should be sufficient for most lightweight applications.

Table 5 A4988 and DRV8825 differences

Device	V-Motor	Current	Steps	Features
A4988	8V to 35V	2.0A/phase	1 to 1/16	thermal protection
DRV8825	8.2V to 35	2.5A/phase	1 to 1/32	configurable decay modes,

Nema 17 Stepper Motor connectivity

There are a few pitfalls here. There are a number of different Nema17 motor cables that may either be supplied or bought separately. The cable supplied with the Nema17 stepper motor may not have the correct connectivity to work with your driver board (either the expander board or the one shown in Figure 17 on page 12). The two correct cables that can be used are shown below. In Figure 20 on the bottom left you will see that signals 2A and 1A have been swapped over in the 4-pin connector. Likewise, in Figure 21 below you see that the wires have been swapped to match the signals correctly to the 4-pin connector.

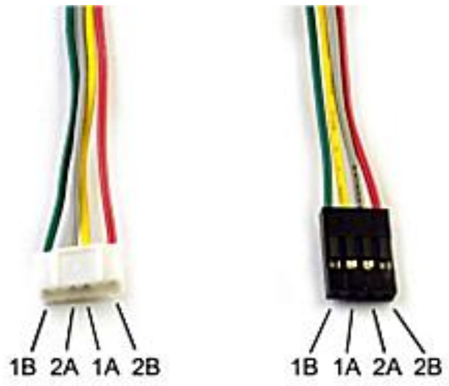


Figure 20 JST XH2.54 Female-Female 4-Pin Cable

Figure 21 JST PH2.0 to XH2.54 Female-Female Cable

If you connect a cable and you find that the motor is working in reverse to what you expect then first try swapping the windings around. If you are not in a position to easily swap the cable around then you can always reverse the polarity of the motor around using the **Motor.reverse()** call.
For more information see the section called *The motor is running the wrong way around* on page 30 in the troubleshooting section.

Software installation

This procedure assumes that the Raspberry Pi is installed with **Raspberry Pi OS** the latest at the time of writing is either **Bullseye** or **Bookworm OS** and with a working Internet Connection. T

The code for driving the motor comes as a number of separate source files. The source for this project can be downloaded from either the **bobrathbone.com** Web site or from the **GitHub** repository.

Download from bobrathbone.com

To extract software on the Raspberry Pi first download with **wget** and then extract it with **tar.**

```
$ mkdir pistepper
$ cd pistepper
$ wget https://bobrathbone.com/raspberrypi/packages/pi_stepper_motor.tar.gz
$ tar -xvf pi_stepper_motor.tar.gz
```

Downloading source files from GitHub

The software is maintained in the following GitHub repository.
https://github.com/bobrathbone/pistepper

To download the software, go to your home directory and download the software using the **git clone** command shown below:

```
$ cd
$ git clone https://github.com/bobrathbone/pistepper
```

This will download the following files into the **pistepper** directory. Change directory to **pistepper.**

```
$ cd pistepper
```

The **pistepper** directory contains the following files:
bipolar_class.py, bipolar_lgpio_class.py, motor_daemon.py, single_motor.py, test_nema17.py, create_tar.sh, motord.py, test_26_nema17.py, test_position.py, motor_i2c_class.py, test_bipolar_class.py, test_stepper.py, LICENSE, README, test_gpios.py, test_unipolar_class.py, log_class.py, README.md, test_motor_i2c_class.py, unipolar_class.py, nema17_screen.py.

Read the README file for information about this particular software release.

Enabling the I2C interface

If you are using the **motor_i2c_class.py** and **test_motor_i2c_class.py** programs it is necessary to install the I2C libraries. If not then skip this section. As the hardware required to run these programs appears to be no longer available, they have not been converted to Python3 but are included in this release if you have the old hardware.

Run raspi-config

```
$ sudo raspi-config
```

Select option **3 Interface Options Configure connections to peripherals**

```
Raspberry Pi Software Configuration Tool (raspi-config)

1 System Options       Configure system settings
2 Display Options      Configure display settings
3 Interface Options    Configure connections to peripherals
4 Performance Options  Configure performance settings
5 Localisation Options Configure language and regional settings
6 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool

          <Select>                          <Finish>
```

Figure 22 Enabling the I2C interface

Now select option **I5 I2C Enable/disable automatic loading of I2C kernel module**

```
Raspberry Pi Software Configuration Tool (raspi-config)

I1 SSH         Enable/disable remote command line access using SSH
I2 RPi Connect Enable/disable Raspberry Pi Connect
I3 VNC         Enable/disable graphical remote desktop access
I4 SPI         Enable/disable automatic loading of SPI kernel module
I5 I2C         Enable/disable automatic loading of I2C kernel module
I6 Serial Port Enable/disable shell messages on the serial connection
I7 1-Wire      Enable/disable one-wire interface
I8 Remote GPIO Enable/disable remote access to GPIO pins

          <Select>                          <Back>
```

Select Yes to enable I2C.

```
Would you like the ARM I2C interface to be enabled?

          <Yes>                    <No>
```

Now run i2cdetect. If you are using a version 2 Raspberry Pi (All latest models 3B, 4B model 5 etc.)

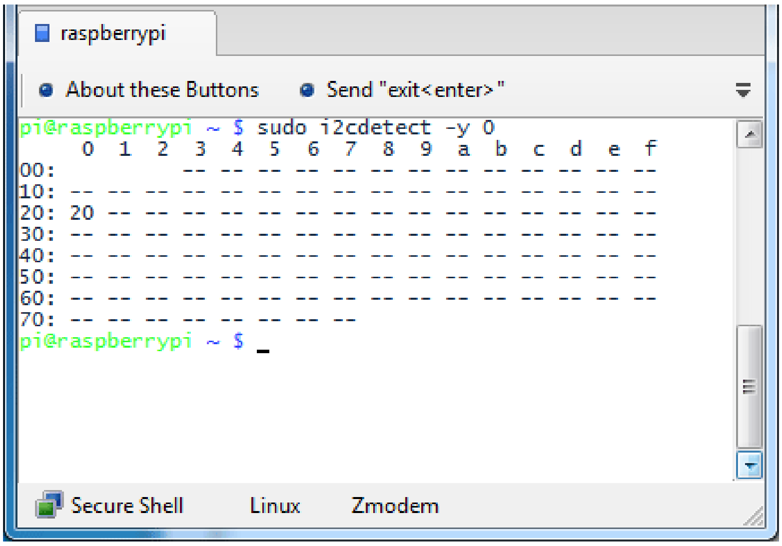
```
sudo i2cdetect -y 1
```

If you are using a version 1 Raspberry Pis (mainly RPi's with 26-pin header)

```
sudo i2cdetect -y 0
```

This will search **/dev/i2c-0** or **/dev/i2c-1** for all address, and if the **ModMyPi I2C** interface is correctly connected, it should show up at **0x20.**

In either case the following screen will be displayed:



Once both of these packages have been installed, you have everything you need to get started accessing I2C and SMBus devices in Python.

Configure motord.py program log rotation

If you will not be using the **radiod.py** daemon then skip this section.

The Radio program logs to a file called **/var/log/motor.log**. This can eventually fill the SD card. Create a file called **/etc/logrotate.d/motor** with the following lines:

```
/var/log/motor.log {
    weekly
    missingok
    rotate 7
    compress
    notifempty
    copytruncate
    create 600
}
```

This will rotate the log files every week so prevent the SD card from eventually filling up.

Source code and usage

unipolar_class.py

This is the actual code that drives the **28BYJ-48** motor using the GPIO pins. It is called by various other 28BYJ-48 driver programs. To use the class in a program first import it and define the motor(s). In the following we define two motors and the GPIO pins they will be using.

```
from unipolar_class import Motor
motora = Motor(17,18,27,22)
motorb = Motor(4,25,24,23)
```

Before a motor can be used it must be initialised. This sets up the GPIO pins.

```
motora.init()
```

To turn the motor one revolution clockwise:

```
motora.turn(1*Motor.REVOLUTION, Motor.CLOCKWISE)
```

To turn the motor two revolutions anti-clockwise:

```
motora.turn(2*Motor.REVOLUTION, Motor.ANTICLOCKWISE)
```

To turn the motor two steps anti-clockwise:

```
motora.turn(2, Motor.ANTICLOCKWISE)
```

The above turns the shaft 0.7 degrees per step (360/512 = 0.703125 Degrees)
The motor has 512 positions. To turn the motor to a particular position (200 in this case):

```
motora.goto(200)
```

To lock the motor in its current position:

```
motora.lock()
```

To stop an already turning motor:

```
motora.interrupt()
```

To set the type of stepping (See *Stepper Motor Theory* on page 2) use one of the following calls.

```
motora.setFullStepDrive()
motora.setHalfStepDrive()
motora.setWaveDrive()
```

The default is *Full Step Drive*. It isn't necessary to set this as it is the default.

test_unipolar_class.py

This contains simple examples of driving two motors using the dual motor driver board.

motor_i2c_class.py

This class does the same as the **motor_class.py** code but uses the i2C interface.

test_motor_i2c_class.py

This class does the same as the **test_motor_class.py** code but uses the i2C interface. However, the motor definitions are different. The MCP23017 I/O expander chip has two banks of eight I/O ports making sixteen in all which allows up to four motors to be driven per MCP23017 I/O expander.

```
address = 0x20 # I2C address of MCP23017
motora = Motor(address,Motor.MOTOR_A)
motorb = Motor(address,Motor.MOTOR_B)
motorc = Motor(address,Motor.MOTOR_C)
motord = Motor(address,Motor.MOTOR_D)
```

The address parameter is normally Hex 0x20 for the MCP23017 I/O expander chip. See *The MCP23017* chip on page 27 and *Enabling the I2C* on page 16 for more information.

motord.py system daemon

The **motord.py** program is a more complex example of driving two motors concurrently. It runs as a system daemon. Each motor is handled by a separate (forked) process. This allows the motors to be turned at the same time.
Just invoking the program displays its usage.

```
$ sudo ./motord.py
usage: ./motord.py start|stop|restart|status|version
```

To start and stop the motor daemon, use the following code.

```
$ sudo ./motord.py start
$ sudo ./motord.py stop
```

Note: The current motor command will be always completed when the stop command is issued.
If the **motord** daemon is running then issuing the status command will display its PID.

```
$ sudo ./motord.py status
Motor daemon running pid 2813
```

The **pid** will be different each time the **motord** program is run.
The version command shows the current version of the software.

```
$ sudo ./motord.py version
Version 1.0
```

motor_daemon.py

This is the code to create the daemon process and to start and stop it. It is used by the **motord.py** program only.

The Log class

The *log_class.py* routine provides logging of events to **/var/log/motor.log** file. It is used by the **motord.py** program only. It logs to **/var/log/motor.log.** The log level needs to be set up in **/var/lib/motor/loglevel** file and should contain one of the following:

INFO, WARNING, ERROR or DEBUG

The bipolar class

This is the low-level driver for the NEMA17 high torque stepper motor using **RPi.GPIO** calls. Any high-level program such as **test_nema17.py** must first import this class as shown below:

```
from bipolar_class import Motor
```

This makes use of six GPIOs to drive the A4988 H-Bridge circuit. They are defined the following, for example **GPIO 20** defines the **step** signal. Below are the definitions for Raspberry Pi's with a 40-pin header.

```
# 40 pin header for newer Raspberry Pi's
step = 21
direction = 20
enable = 25      # Not required - leave unconnected
ms1 = 18
ms2 = 15
ms3 = 14
```

The test program is **test_nema17.py**.

There are also definitions for Raspberry Pi's with a 26-pin header or interface boards with 26 pins.

```
# 26 pin header for older Raspberry Pi's
step = 24
direction = 4
enable = 25
ms1 = 23
ms2 = 22
ms3 = 27
```

The test program for a 26-pin header is **test_26_nema17.py**.

The bipolar_lgpio_class driver program

This program also drives the NEMA17 stepper motor but instead of using **RPi.GPIO** calls it uses the newer LGPIO library (see https://abyz.me.uk/lg/py_lgpio.html). Since the **RPi.GPIO** library does not work on a **Raspberry Pi model 5** or on **Bookworm 32-bit OS**, you should use this driver to run the **NEMA17** stepper motor.

It contains both the motor driver class and the test routine (see the **__main__** routine at the end of the file. To run the code, enter the following commands:

```
$ cd pistepper
$ ./bipolar_lgpio_class.py
```

This will turn the motor using a variety of directions and speeds.

The **bipolar_lgpio_class.py** program can be used on Raspberry Pi's with either a 26 or 40-pin header. To use the 26-pin version go to the **__main__** test routine in the **bipolar_lgpio_class.py** code and uncomment out the 26-pin GPIO definitions and comment out the GPIO 40-pin definitions.

```
# Test routine
if __name__ == '__main__':

    # GPIO assignments for 26-pin header for older Raspberry Pi's
    '''
    step = 24
    direction = 4
    enable = 25
    ms1 = 23
    ms2 = 22
    ms3 = 27
    '''

    # 40 pin header for newer Raspberry Pi's
    step = 21
    direction = 20
    enable = 25   # Not necessarily required (connect to control enable pin)
    ms1 = 18
    ms2 = 15
    ms3 = 14
```

The test_nema17 test program

This is the top-level program to drive the NEMA17 stepper motor. It uses the bipolar_class.py driver. It uses the Raspberry Pi 40-pin header but can be modified to use RPi's with 26-pin headers.

The test_26_ema17 test program

This is the same as the above program but uses the Raspberry Pi 26 pin header wiring.

The control_nema17.py program

This program shows how to control the **Nema17 Stepper Motor** using Rotary Encoders and Push Buttons or Limit switches whilst displaying information on a TFT display. It uses the **rotary_class_gpiozero.py** and **button_class_gpiozero.py** programs for user input and limit switch. When the program is invoked, it first displays the displays the GPIO settings for the Stepper Motor, Rotary Encoder and Limit Switch/Button.

```
$ ./control_nema17.py
Test Neva17 bipolar motor
Motor GPIO settings
  step 21
  direction 20
  enable 25
  ms1 18
  ms2 15
  ms3 14
Rotary encoder
  Rotary SIA signal GPIO 23
  Rotary SIB signal GPIO 24
  Rotary Knob button GPIO 17
  Limit switch GPIO 26
```

It almost immediately starts rotating the motor anticlockwise to simulate traveling backwards twenty revolutions until it hits a limit switch or a stop button connected to GPIO26 is pressed.

```
Motor A Clockwise Sixteenth step
Motor A Clockwise Sixteenth step
Limit switch event 26 pressed
```

It now waits for user input. If the Rotary Encoder knob is turned in either direction it rotates the motor in the relevant direction one step at a time. The following is displayed.

```
Rotary event  1 CLOCKWISE
Rotary event  1 CLOCKWISE
Rotary event  2 ANTICLOCKWISE
Rotary event  2 ANTICLOCKWISE
```

This is simulating fine setting of the start point of the motor and sets the position to 1 each time the Rotary Encoder is turned. The revolution number of steps is set to 200 (FULL step size).

If the Rotary Encoder button is pressed the motor rotates 12-steps in the clockwise direction.
It does this each time the button is pressed up to 16 times.

```
Rotary event  4 BUTTON UP
Rotary event  3 BUTTON DOWN
Rotary event  4 BUTTON UP
Rotary event  3 BUTTON DOWN
Rotary event  4 BUTTON UP
```

The next press causes it to rotate anticlockwise back to position 1. Pressing **Ctrl-C** on the keyboard disables the motor and ends the program.

Remember this is a simulation only to demonstrate how to control the motor using the likes of Buttons, Limit Switches and Rotary encoders.
See the section called *Writing your own software* on page 31 for more information.

The nema17_screen program

The **nema17_screen.py** program has similar functionality of the **control_nema17.py** program but also displays information on a **TFT** screen. It uses the **luma_class.py** TFT driver code for the TFT screen. It simulates a UK based University project to rotate a circular disk with sixteen high power LEDs of different frequencies (not shown) to irradiate a mineral sample for each frequency and to photograph the result. The right-hand display shows a Waveshare TFT display with a rotary encoder with push button and two separate buttons marked BACK and CONFIRM.

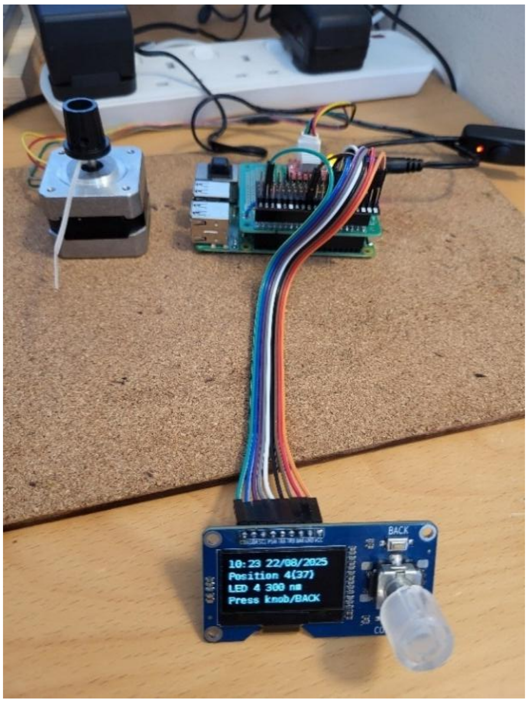


Figure 23 The nema17_screen.py program

Figure 24 Waveshare TFT with rotary encoder/buttons

When the program is started, it rotates the stepper motor anti-clockwise until the CONFIRM button is pressed to simulate a limit switch being activated. The rotary encoder can then be turned to fine adjust the rotating plate to its start position. When the Rotary encoder button is pressed this will turn the plate to the next LED position or back to the previous position if the BACK button is pressed. Details of the current LED characteristics are displayed on the third line of the display.

The rotary_class_gpiozero.py and button_class_gpiozero.py classes

The **rotary_class_gpiozero.py** and **button_class_gpiozero.py** programs are both used by the **control_nema17.py** program. The **button_class_gpiozero.py** program is used either for Limit Switches and or Push buttons. To use these classes first import the classes.

```
from rotary_class_gpiozero import RotaryEncoderClass
from button_class_gpiozero import ButtonClass
```

Example limit switch definition. First define the **Event** routine.

```
def limit_event(event):
    global halt
    if limit_button.pressed():
        print("Limit switch event",event,"pressed")
        motora.stop()    # Stops the motor
        halt = True      # The halt flag is used to exit any loops
```

Define the limit switch GPIO and create it from the Button Class.

```
limit_switch = 26
limit_button = ButtonClass(limit_switch,limit_event,GPIO.PUD_UP)
```

You can define multiple Limit Switches and Buttons only limited by the number of available GPIOS, for example:

```
limit_switch_A = 26
limit_button_A = ButtonClass(limit_switch_A,limit_event,GPIO.PUD_UP)
limit_switch_B = 21
limit_button_B = ButtonClass(limit_switch_B,limit_event,GPIO.PUD_UP)
```

Note that both switches use the same event routine. They both pass the their GPIO number to the **limit_event** routine as the event parameter allowing a specific action to be carried out depending upon which limit switch was activated. For example:

```
def limit_event(event):
    # event is either 26 or 21
    print("Limit switch event",event)
    if limit_button_A.pressed():
        motora.stop()    # Stops the motor
        # Do any other actions for Limit Swich A
    elif limit_button_B.pressed():
        motora.stop()    # Stops the motor
        # Do any other actions for Limit Swich B
```

The test_gpios.py program

Given the apparently random numbering of GPIO pins on the Raspberry Pi 40-pin header it can be confusing how a rotary encoders or push buttons have been connected. The **test_gpios.py** program can quickly sort this out.

```
$ ./test_gpios.py --pull_up
GPIO: 2 State:High
GPIO: 3 State:High
: {output omitted}
GPIO: 24 State:High
GPIO: 25 State:High
GPIO: 26 State:High
GPIO: 27 State:High
Waiting for input events:
```

For example: operating the Limit Switch gives the following output.

```
GPIO 26 falling
GPIO 26 rising
```

For the Rotary Encoder the following will be seen when it is turned.

```
GPIO 26 rising
GPIO 23 falling
```

Other source files

single_motor.py	Test a single **28BJY48** unipolar motor
test_position.py	Unipolar Positional tests based upon number of steps. One revolution = 256 steps

The RPi.GPIO.py program

In the source directory there is a directory called **RPi** which contains a program called **GPIO.py**. It is mainly used for programs for the Unipolar **28BYJ-48** stepper motor which are using **RPi.GPIO** calls as shown below:

```
from RPi.GPIO import GPIO
```

If running on a **Raspberry Pi model 5** or if running **Bookworm (32-bit)** or later then enable the **RPi.GPIO** code with the following instruction.

```
cd
touch pistepper/RPi/__init__.py
```

The above instruction will cause the code using the **GPIO** calls to see directory **RPi** as a package. For earlier models such as the **3B** or **4B** disable the package unless running on **Bookworm 32-bit OS** using the instruction below:

```
rm pistepper/RPi/__init__.py
```

The above is unnecessary for the

Stop bipolar motor from getting hot after boot-up

When the Raspberry Pi is rebooted all of the GPIO pins are set inputs and pulled down low. This includes the A4988 enable pin which then causes heavy current be drawn through the motor which makes the motor very hot to the point that it is too hot to touch. To prevent this from happening edit the **/boot/firmware/config.txt** file and add the **gpio** command to set **GPIO 25** high (disable the enable pin) after the line **[all]**. If you are using a different pin for the enable signal amend the following command as required.

```
[all]
gpio=25=op,dh
```

The MCP23017 expander board

If you are connecting the stepper motor using the I2C interface then you will need an I/O expander board.

There are a number of expander boards available as shown in the following figure. For this project we are using the one shown on the right from Ciseco.

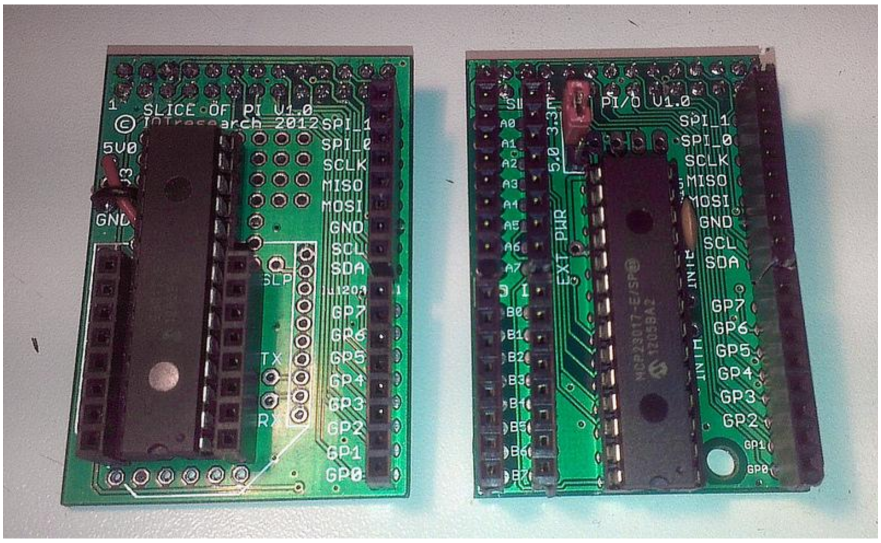


Figure 25 MCP23017 expander boards

Please note in this picture the B0 to B7 outputs are labelled the wrong way round. B7 at the bottom should be B0 and so on.

The MCP23017 chip

The following diagram shows the pin outs for the MCP23017

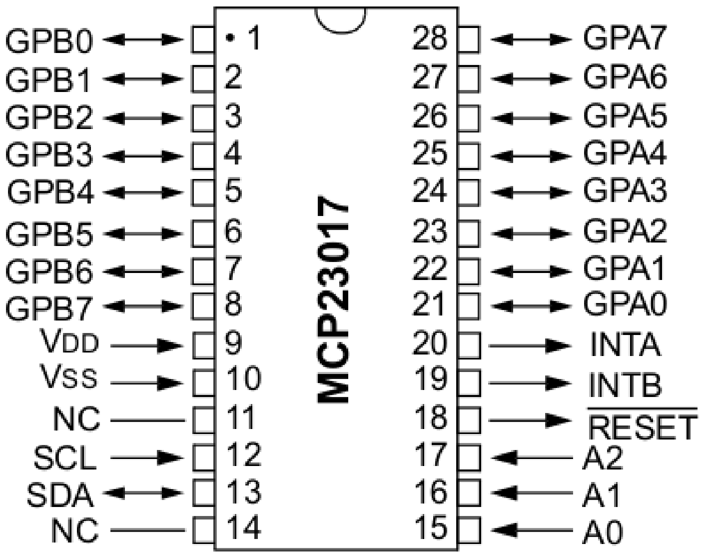


Figure 26 The MCP23017 chip

There are two output banks A and B of 8 pins each. The **I²C** interface consists of a data (SDA) and clock (SCL). The chip has up to eight addresses by biasing the A0, A1 and A2 lines. The full specification for the MCP23017 chip can be found at:
http://ww1.microchip.com/downloads/en/devicedoc/21952b.pdf

MCP2317 and Ciseco Board signals

The MCP2317 chip has 16 input/output pins as shown in the following table. This shows the Hex, decimal or binary values that have to be written to the MCP2317 chip enable the outputs. The banks A and B are addressed by 0x12 and 0x13 respectively (See example program listings in appendix A).

I/O	MCP23017	Pin	Ciseco Board	Bank	Hex	Decimal	Binary
1	GPA0	1	A0	0x12	0x01	1	00000001
2	GPA1	2	A1	0x12	0x02	2	00000010
3	GPA2	3	A2	0x12	0x04	4	00000100
4	GPA3	4	A3	0x12	0x08	8	00001000
5	GPA4	5	A4	0x12	0x10	16	00010000
6	GPA5	6	A5	0x12	0x20	32	00100000
7	GPA6	7	A6	0x12	0x40	64	01000000
8	GPA7	8	A7	0x12	0x80	128	10000000
9	GPB0	21	B0 (B7)	0x13	0x01	1	00000001
10	GPB1	22	B1 (B6)	0x13	0x02	2	00000010
11	GPB2	23	B2 (B5)	0x13	0x04	4	00000100
12	GPB3	24	B3 (B4)	0x13	0x08	8	00001000
13	GPB4	25	B4 (B3)	0x13	0x10	16	00010000
14	GPB5	26	B5 (B2)	0x13	0x20	32	00100000
15	GPB6	27	B6 (B1)	0x13	0x40	64	01000000
16	GPB7	28	B7 (B0)	0x13	0x80	128	10000000

Note: The first batch of Ciseco expander boards have B0-B7 labelled the wrong way round. The numbers in brackets are the incorrect labelling. Watch out for this. Later batches of this board should be correct.

If for example you wish to enable I/O 9 (GPB0) and 10 (GPB1) together you must enable bank 1 and then either add the decimal values together. 1 + 2 = 3 = Hex 0x3 = binary 00000011.

Troubleshooting

Troubleshooting the 28BYJ-48 unipolar stepper motor

You should not normally have any wiring problems if you are using the standard ULN2803A driver board. If you have wired up your own interface board then checking the wiring is the first obvious thing to try.

Use the **test_unipolar_unipolar.py** program to test the motor.

Troubleshooting the Nema17 bipolar stepper motor

The motor doesn't turn

Causes:

1. No +12 Volt supply connected to the VMOT pin on the A4988 driver card
2. RESET and SLEEP signals not wired together on the A4988 driver card
3. ENABLE pin on the A4988 driver card is held LOW particularly on reboot
4. Missing +5 Volt to the VDD pin on the A4988 driver card
5. GND for VMOT and/or VDD not connected to ground of the 12V or 5V supply

The motor twitches uncontrollably or rotates and stops

The MS1, MS2, MS3, STEP and DIR not correctly wired to the Raspberry Pi. More likely the motor windings have been cross-wired to the H-Circuit 1A,2A,1B,2B connections. For example, A1 to 2B and 1B to 2A etc. Recheck motor connection wiring.

The motor gets very hot

The stepper motor gets <u>very</u> hot to the touch possibly enough to cause minor injury or damage. The Bob Rathbone Consultancy will not be held responsible for any loss or injury however caused. Also see the *Disclaimer* on page 35.

This is to some degree, completely normal and expected. From the datasheet for a typical NEMA 17 stepper, the rated temperature rise is 80 °C above ambient and the maximum operating temperature is 130 °C (implying an ambient temperature of 50 °C). It is normal that stepper motors (in general) get a bit hot.

"Too hot to touch" is still relatively cold. 60 °C is already too hot to touch, and that's only a 40 °C rise above a 20 °C ambient temperature.

You can reduce the temperature rise of the motors by reducing the current they receive. The bipolar driver boards in this tutorial have a small potentiometer that can be turned to adjust the current, but keep in mind that doing so will also reduce the torque of the motors and thus they might skip steps if you reduce the current too much.

Another way to reduce the running temperature of the motor is to bolt it to a metal frame which will dissipate some of the heat.

There are several reasons why the motor may run hot:

- Wrong program being used to drive the motor.
  Only use the <u>correct</u> **test_nema17_class.py** and **test_26_nema17_class.py** programs depending on whether or not you have 40-pin or 26-pin wiring respectfully.

- The motor is being locked with the driver signals being enabled for a long period of time. This may be required to lock the motor. However, the motor has natural indents which may provide sufficient locking. If not and you need to lock the motor by driving the coils, try to do so sparingly. This can also happen if the program is exited without disabling the driver signals.
- When the Raspberry Pi is rebooted the A4988 enable pin is pulled low. To prevent this from happening edit the **/boot/firmware/config.txt** file and add the **gpio** command to set **GPIO 25** high (disable) after the line [all]. If you are using a different GPIO for the enable signal then amend the following command as required.

```
[all]
gpio=25=op,dh
```

To disable the signals call **motora.reset()** or **motora.unlock()** during periods of inactivity as shown in the example below.

```
# Reset the motor otherwise it will become hot
motora.reset()
```

or

```
motora.unlock()
```

Always call the motor reset routine when exiting the program.

The motor is running the wrong way around

The test program prints clockwise but the motor is turning anticlockwise and vice versa.

There are two ways of correcting this.

1. Correct polarity by swapping the two pairs of motor connections. Take care not to reverse the wires on an individual pair.
2. Use the **Motor.reverse(True)** call as shown below.

```
motora = Motor(step,direction,enable,ms1,ms2,ms3)
motora.init()
motora.reverse(True)
```

The **reverse()** motor call can be used anytime after the initial **init()** call. To put the motor back to normal polarity during a program run, call the following:

```
motora.reverse(False)
```

Writing your own software

Using the bipolar_lgpio_class.py

Don't modify the **bipolar_lgpio_class.py** code unless you know what you are doing. Copy the **test_nema17.py** or **control_nema17** to your own program file and modify this or write a new one from scratch.
If writing your own software from scratch, first import the appropriate class.

```
from bipolar_lgpio_class import Motor
```

Define the motor driver signals using the GPIO numbers that match your wiring scheme.

```
# 40 pin header for newer Raspberry Pi's
step = 21
direction = 20
enable = 25
ms1 = 18
ms2 = 15
ms3 = 14
```

Define your motor and initialise it.

```
motora = Motor(step,direction,enable,ms1,ms2,ms3)
motora.init()
```

Set the step size (FULL, HALF, QUARTER, EIGHT or SIXTEENTH. Turn the motor CLOCKWISE a full revolution or in thee case of the Expander card set it with the ms1,ms2 and ms3 DIP switches.

```
revolution = motora.setStepSize(Motor.FULL)
motora.turn(revolution, Motor.CLOCKWISE)
```

Turn the motor ANTICLOCKWISE.

```
motora.turn(revolution, Motor.ANTICLOCKWISE)
```

The Nema17 stepper has 200 **FULL** steps per **one revolution**. Other step size settings change this as shown in the table below:

Step size	Degrees	Steps	MS1	MS2	MS3
FULL	1.8º	200	0	0	0
HALF	0.9º	400	0	0	1
QUARTER	0.45º	800	0	1	0
EIGHTH	0.225º	1600	0	1	1
SIXTEENTH	0.1125º	3200	1	0	0

For example, if you set the step size Motor.QUARTER the number of available steps will be 800

```
revolution = motora.setStepSize(Motor.QUARTER)
print("Revolution", revolution)
motora.turn(revolution/2, Motor.CLOCKWISE)
```

The above code will print the following and turn the motor 400 steps (a half revolution)

```
Revolution 400
```

Table 6 Bipolar LGPIO Class calls (bipolar_lgpio_class.py)

Routine	Description
get_chip()	Gets the GPIO chip handle (Not used by user)
__init__(step, direction, enable, ms1, ms2, ms3)	Motor declaration call
init()	Sets the A4988 GPIOs initial default state
reverse(True/False)	Reverse Motor polarity if 'True'
isReversed()	Returns True (motor polarity reversed) or False (normal)
setStepSize(size)	Set step size to FULL, HALF, QUARTER, EIGHTH or SIXTEENTH
startPosition()	Set start position to 1
getPosition()	Get current position
getRevolution()	Get the number of steps in a revolution
turn(steps,direction)	Turn n steps in direction clockwise or anti clockwise
goto(position=1,stepsize=FULL)	Go to a specific position (step size dependent)
lock()	Lock the motor in the current position
unlock()	Unlock the motor in the current position (prevents over-heating)
setStepResolution(stepres)	Internal use – Write step resolution to ms1, ms2 and ms3
setDebug(level)	Print debug statements if level = True
close()	Close the GPIO chip
stop()	Stop current motor operation, for example if limit reached
start()	Allow motor to carry out new operations
reset()	Put motor back to a know state

Running the program as a system daemon

It may be more convenient to run your program as a system daemon. A daemon is usually started during system startup. There is an example called **motord.py** for the **28BYJ-48** bipolar stepper motor. See section called *motord.py system daemon* on page 20 for further information.

Unfortunately, there isn't one for the Nema17 stepper motor in this release. The **motord.py** program can be copied and modified to support the nema17 stepper motor.

Appendix A Code Listings

All code can be downloaded from the following URL:

https://bobrathbone.com/raspberrypi/packages/pi_stepper_motor.tar.gz

The following table lists all of the available software and its function.

Table 7 Source files

File name	Driver Class	Type	Description
test_unipolar_class.py	unipolar_class.py	Unipolar	28BYJ-48 stepper dual motor driver
single_motor.py	unipolar_class.py	Unipolar	28BYJ-48 stepper single motor
test_motor_i2c_class.py	test_motor_i2c_class.py	Unipolar	28BYJ-48 stepper motor I2C driver
test_position.py	unipolar_class.py	Unipolar	28BYJ-48 test position setting
motord.py	motor_daemon.py	Unipolar	28BYJ-48 background daemon
log_class.py	n/a	n/a	Logging class for motord.py
test_nema17.py	bipolar_class.py	Bipolar	Nema17 stepper motor driver
test_26_nema17.py	bipolar_class.py	Bipolar	Nema17 driver 26-pin header
See **__main__** in the driver class.	bipolar_lgpio_class.py	Bipolar	Nema17 driver using LGPIO
test_motor_i2c_class.py	motor_i2c_class.py	Unipolar	I2C 28BYJ-48 stepper dual motor driver (1)
test_expander.py	bipolar_lgpio_class.py	Bipolar	Test Driver Expander Board
nema17_screen.py	bipolar_lgpio_class.py	Bipolar	Test with TFT screen, rotary encoders and push buttons

Note 1: The hardware for the I2C interface appears no longer to be available but the I2C programs are included in case you have this old hardware. The I2C programs written in Python 2 have not yet been tested with Python 3.

Appendix B – Specifications

Appendix B.1 - 28BYJ-48 – 5V Stepper Motor

- Operating Voltage 5VDC
- Operating Current 240mA (typical)
- Number of phases 4
- Gear Reduction Ratio 64:1
- Step Angle 5.625°/64
- Frequency 100Hz
- In-traction Torque >34.3mN.m(120Hz)
- Self-positioning Torque >34.3mN.m
- Friction torque 600-1200 gf.cm
- Pull in torque 300 gf.cm

The 28BYJ-48 data sheet can be found at:
https://www.mouser.com/datasheet/2/758/stepd-01-data-sheet-1143075.pdf

Appendix B.2 – Nema17 2-phase Stepper Motor

- Rated Voltage: 12V DC
- Current: 1.2A at 4V
- Step Angle: 1.8 deg.
- No. of Phases: 4
- Motor Length: 1.54 inches
- 4-wire, 8-inch lead
- 200 steps per revolution, 1.8 degrees
- Operating Temperature: -10 to 40 °C
- Unipolar Holding Torque: 22.2 oz-in

The Nema17 data sheet can be found at:
https://media.pbclinear.com/pdfs/pbc-linear-data-sheets/data-sheet-stepper-motor-support.pdf

Appendix C Licences

The software and documentation for this project is released under the GNU General Public Licence.

The GNU General Public License (GNU GPL or GPL) is the most widely used free software license, which guarantees end users (individuals, organizations, companies) the freedoms to use, study, share (copy), and modify the software. Software that ensures that these rights are retained is called free software. The license was originally written by Richard Stallman of the Free Software Foundation (FSF) for the GNU project.

The GPL grants the recipients of a computer program the rights of the Free Software Definition and uses *copyleft* to ensure the freedoms are preserved whenever the work is distributed, even when the work is changed or added to. The GPL is a *copyleft* license, which means that derived works can only be distributed under the same license terms. This is in distinction to permissive free software licenses, of which the BSD licenses are the standard examples. GPL was the first *copyleft* license for general use. This means that you may modify and distribute the software and documentation subject to the conditions of the licences.

See http://www.gnu.org/licenses for further information on the GNU General Public License.

The licences for the source and documentation for this project are:

GNU General Public License.	See http://www.gnu.org/licenses/gpl.html
GNU AFFERO General Public License.	See http://www.gnu.org/licenses/agpl.html
GNU Free Documentation License.	See http://www.gnu.org/licenses/fdl.html

Acknowledgements

Thanks to the numerous Raspberry Pi contributors who have placed articles about driving stepper motors using the Raspberry Pi on their websites and blogs for the benefit of the community.

The code for the 28BYJ48 stepper motor is based upon original code from PiHut.

Some diagrams came from Aleksas Pielikis at https://github.com/aleksas/zero-stepper

Thank you to Jian-You Lin, Brandeis University, Waltham, Massachusetts, USA, for his help in configuring, testing and documentation improvements for the Nema17 stepper motor project.

Disclaimer

Glossary

FET	Field Effect Transistor
GND	Ground (0 Volts)
GPIO	General Purpose IO (On the Raspberry Pi)
IC	Integrated Circuit
IGBT	Insulated-Gate Bipolar Transistor
LC	Inductive (L) Capacitive(C) voltage spikes, usually at switch on and can be damaging
LED	Light Emitting Diode
NEMA	The US National Electrical Manufacturers Association
PID	Process Identification Number
VDD	Voltage Drain Drain (In this project +5 Volts to the **A4988 H-Bridge** circuit driver board)
VMOT	Voltage Motor (In this project +8 to 12 Volt power to the **Nema17** stepper motor)

Index