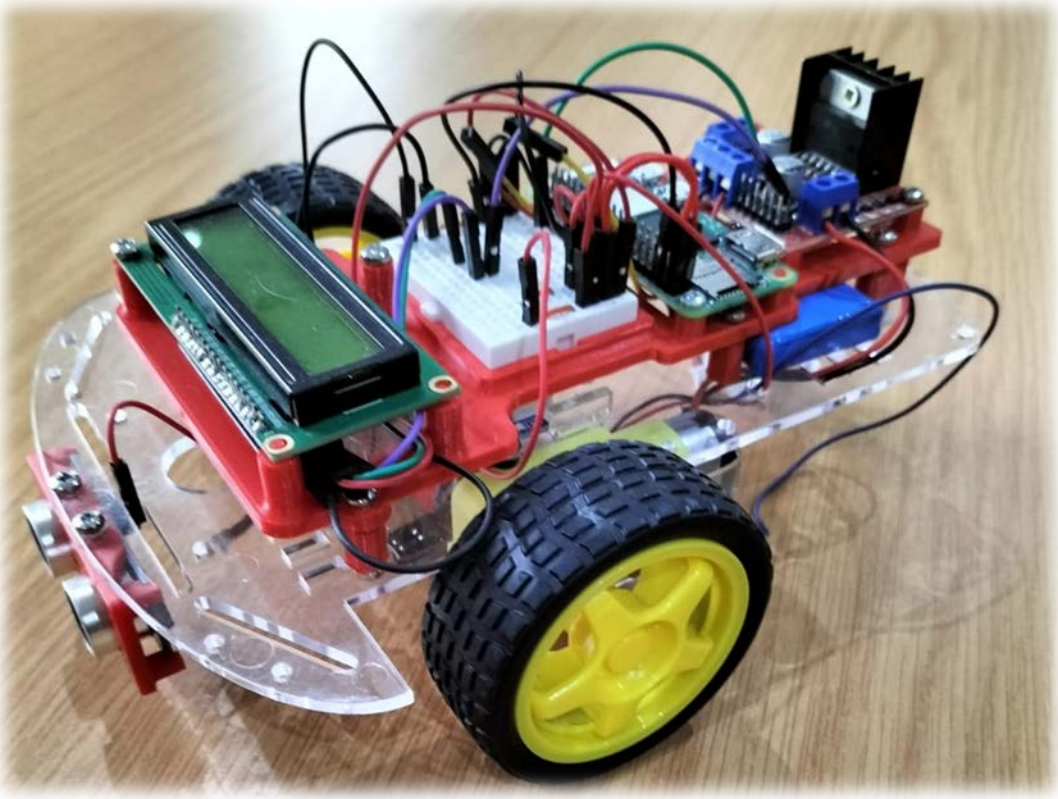
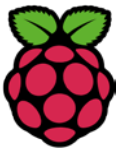
# RPi Robot

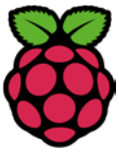
## Python



*Johan Benade - 2022*



# Table of Contents



# 1 What is a robot?

*A robot is an autonomous machine capable of sensing its environment, carrying out computations to make decisions, and performing actions in the real world.*

## 1.1 Main components of a robot

Robots are built to present solutions to a variety of needs and fulfill several different purposes, and therefore, require a variety of specialized components to complete these tasks. However, there are several components that are central to every robot's construction, like a power source or a central processing unit. The components of a robot fall into five categories:

### Control system

Computation includes all the components that make up a robot's central processing unit, often referred to as its control system. Control systems are programmed to tell a robot how to utilize its specific components, similar in some ways to how the human brain sends signals throughout the body, to complete a specific task. These robotic tasks could comprise anything from minimally invasive surgery to assembly line packing.

### Sensors

Sensors provide a robot with stimuli in the form of electrical signals that are processed by the controller and allow the robot to interact with the outside world. Common sensors found within robots include video cameras that function as eyes, photoresistors that react to light and microphones that operate like ears. These sensors allow the robot to capture its surroundings and process the most logical conclusion based on the current moment and allows the controller to relay commands to the additional components.

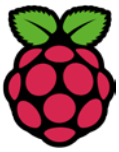
### Actuators

As previously stated, a device can only be a robot if it has a movable frame or body. Actuators are the components that are responsible for this movement. These components are made up of motors that receive signals from the control system and move in tandem to carry out the movement necessary to complete the assigned task. Actuators can be made of a variety of materials, such as metal or elastic, and are commonly operated by use of compressed air (pneumatic actuators) or oil (hydraulic actuators,) but come in a variety of formats to best fulfill their specialized roles.

### Power Supply

Like the human body requires food to function, robots require power. Stationary robots, such as those found in a factory, may run on AC power through a wall outlet but more commonly, robots operate via an internal battery. Most robots utilize lead-acid batteries for their safe qualities and long shelf life while others may utilize the more compact but also more expensive silver-cadmium variety. Safety, weight, replaceability and lifecycle are all important factors to consider when designing a robot's power supply.

Some potential power sources for future robotic development also include pneumatic power from compressed gasses, solar power, hydraulic power, flywheel energy storage organic garbage through anaerobic digestion and nuclear power.



### End Effectors

End effectors are the physical, typically external components that allow robots to finish carrying out their tasks. Robots in factories often have interchangeable tools like paint sprayers and drills, surgical robots may be equipped with scalpels and other kinds of robots can be built with gripping claws or even hands for tasks like deliveries, packing, bomb diffusion and much more.

# 2 Introduction to mobile robots

A mobile robot is a robot that can move in the surrounding (locomotion). Mobile robotics is usually considered a subfield of robotics and information engineering.

A spying robot is an example of a mobile robot capable of movement in a given environment.

Mobile robots have the capability to move around in their environment and are not fixed to one physical location. Mobile robots can be "autonomous" (AMR - autonomous mobile robot) which means they are capable of navigating an uncontrolled environment without the need for physical or electro-mechanical guidance devices. Alternatively, mobile robots can rely on guidance devices that allow them to travel a pre-defined navigation route in relatively controlled space. By contrast, industrial robots are usually more-or-less stationary, consisting of a jointed arm (multi-linked manipulator) and gripper assembly (or end effector), attached to a fixed surface. The joint.

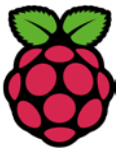
Mobile robots have become more commonplace in commercial and industrial settings. Hospitals have been using autonomous mobile robots to move materials for many years. Warehouses have installed mobile robotic systems to efficiently move materials from stocking shelves to order fulfillment zones. Mobile robots are also a major focus of current research and almost every major university has one or more labs that focus on mobile robot research. Mobile robots are also found in industrial, military and security settings.

## 2.1 Mobile Robots – Where to find


*Figure 1 Ubtech displayed several mobile robots for service applications at CES 2020. Source: Ubtech*

In 2019, applications for mobile robots advanced across all sectors. From manufacturing and retail to customer service, we saw collaborative robotics do some remarkable things. Yet the technology has so much more to offer.

Broadly speaking, autonomous mobile robots can carry out a variety of tasks with a minimum of human input, and many are designed to operate safely around people. Amazon's Kiva robots are a well-known



example. They play a crucial role in the e-commerce giant's order-fulfillment process, zooming around company warehouses to find products and bringing them to the necessary drop-off stations.

Thanks to these market-defining robots, Amazon can fill more orders faster than ever before, with a remarkable degree of accuracy.

2020 is sure to be an incredible year for robotics adoption, especially for mobile platforms. Amazon will be using 120,000 robots by the end of this year, and the global market for mobile robots will surpass $3 billion, predicts Ash Sharma, research director at Interact Analysis.

Here are just some of the places where one can expect to see mobile robots in 2020.

## 2.2 Application of mobile robots

## Mobile robots in medical facilities

Nurses and other medical professionals spend a lot of time running back and forth around medical facilities, doing things like tending to patients, grabbing supplies, and answering doctors' calls.

Mobile robots can alleviate much of this legwork. They could move patients, gather and transport supplies, assist with surgical procedures, and even disinfect rooms. Robots could reduce the number of tedious tasks, reduce workplace strain and injuries, and ensure more consistent quality of care.

For instance, the Texas Medical Research Innovation Institute in Houston has tested ABB's YuMi collaborative robot as a roving laboratory technician. The two-armed mobile robot can carry out tasks like pipetting liquids, sorting test tubes and carrying equipment.

According to ABB's performance data, Yunmi can help speed up hospital and research lab operations by as much as 50%, working a full 24 hours a day.

Not only can mobile robots accelerate healthcare operations, but also they could pave the way for genuinely continuous, all-hour coverage. Nurses and doctors need to rest, but mobile robots do not.

## Hospitality and customer service

While chatbots and software-based communication tools have been a staple of the hospitality and retail industries for years now, the technology has leaped into forms that are more physical. From Japan's robot-staffed hotel to Rollbot, which can bring guests a roll of toilet paper, the technology promises to improve customer experiences.

With shortages of personnel, mobile robots can help attend to customer needs. Imagine robotic bellhops that can bring your luggage to your room, round-the-clock room service (perhaps supplied by a robotic kitchen), or autonomous shopping carts that will drive themselves around a store.

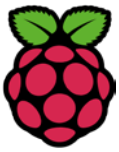
For another real-world example, Travelmate has created a robot suitcase that will follow on your heels, allowing for hands-free travel experiences.

At CES 2020, Ubtech's Cruzr service robot was among the many designed for customer service.

## Agriculture

In the past year, farmers and agribusinesses have started turning to mobile robots to maintain, measure, and harvest fresh produce, among many other applications.

> Cambridge Consultants' Mamut autonomous robot is one such machine, designed to explore active fields and capture useful data, which can help improve crop yield. Because the data collection process is fully automated, it eliminates the need for farmers and crews to pore over fields.



Mamut is more useful than drones because it's ground-based for greater endurance, and it can collect more information.

Whether on their own, as enhancements to existing equipment, or in combination with aerial drones, mobile agriculture robots can help handle pest infestations, measure soil quality and pesticides, and harvest fruits and vegetables.

## Warehousing and order fulfillment

From the factory to order delivery, many links in supply chains are being automated. After Amazon acquired Kiva Systems, many other robots offer to serve warehouses and order-fulfillment businesses. They include 6 River Systems' Chuck, Locus Robotics' LocusBots, Swisslog's CarryPick, and Hitachi's Racrew.

From mobile carts that accompany human pickers to more autonomous systems that move around warehouses on their own, there is an incredible variety of mobile robots in supply chain and logistics today.

In addition to retailers such as Best Buy, Shopify, and Walmart gaining greater efficiency and visibility into their operations, customers can also reap the benefits of robotics. They include shorter e-commerce order fulfillment, more accurate picking, and possibly even decreased shipping prices.

## Package delivery

For a couple of years now, companies such as Alphabet, Amazon, FedEx, and UPS have been working on drones for local package deliveries. Safety and regulatory concerns, a variety of competing alternatives, and technical challenges have slowed progress.

Still, mobile robots and drones could not only speed up delivery times, but they could also reduce the environmental impact of the many trucks on the road.

For instance, Amazon has unleashed a small army of wheel-based delivery robots called Amazon Scout. It's not a stretch to predict that there will be much more of this over the coming year, particularly from some of the other companies in the shipping field.

FedEx already has an autonomous delivery robot in use, and it first appeared in the summer of 2019. While UPS has not yet followed suit, other companies have, including Starship Technologies, grocery brand Kroger, General Motors, and DoorDash's Nuro.

## Mobile robots for education

A wide variety of STEM (science, technology, engineering, and mathematics) kits are intended to inspire young people (and adults) and help them learn how to code.

Ozobot, for example, is designed for use in educational environments, aimed at teachers and students alike. The robot will react to colors and lines drawn on a tablet, paper, or even custom surfaces. It can also play games with users, many of which explore the concepts above.

As technology has become an incredibly vital part of modern life, many more educational applications will appear over the coming year.

## Disaster recovery and emergency response

Australia's massive fires are just the latest disaster to remind us of the need for societal resilience and the possibilities for automated threat response. Mobile robots can enter spaces the average human

cannot, as well as endure direct exposure to hazardous chemicals, heat and flames, and even widespread debris. Robotics is ideally suited for disaster recovery and emergency rescue situations.

From underwater recovery and firefighting to ambulatory operations — much like Zebro can do — autonomous robots could swarm into the field to save human lives. The deployment of mobile robots in 2020 could enhance human survivability.

## 2.3 Opportunities for mobile robots

Based on the sheer number of applications that currently exist for mobile robots, the future is bright. The technology offers near-limitless opportunities, as it can play a role not just across dozens of industries, but also in so many ways within each field.

From customer service to disaster recovery, mobile robots are going to play a starring role in our future.

## 2.4 What Types of Mobile Robots do exist?

A custom robot design often starts with a "vision" of what the robot will look like and what it will do. The types of robots possible are unlimited, though the more popular are:

- Land-based wheeled robot
- Land-based tracked robot
- Land-based legged robot
- Air-based: plane, helicopter, blimp
- Water-based; boat, submarine
- Misc. and combination robot
- Stationary robot (arm, manipulator etc.)

Land-based wheeled robots are the most popular mobile robots among beginners as they usually require the least investment while providing significant exposure to robotics. The most complex type of robots is the autonomous humanoid (resembling a human), as it requires many degrees of freedom, synchronizing many motors and many sensors. This section is intended to help you decide what type of robot to build. Once you have chosen the type of robot, you can brainstorm what tasks/functions you want it to complete.

## Wheeled Robots

Wheels are by far the most popular method of providing robot mobility and are used to propel many different sized robots and robotic platforms. Wheels can be just about any size, from fractions of an inch to 10 to 12 inches. Tabletop robots tend to have the smallest wheels, usually less than 2 inches in diameter. Robots can have just about any number of wheels, although 3 and 4 are the most common.

Normally a three wheeled robot uses two wheels and a caster at one end. More complex two wheeled robots use gyroscopic stabilization. It is rare that a wheeled robot uses anything but skid steering (like that of a tank).

Rack and pinion steering such as that found on a car requires too many parts and its complexity and cost outweigh most of its advantages.

Four and six wheeled robots have the advantage of using multiple drive motors (one connected to each wheel) which reduces slip. Omni-directional wheels, mounted properly, can give the robot significant mobility advantages.

A common misconception about building a wheeled robot is that large, low-cost DC motors can propel a medium sized robot. However, there is a lot more involved than just selecting and using a motor.

**Advantages**

- Low-cost
- Simple design and construction
- Near infinite different dimensions cater to your specific project
- Six wheels can replace a track system
- Diameter, width, material, weight, tread etc. can all be custom to your needs
- Excellent choice for beginners

**Disadvantages**

- May lose traction (slip)

Small contact area (small rectangle or line)

## Tracked Robots

Tracks (or treads) are like what tanks use. Track drive is best for robots used outdoors and on soft ground. Although tracks do not provide added "force", they do reduce slip and more evenly distribute the weight of the robot, making them useful for loose surfaces such as sand and gravel. Most people tend to agree that tank tracks add an "aggressive" look to the robot as well.

**Advantages**

- Constant contact with the ground prevents slipping that might occur with wheels
- Evenly distributed weight helps your robot tackle a variety of surfaces

**Disadvantages**

- When turning, there is a sideways force that acts on the ground; this can cause damage to the surface the robot is being used on and cause the tracks to wear.
- Not many different tracks are available (robot is usually constructed around the tracks)
- Increased mechanical complexity and connections

## Legged Robots

An increasing number of robots use legs for mobility. Legs are often preferred for robots that must navigate on very uneven terrain. Most amateur robots are designed with six legs, which allow the robot to be statically balanced (always balanced on 3 legs). Robots with fewer legs are harder to balance. Researchers have experimented with monopod (one legged "hopping") designs, though bipeds (two legs) and quadrupeds (four legs) and hexapods (6 legs) are most popular.

**Advantages**

- Closer to organic/natural motion - Can potentially overcome large obstacles and navigate very rough terrain

**Disadvantages**

- Increased mechanical, electronic and coding complexity
- Lower battery size despite increased power demands
- Higher cost to build

## Air-based Robots

An AUAV (Autonomous Unmanned Aerial Vehicle) is very appealing and is entirely within the capability of many robot enthusiasts. However, the advantages of building an autonomous unmanned aerial vehicle, especially if you are a beginner, have yet to outweigh the risks.

High-altitude AUAV blimps and aircraft may one day be used for communication. When considering an aerial vehicle, most hobbyists still use existing commercial remote-controlled aircraft.

Aircraft such as the US military Predator were initially semi-autonomous though in recent years Predator aircraft have flown missions autonomously.

**Advantages**

- Remote controlled aircraft have been in existence for decades (good community)
- Excellent for surveillance

**Disadvantages**

- Entire investment can be lost in one crash.
- Very limited robotic community to provide help for autonomous control

## Water-based Robots

An increasing number of hobbyists, institutions and companies are developing unmanned underwater vehicles. There are many obstacles yet to overcome to make underwater robots attractive to the wider robotic community though in recent years, several companies have commercialized pool cleaning robots. Underwater vehicles can use ballast (compressed air and flooded compartments), thrusters, tail and fins or even wings to submerge. Other aquatic robots such as pool cleaners are useful commercial products.

**Advantages**

- Most of our planet is water
- Design is almost guaranteed to be unique
- Can be used and/or tested in a pool

**Disadvantages**

- Robot can be lost many ways (sinking, leaking, tangled...)
- Most electronic parts do not like water (also consider water falling on electronics when accessing the robot after a dive)
- Surpassing depths of 10m or more can require significant research and investment
- Very limited robotic community to provide help
- Limited wireless communication options

## Miscellaneous and combination / hybrid

Your idea for a robot may not fall nicely into any of the above categories or may be comprised of several different functional sections. Note again that this guide is intended for mobile robots as opposed to stationary or permanently fixed designs (other than robotic arms and grippers). It is wise to consider

when building a combination / hybrid design, to use a modular design (each functional part can be taken off and tested separately). Miscellaneous designs can include hovercraft, snake-like designs, turrets and more.

**Advantages**

- Designed and built to meet specific needs
- Multi-tasking and can be comprised of modules
- Can lead to increased functionality and versatility

**Disadvantages**

- Increased complexity and cost
- Often parts must be custom designed and built

## Arms & Grippers

Although these do not fall under the category of "mobile" robotics, the field of robotics essentially started with arms and end-effectors (devices that attach to the end of an arm such as grippers, magnets etc.). Arms and grippers are the best way for a robot to interact with the environment it is exploring. Simple robot arms can have just one motion, while more complex arms can have a dozen or more unique degrees of freedom.

**Advantages**

- Very simple to very complex design possibilities
- Easy to make a 3 or 4 degree of freedom robot arm (two joints and turning base)

**Disadvantages**

- Stationary unless mounted on a mobile platform
- Cost to build is proportional to lifting capability

# 3 Software

The PiBot will be connected to a computer or laptop via a wireless connection. The development of the software to operate the PiBot, will be done on a computer and then transferred to the PiBot and from there the code will be executed.

To connect to the PiBot wirelessly from a Windows computer, the following software packages will be needed, download and install all of them:

- Raspberry Pi Imager – installation of the OS on the SD Card
- Angry IP Scanner or Advance IP Scanner – Determine the RPi IP address
- Putty – Secure Shell (SSH) connection to the RPi
- Thonny – Integrated Development Environment, code development environment.

## 3.1 Raspberry Pi Imager

The Raspberry Pi needs an operating system to work. Raspberry Pi OS (previously called Raspbian) is the official supported operating system. Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with the Raspberry Pi. The Raspberry Pi Imager software can be downloaded from the following link: https://www.raspberrypi.com/software/

A computer with a SD card reader will be needed to install image.

To create a bootable image on the SD card the following steps needs to be followed:

1) Connect an SD Card reader with the SD card inside to the computer.
    a. (On laptop computers with an internal SD card reader, insert the SD card into the reader)
2) Open Raspberry Pi Imager and choose the required OS from the list presented.
3) Choose the SD card onto which the OS must be installed.
4) Complete the wireless settings and save it.
5) Review all selections and click on the Write button to begin writing the data to the SD card.
6) When the writing process is completed, remove the SD card from the reader, insert it into the RPi and power it up.
7) Use AngryIPScanner or AdvanceIPScanner to determine the IP Address of the RPi.
8) Once you have the IP address, use Putty to ssh into the RPI.
9) Run Thonny to develop python code, connect to the RPI to run the code on the RPi. All files created in Thonny can be saved on the local computer or on the Rpi.

**Step 2)** – Open Raspberry Pi Imager and select the operating image:


*Figure 2 On the landing page for Raspberry Pi Imager, Choose OS.*

There are several options to choose from:

- If you will be using the RPi with a screen, keyboard and mouse connected to it, then choose the default option Raspberry Pi OS (32-bit) option. This is a 1.1GB download and it runs the GUI on the Pi that use a huge amount of resources. The RPi responses will be a bit slow.

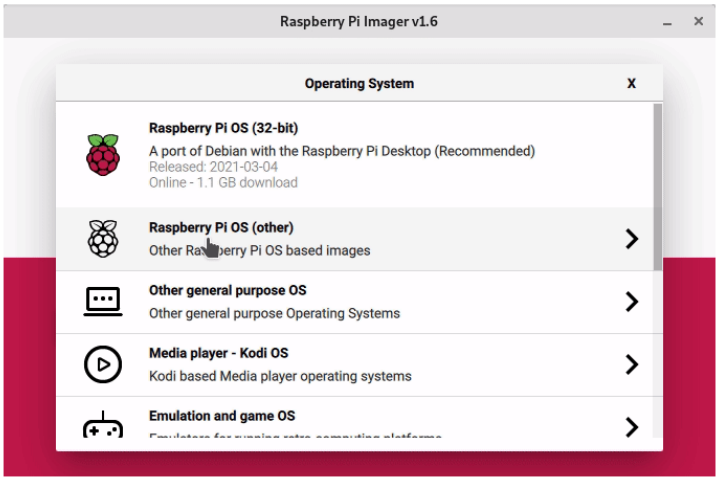



*Figure 3 Default OS for RPi*

- In our application with the PiBot, we do not need the graphical user interface, therefore we can choose Raspberry Pi OS (other), and select Raspberry Pi OS Lite (32-bit). This will then install the OS with no desktop environment. It is a 0.4GB download. When this is run on the RPi it only uses resources required for a headless environment. Responses from the RPi will be faster compared to the full OS with the full Desktop.

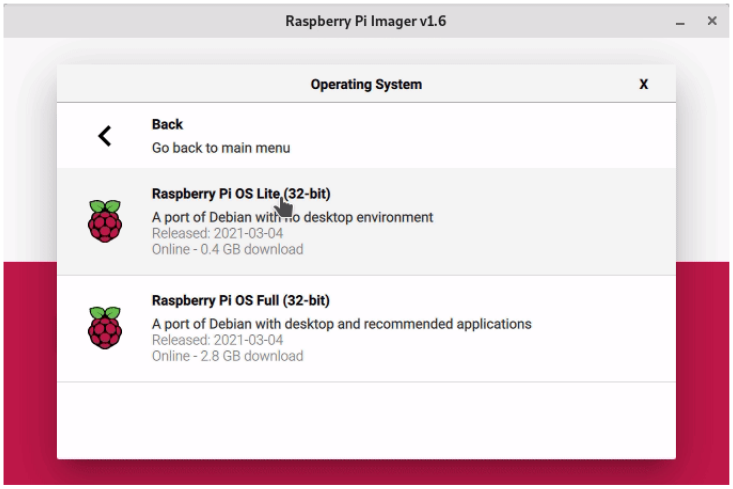



*Figure 4 Selecting Raspberry Pi OS Lite(32-bit)*

**Step 3) C**hoose the storage location of SD Card where the Raspberry Pi OS must be installed.

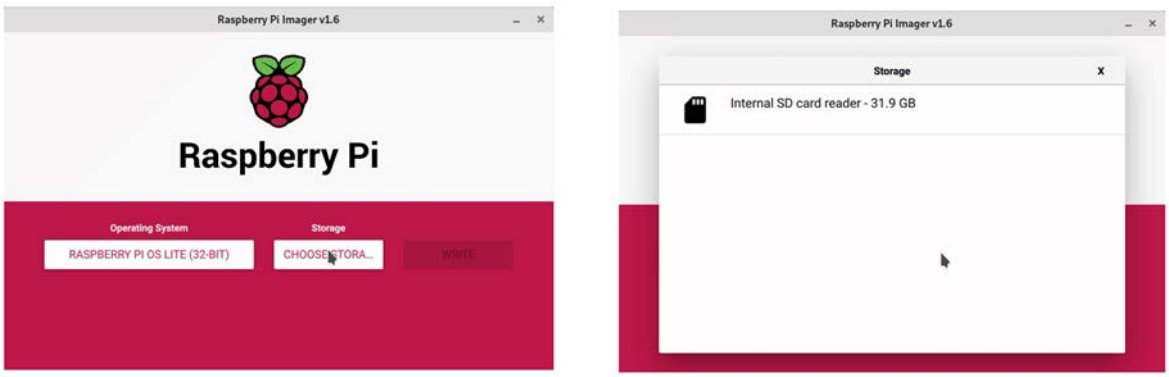



*Figure 5 Choosing storage location*

**Step 4)** To setup the RPi for a headless environment, use the setup button located on the bottom right of the screen.

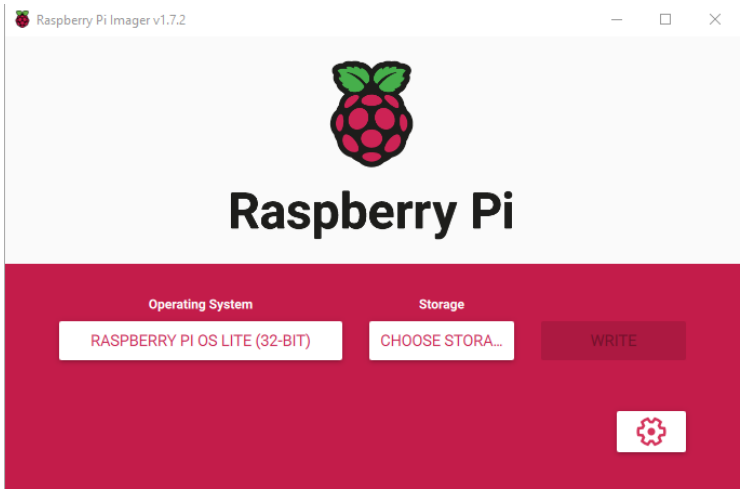



*Figure 6 Step 4), click on the setup icon bottom right.*

The first option is to set the Host name for your RPi. Ths is also the name that will appear when you are determining the IP Address for your RPi. A unique Hostname will be assigned to you during the training session. If you are using it in a stand alone or at home environment, you can keep the default host name.

Enable SSH and choose “Use password authentication” as indicated in Figure 7.

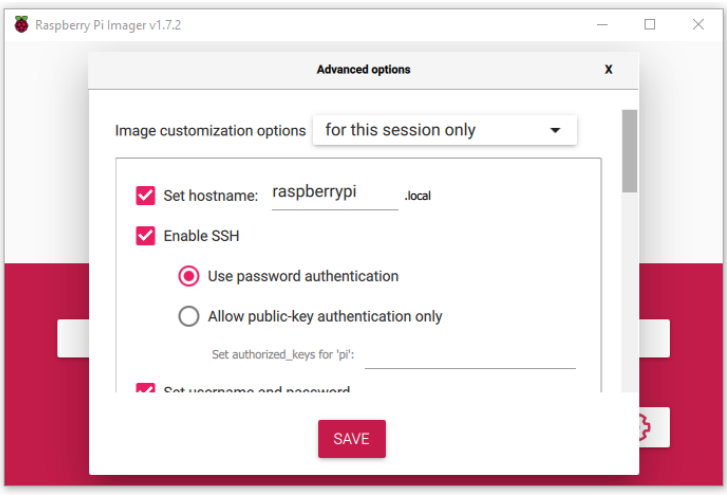



*Figure 7 Choose enable SSH and Use password authentication.*

Next step is to set the username and password. To simplify logging into the pi the first time, keep the username as “pi”, and the password as “pi”.

Complete the details of the wifi network in the Configure wireless LAN. See Figure 8. Enter the SSID and the WiFi password.

The locale setting can be changed as well.

Then click on SAVE to go to the next step.

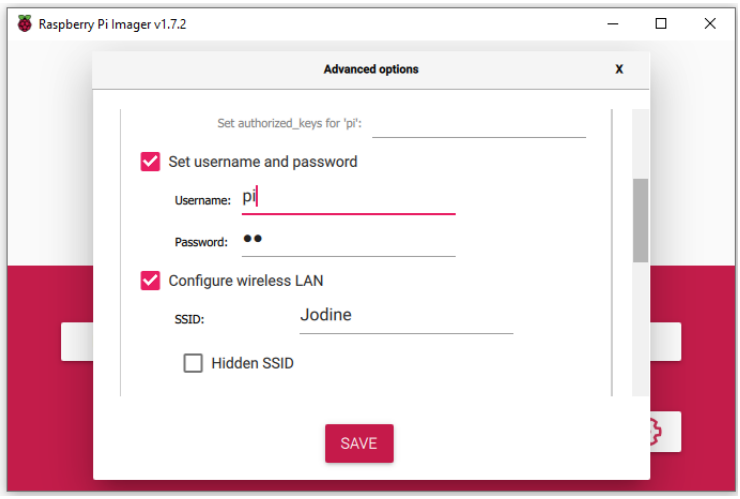



*Figure 8 Enter username and password, configure the WiFi setting.*

**Step 5) –** Review all options and settings and click on WRITE to start the writing of the OS to the SD card. See Figure 9.


*Figure 9 Start the writing to the SD card process by clicking WRITE button.*

**Step 6)** – Once the OS writing process has finished, remove the SD card from your computer or laptop. Insert the SD card into the RPi and power it up.

The power LED wil startup initially as solid green for a couple of seconds and will then start blinking.

The blinking indicates access to the SD card. Wait a minute or two before the next step.

## 3.2 IP Scanners

**Step 7)** – During this step, the network IP range and addresses needs to be determined. On a windows computer open the command prompt (cmd) and run “ipconfig”.

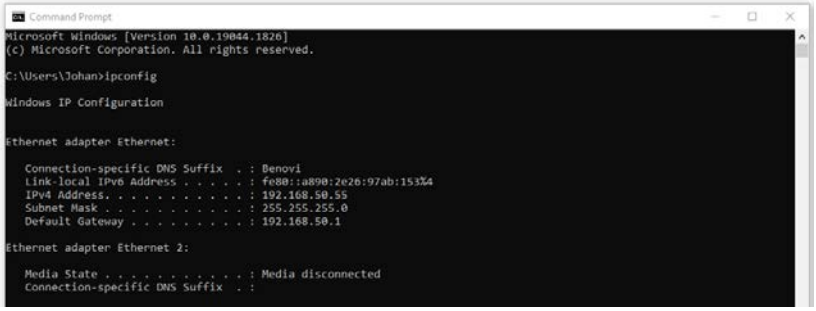



*Figure 10 Run ipconfig in a command prompt window and you will see the IPv4 Address and subnet mask.*

Open Advance IPscanner and make sure the IP address range is similar to your ipconfig results. In Figure 10, an IP address of 192.168.50.55 is shown. In AdvanceIPscanner as well as in AngryIPscanner set the IP address range as 192.168.50.1 to 192.168.50.254. Both these scanners will then detect all devices in this IP range. See Figure 11 and Figure 12 for Angry IP scanner and Advance IP Scanner range settings and results.

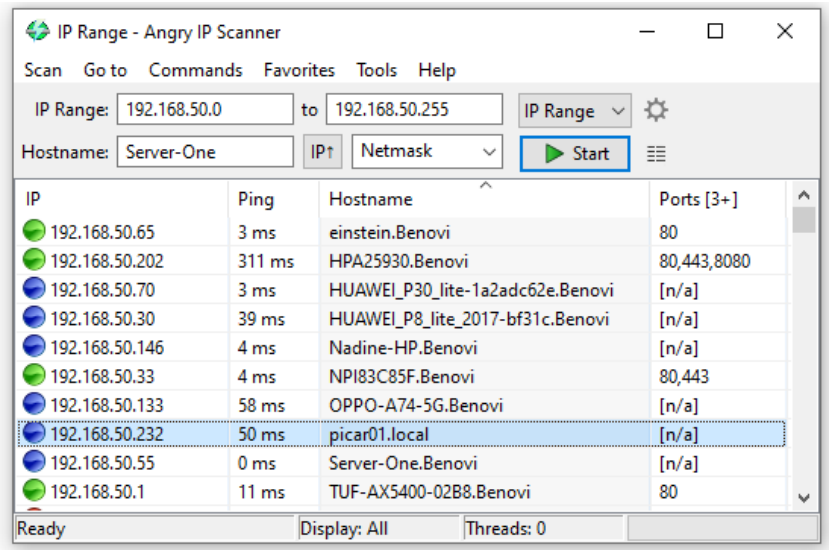



*Figure 11 Angry IP scanner range settings.*

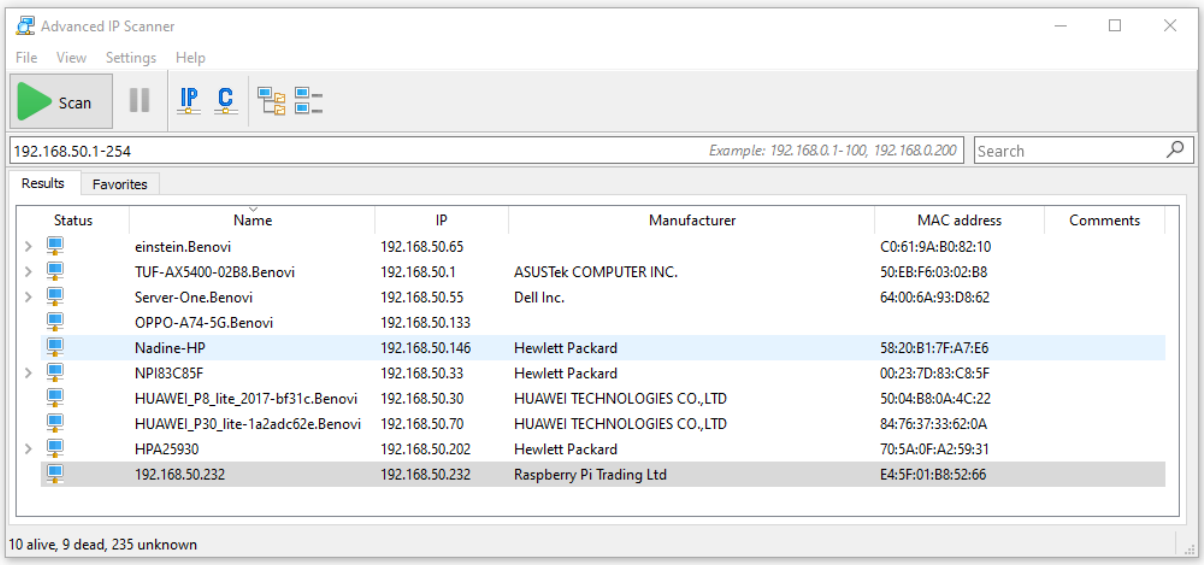



*Figure 12 Advance IP scanner range settings and results.*

The host name of the PiBot being used is picar01. In Figure 11 and Figure 12 the IP address of the picar01 is indicated as 192.168.50.232.

## 3.3 SSH - Putty

**Step 8)** Open Putty and enter the login details of the PiBot you are using. You can either enter the IP address or you can enter the hostname of the PiBot you are using.

In this case we are using host name picar01 or the IP address 192.168.50.232 see Figure 13.

The first time you log into the PiBot, a security alert will appear, Figure 14. Click on accept to move to the next step.

Enter your username “pi” and password “pi”, Figure 15. Figure 16 shows a successful login attempt.

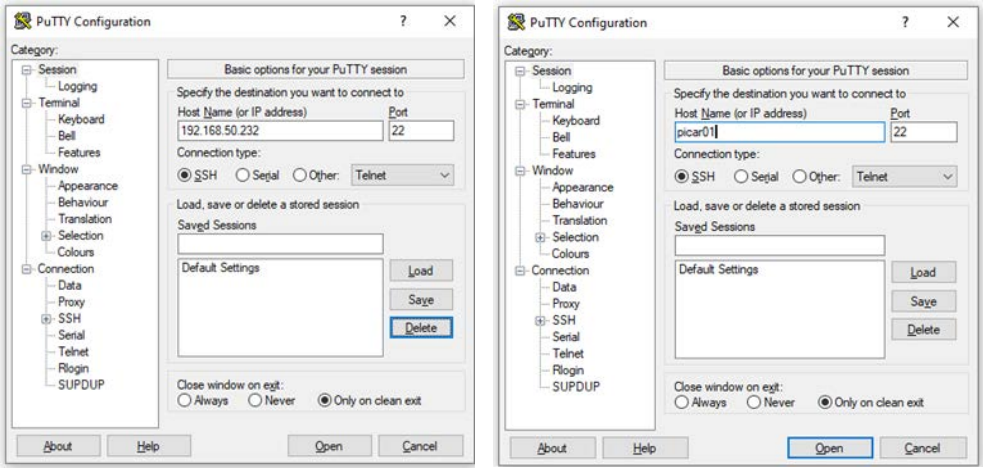



*Figure 13 Using Putty to login to the PiBot*

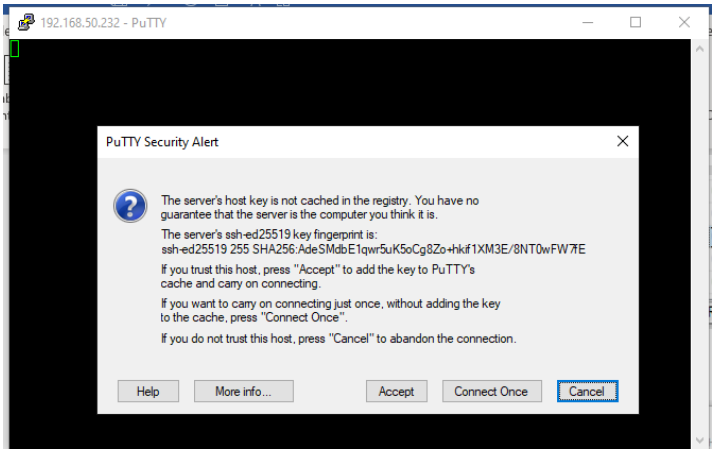



*Figure 14 First time login to the PiBot. Click on Accept.*

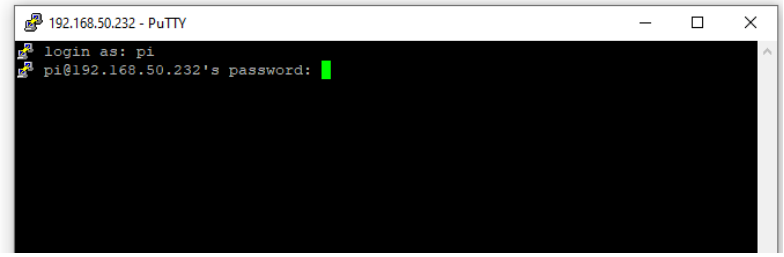



*Figure 15 Login as "pi" and enter password "pi".*

## 3.4 Thonny

The programming editor will be discussed in 6.2 **The Code Editor and Shell (Thonny)**

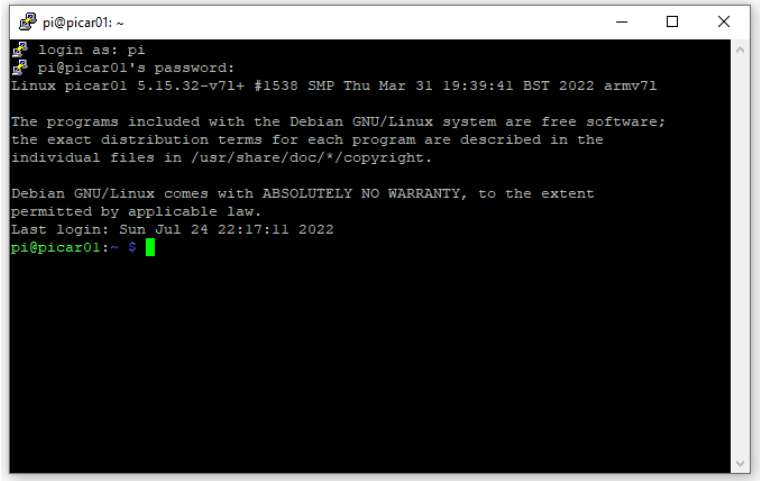



*Figure 16 Successful login to the PiBot.*

**Step 9)** Launch Thonny IDE. This will be used to develop the Python code that can run on the RPi.

Figure 17**,** shows the Thonny IDE after launching.

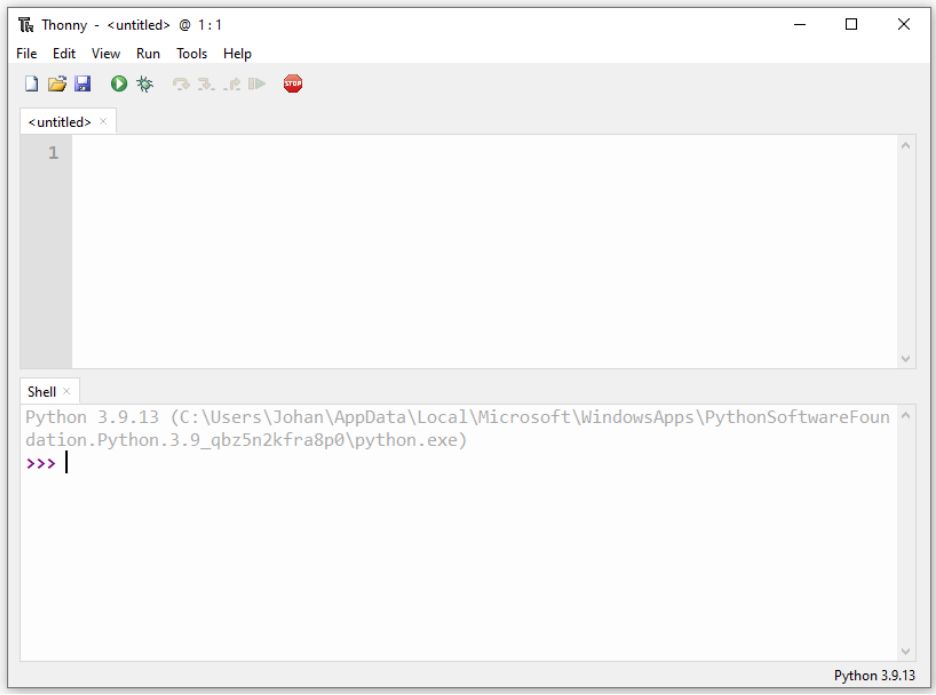



*Figure 17 Thonny IDE*

The default interpreter will be on the local computer or laptop. This needs to be changed to the PiBot. In the bottom right corner, the interpreter is indicated. Click on it and choose "Configure interpreter…". Choose "Remote Python 3 (SSH)" option in "*Which interpreter or device Thonny use for running your code?"*

Enter the host IP address, username and select Authentication method as password. See Figure 18.

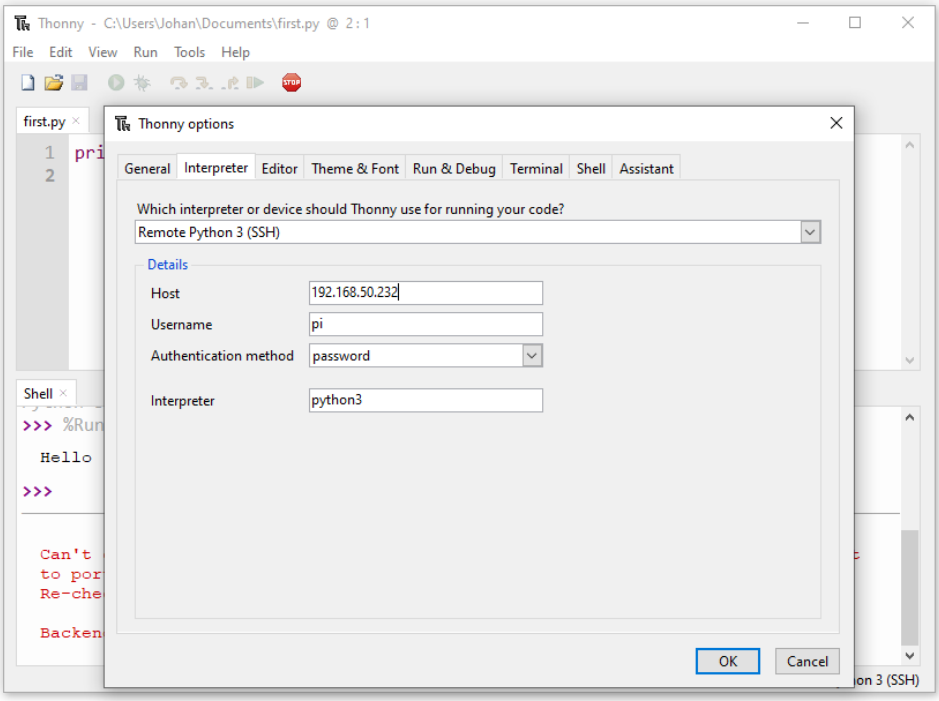



*Figure 18 Thonny setup to connect to the PiBot.*

You are now setup to start programming your PiBot.

# 4 Introduction to the RPI Zero W

The Raspberry Pi is a popular Single Board Computer (SBC) in that it is a full computer packed into a single board. Many may already be familiar with the Raspberry Pi 3 and its predecessors, which comes in a form factor that has become as highly recognizable.

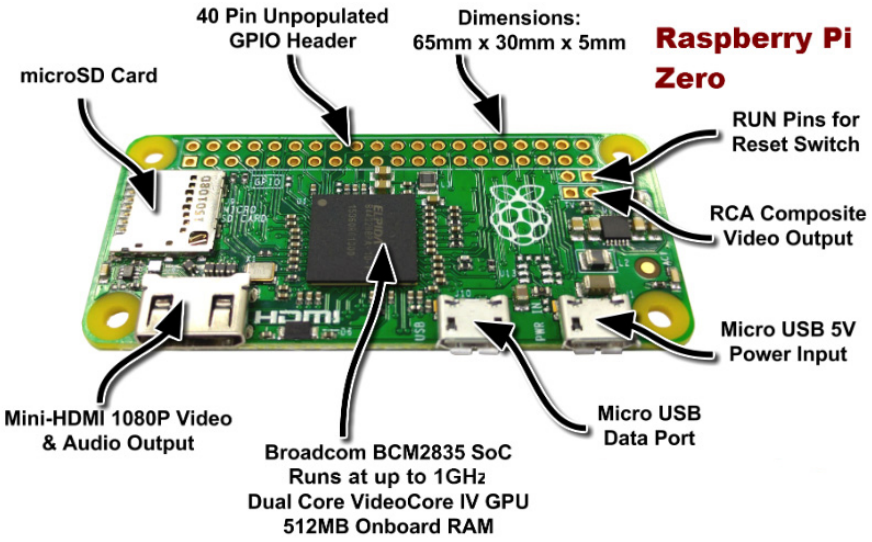



*Figure 19 Raspberry Pi Zero W SBC*

The Raspberry Pi comes in an even smaller form factor. The introduction of the Raspberry Pi Zero allowed one to embed an entire computer in even smaller projects. This guide will cover the latest version of the Zero product line, the Raspberry Pi Zero - Wireless, which has an onboard Wi-Fi module. While these directions should work for most any version and form factor of the Raspberry Pi, it will revolve around the Pi Zero W.

## 4.1 Hardware Overview

Let's go over some of the most noticeable differences between the Raspberry Pi Zero (and Pi Zero W) and the Raspberry Pi 3.

Both boards are identical in features except that the W has built in Wi-Fi and Bluetooth. Getting started with the Pi Zero board can be a little more cumbersome than with the Pi 3 because many of the connectors need adapters to connect to standard size connectors. Otherwise, to get started, all you need is an SD card with a Raspberry Pi image on it and power.

| | |
|---|---|
| **Mini HDMI**<br>Unlike the previous models of the Raspberry Pi which use a standard HDMI connector, the Zero uses a mini-HDMI connector to save space. To connect the Zero to a monitor or television, you will need a mini-HDMI to HDMI adapter or cable. |  |

**USB On-the-Go**

The Raspberry Pi 3 and other models have traditionally had 2-4 standard size female USB connectors, which allowed for all variety of devices to connect including mice, keyboards, and Wi-Fi dongles. To save space, the Zero has opted for a USB On-the-Go (OTG) connection. The Pi Zero uses the same Broadcom IC that powered the original Raspberry Pi A and A+ models. This IC connects directly to the USB port allowing for OTG functionality, unlike the Pi B, B+, 2 and 3 models, which use an onboard USB hub to allow for multiple USB connections.

To connect a device with a standard male USB connection, you will need a USB OTG cable. Plug the microUSB end into the Pi Zero and plug your USB device into the standard female USB end.

For use with other standard USB devices, it is recommended that you use a powered USB hub. Wireless keyboard and mouse combos work best as they have one USB dongle for both devices.

**Power**

Like other Pi's, power is provided through a microUSB connector. Voltage supplied to the power USB should be in the range of 5-5.25V.

The Raspberry Pi does not have a power switch, it will turn on once connected to a power supply.

On the GPIO, the 5V pins (Pins 2 and 4) are directly connected to the 5V0 rail (power supply). This means that you can, in theory, power the raspberry pi via one of these pins.

**microSD Card Slot**

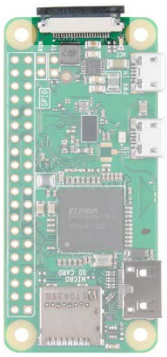
Another familiar interface is the microSD card slot. Insert your microSD cards that contains your Raspberry Pi image file here.

**WiFi and Bluetooth**

As with the Raspberry PI 3, the Zero W offers both 802.11n wireless LAN and Bluetooth 4.0 connectivity. This frees up many of the connections that would have been made over USB, such as a Wi-Fi dongle and a USB keyboard and mouse if substituting a Bluetooth keyboard/mouse.

**Camera Connector**

The Raspberry Pi Zero V1.3+ and all Zero W's have an onboard camera connector. This can be used to attach the Raspberry Pi Camera module. However, the connector is a 22pin 0.5mm and different than the standard Pi. You will need a different cable to connect the camera to the Pi Zero W.



**GPIO**

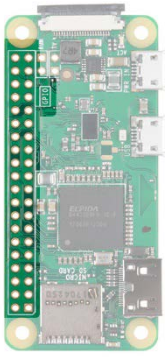
The Raspberry Pi features forty GPIO pins along the top edge of the board. You can use these GPIO pins to connect the Raspberry Pi to external components.

Powering the raspberry through the GPIO also means that your raspberry pi is unprotected and prone to voltage and current surges. Be careful!

GROUND pins are used to ground devices. All ground pins are connected to the same line.

The Raspberry Pi features five different types of pins:

1. **GPIO**: These are general-purpose pins that can be used for input or output.
2. **3V3**: These pins supply a 3.3 V power source for components. 3.3 V is also the internal voltage that all GPIO pins supply.
3. **5V**: These pins supply a 5 V power source, the same as the USB power input that powers the Raspberry Pi. Some components, such as the passive infrared motion sensor, require 5 V.
4. **GND**: These pins provide a ground connection for circuits.
5. **ADV**: These special-purpose pins are advanced.




**Additional Connections**

Last, you may notice two sets of thru hole pads labeled **TV** and **Run**.

The **TV** pads allow you to connect an RCA jack to the board instead of using the HDMI out.

The **Run** pins connect to the chips reset pin and will either turn the board off or turn it back on once it has been shut down. Connecting a button here is a good way to power cycle your board.

For a complete description of each pin on the GPIO header and all the connectors on the PI Zero, consult the graphical datasheet below:

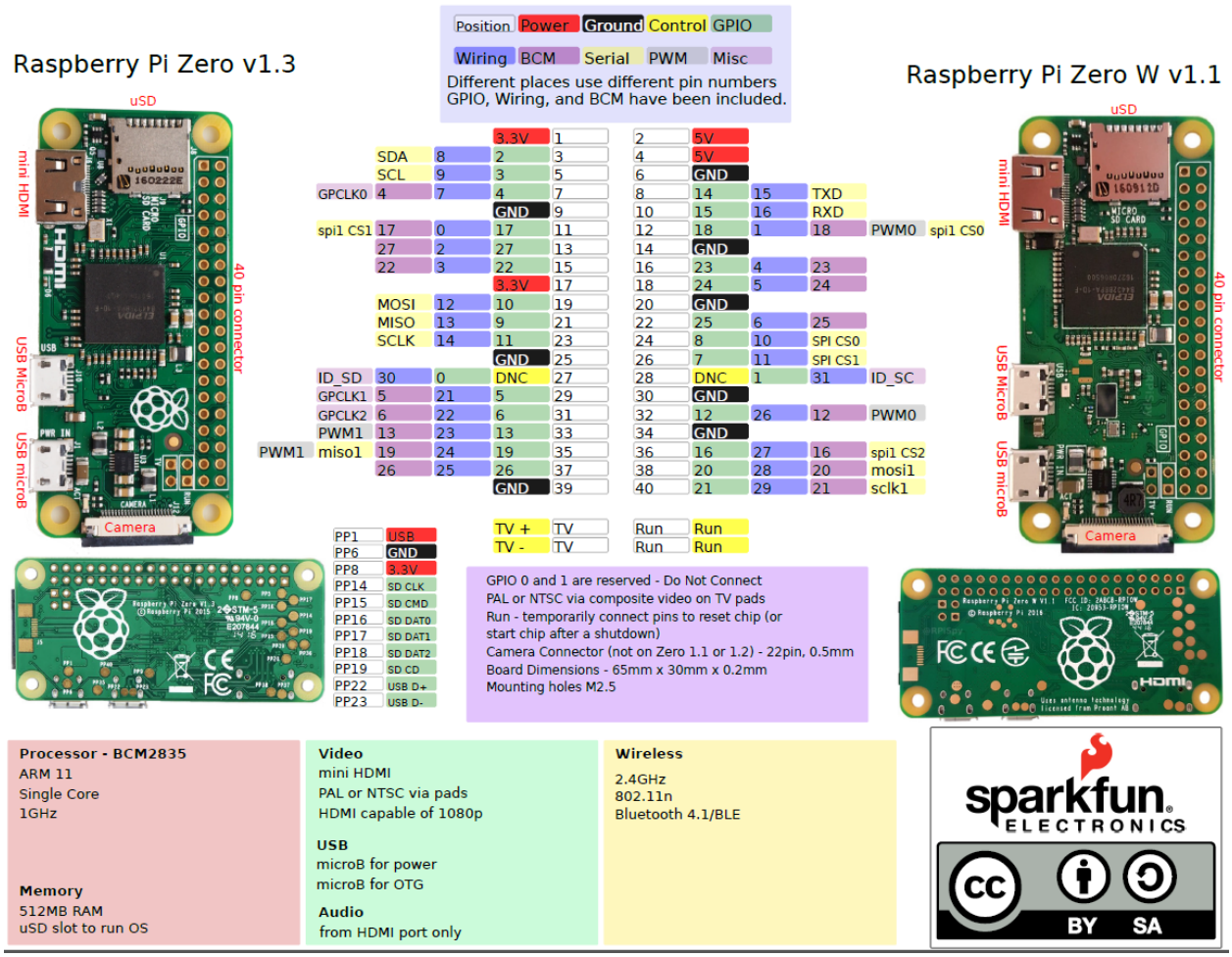



*Figure 20 GPIO Pinout on the RPi Zero W*

# 5 Basic Linux commands

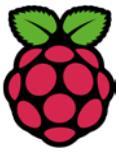
The Raspberry Pi is a small single board computer that runs mostly on Linux. This SBC can be connected to a standard display screen, keyboard and mouse. By doing this it becomes a full functional computer system.

To know your way around the raspberry pi system, a good understanding of the Linux file system is needed: where software is installed, where the danger zones are, how to see what is connected to the rpi, how to navigate the file system and so on. The following section will list and describe the most used Linux commands.

## 5.1 File System

| Command | Description |
|---|---|
| ***ls*** | The ***ls*** command lists the content of the current directory (or one that is specified).<br>It can be used with the **-l** flag to display additional information (permissions, owner, group, size, date and timestamp of last edit) about each file and directory in a list format.<br>The **-a** flag allows you to view files beginning with . (i.e. dotfiles). |
| ***cd*** | Using *cd* changes the current directory to the one specified. |

| | |
|---|---|
| | Relative (i.e., *cd directoryA*) or absolute (i.e., *cd /home/pi/directoryA*) paths can be used. |
| ***pwd*** | The *pwd* command displays the name of the present working directory:<br>On a Raspberry Pi, entering *pwd* will output something like /home/pi. |
| ***mkdir*** | You can use *mkdir* to create a new directory,<br>Example: *mkdir newDir* would create the directory newDir in the present working directory. |
| ***rmdir*** | To remove empty directories, use *rmdir*.<br>For example, *rmdir oldDir* will remove the directory oldDir only if it is empty. |
| ***rm*** | The command *rm* removes the specified file (or recursively from a directory when used with -r).<br>Be careful with this command: files deleted in this way are mostly gone for good! |
| ***cp*** | Using *cp* makes a copy of a file and places it at the specified location (this is similar to copying and pasting). For example, cp ~/fileA /home/otherUser/ would copy the file fileA from your home directory to that of the user otherUser (assuming you have permission to copy it there).<br>This command can either take FILE FILE (*cp fileA fileB*), FILE DIR (*cp fileA /directoryB/*) or -r DIR DIR (which recursively copies the contents of directories) as arguments. |
| ***mv*** | The *mv* command moves a file and places it at the specified location (so where cp performs a 'copy-paste', mv performs a 'cut-paste').<br>The usage is similar to *cp*. So *mv ~/fileA /home/otherUser/* would move the file fileA from your home directory to that of the user otherUser.<br>This command can either take FILE FILE (mv fileA fileB), FILE DIR (mv fileA /directoryB/) or DIR DIR (mv /directoryB /directoryC) as arguments.<br>This command is also useful as a method to rename files and directories after they've been created. |
| ***cat*** | *cat* can be used to list the contents of file(s), e.g. *cat* thisFile will display the contents of thisFile. Can be used to list the contents of multiple files, i.e. *cat *.txt* will list the contents of all .txt files in the current directory. |
| ***chmod*** | *chmod* is used to change the permissions for a file. The *chmod* command can use symbols: u (user that owns the file), g (the files group), and o (other users) and the permissions r (read), w (write), and x (execute).<br>Using *chmod u+x* filename will add execute permission for the owner of the file. |
| ***chown*** | The *chown* command changes the user and/or group that owns a file.<br>It normally needs to be run as root using sudo e.g. *sudo chown pi:root* filename will change the owner to pi and the group to root. |
| ***ssh*** | *ssh* denotes the secure shell.<br>Connect to another computer using an encrypted network connection. |
| ***scp*** | The *scp* command copies a file from one computer to another using ssh. |
| ***sudo*** | The *sudo* command enables you to run a command as a superuser, or another user. |



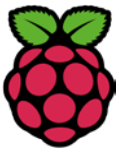
| | |
|---|---|
| | Use *sudo -s* for a superuser shell |
| ***dd*** | The *dd* command copies a file converting the file as specified.<br>It is often used to copy an entire disk to a single file or back again.<br>For example, *dd if=/dev/sdd of=backup.img* will create a backup image from an SD card or USB disk drive at /dev/sdd.<br>Make sure to use the correct drive when copying an image to the SD card as it can overwrite the entire disk. |
| ***df*** | Use ***df*** to display the disk space available and used on the mounted filesystems.<br>Use ***df -h*** to see the output in a human-readable format using M for MBs rather than showing number of bytes. |
| ***tree*** | Use the *tree* command to show a directory and all subdirectories and files indented as a tree structure. |
| ***&*** | Run a command in the background with *&*, freeing up the shell for future commands. |

## 5.2 Search

| Command | Description |
|---|---|
| **grep** | Use *grep* to search inside files for certain search patterns.<br>For example, *grep "search" *.txt* will look in all the files in the current directory ending with .txt for the string search.<br>The *grep* command supports regular expressions which allows special letter combinations to be included in the search. |
| **find** | The *find* command searches a directory and subdirectories for files matching certain patterns. |
| **whereis** | Use *whereis* to find the location of a command.<br>It looks through standard program locations until it finds the requested command. |

## 5.3 Networking

| Command | Description |
|---|---|
| **ping** | The ping utility is usually used to check if communication can be made with another host. It can be used with default settings by just specifying a hostname<br>(e.g., *ping raspberrypi.com*) or an IP address (e.g. *ping 8.8.8.8*).<br>It can specify the number of packets to send with the -c flag. |
| **nmap** | *nmap* is a network exploration and scanning tool. It can return port and OS information about a host or a range of hosts. Running just *nmap* will display the options available as well as example usage. |
| **hostname** | The *hostname* command displays the current hostname of the system.<br>A privileged (super) user can set the hostname to a new one by supplying it as an argument (e.g. *hostname new-host*). |



| | |
|---|---|
| **ifconfig** | Use *ifconfig* to display the network configuration details for the interfaces on the current system when run without any arguments (i.e. *ifconfig*).<br>By supplying the command with the name of an interface (e.g. *eth0* or *lo*) the configuration can be altered. |

# 6 Python Language

Python is a programming language that uses a very simple and clean syntax and dynamic typing that is easy to learn. It uses a simple object orientated programming approach and very high-level data structures.

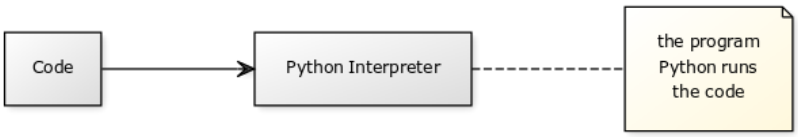
It used in many application domains including Web and Internet Development, Scientific apps, Desktop apps, education and general software applications.

One of the key benefits of Python Programming is its interpretive nature. The Python interpreter and standard library are available in binary or source form from the Python website and can run seamlessly on all major operating systems. Python Programming language is also freely distributable, and the same site even has tips and other third-party tools, programs, modules and more documentation.

The Python interpreter can easily be extended with new data types or functions in C++, C or any other language callable from C. The Python Programming language works as an extension for customizable applications. It uses English keywords rather than punctuation, and it has fewer syntax constructions than other programming languages, this makes it so easy to learn the language.

To run Python programs, you will need the Python interpreter and possibly a graphical editor.

A Python interpreter executes Python code (sometimes called programs).




A program can be one or more Python files. Code files can include other files or modules. To run a program, you need to specify a parameter when executing Python.

There are two ways of using Python to run your program - using the interactive interpreter prompt or using a source file.

## 6.1 Using The Interpreter Prompt

Open the terminal in your operating system and then open the Python prompt by typing *python3* and pressing *enter* key.

After starting Python, you should see **>>>** where you can start typing stuff. The **>>>** is called the Python interpreter prompt.

At the Python interpreter prompt, type:

```
print ("Hello World")
```

followed by the **[enter]** key. You should see the words *Hello World* printed to the screen.

Here is an example of the result. The details about the Python software will differ based on your computer, but the part from the prompt (i.e. from >>> onwards) should be the same regardless of the operating system.

```
$ python3
```

```
Python 3.6.0 (default, Jan 12 2017, 11:26:36)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
```

Python provides the output of the line immediately. The line entered is a single Python *statement*. *print* is used to write something to the screen.

**How to Quit the Interpreter Prompt**

In a GNU/Linux shell the command interpreter, *[ctrl+d]* can be used to exit the command interpreter. Entering *exit()* can also be used. In a Windows command prompt, *[ctrl+z]* followed by the enter key will exit the command interpreter.

At the command interpreter a program cannot be typed out every time, in most cases this will not even be possible. The program must be developed in an editor and then saved to a file. This file can then be executed from the command interpreter any number of times.

Thonny is a free Python Integrated Development Environment (IDE). It has a built-in debugger that can help when bugs are experienced, and it offers the ability to do step through expression evaluation. It can be downloaded and installed from this link: *https://realpython.com/courses/python-thonny/*

**The USER interface**

Thonny is the workroom in which Python projects can be created. It contains a toolbox with many tools that can be used to develop programs. The following section will cover the features available inside the Thonny workroom.

## 6.2 The Code Editor and Shell (Thonny)

After Thonny is started, a window with several icons at the top will be visible. In the window there are two white areas:

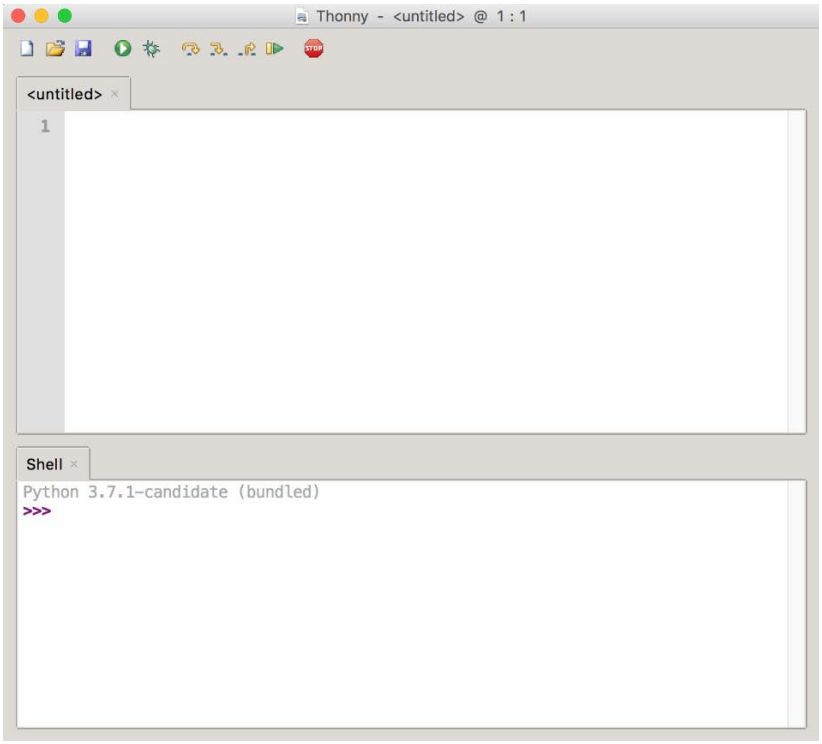



*Figure 21 Thonny workroom with Code editor and the Shell.*

There are the two main sections in the window. The top section is the code editor, where the program will be created with python code. The bottom half is the Shell, where outputs from the developed code will be visible.

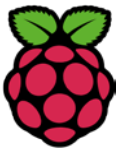
**The ICONS**

Directly above the editor white area, several icons are visible.


*Figure 22 Editor icons.*

From left to right, the description of each of the icons in the image is as follow:

| | |
|---|---|
| **A:** | The paper icon allows you to create a new file. Typically, in Python you want to separate your programs into separate files. |
| **B:** | The open folder icon allows you to open a file that already exists on your computer. This might be useful if you come back to a program that you worked on previously. |
| **C:** | The floppy disk icon allows you to save your code. Press this early and often. |
| **D:** | The play icon allows you to run your code. Remember that the code you write is meant to be executed. Running your code means you're telling Python, "Do what I told you to do!" (In other words, "Read through my code and execute what I wrote.") |
| **E:** | The bug icon allows you to debug your code. It's inevitable that you will encounter bugs when you're writing code. A bug is another word for a problem. Bugs can come in many |



| | |
|---|---|
| | forms, sometimes appearing when you use inappropriate syntax and sometimes when your logic is incorrect. |
| **Thonny's bug button is typically used to spot and investigate bugs. You'll work with this later in the tutorial. By the way, if you're wondering why they're called bugs, there's also a fun story of how it came about!** | |
| **F-H:** | The arrow icons allow you to run your programs step by step. This can be very useful when you're debugging or, in other words, trying to find those nasty bugs in your code. These icons are used after you press the bug icon. You'll notice as you hit each arrow, a yellow highlighted bar will indicate which line or section Python is currently evaluating: |
| | The F arrow tells Python to take a big step, meaning jumping to the next line or block of code. |
| | The G arrow tells Python to take a small step, meaning diving deep into each component of an expression. |
| | The H arrow tells Python to exit out of the debugger. |
| **I:** | The resume icon allows you to return to play mode from debug mode. This is useful in the instance when you no longer want to go step by step through the code, and instead want your program to finish running. |
| **J:** | The stop icon allows you to stop running your code. This can be particularly useful if, let's say, your code runs a program that opens a new window, and you want to stop that program. |

## 6.3 Basics of the Python language

## Comments

Comments are any text to the right of the # symbol and is mainly useful as notes for the reader of the program.

For example:

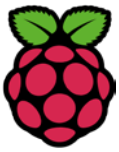
```
print('hello world') # Note that print is a function
```

or:

```
# Note that print is a function
print('hello world')
```

Use as many useful comments as you can in your program to:

- explain assumptions
- explain important decisions
- explain important details
- explain problems you're trying to solve
- explain problems you're trying to overcome in your program, etc.



Code tells you how, comments should tell you why.

This is useful for readers of your program so that they can easily understand what the program is doing. Remember, that person can be yourself after six months!

**Literal Constants**

An example of a literal constant is a number like 5, 1.23, or a string like 'This is a string' or "It's a string!".

It is called a literal because it is literal - you use its value literally. The number 2 always represents itself and nothing else - it is a constant because its value cannot be changed. Hence, all these are referred to as literal constants.

**Numbers**

Numbers are mainly of two types - integers and floats.

An example of an integer is 2 which is just a whole number.

Examples of floating-point numbers (or floats for short) are 3.23 and 52.3E-4. The E notation indicates powers of 10. In this case, 52.3E-4 means 52.3 * 10^-4^.  The int type can be an integer of any size.

**Strings**

A string is a sequence of characters. Strings are basically just a bunch of words. You will be using strings in almost every Python program that you write, so pay attention to the following part.

**Single Quote**

You can specify strings using single quotes such as 'Quote me on this'.

All white space i.e. spaces and tabs, within the quotes, are preserved as-is.

**Double Quotes**

Strings in double quotes work the same way as strings in single quotes. An example is "What's your name?".

**Triple Quotes**

You can specify multi-line strings using triple quotes - (""" or '''). You can use single quotes and double quotes freely within the triple quotes.

The following is an example:

```
'''This is a multi-line string. This is the first line.
This is the second line.
"What's your name?," I asked.
He said "Bond, James Bond."
'''
```

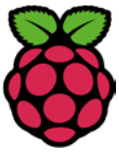
## 6.4 Operators and Expressions

Most statements (logical lines) that you write will contain expressions. A simple example of an expression is *2 + 3*. An expression can be broken down into operators and operands.

Operators are functionality that do something and can be represented by symbols such as + or by special keywords. Operators require some data to operate on and such data is called operands. In this case, 2 and 3 are the operands.

**Operators**

A brief look at operators and their usage.

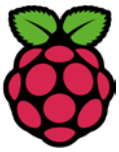


Note that you can evaluate the expressions given in the examples using the interpreter interactively. For example, to test the expression 2 + 3, use the interactive Python interpreter prompt:

```
>>> 2 + 3
5
>>> 3 * 5
15
>>>
```

Available operators:

| | |
|---|---|
| **+** | **Adds two objects: 3 + 5 gives 8. 'a' + 'b' gives 'ab'.** |
| **-** | Gives the subtraction of one number from the other; if the first operand is absent it is assumed to be zero. -5.2 gives a negative number and 50 - 24 gives 26. |
| ***** | Gives the multiplication of the two numbers or returns the string repeated that many times.<br>2 * 3 gives 6. 'la' * 3 gives 'lalala'. |
| ****** | Returns x to the power of y, 3 ** 4 gives 81 (i.e. 3 * 3 * 3 * 3) |
| **/** | Divide x by y: 13 / 3 gives 4.333333333333333 |
| **//** | Divide x by y and round the answer down to the nearest integer value. Note that if one of the values is a float, you'll get back a float.<br>13 // 3 gives 4<br>-13 // 3 gives -5<br>9//1.81 gives 4.0 |
| **%** | Returns the remainder of the division: 13 % 3 gives 1. -25.5 % 2.25 gives 1.5. |
| **<<** | Shifts the bits of the number to the left by the number of bits specified. (Each number is represented in memory by bits or binary digits i.e. 0 and 1): 2 << 2 gives 8. 2 is represented by 10 in bits. |
| **>>** | Shifts the bits of the number to the right by the number of bits specified.<br>11 >> 1 gives 5.<br>11 is represented in bits by 1011 which when right shifted by 1 bit gives 101 which is the decimal 5. |
| **&** | Bit-wise AND of the numbers: if both bits are 1, the result is 1. Otherwise, it's 0.<br>5 & 3 gives 1 (0101 & 0011 gives 0001) |
| **\|** | Bitwise OR of the numbers: if both bits are 0, the result is 0. Otherwise, it's 1.<br>5 \| 3 gives 7 (0101 \| 0011 gives 0111) |
| **^** | Bitwise XOR of the numbers: if both bits (1 or 0) are the same, the result is 0. Otherwise, it's 1.<br>5 ^ 3 gives 6 (O101 ^ 0011 gives 0110) |
| **~** | The bit-wise inversion of x is -(x+1) |



| | |
|---|---|
| | ~5 gives -6 |
| < | Returns whether x is less than y. All comparison operators return True or False. Note the capitalization of these names.<br>5 < 3 gives False and 3 < 5 gives True.<br>Comparisons can be chained arbitrarily: 3 < 5 < 7 gives True. |
| > | Returns whether x is greater than y<br>5 > 3 returns True. If both operands are numbers, they are first converted to a common type. Otherwise, it always returns False. |
| <= | Returns whether x is less than or equal to y: x = 3; y = 6; x <= y returns True |
| >= | Returns whether x is greater than or equal to y: x = 4; y = 3; x >= 3 returns True |
| == | Compares if the objects are equal<br>x = 2; y = 2; x == y returns True<br>x = 'str'; y = 'stR'; x == y returns False<br>x = 'str'; y = 'str'; x == y returns True |
| != | Compares if the objects are not equal<br>x = 2; y = 3; x != y returns True |
| not | f x is True, it returns False. If x is False, it returns True. x = True; not x returns False. |
| and | x and y returns False if x is False, else it returns evaluation of y<br>x = False; y = True; x and y returns False since x is False. In this case, Python will not evaluate y since it knows that the left-hand side of the 'and' expression is False which implies that the whole expression will be False irrespective of the other values. This is called short-circuit evaluation. |
| or | If x is True, it returns True, else it returns evaluation of y<br>x = True; y = False; x or y returns True. Short-circuit evaluation applies here as well. |

**Shortcut for math operation and assignment**

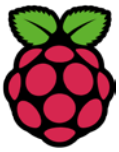
It is common to run a math operation on a variable and then assign the result of the operation back to the variable, hence there is a shortcut for such expressions:

| | |
|---|---|
| a = 2<br>a = a * 3 | a = 2<br>a *= 3 |

Notice that var = var operation expression becomes var operation= expression.

**Evaluation Order**

If you had an expression such as 2 + 3 * 4, is the addition done first or the multiplication? Our high school math's tells us that the multiplication should be done first. This means that the multiplication operator has higher precedence than the addition operator.



The following table gives the precedence table for Python, from the lowest precedence (least binding) to the highest precedence (most binding). This means that in each expression, Python will first evaluate the operators and expressions lower in the table before the ones listed higher in the table. The following table, taken from the Python reference manual, is provided for the sake of completeness. It is far better to use parentheses to group operators and operands appropriately to explicitly specify the precedence. This makes the program more readable.

| |
|---|
| lambda : Lambda Expression |
| if - else : Conditional expression |
| or : Boolean OR |
| and : Boolean AND |
| not x : Boolean NOT |
| in, not in, is, is not, <, <=, >, >=, !=, == : Comparisons, including membership tests and identity tests |
| \| : Bitwise OR |
| ^ : Bitwise XOR |
| & : Bitwise AND |
| <<, >> : Shifts |
| +, - : Addition and subtraction |
| *, /, //, % : Multiplication, Division, Floor Division and Remainder |
| +x, -x, ~x : Positive, Negative, bitwise NOT |
| ** : Exponentiation |
| x[index], x[index:index], x(arguments...), x.attribute : Subscription, slicing, call, attribute reference |
| (expressions...), [expressions...], {key: value...}, {expressions...} : Binding or tuple display, list display, dictionary display, set display |

Operators with the same precedence are listed in the same row in the above table. For example, + and - have the same precedence.

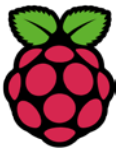
**Changing the Order of Evaluation**

To make the expressions more readable, we can use parentheses. For example, 2 + (3 * 4) is easier to understand than 2 + 3 * 4 which requires knowledge of the operator precedence's. As with everything else, the parentheses should be used reasonably (do not overdo it) and should not be redundant, as in (2 + (3 * 4)).

There is an additional advantage to using parentheses - it helps us to change the order of evaluation. For example, if you want addition to be evaluated before multiplication in an expression, then you can write something like (2 + 3) * 4.

**Associativity**

Operators are usually associated from left to right. This means that operators with the same precedence are evaluated in a left to right manner. For example, 2 + 3 + 4 is evaluated as (2 + 3) + 4.



**Expressions**

Example (save as expression.py):

```
length = 5
breadth = 2
area = length * breadth
print('Area is' area)
print('Perimeter is' 2 * (length + breadth))
```

Output:

```
$ python expression.py
Area is 10
Perimeter is 14
```

**How It Works**

The length and breadth of the rectangle are stored in variables by the same name. We use these to calculate the area and perimeter of the rectangle with the help of expressions. We store the result of the expression length * breadth in the variable area and then print it using the print function. In the second case, we directly use the value of the expression 2 * (length + breadth) in the print function.

Also, notice how Python pretty-prints the output. Even though we have not specified a space between 'Area is' and the variable area, Python puts it for us so that we get a nice clean output, and the program is much more readable this way (since we don't need to worry about spacing in the strings we use for output). This is an example of how Python makes life easy for the programmer.

## 6.5 Control Fow

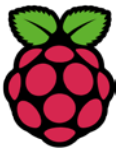
In the programs we have seen till now, there has always been a series of statements faithfully executed by Python in exact top-down order. What if you wanted to change the flow of how it works? For example, you want the program to take some decisions and do different things depending on different situations, such as printing 'Good Morning' or 'Good Evening' depending on the time of the day?

As you might have guessed, this is achieved using control flow statements. There are three control flow statements in Python - *if, for* and *while*.

**The if statement**

The if statement is used to check a condition: if the condition is true, we run a block of statements (called the if-block), else we process another block of statements (called the else-block). The else clause is optional.

Example (save as if.py):



```
number = 23
guess = int(input('Enter an integer : '))
if guess == number:
    # New block starts here
    print('Congratulations, you guessed it.')
    print('(but you do not win any prizes!)')
    # New block ends here
elif guess < number:
    # Another block
    print('No, it is a little higher than that')
else:
    print('No, it is a little lower than that')
print('Done')
# This last statement is always executed, after the if statement is executed.
```

Output:

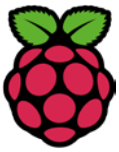
```
$ python if.py
Enter an integer : 50
No, it is a little lower than that
Done
$ python if.py
Enter an integer : 22
No, it is a little higher than that
Done
$ python if.py
Enter an integer : 23
Congratulations, you guessed it.
(but you do not win any prizes!)
Done
```

**How It Works**

In this program, we take guesses from the user and check if it is the number that we have. We set the variable number to any integer we want, say 23. Then, we take the user's guess using the input() function. Functions are just reusable pieces of programs.

We supply a string to the built-in input function which prints it to the screen and waits for input from the user. Once we enter something and press kbd:[enter] key, the input() function returns what we entered, as a string. We then convert this string to an integer using int and then store it in the variable guess. Actually, the int is a class but all you need to know right now is that you can use it to convert a string to an integer (assuming the string contains a valid integer in the text).

Next, we compare the guess of the user with the number we have chosen. If they are equal, we print a success message. Notice that we use indentation levels to tell Python which statements belong to which



block. This is why indentation is so important in Python. I hope you are sticking to the "consistent indentation" rule. Are you?

Notice how the if statement contains a colon at the end - we are indicating to Python that a block of statements follows.

Then, we check if the guess is less than the number, and if so, we inform the user that they must guess a little higher than that. What we have used here is the elif clause which actually combines two related if else-if else statements into one combined if-elif-else statement. This makes the program easier and reduces the amount of indentation required.

The elif and else statements must also have a colon at the end of the logical line followed by their corresponding block of statements (with proper indentation, of course)

You can have another if statement inside the if-block of an if statement and so on - this is called a nested if statement.

Remember that the elif and else parts are optional. A minimal valid if statement is:

```
if True:
    print('Yes, it is true')
```

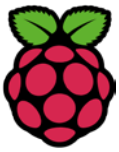
After Python has finished executing the complete if statement along with the associated elif and else clauses, it moves on to the next statement in the block containing the if statement. In this case, it is the main block (where execution of the program starts), and the next statement is the print('Done') statement. After this, Python sees the ends of the program and simply finishes up.

Even though this is a very simple program, I have been pointing out a lot of things that you should notice. All these are straightforward (and surprisingly simple for those of you from C/C++ backgrounds). You will need to become aware of all these things initially, but after some practice you will become comfortable with them, and it will all feel 'natural' to you.

**The while Statement**

The while statement allows you to repeatedly execute a block of statements as long as a condition is true. A while statement is an example of what is called a looping statement. A while statement can have an optional else clause.

Example (save as while.py):



```
number = 23
running = True
while running:
    guess = int(input('Enter an integer : '))
    if guess == number:
        print('Congratulations, you guessed it.')
        # this causes the while loop to stop
        running = False
    elif guess < number:
        print('No, it is a little higher than that.')
    else:
        print('No, it is a little lower than that.')
else:
    print('The while loop is over.')
    # Do anything else you want to do here
print('Done')
```

Output:

```
$ python while.py
Enter an integer : 50
No, it is a little lower than that.
Enter an integer : 22
No, it is a little higher than that.
Enter an integer : 23
Congratulations, you guessed it.
The while loop is over.
Done
```

**How It Works**

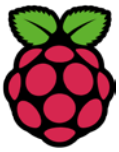
In this program, we are still playing the guessing game, but the advantage is that the user is allowed to keep guessing until he guesses correctly - there is no need to repeatedly run the program for each guess, as we have done in the previous section. This aptly demonstrates the use of the while statement.

We move the input and if statements to inside the while loop and set the variable running to True before the while loop. First, we check if the variable running is True and then proceed to execute the corresponding while-block.

After this block is executed, the condition is again checked which in this case is the running variable. If it is true, we execute the while-block again, else we continue to execute the optional else-block and then continue to the next statement.

The else block is executed when the while loop condition becomes False - this may even be the first time that the condition is checked. If there is an else clause for a while loop, it is always executed unless you break out of the loop with a break statement.

The True and False are called Boolean types and you can consider them to be equivalent to the value 1 and 0 respectively.



**The for loop**

The for..in statement is another looping statement which iterates over a sequence of objects i.e. go through each item in a sequence.

Example (save as for.py):

```
for i in range(1, 5):
    print(i)
else:
    print('The for loop is over')
```

Output:

```
$ python for.py
1
2
3
4
The for loop is over
```

**How It Works**

In this program, we are printing a sequence of numbers. We generate this sequence of numbers using the built-in range function.

What we do here is supply it two numbers and range returns a sequence of numbers starting from the first number and up to the second number. For example, range(1,5) gives the sequence [1, 2, 3, 4]. By default, range takes a step count of 1. If we supply a third number to range, then that becomes the step count. For example, range(1,5,2) gives [1,3]. Remember that the range extends up to the second number i.e. it does not include the second number.

Note that range() generates only one number at a time, if you want the full list of numbers, call list() on the range(), for example, list(range(5)) will result in [0, 1, 2, 3, 4]. Lists are explained in the data structures chapter.

The for loop then iterates over this range - for i in range(1,5) is equivalent to for i in [1, 2, 3, 4] which is like assigning each number (or object) in the sequence to i, one at a time, and then executing the block of statements for each value of i. In this case, we just print the value in the block of statements.

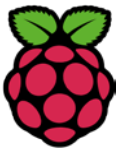
Remember that the else part is optional. When included, it is always executed once after the for loop is over unless a break statement is encountered.

Remember that the for..in loop works for any sequence. Here, we have a list of numbers generated by the built-in range function, but in general we can use any kind of sequence of any kind of objects! We will explore this idea in detail in later chapters.

**The break Statement**

The break statement is used to break out of a loop statement i.e. stop the execution of a looping statement, even if the loop condition has not become False or the sequence of items has not been completely iterated over.

An important note is that if you break out of a for or while loop, any corresponding loop else block is not executed.



Example (save as break.py):

```
while True:
    s = input('Enter something : ')
    if s == 'quit':
        break
    print('Length of the string is', len(s))
print('Done')
```

Output:

```
$ python break.py
Enter something : Programming is fun
Length of the string is 18
Enter something : When the work is done
Length of the string is 21
Enter something : if you wanna make your work also fun:
Length of the string is 37
Enter something : use Python!
Length of the string is 11
Enter something : quit
Done
```

**How It Works**

In this program, we repeatedly take the user's input and print the length of each input each time. We are providing a special condition to stop the program by checking if the user input is 'quit'. We stop the program by breaking out of the loop and reach the end of the program.

The length of the input string can be found out using the built-in len function.

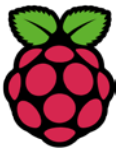
Remember that the break statement can be used with the for loop as well.

### The continue Statement

The continue statement is used to tell Python to skip the rest of the statements in the current loop block and to continue to the next iteration of the loop.

Example (save as continue.py):

```
while True:
    s = input('Enter something : ')
    if s == 'quit':
        break
    if len(s) < 3:
        print('Too small')
        continue
    print('Input is of sufficient length')
    # Do other kinds of processing here...
```



Output:

```
$ python continue.py
Enter something : a
Too small
Enter something : 12
Too small
Enter something : abc
Input is of sufficient length
Enter something : quit
```

**How It Works**

In this program, we accept input from the user, but we process the input string only if it is at least 3 characters long. So, we use the built-in len function to get the length and if the length is less than 3, we skip the rest of the statements in the block by using the continue statement. Otherwise, the rest of the statements in the loop are executed, doing any kind of processing we want to do here.

Note that the continue statement works with the for loop as well.

## 6.6 Functions

Functions are reusable pieces of programs. They allow you to give a name to a block of statements, allowing you to run that block using the specified name anywhere in your program and any number of times. This is known as calling the function. We have already used many built-in functions such as *len* and *range.*

The function concept is probably the most important building block of any non-trivial software (in any programming language).

Functions are defined using the *def* keyword. After this keyword comes an identifier name for the function, followed by a pair of parentheses which may enclose some names of variables, and by the final colon that ends the line. Next follows the block of statements that are part of this function. An example will show that this is actually very simple:

Example (save as function1.py):

```
def say_hello():
    # block belonging to the function
    print('hello world')
# End of function
say_hello()  # call the function
say_hello()  # call the function again
```

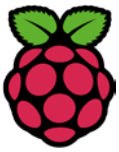
Output:

```
$ python function1.py
hello world
hello world
```

**How It Works**

We define a function called say hello using the syntax as explained above. This function takes no parameters and hence there are no variables declared in the parentheses. Parameters to functions are just input to the function so that we can pass in different values to it and get back corresponding results.

Notice that we can call the same function twice which means we do not have to write the same code again.



**Function Parameters**

A function can take parameters, which are values you supply to the function so that the function can do something utilizing those values. These parameters are just like variables except that the values of these variables are defined when we call the function and are already assigned values when the function runs.

Parameters are specified within the pair of parentheses in the function definition, separated by commas. When we call the function, we supply the values in the same way. Note the terminology used - the names given in the function definition are called parameters whereas the values you supply in the function call are called arguments.

Example (save as function_param.py):

```
def print_max(a, b):
    if a > b:
        print(a, 'is maximum')
    elif a == b:
        print(a, 'is equal to', b)
    else:
        print(b, 'is maximum')
# directly pass literal values
print_max(3, 4)
x = 5
y = 7
# pass variables as arguments
print_max(x, y)
```

Output:

```
$ python function_param.py
4 is maximum
7 is maximum
```

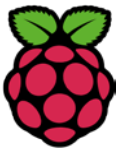
**How It Works**

Here, we define a function called print_max that uses two parameters called a and b. We find out the greater number using a simple if..else statement and then print the bigger number.

The first time we call the function print_max, we directly supply the numbers as arguments. In the second case, we call the function with variables as arguments. print_max(x, y) causes the value of argument x to be assigned to parameter a and the value of argument y to be assigned to parameter b. The print_max function works the same way in both cases.

**Local Variables**

When you declare variables inside a function definition, they are not related in any way to other variables with the same names used outside the function - i.e. variable names are local to the function. This is called the scope of the variable. All variables have the scope of the block they are declared in starting from the point of definition of the name.



Example (save as function_local.py):

```
x = 50
def func(x):
    print('x is', x)
    x = 2
    print('Changed local x to', x)
func(x)
print('x is still', x)
```

Output:

```
$ python function_local.py
x is 50
Changed local x to 2
x is still 50
```

**How It Works**

The first time that we print the value of the name x with the first line in the function's body, Python uses the value of the parameter declared in the main block, above the function definition.

Next, we assign the value 2 to x. The name x is local to our function. So, when we change the value of x in the function, the x defined in the main block remains unaffected.

With the last print statement, we display the value of x as defined in the main block, thereby confirming that it is unaffected by the local assignment within the previously called function.

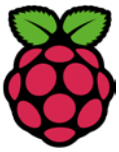
**The global statement**

If you want to assign a value to a name defined at the top level of the program (i.e. not inside any kind of scope such as functions or classes), then you have to tell Python that the name is not local, but it is global. We do this using the global statement. It is impossible to assign a value to a variable defined outside a function without the global statement.

You can use the values of such variables defined outside the function (assuming there is no variable with the same name within the function). However, this is not encouraged and should be avoided since it becomes unclear to the reader of the program as to where that variable's definition is. Using the global statement makes it amply clear that the variable is defined in an outermost block.

Example (save as function_global.py):

```
x = 50
def func():
    global x
    print('x is', x)
    x = 2
    print('Changed global x to', x)
func()
print('Value of x is', x)
```



Output:

$ python function_global.py

```
x is 50
Changed global x to 2
Value of x is 2
```

**How It Works**

The global statement is used to declare that x is a global variable - hence, when we assign a value to x inside the function, that change is reflected when we use the value of x in the main block.

You can specify more than one global variable using the same global statement e.g. global x, y, z.

**Default Argument Values**

For some functions, you may want to make some parameters optional and use default values in case the user does not want to provide values for them. This is done with the help of default argument values. You can specify default argument values for parameters by appending to the parameter name in the function definition the assignment operator (=) followed by the default value.

Note that the default argument value should be a constant. More precisely, the default argument value should be immutable - this is explained in detail in later chapters. For now, just remember this.

Example (save as function_default.py):

```
def say(message, times=1):
    print(message * times)
say('Hello')
say('World', 5)
```

Output:

```
$ python function_default.py
Hello
WorldWorldWorldWorldWorld
```

**How It Works**

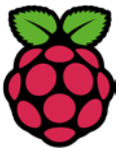
The function named say is used to print a string as many times as specified. If we don't supply a value, then by default, the string is printed just once. We achieve this by specifying a default argument value of 1 to the parameter times.

In the first usage of say, we supply only the string, and it prints the string once. In the second usage of say, we supply both the string and an argument 5 stating that we want to say the string message 5 times.

**CAUTION**

Only those parameters which are at the end of the parameter list can be given default argument values i.e. you cannot have a parameter with a default argument value preceding a parameter without a default argument value in the function's parameter list.

This is because the values are assigned to the parameters by position. For example, def func(a, b=5) is valid, but def func(a=5, b) is not valid.



**Keyword Arguments**

If you have some functions with many parameters and you want to specify only some of them, then you can give values for such parameters by naming them - this is called keyword arguments - we use the name (keyword) instead of the position (which we have been using all along) to specify the arguments to the function.

There are two advantages

- One, using the function is easier since we do not need to worry about the order of the arguments.
- Two, we can give values to only those parameters to which we want to, provided that the other parameters have default argument values.

Example (save as function_keyword.py):

```
def func(a, b=5, c=10):
  print('a is', a, 'and b is', b, 'and c is', c)
func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

Output:

```
$ python function_keyword.py
a is 3 and b is 7 and c is 10
a is 25 and b is 5 and c is 24
a is 100 and b is 5 and c is 50
```

**How It Works**

The function named func has one parameter without a default argument value, followed by two parameters with default argument values.

In the first usage, func(3, 7), the parameter a gets the value 3, the parameter b gets the value 7 and c gets the default value of 10.

In the second usage func(25, c=24), the variable a gets the value of 25 due to the position of the argument. Then, the parameter c gets the value of 24 due to naming i.e. keyword arguments. The variable b gets the default value of 5.

In the third usage func(c=50, a=100), we use keyword arguments for all specified values. Notice that we are specifying the value for parameter c before that for a even though a is defined before c in the function definition.

**VarArgs parameters**

Sometimes you might want to define a function that can take any number of parameters, i.e. variable number of arguments, this can be achieved by using the stars (save as function_varargs.py):

```
def total(a=5, *numbers, **phonebook):
  print('a', a)
  #iterate through all the items in tuple
  for single_item in numbers:
    print('single_item', single_item)
  #iterate through all the items in dictionary
  for first_part, second_part in phonebook.items():
```

```
    print(first_part,second_part)
total(10,1,2,3,Jack=1123,John=2231,Inge=1560)
```

Output:

$ python function_varargs.py

```
a 10
single_item 1
single_item 2
single_item 3
Inge 1560
John 2231
Jack 1123
```

**How It Works**

When we declare a starred parameter such as *param, then all the positional arguments from that point till the end are collected as a tuple called 'param'.

Similarly, when we declare a double-starred parameter such as **param, then all the keyword arguments from that point till the end are collected as a dictionary called 'param'.

**The return statement**

The return statement is used to return from a function i.e. break out of the function. We can optionally return a value from the function as well.

Example (save as function_return.py):

```
def maximum(x, y):
   if x > y:
      return x
   elif x == y:
      return 'The numbers are equal'
   else:
      return y
print(maximum(2, 3))
```

Output:

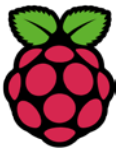
```
$ python function_return.py
3
```

**How It Works**

The maximum function returns the maximum of the parameters, in this case the numbers supplied to the function. It uses a simple if..else statement to find the greater value and then returns that value.

Note that a return statement without a value is equivalent to return None. None is a special type in Python that represents nothingness. For example, it is used to indicate that a variable has no value if it has a value of None.

Every function implicitly contains a return None statement at the end unless you have written your own return statement. You can see this by running print(some_function()) where the function some_function does not use the return statement such as:



*def some_function():*

*pass*

The pass statement is used in Python to indicate an empty block of statements.

*TIP: There is a built-in function called max that already implements the 'find maximum' functionality, so use this built-in function whenever possible.*

**DocStrings**

Python has a nifty feature called documentation strings, usually referred to by its shorter name docstrings. DocStrings are an important tool that you should make use of since it helps to document the program better and makes it easier to understand. Amazingly, we can even get the docstring back from, say a function, when the program is running!

Example (save as function_docstring.py):

```
def print_max(x, y):
    '''Prints the maximum of two numbers.
    The two values must be integers.'''
    # convert to integers, if possible
    x = int(x)
    y = int(y)
    if x > y:
        print(x, 'is maximum')
    else:
        print(y, 'is maximum')
print_max(3, 5)
print(print_max.__doc__)
```

Output:

```
$ python function_docstring.py
5 is maximum
Prints the maximum of two numbers.
    The two values must be integers.
```

**How It Works**

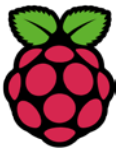
A string on the first logical line of a function is the docstring for that function. Note that DocStrings also apply to modules and classes, which we will learn about in the respective chapters.

The convention followed for a docstring is a multi-line string where the first line starts with a capital letter and ends with a dot. Then the second line is blank followed by any detailed explanation starting from the third line. You are strongly advised to follow this convention for all your docstrings for all your non-trivial functions.

We can access the docstring of the print_max function using the __doc__ (notice the double underscores) attribute (name belonging to) of the function. Just remember that Python treats everything as an object and this includes functions. We'll learn more about objects in the chapter on classes.

If you have used help() in Python, then you have already seen the usage of docstrings! What it does is just fetch the __doc__ attribute of that function and displays it in a neat manner for you. You can try it out on the function above - just include help(print_max) in your program. Remember to press the q key to exit help.

Automated tools can retrieve the documentation from your program in this manner.



# 7 Robot Car Chassis

Each mobile robot has its own chassis design. It is designed for the specific task the robot will execute. This mobile robot is a small version of a mobile robot that can be used in a known and unknown environment. It mostly consists of the following items:

| Component | **Description** |
|---|---|
| **The chassis base:** | This is a pre-cut Acrylic or Perspex base with mounting holes to mount various components. |
| **Motor Mounts:** | Simple pre-cut Acrylic or Perspex small components are provided to mount the motors to the chassis |
| **DC Motors:** | Two DC motors are included in the kit. They are powered by 3-6 volts and they way they are mounted on the chassis will provide differential steering to the robot. |
| **Wheels:** | Two plastics wheels, with tire treads are included in the kit. |
| **Rear Castor:** | The differential steering system requires a free running wheel. A small caster wheel is included in the kit to provide the free running wheel. |
| **Encoder Wheels:** | These two plastic disks are meant to mount to the motor shafts, on the opposite sides from the wheels. They have a series of slots in them as they are designed to be used with an optical source-sensor to provide feedback on motor speed and wheel position. |
| **Mounting Hardware:** | The nuts, bolts and spacers that is needed to put the chassis together. 3D printed mounting brackets for the electronic components, such as the LCD screen, main controller, motor controller, ultrasonic and line sensors, are provided to position and hold the components in place. |
| **Misc. Parts:** | Common additions are power switches, wires and pan-and-tilt mechanisms for mounting sensors or cameras. |

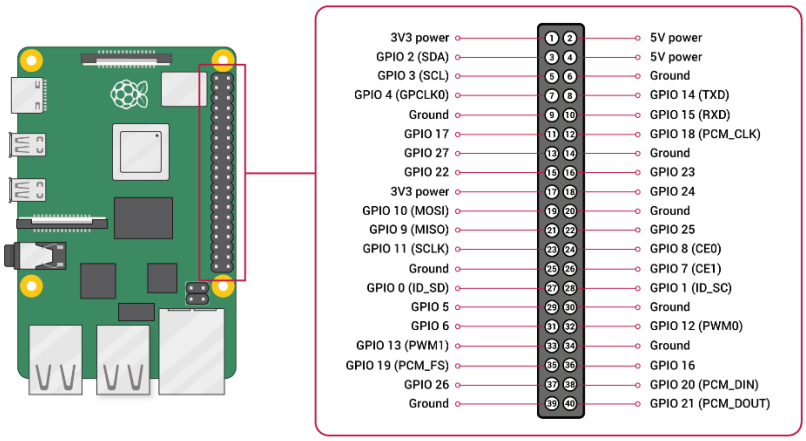
# 8 Hardware setup on the RPi

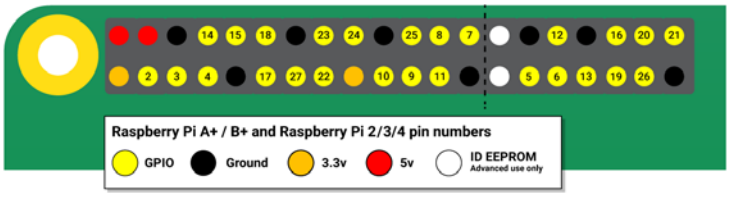
## 8.1 GPIO on the RPi Zero W

GPIO is an abbreviation for General Purpose Input/Output.

This is a powerful feature of the Raspberry Pi. A 40-pin GPIO header is found on all current Raspberry Pi boards (unpopulated on Raspberry Pi Zero, Raspberry Pi Zero W and Raspberry Pi Zero 2 W). Any of the GPIO pins can be designated (in software) as an input or output pin and used for a wide range of purposes.

GPIO pins allow adding extensions to your Raspberry Pi. These extensions could be inputs devices such as sensors or output devices such as controllers, relays, LED's, etc. To use these extensions, the easiest way is to create Python scripts. GPIO pins allow adding extensions to your Raspberry Pi. To use these extensions, the easiest way is to create Python scripts.

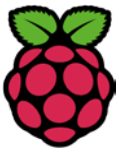



The numbering of the GPIO pins is not in numerical order; GPIO pins 0 and 1 are present on the board (physical pins 27 and 28) but are reserved for advanced use.

**Voltages**

Two 5V pins (2 and 4) and two 3.3V pins (1 and 3) are present on the board, as well as several ground pins (0V), which are unconfigurable. The remaining pins are all general purpose 3.3V pins, meaning outputs are set to 3.3V and inputs are 3.3V-tolerant. Check the voltage output of your sensors before connecting them to the RPi I/O pins. If the sensor output is 5V then a voltage divider is needed to bring it down to 3.3V, which is acceptable for the RPi I/O.



**Outputs**

A GPIO pin designated as an output pin can be set to high (3.3V) or low (0V). Check the volage requirement of your devices and make sure they can operate with the 3.3V output on the GPIO pins.

**Inputs**

A GPIO pin designated as an input pin can be read as high (3.3V) or low (0V). This is made easier with the use of internal pull-up or pull-down resistors. Pins GPIO2 and GPIO3 have fixed pull-up resistors, but for other pins this can be configured in software.

**More**

The GPIO pins can be used with a variety of alternative functions, some are available on all pins, others on specific pins.

| | |
|---|---|
| **PWM**<br>**(Pulse-width modulation)** | Software PWM available on all pins<br>Hardware PWM available on GPIO12, GPIO13, GPIO18, GPIO19 |
| **SPI** | SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7)<br>SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16) |
| **I2C** | Data: (GPIO2); Clock (GPIO3)<br>EEPROM Data: (GPIO0); EEPROM Clock (GPIO1) |
| **Serial** | TX (GPIO14); RX (GPIO15) |

**Raspberry Pi Configuration**

The Raspberry Pi needs to be configured to allow access to the GPIO pins. The following steps will configure the Raspberry Pi to allow access to the GPIO Pins.

**Start by updating your system**

```
sudo apt update && sudo apt upgrade
```

- **Install the rpi.gpio package**

```
- sudo apt install rpi.gpio
```

- **Enable I2C and SPI in raspi-config**

```
- sudo raspi-config
```

Go into “Interface Options”.
And enable I2C and SPI in each submenu.

I2C and SPI pins are specific GPIO pins.

## 8.2 LCD Display Using I2C Module

In the module left side, we have 4 pins, and two are for power (VCC and GND), and the other two are the interface I2C (SDA and SCL). The plate pot is for display contrast adjustment, and the jumper on the opposite side allows the back light is controlled by the program or remain off for power saving.

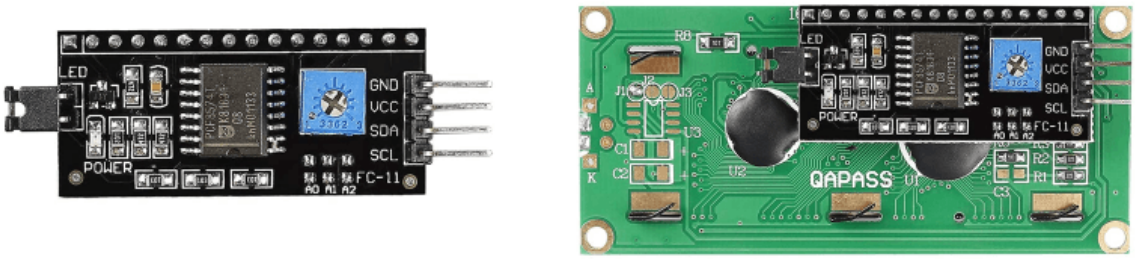



*Figure 23 I2C interface and 2x16 LCD screen*

## Introduction to LCD display

Hitachi's HD44780 based 16×2-character LCD are very cheap and widely available. Using the LCD backpack module, desired data can be displayed on the LCD through the I2C bus. this is a general purpose bidirectional 8 bit I/O port expander that uses the I2C protocol.

The PCF8574 is a silicon CMOS circuit provides general purpose remote I/O expansion (an 8-bit quasi-bidirectional) for most microcontroller families. This I2C module are centered around PCF8574T (SO16 package of PCF8574 in DIP16 package) with a default slave address of 0x27.

If your module holds a PCF8574AT chip, then the default slave address will change to 0x3F. In short, your backpack is based on PCF8574T and the address connections (A0-A1- A2) are not bridged with solder it will have the slave address 0x27.

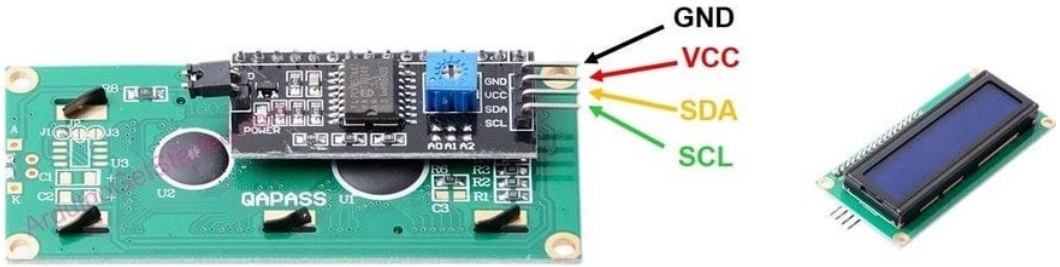



*Figure 24 I2C pinouts*

LCD stands for Liquid crystal display. 16×2 LCD is named so because; it has 16 Columns and 2 Rows. There are a lot of combinations available like 8×1, 8×2, 10×2, 16×1, etc. but the most used one is the 16×2 LCD. So, it will have 16×2 = 32 characters in total and each character will be made of 5×8 Pixel Dots.

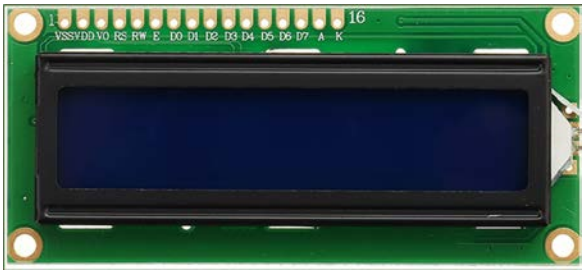


*Figure 25 16x2 LCD screen*

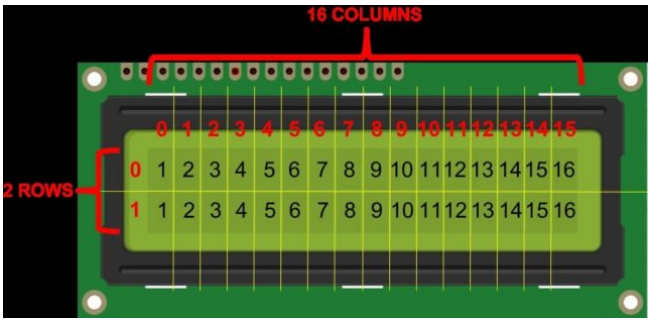



*Figure 26 Rows and Columns on the LCD screen*

16×2 LCD is named so because it has 16 Columns and 2 Rows. So, it will have (16×2=32) 32 characters in total and each character will be made of 5×8 Pixel Dots. A Single character with all its Pixels is shown in the below picture.

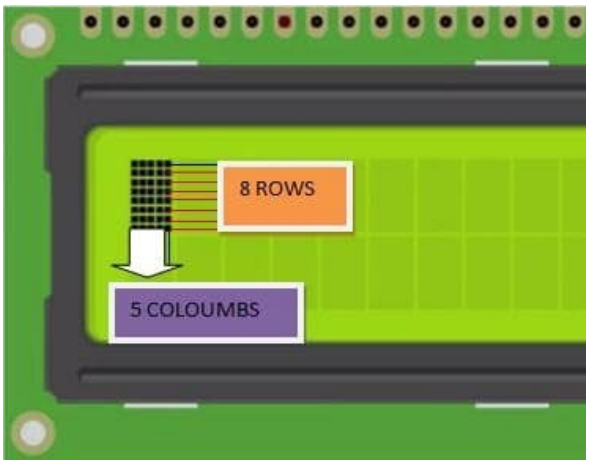



*Figure 27 Single character with all pixels*

Each character has (5×8=40) 40 Pixels and for 32 Characters we will have (32×40) 1280 Pixels. Further, the LCD should also be instructed about the Position of the Pixels.

Therefore, it will be a complicated task to handle everything with the help of a microcontroller. The LCD uses an interface IC like HD44780. This IC is mounted on the backside of the LCD Module.

The function of this IC is to get the Commands and Data from the MCU and process them to display meaningful information on LCD Screen. The LCD operating Voltage is 4.7V to 5.3V & the Current consumption is 1mA without a backlight.

It can work on both 8-bit and 4-bit mode. It can also display any custom-generated characters.

These LCDs are available in Green and Blue Backlight.

## Interfacing 16X2 LCD Display with Raspberry Pi

Components required:

- 16×2 LCD Display
- I2C Module (PCF8574)
- *Raspberry Pi Zero W with Raspbian OS Installed*
- Min 8GB microSD card
- Pi-compatible power sources
- 5V DC Adapter
- Breadboard
- Jumper Wires
- Circuit Diagram

I2C Module is designed suitably to connect all 17 pins of 16X2 LCD Module from Backside to make interfacing easy.

The I2C Module has only 4pins which will be connected directly to the GPIO side of the Raspberry PI, and these pins are:

- GND >> to >> GND
- VCC >> to >> 5V
- SDA >> to >>GPIO
- SCL >> to >>GPIO

The I2C module is equipped with an onboard potentiometer to adjust the brightness as per requirement.

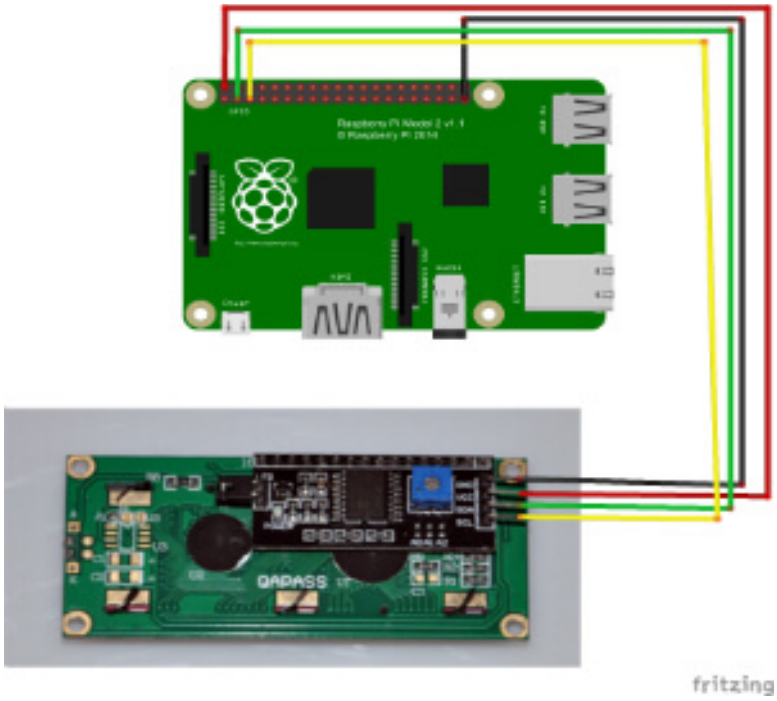



*Figure 28 I2C wiring to the RPi*

The following steps show how to Interface the 16X2 LCD Display with Raspberry Pi using I2C Module, Python and GPIO's.

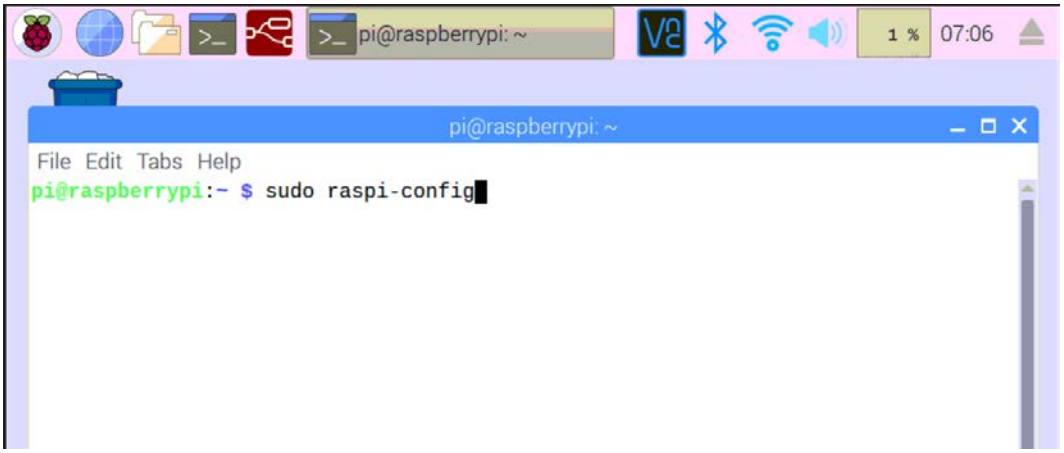
## Step-1: Enable i2c using raspi-config utility

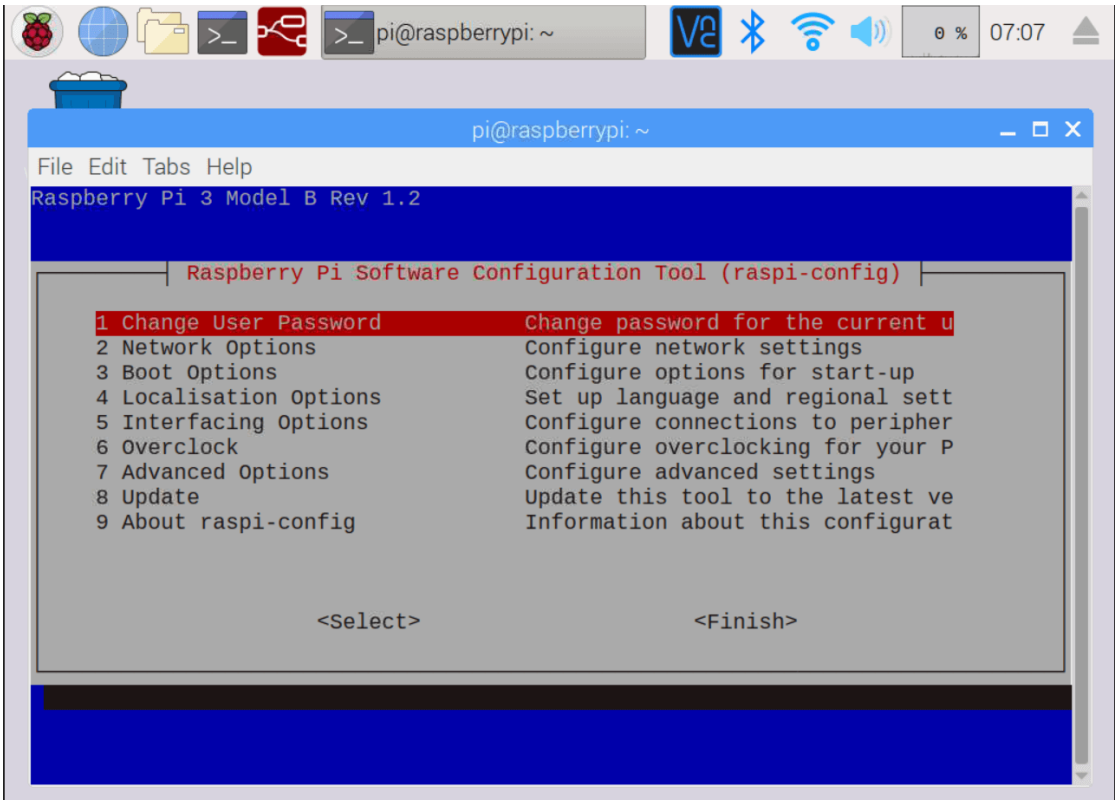
First, Update and upgrade the Raspberry PI with below command

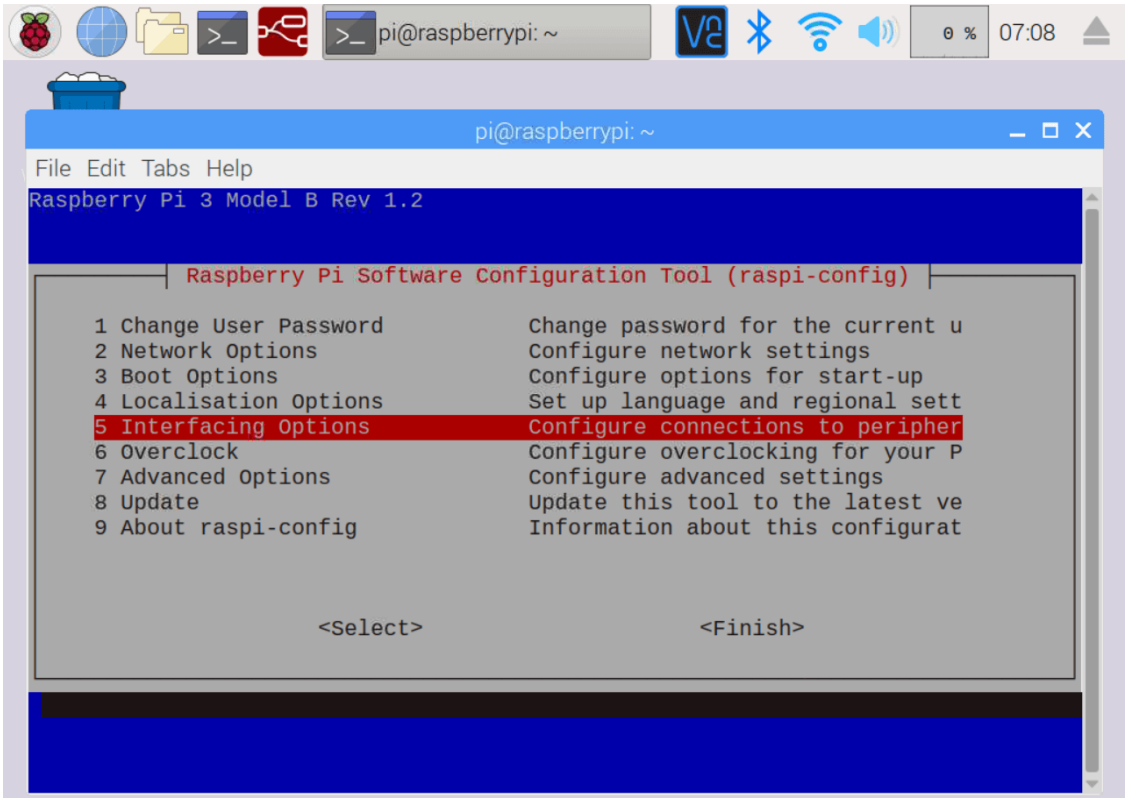
```
sudo apt update -y && sudo apt upgrade -y
```

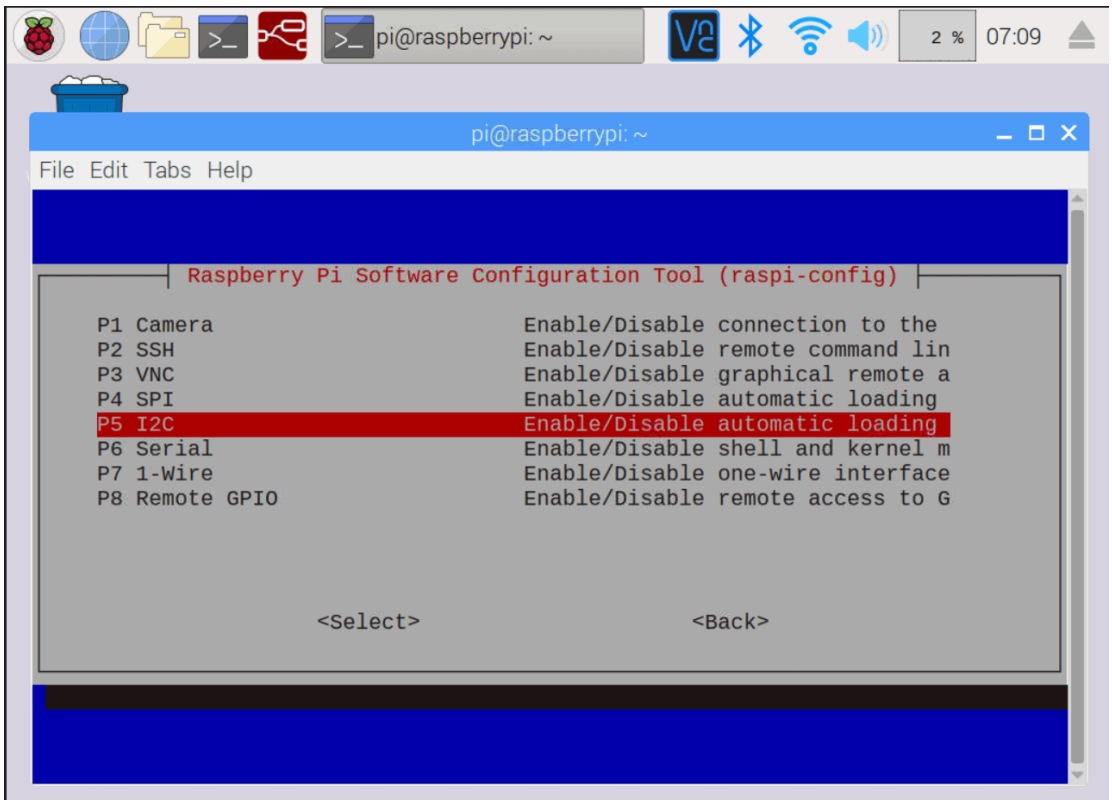
Then, make sure that the I2C protocol is enabled on the Raspberry Pi. Use *sudo raspi-config* command to enable raspi-config utility to enable I2C protocol on Raspberry PI.

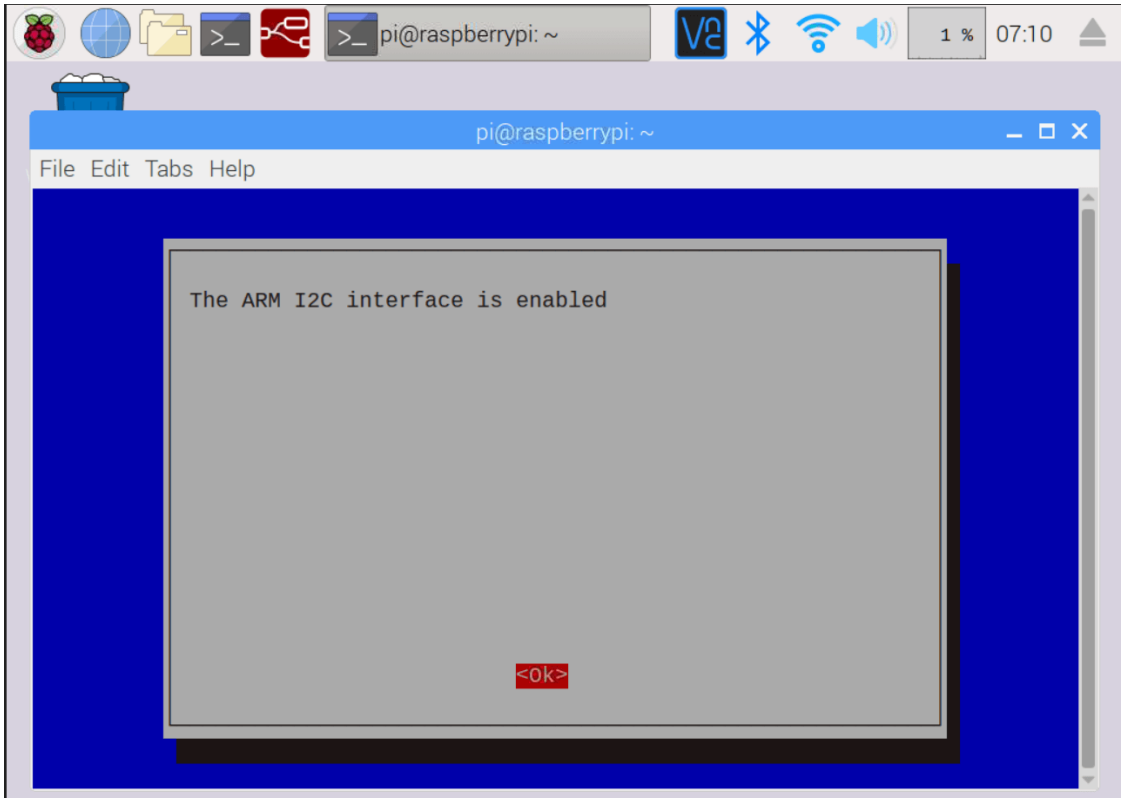
```
sudo raspi-config
```




pi@raspberrypi: ~
0 %
07:08
pi@raspberrypi: ~
File Edit Tabs Help
Raspberry Pi 3 Model B Rev 1.2
Raspberry Pi Software Configuration Tool (raspi-config)
1 Change User Password
Change password for the current u
2 Network Options
Configure network settings
3 Boot Options
Configure options for start-up
4 Localisation Options
Set up language and regional sett
5 Interfacing Options
Configure connections to peripher
6 Overclock
Configure overclocking for your P
7 Advanced Options
Configure advanced settings
8 Update
Update this tool to the latest ve
9 About raspi-config
Information about this configurat
<Select>
<Finish>


pi@raspberrypi: ~
2 %
07:09
pi@raspberrypi: ~
File Edit Tabs Help
Raspberry Pi Software Configuration Tool (raspi-config)
P1 Camera
Enable/Disable connection to the
P2 SSH
Enable/Disable remote command lin
P3 VNC
Enable/Disable graphical remote a
P4 SPI
Enable/Disable automatic loading
P5 I2C
Enable/Disable automatic loading
P6 Serial
Enable/Disable shell and kernel m
P7 1-Wire
Enable/Disable one-wire interface
P8 Remote GPIO
Enable/Disable remote access to G
<Select>
<Back>



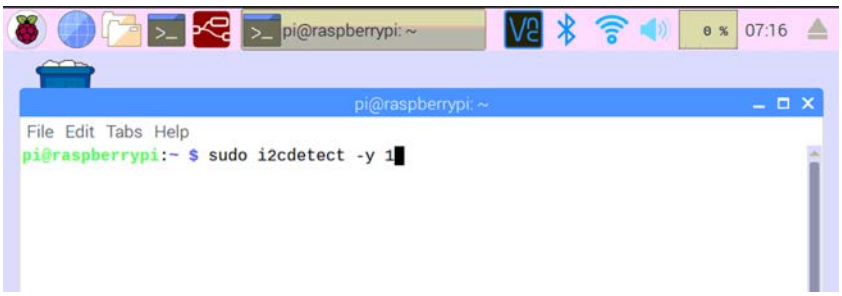

Click on the Finish Button and then reboot the RPi.

```
sudo reboot now
```

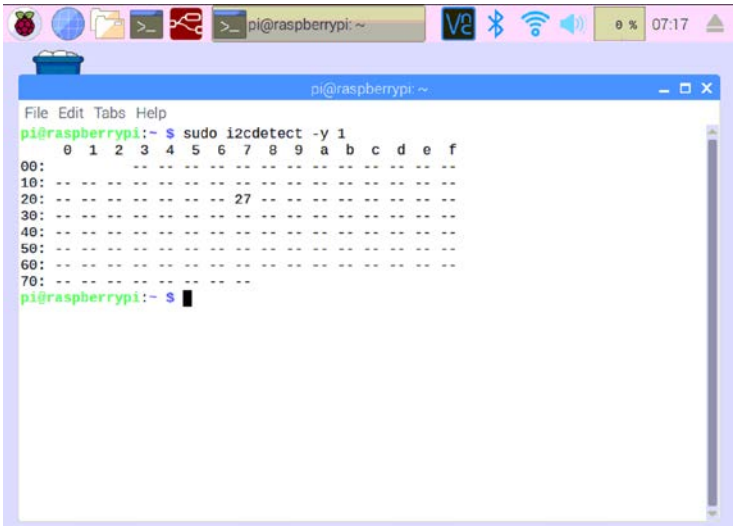
## Step-2: Detect I2C Bus and Device address

Once the system has finished the reboot process and is back, check if the I2C bus is active, I2C protocol supports multiple devices connected to the same bus, and it identifies each device by their hardware address. The i2cdetect command can be used to see what is connected and what the addresses are.

```
sudo apt install i2c-tools
sudo i2cdetect -y 1
```




The device address of the I2C Module connected with Raspberry PI is found at 0x27. The same Address will be used in the Python script to enable I2C communication between Raspberry Pi and I2C module.




## Step-3: Installing rpi-lcd library

The easiest way to program this 16×2 I²C LCD display in Python is by using a dedicated library. This library supports LCD text displays (20x4, 16x2 and other) via I²C converter.

It requires the *python-smbus* package installed on your Raspberry Pi and enabled ARM I2C interface.

Get the package from *pypi*

```
pip install rpi-lcd
```

Once the installation is finished, you can start developing code to display text, numbers and symbols on an LCD screen.

All you must do now, is to import the library in your code and use the various commands to manipulate the screen to display characters on it.

## Program for interfacing 16×2 LCD with Raspberry PI

The code below is to display the words “Hello” and “Raspberry Pi” on the LCD screen. Create a new file in Thonny and enter the code as is:

```
pi@raspberrypi:~/lcd $ sudo nano demo_clock.py
```

If the command below is executed, and the connection between the RPI, the I2C Module and LCD , on the LCD Display the Data that was edited will print on the display.

```
from rpi_lcd import LCD
from time import sleep
lcd = LCD()

lcd.text('Hello World!', 1)
lcd.text('Raspberry Pi', 2)

```

```
sleep(5)
lcd.clear()
lcd.text('This is a message that has more ‘than 16 chars and does not fit in one line', 1)
sleep(2)
lcd.clear()
```

## 8.3 L298N (DC Motor Driver Module)

A DC motor is powered by connecting the motor terminals to a positive and negative voltage. Figure 29 shows when you hook a dc motor up to {+} wire on one lead and {-} wire on the other lead, the motor turns a certain direction. When you switch the wires to other leads, the motor turns the other direction. When these motors are installed on a robot, the terminals cannot be physically switched every time the direction of the motor needs to change.

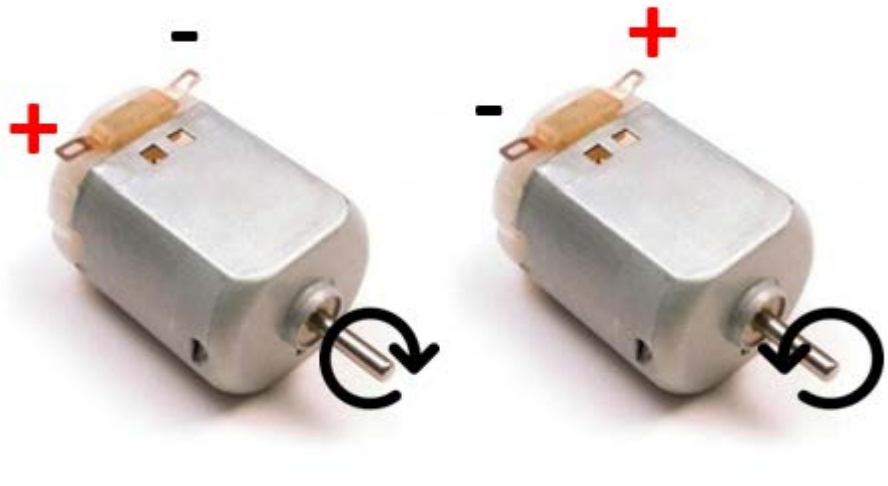



*Figure 29 Switching DC motor terminals*

This change needs to be done electronically as well as through software manipulation. The electronic circuit that can make this switch is called an H-Bridge. Using this circuit current can be supplied in two directions. The LN298N, is an H-Bridge with two possible outputs. In other words, we can connect and control two DC motors with it.

Let’s investigate a simple H-Bridge circuit.

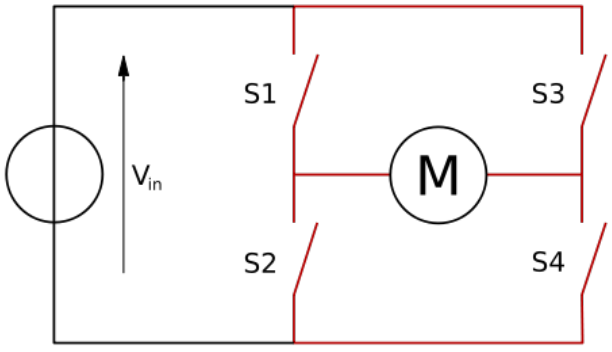



*Figure 30 An H-Bridge built with switches*

S1, S2, S3 and S4 are simple switches. If S1 and S2 are closed, a short circuit between the + and – will be caused. The same will happen if S3 and S4 are closed. This condition, in technical terms, is called a shoot through. Therefore, a shoot through is not an option.

If switches S1 and S4 are closed, current flows through the motor from left to right. If S3 and S2 are closed, current flows from right to left. In these two conditions, the direction of rotation is different.

The following table summarizes operation, with S1-S4 corresponding to the diagram above. In the table below, "1" is used to represent "on" state of the switch, "0" to represent the "off" state.

| S1 | S2 | S3 | S4 | Result |
|---|---|---|---|---|
| **1** | 0 | 0 | **1** | Motor moves right |
| 0 | **1** | **1** | 0 | Motor moves left |
| 0 | 0 | 0 | 0 | Motor coasts |
| **1** | 0 | 0 | 0 | |
| 0 | **1** | 0 | 0 | |
| 0 | 0 | **1** | 0 | |
| 0 | 0 | 0 | **1** | |
| 0 | **1** | 0 | **1** | Motor brakes |
| **1** | 0 | **1** | 0 | |
| x | x | **1** | **1** | Short circuit |
| **1** | **1** | x | x | |

*Figure 31 Effects of closing the switches.*

This is exactly what is needed in most robotics projects. But having physical switches would be very inconvenient. More motors will be needed to close and open switches. And to control those motors even more switches will be needed.

An electronically controlled switch, called a transistor, is therefore needed and used in the circuit Figure 32.

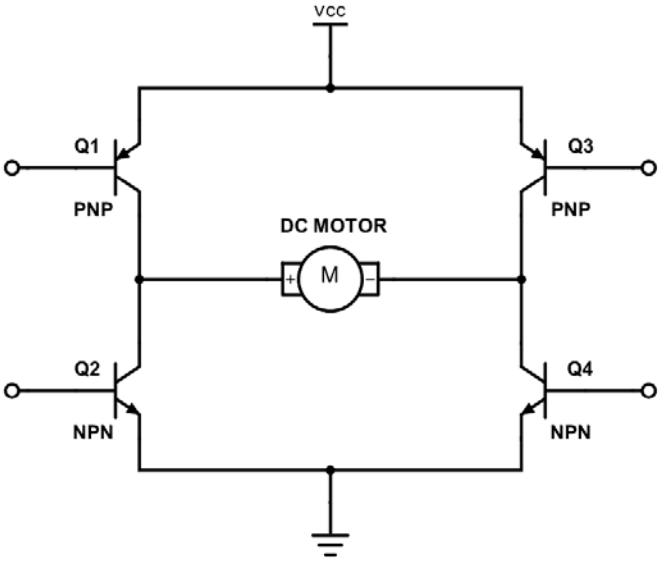



*Figure 32 Transistor H-Bridge*

## Introduction to the LN298N H-Bridge

The L298N is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A. The L298N motor driver uses PWM to change the speed of the motor and the dual H-bridge to change the direction of the motor.

NOTE: If more than 12V is connected to the VCC supply in pin then the jumper near the supply must be disconnected to disconnect the 5V on board regulator since it cannot use anything more than 12V.

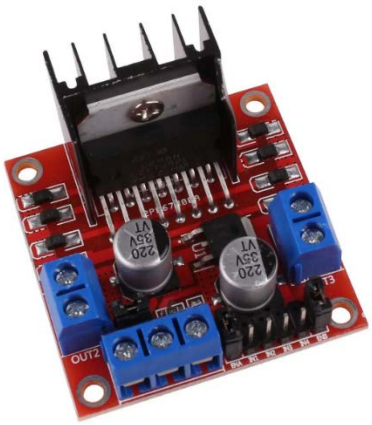


*Figure 33 L298N Motor Driver*

The L298N module has 13 connectors

| Pin | Description |
|---|---|
| **VCC:** | Power supply pin. Connect this pin to +7.4V DC supply |
| **GND:** | Ground pin. Connect this pin to ground connection. |
| **5V:** | 5V output when the board is powered through the 12V input supply. |
| **ENA:** | Takes PWM input to control the speed of motor 1 (jumped to 5V by default) |
| **IN1 & IN2:** | They take a logical 5V and controls the direction of motor 1 |
| **IN3 & IN4:** | They take a logical 5V and controls the direction of motor 2 |
| **ENB:** | Takes PWM input to control the speed of motor 2 (jumped to 5V by default) |
| **OUT1 & OUT2** | Output 10V to motor 1 |
| **OUT3 & OUT4** | Output 10V to motor 2 |

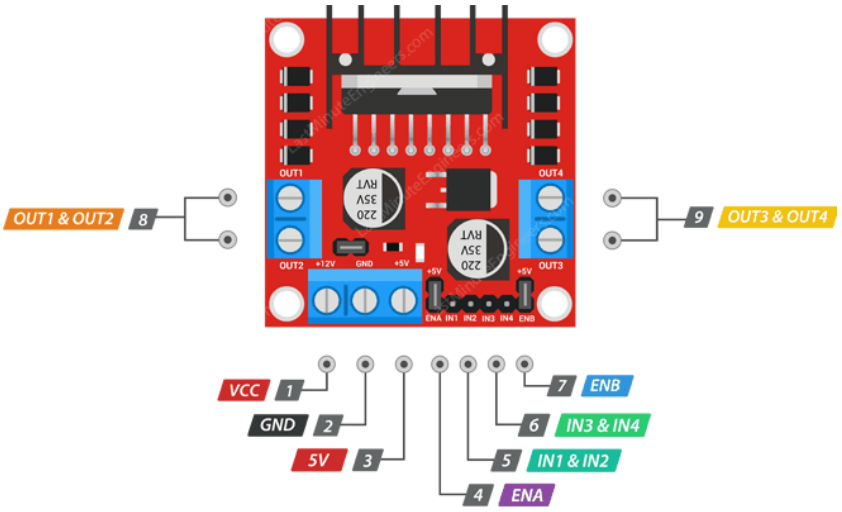



*Figure 34 Pin layout or the H-Bridge*

## Hardware interfacing with the RPI

- Connect IN1 on the L298 to the Raspberry Pi pin number 26.
- Connect IN2 on the L298 to Raspberry Pi  pin number 20.
- Connect the ENA and VCC the 7-volt battery.
- Make sure the grounds of the battery, Raspberry Pi, and L298 are common.

## Demo Code

```
import sys
import time
import RPi.GPIO as GPIO

mode=GPIO.getmode()

GPIO.cleanup()

Forward=26
Backward=20
sleeptime=1

GPIO.setmode(GPIO.BOARD)
```

```
GPIO.setup(Forward, GPIO.OUT)
GPIO.setup(Backward, GPIO.OUT)

def forward(x):
GPIO.output(Forward, GPIO.HIGH)
print("Moving Forward")
time.sleep(x)
GPIO.output(Forward, GPIO.LOW)

def reverse(x):
GPIO.output(Backward, GPIO.HIGH)
print("Moving Backward")
time.sleep(x)
GPIO.output(Backward, GPIO.LOW)

while (1):
forward(5)
reverse(5)
GPIO.cleanup()
```

## 8.4 Encoder Module

## Introduction

The IR Optocoupler module consists of an infrared light-emitting diode and an NPN phototriode with a slot width of 5.9mm. If the non-transparent object passes through the slot type, it can trigger (used with the car code disk) to output the TTL low level.


*Figure 35 Optocoupler Sensor and IR Optocoupler Interrupter wheel disk*

## IR Optocoupler Sensor Module Pin Description

The IR Optocoupler module has 3 pins,

| | |
|---|---|
| **OUT:** | Output pin. Connect to GPIO pin as an input on the raspberry pi |
| **VCC:** | Power supply pin. Connect this pin to +3.3V DC supply |
| **GND:** | Ground pin. Connect this pin to ground connection. |

Note that the output voltage of the IR optocoupler module is 5V when it is powered with 5V and 3.3V when it is powered with 3.3V but the maximum voltage for the raspberry pi is 3.3V on the GPIO pins, therefore it should be powered with 3.3V

## Calculations

Now suppose the sensor interrupter disk has 20 slots, then for every 20 pulses the motor has made one full rotation.

Then assuming the wheel makes 20 pulses per second,

$$RPM = \frac{pulses \times 60}{20}$$

then,

$$wheel\ diameter = 66.1\ mm$$

$$wheel\ circumference = 66.1 \times \pi = 207.6\ mm$$

theretofore,

$$Distance\ travelled = \frac{pulses}{20} \times 207.6$$

The distance is in millimeters.

**For example,**

Suppose $ulses = 73$ , Then $Distance\ travelled = \frac{73}{20} \times 207.6 = 757.74\ mm$

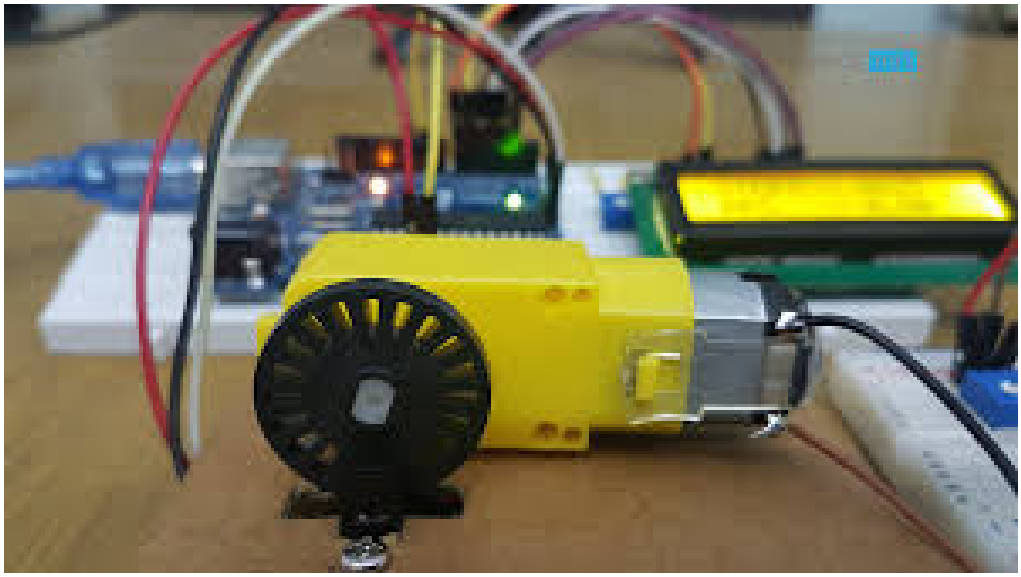


*Figure 36 Optical encoder installation*

## Demo Code

```
#Libraries
import RPi.GPIO as GPIO
import time

#GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BOARD)
#set GPIO Pins
GPIO_TRIGGER = 16
GPIO_ECHO = 18

#set GPIO direction (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
    # set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)
```

```
    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)
    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()
    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime
    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2
    return distance

while True:
    dist = distance()
    print ("Measured Distance = %.1f cm" % dist)
    time.sleep(1)
    if dist < 5:
        print("Stop")
        time.sleep(2)
```

## 8.5 HC-SR04 (Ultrasonic Distance Sensor Module)

## Introduction

The HC-SR04 Ultrasonic Distance Sensor is a sensor used for detecting the distance to an object using sonar. It's ideal for any robotics projects which are required to avoid objects, by detecting how close they are.


*Figure 37 HC-SR04 Sensor*

The HC-SR04 uses non-contact ultrasound sonar to measure the distance to an object, and consists of an ultrasonic transmitter, a receiver, and a control circuit. The transmitter emits a high frequency ultrasonic sound, which bounce off any nearby solid objects, and the receiver listens for any return echo. The echo is then processed by the control circuit to calculate the time difference between the signal being transmitted and received. This time can subsequently be used, along with some math, to calculate the distance between the sensor and the reflecting object.

## HC-SR04 Module Pin Description

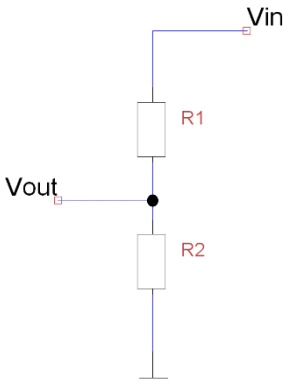
The HC-SR04 module has 4 pins,

| | |
|---|---|
| **VCC:** | Power supply pin. Connect this pin to +5V DC supply. |
| **TRIGGER:** | Trigger pin is an Input pin. This pin has to be kept high for 10us to initialize measurement by sending ultrasonic wave |
| **ECHO:** | Echo pin is an Output pin. This pin goes high for a period of time which will be equal to the time taken for the ultrasonic wave to return back to the sensor. Connect this pin to a voltage divider to convert the output to 3.3V. |
| **GND:** | Ground pin. Connect this pin to ground connection. |

## Calculations

Note that the output voltage of the HC-SR04 is 5V but the maximum voltage for the raspberry pi is 3.3V on the GPIO pins. The output voltage of the HC-SR04 should be reduced to 3.3V via a logic level converter or a voltage divider.

Now suppose there is a voltage divider,




then,

$$Vout = Vin \times \frac{R2}{R1 + R2}$$

$$\frac{Vout}{Vin} = \frac{R2}{R1 + R2}$$

$$\frac{3.3}{5} = \frac{R2}{1000 + R2}$$

$$0.66 = \frac{R2}{1000 + R2}$$

$$0.66(1000 + R2) = R2$$

$$660 + 0.66R2 = R2$$

$$660 = 0.34R2$$

$$1941 = R2$$

Therefore, assuming that $R1 = 1000\ \Omega$ :

So, let it be 1kΩ for R1 and a 2kΩ resistor as R2.

The HC-SR04 sensor requires a short 10uS pulse to trigger the module, which will cause the sensor to shoot an ultrasound wave at 40 kHz to obtain an echo response. So, to create our trigger pulse, we set out trigger pin high for 10uS then set it low again.

The sensor sets ECHO pin to HIGH for time it takes for the pulse to go and come back therefore the amount of time that the ECHO pin stays high is measured.

$$Speed = \frac{Distance}{Time}$$

To get the distance in centimeters the speed of sound in air, at sea level and at room temperature must be used. The speed is 343m/s therefore:

Where speed is the speed of sound (343m/s) and time is the amount of time it takes the pulse to go and come back.

$$34300 = \frac{Distance}{Time/2}$$

$$17150 = \frac{Distance}{Time}$$

$$17150 \times Time = Distance$$

Time is divided by 2 because we only want the time it takes to comeback after bouncing of an object which is half the time it takes the pulse to go and come back.

**NOTE:** The distance obtained from these calculations is in centimeters.

**For example,**

Suppose $Time = 0.005 \times 10^{-3}$ seconds

Then $Distance = 17150 \times 0.005 \times 10^{-3} = 0.08575\, m = 8.58\, cm$

## Demo Code

```
Libraries
import RPi.GPIO as GPIO
import time
#GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

#set GPIO Pins
GPIO_TRIGGER = 18
GPIO_ECHO = 24
#set GPIO direction (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
    # set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)
```

```
    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()
    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime
    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2

    return distance
if __name__ == '__main__':
    try:
        while True:
            dist = distance()
            print ("Measured Distance = %.1f cm" % dist)
            time.sleep(1)
        # Reset by pressing CTRL + C
    except KeyboardInterrupt:
```

```
    print("Measurement stopped by User")
    GPIO.cleanup()
```

## 8.6 FC-123 (Line Sensor Module)

### Introduction

The module is a white/black lines sensor that are usually seen in a robot path. The sensor also has a good sensitivity setting and a reduced susceptibility to interference. When a 5V input is detected, the module will output LOW when a white object is detected while it is not detected. This happens if the sensor is too close to the ground. A clearance height of 2.5 cm is required for optimum performance. Ambient light also influences the response of the sensor, a very brightly lit room or a very dim room will affect the performance of the sensor


*Figure 38 Line Sensor*

### FC-123 Module Pin Description

The FC-123 module has 3 pins,

| Pin | Description |
|---|---|
| **OUT:** | Output pin. Connect to GPIO pin as an input on the raspberry pi |
| **VCC:** | Power supply pin. Connect this pin to +3.3V DC supply |
| **GND:** | Ground pin. Connect this pin to ground connection. |

### Calculations

There are no calculations involved it is plug and play. The raspberry pi must listen for signal coming from the OUT pin.

### Demo Code

```
#!/usr/bin/python

from RPi import GPIO
from time import sleep
```

```
GPIO.setmode(GPIO.BCM)

left_sensor = 11
right_sensor = 9

GPIO.setup(left_sensor, GPIO.IN)
GPIO.setup(right_sensor, GPIO.IN)

try:
        while True:
                if not GPIO.input(left_sensor):
                        print("Robot is straying off to the right, move left captain!")
                elif not GPIO.input(right_sensor):
                        print("Robot is straying off to the left, move right captain!")
                else:
                        print("Following the line!")
                sleep(0.2)
except:
        GPIO.cleanup()
```

## 8.7MPU6050 Sensor Module

## Introduction

MPU6050 sensor module is complete 6-axis Motion Tracking Device. It combines 3-axis Gyroscope, 3-axis Accelerometer and Digital Motion Processor all in small package. Also, it has additional feature of on-chip Temperature sensor. It has I2C bus interface to communicate with the microcontrollers.


*Figure 39MPU6050 Sensor*

It has Auxiliary I2C bus to communicate with other sensor devices like 3-axis Magnetometer, Pressure sensor etc. If 3-axis Magnetometer is connected to auxiliary I2C bus, then MPU6050 can provide complete 9-axis Motion Fusion output.

Let's see MPU6050 inside sensors.

## 3-Axis Gyroscope

The MPU6050 consist of 3-axis Gyroscope with Micro Electromechanical System (MEMS) technology. It is used to detect rotational velocity along the X, Y, Z axes as shown in below figure.

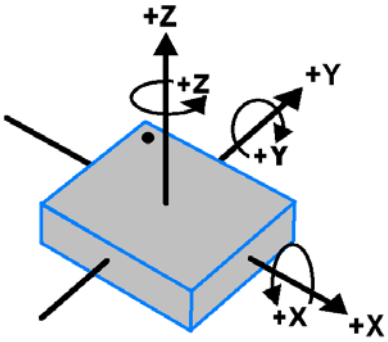



*Figure 40 MPU-6050 Orientation And Polarity Of Rotation*

- When the gyros are rotated about any of the sense axes, the Coriolis Effect causes a vibration that is detected by a MEM inside MPU6050.
- The resulting signal is amplified, demodulated, and filtered to produce a voltage that is proportional to the angular rate.
- This voltage is digitized using 16-bit ADC to sample each axis.
- The full-scale range of output are +/- 250, +/- 500, +/- 1000, +/- 2000.
- It measures the angular velocity along each axis in degree per second unit.
-

## 3-Axis Accelerometer

The MPU6050 consist of 3-axis Accelerometer with Micro Electromechanical (MEMs) technology. It used to detect angle of tilt or inclination along the X, Y and Z axes as shown in below figure.

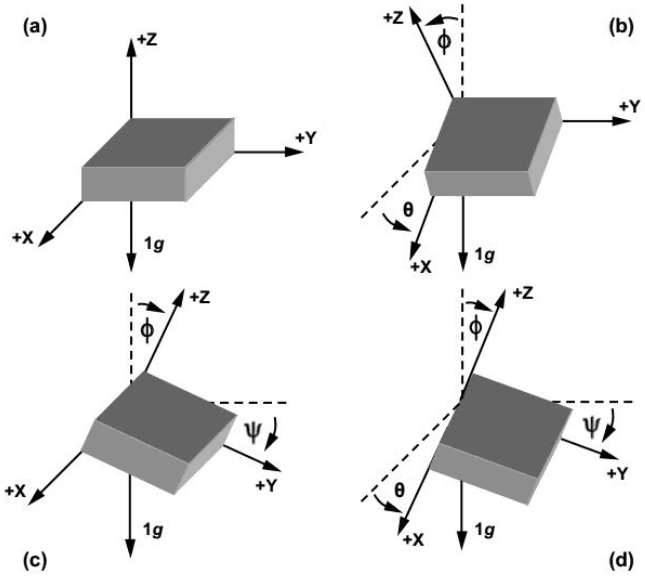



*Figure 41 MPU-6050 Angle of Inclination*

- Acceleration along the axes deflects the movable mass.
- This displacement of moving plate (mass) unbalances the differential capacitor which results in sensor output. Output amplitude is proportional to acceleration.
- 16-bit ADC is used to get digitized output.
- The full-scale range of acceleration are +/- 2g, +/- 4g, +/- 8g, +/- 16g.
- It measured in g (gravity force) unit.
- When device placed on flat surface it will measure 0g on X and Y axis and +1g on Z axis.

## DMP (Digital Motion Processor)

The embedded Digital Motion Processor (DMP) is used to compute motion processing algorithms. It takes data from gyroscope, accelerometer, and additional 3rd party sensor such as magnetometer and processes the data. It provides motion data like roll, pitch, yaw angles, landscape and portrait sense etc. It minimizes the processes of host in computing motion data. The resulting data can be read from DMP registers.

## On-chip Temperature Sensor

On-chip temperature sensor output is digitized using ADC. The reading from temperature sensor can be read from sensor data register.

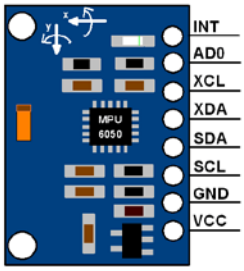



*Figure 42 MPU-6050 Module*

## MPU-6050 Module Pin Description

The MPU-6050 module has 8 pins,

| | |
|---|---|
| INT: | Interrupt digital output pin. |
| AD0: I2C | Slave Address LSB pin. This is 0th bit in 7-bit slave address of device. If connected to VCC then it is read as logic one and slave address changes. |
| XCL: | Auxiliary Serial Clock pin. This pin is used to connect other I2C interface enabled sensors SCL pin to MPU-6050. |
| XDA: | Auxiliary Serial Data pin. This pin is used to connect other I2C interface enabled sensors SDA pin to MPU-6050. |
| SCL: | Serial Clock pin. Connect this pin to microcontrollers SCL pin. |
| SDA: | Serial Data pin. Connect this pin to microcontrollers SDA pin. |
| GND: | Ground pin. Connect this pin to ground connection. |
| VCC: | Power supply pin. Connect this pin to +5V DC supply. |

MPU-6050 module has Slave address (When AD0 = 0, i.e., it is not connected to VCC) as,

- Slave Write address (SLA+W): 0xD0
- Slave Read address (SLA+R): 0xD1

MPU-6050 has various registers to control and configure its mode of operation. So, kindly go through MPU-6050 datasheet and MPU-6050 Register Map.

## Setting up with the RPI Zero W

- Connect the PIN 3V3 (Pin 1) of the Raspberry-Pi to the VCC of the MPU6050
- Connect the GND pin (Pin 39) to the GND of the MPU6050
- Connect Pin I2C1 SDA (Pin 3) to the MPU6050's SDA
- Connect the SCL Pin (Pin 5) to the SCL of the MPU6050

Make sure that /dev/i2c-1 exists on your Raspberrry-Pi. If not, check our article on the i2c bus of the Raspberry-Pi.

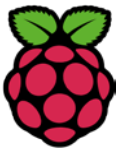
MPU6050's default address is 0x68, you can test it responds correctly by asking for its address.

```
i2cget -y 1 0x68 0x75
0x68 # Yes, we know that but thanks!
```

## Demo Code

```
import smbus
from time import sleep
import math
import RPi.GPIO as GPIO
import sys


PWR_MGMT_1   = 0x6B
```

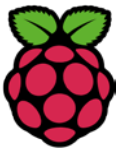


```
SMPLRT_DIV   = 0x19
CONFIG       = 0x1A
GYRO_CONFIG = 0x1B
INT_ENABLE   = 0x38
ACCEL_XOUT_H = 0x3B
ACCEL_YOUT_H = 0x3D
ACCEL_ZOUT_H = 0x3F
GYRO_XOUT_H = 0x43
GYRO_YOUT_H = 0x45
GYRO_ZOUT_H = 0x47
bus = smbus.SMBus(1)

def MPU_Init():
    bus.write_byte_data(Device_Address, SMPLRT_DIV, 7)
    bus.write_byte_data(Device_Address, PWR_MGMT_1, 1)
    bus.write_byte_data(Device_Address, CONFIG, 0)
    bus.write_byte_data(Device_Address, GYRO_CONFIG, 24)
    bus.write_byte_data(Device_Address, INT_ENABLE, 1)

def read_raw_data(addr):
        high = bus.read_byte_data(Device_Address, addr)
        low = bus.read_byte_data(Device_Address, addr+1)
        value = ((high << 8) | low)
        if(value > 32768):
                value = value - 65536
        return value

def dist(a, b):
    return math.sqrt((a*a) + (b*b))
def get_y_rotation(x, y, z):
```

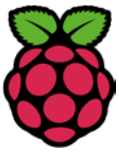


```
    radians = math.atan2(y, z)
    return -(radians * (180.0 / math.pi))

def get_x_rotation(x, y, z):
    radians = math.atan2(x, dist(y, z))
    return -(radians * (180.0 / math.pi))

if __name__ == "__main__":
    Device_Address = 0x68
    MPU_Init()
    print("Reading MPU6050...")
    try:
        while True:
            acc_x = read_raw_data(ACCEL_XOUT_H)
            acc_y = read_raw_data(ACCEL_YOUT_H)
            acc_z = read_raw_data(ACCEL_ZOUT_H)

            acclX_scaled = acc_x * .000061 * 9.80665
            acclY_scaled = acc_y * .000061 * 9.80665
            acclZ_scaled = acc_z * .000061 * 9.80665
            x_angle = get_x_rotation(acclX_scaled, acclY_scaled, acclZ_scaled)
            y_angle = get_y_rotation(acclX_scaled, acclY_scaled, acclZ_scaled)
            print("X rotation: ", x_angle)
            print("Y rotation: ",y_angle)
            sleep(.50)
    except KeyboardInterrupt:
        sys.exit(0)
    except Exception as e:
        print(e)
        sys.exit(0)
```



## Calculations

Note that gyroscope and accelerometer sensor data of MPU6050 module consists of 16-bit raw data in 2's complement form.

Temperature sensor data of MPU6050 module consists of 16-bit data (not in 2's complement form).

Now suppose we have selected,

- Accelerometer full scale range of +/- 2g with Sensitivity Scale Factor of 16,384 LSB(Count)/g.
- Gyroscope full scale range of +/- 250 °/s with Sensitivity Scale Factor of 131 LSB (Count)/°/s.

then,

To get sensor raw data, we need to first perform 2's complement on sensor data of Accelerometer and gyroscope.

After getting sensor raw data we can calculate acceleration and angular velocity by dividing sensor raw data with their sensitivity scale factor as follows,

**Accelerometer values in g (g force)**

Acceleration along the X axis = (Accelerometer X axis raw data/16384) g.

Acceleration along the Y axis = (Accelerometer Y axis raw data/16384) g.

Acceleration along the Z axis = (Accelerometer Z axis raw data/16384) g.

**Gyroscope values in °/s (degree per second)**

Angular velocity along the X axis = (Gyroscope X axis raw data/131) °/s.

Angular velocity along the Y axis = (Gyroscope Y axis raw data/131) °/s.

Angular velocity along the Z axis = (Gyroscope Z axis raw data/131) °/s.

**Temperature value in °/c (degree per Celsius)**

Temperature in degrees C = ((temperature sensor data)/340 + 36.53) °/c.

**For example,**

Suppose after 2' complement we get accelerometer X axes raw value = +15454

Then Ax = +15454/16384 = 0.94 g.

# 9 Sensors

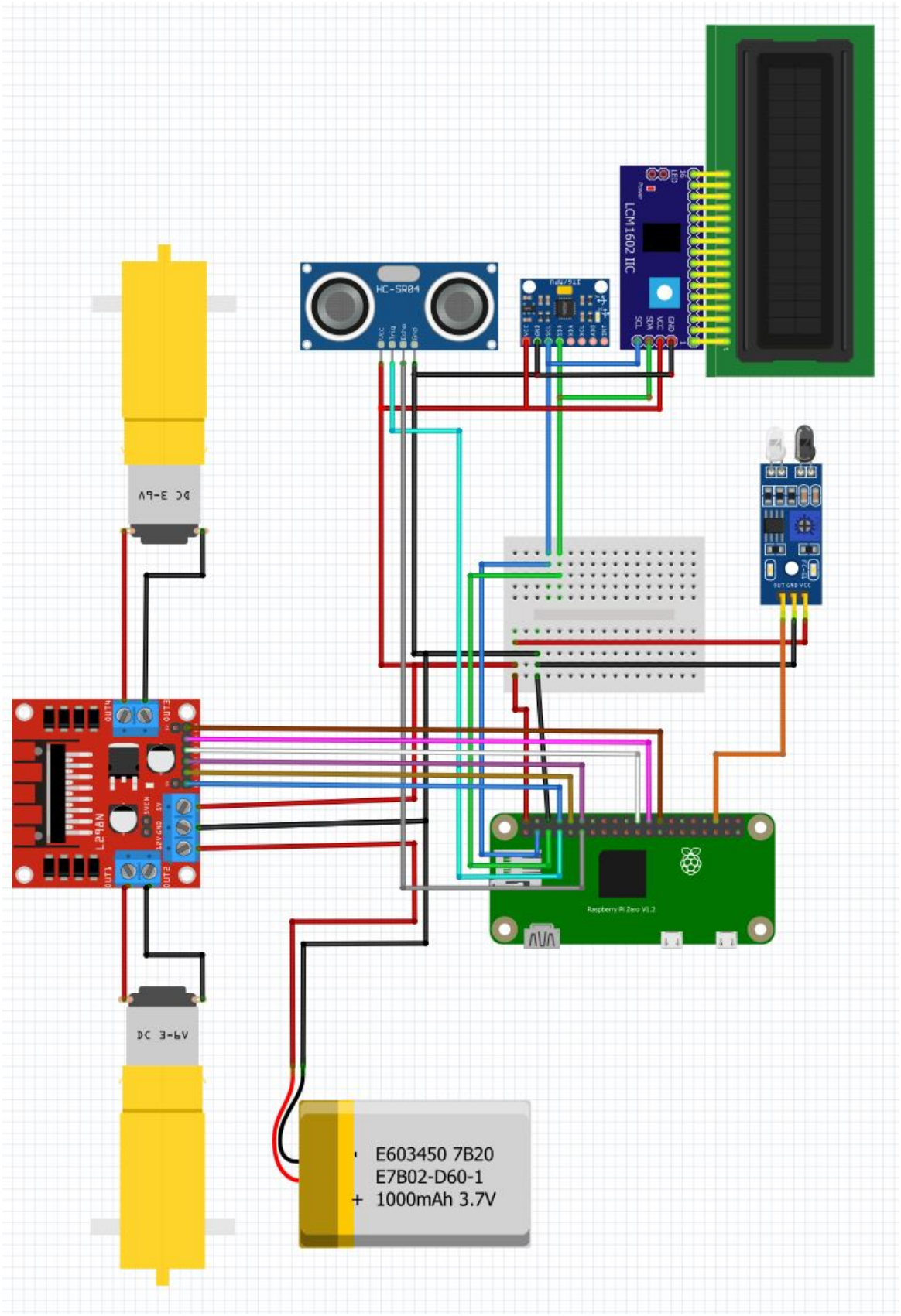
| **Inputs** | | |
|---|---|---|
| Light Interrupter Module |  | **Power Supply: 3.3-5V**<br>Dimension: 26.8mm * 15mm * 18.7mm<br>Mounting holes size: 3mm<br>Gap width: 6mm<br>In the case of working with a MCU:<br>VCC ↔ 3.3V ~ 5V<br>GND ↔ GND<br>DOUT ↔ MCU.IO (digital output) |
| Line Sensor |  | **Power Supply: 5V**<br>Current: <10mA<br>Size: 10 x 35mm |
| Ultrasonic sensor |  | **Power supply: 5V DC**<br>Quiescent current <2mA<br>Effectual angle <15°<br>Ranging distance 2cm – 400 cm<br>Resolution 0.3 cm |
| IMU 6050 |  | **Power Supply :3-5V (internal regulator)**<br>Input Voltage: 2.3 - 3.4V<br>Chipset: Invensense MPU-6050<br>Communication modes: I2C<br>Chip built-in 16-bit AD Converter<br>Data Output - 16 Bit<br>Gyroscope range: ± 250 500 1000 2000 ° / s<br>Acceleration range: ± 2 ± 4 ± 8 ± 16g |
| Outputs | | |
| LCD |  | |
| I2C Interface for LCD |  | **Power Supply and Signal: 5V / 3.3V Ready**<br>Compatible with 16x2, 16x4 and 20x4 Character LCD's<br>Default I2C Address: 0x3F |
| Motors |  | |

# 10 References


1. https://www.cobottrends.com/where-spot-mobile-robots-2020/
2. https://veridian.info/autonomous-mobile-robots/
3. https://en.wikipedia.org/wiki/Mobile_robot
4. https://robots.ieee.org/learn/what-is-a-robot/
5. https://www.mechstudies.com/what-robots-definition-meaning-parts-mechanism-types-robotics/
6. https://www.robotshop.com/community/tutorials/show/basics-what-types-of-mobile-robots-are-there
7. http://robotplatform.com/knowledge/Classification_of_Robots/wheel_control_theory.html
8. https://www.alliedmotion.com/electric-traction-and-steering-for-robotic-vehicles/
9. https://www.smashingrobotics.com/introduction-to-mobile-robot-sensor-systems/
10. https://www.robotshop.com/community/tutorials?mode=list&sort=newest&page=1
11. https://www.studytonight.com/post/difference-between-microprocessor-and-microcontroller#:~:text=These%20are%20the%20following%20key%20difference%20between%20microprocessor,when%20compared%20to%20a%20microprocessor.%20More%20items...%20
12. https://www.guru99.com/difference-between-microprocessor-and-microcontroller.html
13. https://bws428.github.io/raspberry-pi-zero-w-setup/
14. https://desertbot.io/blog/setup-pi-zero-w-headless-wifi
15. https://www.raspberrypi.com/documentation/computers/getting-started.html#setting-up-your-raspberry-pi
16. https://www.woolseyworkshop.com/2018/09/06/interfacing-an-mpu6050-gyroscope-accelerometer-sensor-module-to-a-raspberry-pi/
17. https://microdigisoft.com/mpu6050-accelerometergyroscope-sensor-interfacing-with-raspberry-pi/
18. https://tutorials-raspberrypi.com/raspberry-pi-ultrasonic-sensor-hc-sr04/
19. https://pimylifeup.com/raspberry-pi-distance-sensor/
20. https://openest.io/en/services/mpu6050-accelerometer-on-raspberry-pi/
21. https://www.electronicwings.com/raspberry-pi/mpu6050-accelerometergyroscope-interfacing-with-raspberry-pi
22. https://circuitdigest.com/microcontroller-projects/mpu6050-gyro-sensor-interfacing-with-raspberry-pi
23. 18 Commands That Will Change The Way You Use Linux Forever https://www.youtube.com/watch?v=AVXYq8aL47Q
24. https://pythonbasics.org/getting-started/
25. https://www.educba.com/python-programming-beginners-tutorial/
26. https://www.rohm.com/electronics-basics/photointerrupters/what-is-a-photointerrupter

# 11 Appendix

## 11.1 PiBot Wiring diagram



## 11.2 PiBot Pinout & Breadboard connections

| Device | Pin | Rpi Pin | Bread Board | H-Bridge | Other |
|---|---|---|---|---|---|
| Left Encoder | VCC | | H3 | | |
| | GND | | C20 | | |
| | OUT | 13 | | | |
| Right Encoder | VCC | | I3 | | |
| | GND | | D20 | | |
| | OUT | | | | |
| MPU | VCC | | F1 | | |
| | GND | | A20 | | |
| | SCL | | F6 | | |
| | SDA | | F5 | | |
| Left Line Sensor | VCC | | G3 | | |
| | GND | | I20 | | |
| | OUT | 11 | | | |
| Right Line Sensor | VCC | | F3 | | |
| | GND | | H20 | | |
| | OUT | 12 | | | |
| Ultrasonic Sensor | Vcc | | G1 | | |
| | Gnd | | B20 | | |
| | Trig | 16 | | | |
| | Echo | | A15 | | |
| LCD | VCC | | H1 | | |
| | GND | | F19 | | |
| | SDA | | G5 | | |
| | SCL | | G6 | | |
| H-Bridge | +12V | | | | Switch |
| | GND | | I19 | | |
| | +5V | | J1 | | |
| | ENA | 38 | | | |
| | IN1 | 37 | | | |
| | IN2 | 35 | | | |
| | IN3 | 33 | | | |
| | IN4 | 31 | | | |
| | ENB | 29 | | | |
| Left Motor | VCC | | | OUT4 | |
| | GND | | | OUT3 | |
| Right Motor | VCC | | | OUT2 | |
| | GND | | | OUT1 | |
| Battery | VCC | | | | Switch |
| | GND | | J19 | | |
| 680Ω resistor | | | E15 and G15 | | |
| 1,5kΩ resistor | | | H15 and H19 | | |
| Jumper | | 18 | J15 | | |

| | | | | | |
|---|---|---|---|---|---|
| Jumper | | 1 | J3 | | |
| Jumper | | 2 | I1 | | |
| Jumper | | 3 | J5 | | |
| Jumper | | 5 | J6 | | |
| Jumper | | | G19 and G20 | | |
| Jumper | | | E20 and F20 | | |
| Jumper | | 39 | J20 | | |

## 11.3 GPIO Pinout

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | 3.3V PWR | 1 | 2 | 5V PWR | | |
| MPU | LCD | I2C1 SDA | GPIO 2 | 3 | 4 | 5V PWR | | |
| MPU | LCD | I2C1 SCL | GPIO 3 | 5 | 6 | GND | | |
| | | | GPIO 4 | 7 | 8 | UART0 TX | | |
| | | | GND | 9 | 10 | UART0 RX | | |
| | L Enc in | | GPIO 17 | 11 | 12 | GPIO 18 | PWM | En A |
| | R Enc in | | GPIO 27 | 13 | 14 | GND | | |
| | | | GPIO 22 | 15 | 16 | GPIO 23 | In 2 | |
| | | | 3.3V PWR | 17 | 18 | GPIO 24 | In 1 | |
| | Line 1 | SPI0 MOSI | GPIO 10 | 19 | 20 | GND | | |
| | Line 2 | SPI0 MISO | GPIO 9 | 21 | 22 | GPIO 25 | | |
| | | SPI0 SCLK | GPIO 11 | 23 | 24 | GPIO 8 | SPI0 CS0 | |
| | | | GND | 25 | 26 | GPIO 7 | SPI0 CS1 | |
| | | | Reserved | 27 | 28 | Reserved | | |
| | In 3 | | GPIO 5 | 29 | 30 | GND | | |
| | In 4 | | GPIO 6 | 31 | 32 | GPIO 12 | PWM | |
| | En B | PWM | GPIO 13 | 33 | 34 | GND | | |
| | PWM | SPI1 MISO | GPIO 19 | 35 | 36 | GPIO 16 | SPI1 CS0 | |
| | | | GPIO 26 | 37 | 38 | GPIO 20 | SPI1 MOSI | |
| | | | GND | 39 | 40 | GPIO 21 | SPI1 SCLK | |