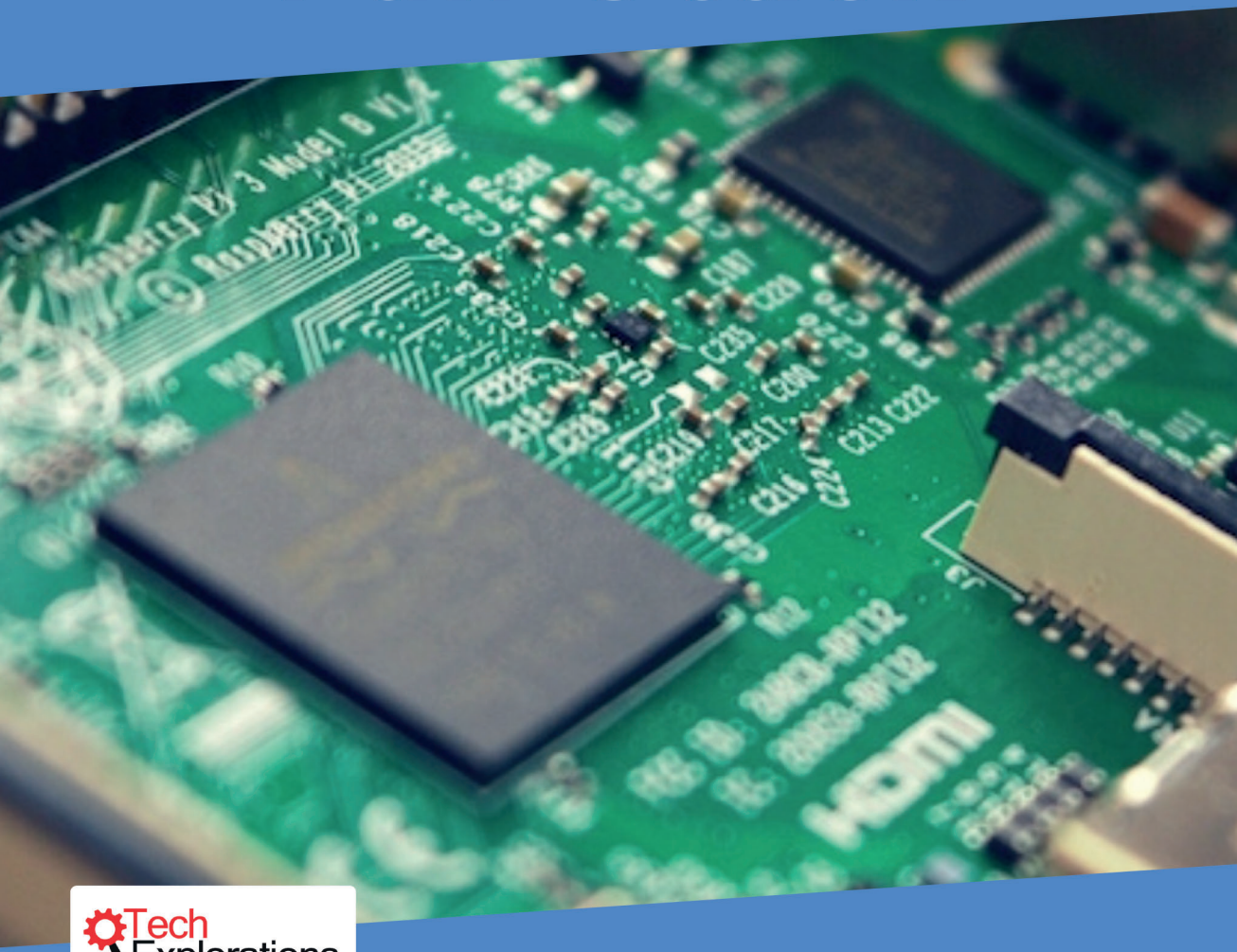
elektor books

A comprehensive course that will teach you how to build a modern IoT application

Raspberry Pi Full Stack



Tech Explorations
www.techexplorations.com

Dr. Peter Dalmaris

elektor
design > share > sell

Raspberry Pi Full Stack

Dr. Peter Dalmaris

elektor

LEARN DESIGN SHARE

This is an Elektor Publication. Elektor is the media brand of
Elektor International Media B.V.
78 York Street, London W1H 1DP, UK
Phone: (+44) (0)20 7692 8344

Declaration

The author and publisher have used their best efforts in ensuring the correctness of the information contained in this book. They do not assume, or hereby disclaim, any liability to any party for any loss or damage caused by errors or omissions in this book, whether such errors or omissions result from negligence, accident or any other cause..

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Cover designer: Harmen Heida
Prepress Production: D-Vision, Julian van den Berg
Printed in the Netherlands
First Printing: 2020

ISBN: 978-1-907920-95-0
ISBN (PDF): 978-3-89576-378-6
ISBN (epub): 978-3-89576-379-3

About the author

Dr. Peter Dalmaris is an educator, electrical engineer, electronics hobbyist, and maker. He is the creator of online video courses on DIY electronics and author of several technical books. Peter is the author of 'Maker Education Revolution', a book about how making is changing the way we learn and teach in the 21st century.

As a Chief Tech Explorer since 2013 at Tech Explorations, the company he founded in Sydney, Australia, Peter's mission is to explore technology and help educate the world.

Tech Explorations offers educational courses and Bootcamps for electronics hobbyists, STEM students, and STEM teachers.

A lifelong learner, Peter's core skill lies in explaining difficult concepts through video and text. With over 15 years of tertiary teaching experience, Peter has developed a simple yet comprehensive teaching style that students from all around the world appreciate.

His passion for technology and the world of DIY open-source hardware has been a dominant driver that has guided his personal development and his work through Tech Explorations.

How to read this book

All examples, descriptions and procedures have been tested on a Raspberry Pi Zero W, running Raspbian Buster. I have also tested the exact same project on a Raspberry Pi 4, Raspberry Pi 3 and Raspberry Pi 1.

This book has a web page with resources designed to maximise the value it delivers to you, the reader. Please read about the book web page, what it offers and how to access it in the section 'The book web page', later in this introductory segment.

Finally, you may be interested in the video course version of this book. This course contains detailed demonstrations and explanations of the Full Stack project. The video lectures capture techniques and procedures that are just not possible to do in text.

Please check in the book web page for updates on this project. Be sure to subscribe to the Tech Explorations email list so I can send you updates.

Requirements

To make the most out of this book, you will need a few things. You probably already have them.

The list of hardware requirements is available on the Tech Explorations website:

https://techexplorations.com/parts/rpifs-parts/

Please source these items before you embark on this project.

The book web page

As a reader of this book, you are entitled access to its online resources.

You can access these resources by visiting the book's web page at http://txplo.re/rpifsp.

The two available resources are:

1. **The book discussion space on the Tech Explorations Community**. This is a place where you can ask book-related questions and have a conversation about your projects. I will be spending time in the forum weekly, answering questions and participating in discussions.

2. **An errata page**. As I correct bugs, I will be posting information about these corrections in this page. Please check this page if you suspect you have found an error. If you have found an error that is not listed on the errata page, please use the error report form in the same page to let me know about it.

From time to time, I will be posting additional Raspberry Pi Full Stack resources and updates on this page, so please check regularly. By subscribing to the Tech Explorations email list, you'll be sure to receive my regular book updates and news. The subscription form is in the book page.

Did you find an error?

Please let us know.

Using any web browser, go to http://txplo.re/rpifsp, and submit a ticket.

Please take care to provide enough details in your ticket so I can find the bug. Please remember to include the page number of the PDF version of the book.

I'll get it fixed right away.

Part 1: Getting started with the Raspberry Pi Full Stack

Chapter 1 • What is this book about?

Think of this book as your guide to a adventure for determined learners.

Yes, an adventure.

An adventure of courage, determination, and collaboration.

If you choose to stay to path, you will be rewarded with knowledge.

And, with my admiration.

Because this adventure is not going to be a walk in the park.

You will struggle.

You will fail.

Many times.

You will curse me :-)

But each time, you will get up, dust yourself off, and continue your journey to complete this full stack application.

I will be there to help you at every step of the way.

I will guide you and show you the way.

But you will have to do the hard walk.

You will have to write the code, solder the circuits, test the connections and search the database

At the end, you will have more than knowledge: you will feel the sweet sense of accomplishment, and you will know that you are strong enough to conquer the next challenge.

The adventure I am talking about is a project, in which your weapon is a Raspberry Pi. Your objective is to create a useful application that runs on the Raspberry Pi and spans the Internet, and your enemy is your lazy self.

While most books are filled with micro-projects, this one is the opposite. The whole book is one BIG project.

This project brings together multiple technologies: electronics, computers, operating systems, networking, multiple programming languages, a database, Internet of Things platforms, and more.

You will learn how to use these technologies to create something that actually works.

This will be an application with many moving parts. In terms of its architecture and the specific components that it contains, it is an accurate analog of commercial Internet of Things applications in domains such as Smart Homes, Smart Cities and Manufacturing.

The application you will build is scalable to a global scale with a few targeted modifications, and it is a platform on which you can build.

Chapter 2 • A walk-through the Full Stack project

Ok, so you know that this project is going to be awesome.

But what exactly is it that you are going to do?

What exactly is it that you are going to learn?

I'm glad you asked. Let's dive in.

The image below depicts the path you will follow through this course (Figure 2.1).

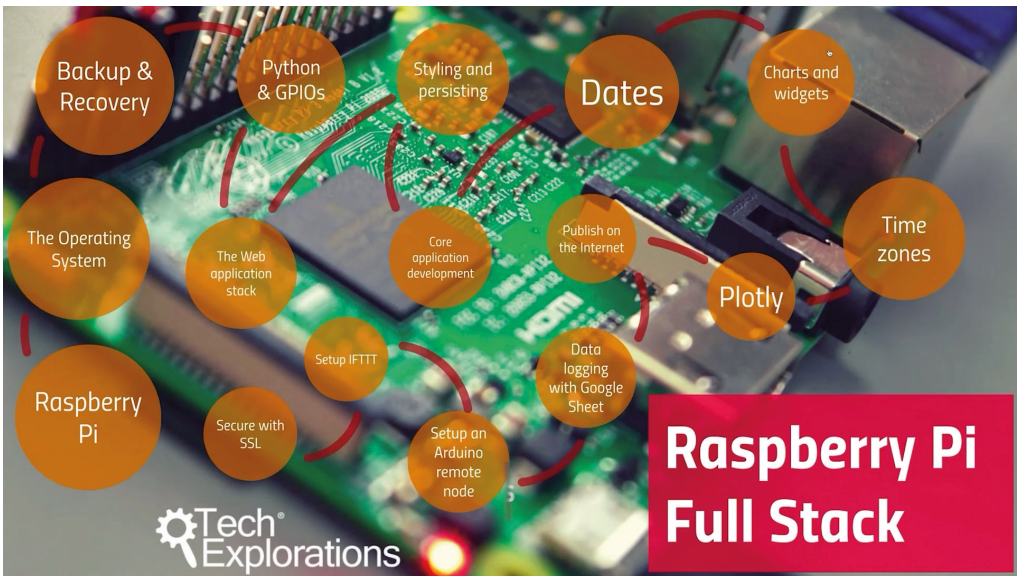


Figure 2.1: The path you will follow through this course.

The journey begins with a look at the base hardware, the Raspberry Pi. The application that you will eventually create works well on any Raspberry Pi, however in this book I have concentrated on the Raspberry Pi Zero W (for "Wireless"). The Raspberry Pi Zero represents an excellent balance of price versus performance versus size. I am amazed by how much you can achieve with this low-cost computer. In the video version of this book, I have implemented the exact same application on the Raspberry Pi 4. The implementation of the application only has a couple of small differences between the Raspberry Pi 4 and the Raspberry Pi Zero, which I will highlight when the time comes.

In the first section of the book, I will introduce you to the Raspberry Pi, and focus on the Raspberry Pi Zero as our base hardware platform. This base hardware represents the lowest level of the Full Stack application.

In the second section of the book, step two of the project, you will learn about the Raspbian operating system. This is the software that makes it possible for us, the application developers, to implement our application using high-level tools like the Python programming language and the SQLite3 database. You will learn that the Raspberry Pi, as a true multi-purpose computer, can work with a wide range of operating systems, including various flavours of Linux and even Microsoft Windows. In this project, you will be using a minimal version of the Raspberry Pi preferred operating system, Raspbian.

Because it is only a matter of "when", and not "if" for disaster to strike, **in section three of this book I will show you how to make backups of your Raspberry Pi's SD card, and then how to restore from a backup file.** Making a mistake during the implementation of a project which makes it hard or even impossible to continue is something that you should expect when you are learning something new. I have been there many times. If you are prepared for such occurrences, with full backups taken at regular intervals, you will be always be able to restore your project to a previous working state and continue from there, instead of having to rebuild from scratch.

The real work begins in section four. There, you will learn about the Raspberry Pi's General Purpose Inputs and Outputs (GPIO), and how to use them with simple Python programs. I will explain how to refer to a specific GPIO and show you the basic Python commands you need to perform simple tasks, like turn an LED on and read the state of a button. All big things start with the first simple step, and for this project, the first step is to blink an LED.

In section five, you will learn about the application stack and the components and services that are present in each level. This is where you will create the framework of this project, and learn about where in the stack various back-end and front-end technologies are placed such as the Flask micro-framework for building Python web applications, the uWSGI application server, the Nginx high-performance web server, the SQLite3 database server, HTML/CSS/JavaScript/JQuery, and, of course, Internet of Things platforms like Google's various Cloud APIs, IFTTT and Plotly. Yes, you will learn how to use all of these technologies but not in one big brain dump. We'll spread the learning throughout the full project, so you can relax.

In section six, you will start building the front end of your application, and give it persistence. Persistence is the ability of an application to store data, and retrieve it later. This is where you will set up the first version of a simple Flask Python-powered web application, and give it a basic (but elegant) user interface using HTML and CSS. For persistence, you will use SQLite3, a simple yet powerful open-source database server.
From section seven and onward, you are really getting into the nitty-gritty of the application development process.

In section seven, you will create the core of your application, and flesh-out that most important functions: you will implement the sensor integration so you can get readings and store them in the database, and automate measurements so that data-login begins.

In section eight, you will learn how to work with dates in the database, retrieve a subset of sensor records from the database using the URL query string, and how to integrate graphical widgets in your application's front end to make it easy for the end user to select date and time ranges.

You are now into the most rewarding and fast-paced part of the project, where with each chapter you implement new features.

In section nine, you will learn how to create charts from the stored data, and display multiple datapoint across selectable date and time ranges graphically.

In section ten, you will revisit the date and time topic that you started in Section eight, but this time at the front-end. You will learn how to use JavaScript/JQuery and Python to convert the date and times stored in your application's database into the correct date/time for your timezone. Of course, once you complete the implementation of this feature, the conversion will happen automatically, without any input from the user.

In section eleven, you will revisit one more topic, the graphical representation of data, and learn how to create temperature and humidity charts using Plotly. This is an opportunity to learn how to interact with a powerful Cloud service, and increase the value of your application by giving the user the opportunity to process their environment data with Plotly's powerful graphical analysis tools.

In section twelve, you will learn how to access your Raspberry Pi Full Stack application from anywhere in the world. By doing so, you will be able to check your application on your phone no matter where you might happen to be in the world.
Most people would stop here and call it a day, but not us. There's more to do.

In section thirteen, you will learn to record data from your sensors on a Google Sheet using the Google Drive and Sheet APIs. This work will give you a unique insight into how you can integrate a sophisticated Cloud service, like Google Drive and Sheet, including the confusing processes around creating and using its authentication and encryption certificates.

In section fourteen you will learn how to expand your Raspberry Pi Full Stack application by connecting (wirelessly) Arduino nodes with their own sensor. With this extension, your Raspberry Pi will be recording data from an arbitrary number of Arduino nodes (as well as from its own sensor), both on its local database, and on Google Sheet.

With the newly expanded application from section fourteen, you will jump into section fifteen and learn how to use another popular Cloud service, If This Then That (IFTTT). You will learn how to implement a configurable email alert app that will email you when the temperature or humidity in one of your application nodes exceeds a threshold. Once you know how to do this, you will be able to integrate any IFTTT capability with your application. It is incredible what you can achieve in just two steps (step 1: "if this", step 2: "then that").

To conclude this project, in Section sixteen, you will learn how to secure your application with SSL. This is important especially if you have chosen to expose your application to the Internet, as you learned in Section twelve. One you implement the security components and configuration on the web server, you will be able to access your application page with the secure HTTPS protocol instead of the un-encrypted HTTP.
There you have it, this is the project you are about to embark on.
Ready?

Chapter 3 • Required hardware

This project is hands-on. While you will be spending most of your implementation time on the software side, you will need to assemble the hardware first. Luckily, the hardware requirements are few and simple.

Below, you will find the list of hardware components you will need. Please use this list as a quick references, but check out the online list of parts to ensure that you have the latest details about the hardware.

You can find the online parts list here: https://techexplorations.com/parts/rpifs-parts/

A question I frequently receive from students of Raspberry Pi Full Stack is "which Raspberry Pi" and "which Arduino" should I use?

Here are my guidelines:

Which Raspberry Pi?
It doesn't really matter. I tested this project on the original Raspberry Pi 1 Model A. In fact, my Raspberry Pi 1 is still running the original code for this project, going strong since 2014.

I also tested with the Raspberry Pi 2, 3 and 4 (all, Model B), as well as the Raspberry Pi Zero W. In all cases, the application works flawlessly.

Your choice of Raspberry Pi for this project comes down to these considerations:

- Do you have a spare Raspberry Pi?
- Do you mind waiting a little longer for the software compilation and installation to complete?
- Do you prefer not to use the more recent and expensive Raspberry Pi's (especially the Raspberry Pi 4 with 4GB RAM).
- Do you prefer a smaller footprint?

Once you have completed the application and it is working on your Raspberry Pi, it will be hard to "feel" the performance difference between the various Raspberry Pis, with the exception of the Raspberry Pi 1. The Raspberry Pi 1 is and feels slow to the end user, so I would recommend against it.

So, let's exclude the Raspberry Pi 1 from the remainder of this discussion.

The application is currently running on a Raspberry Pi 4, 3 and Zero and I can hardly feel the difference.

Next, consider the performance of each computer during development. Installing and compiling C code is computationally intensive. To compile the Python interpreter and RF24 drivers you will need anywhere between 30 minutes to one hour.

This is where the Raspberry Pi 4 beats all other Raspberry Pi's by a long way.

It is fast, and this means you will not have time to get coffee or tea during compilation. The Raspberry Pi Zero is the slowest of the modern Raspberry Pi's because of its low clock speed and single core. Still, I didn't find the performance unworkable. I just went to get coffee or worked on my email queue while the Raspberry Pi Zero was "sweating" with the compilation.

If you are the kind of person that goes for top raw performance, go for the Raspberry Pi 4. It has so much power that you can easily run the Full Stack application using the full GUI version of the operating system, as well as run other applications on it, like the Workbench Automation Computer application, and more.

The Raspberry Pi 4 is super-fast, but it also quickly gets super-hot. This heat tends to effect the temperature sensor which consistently shows a higher temperature on the Raspberry Pi 4, skewing your temperature data by one or two degrees Celsius.

If you are concerned by this, and want to ensure more accurate measurements, you should avoid the Raspberry Pi 4, or just ignore the data from its on-board sensor and instead use data from Arduino nodes.

If you are not thrilled by the expensive Raspberry Pi 4 and the amount of heat it produces, consider the Raspberry Pi Zero W (Wireless). It is not crazy fast, but is very cheap, fast enough during development, and has a tiny footprint. You can use it with a regular USB power supply (instead of the more expensive and specially-designed power supply for the Raspberry Pi 4).

The Raspberry Pi Zero W is my recommendation.

As you can see in Figure 3.2, the Raspberry Pi Zero W, with the custom project HAT that contains the sensor, button, LEDs and RF24 transceiver measures around 7 cm in length, 4 cm in height and 5 cm in width.

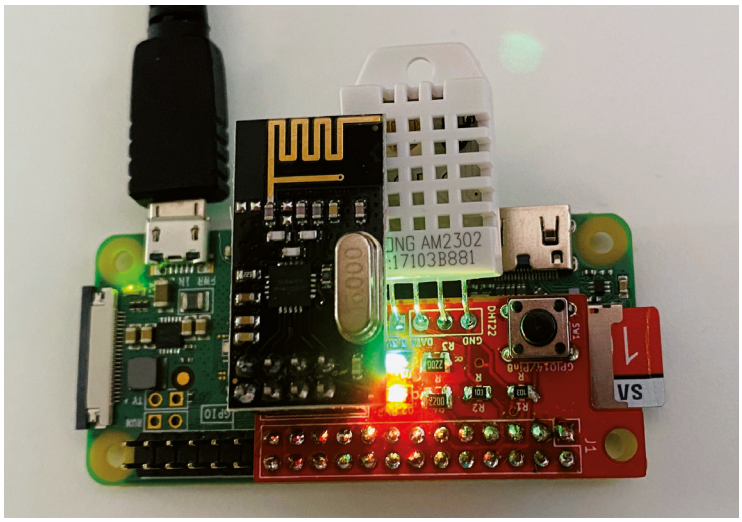


Figure 3.2: My Raspberry Pi Zero W with my custom project HAT.

It emits minimal heat and costs less than $20. Unless you have specific reasons to use a different model , go for the Zero.

Which Arduino?
For the Arduino, the guidelines are much simpler. Just use the Arduino Uno or something compatible, or whichever Atmega328-based Arduino board you happen to have.

I have tested the circuit and sketch with an Arduino Uno and Arduino Pro Mini boards, and the operation in both cases was flawless.

List of hardware
For a more complete and up-to-date list of hardware components, please check https://techexplorations.com/parts/rpifs-parts/. The following is a simplified list that is centered around the Raspberry Pi Zero W.

- A Raspberry Pi Zero W
- An Arduino Uno or 100% compatible
- SanDisk Ultra 8GB (or larger, I use 16GBytes) Class 10 UHS-I MicroSDHC Memory Card with Adapter
- 2 x 10KΩ resistors (a pull-up for the button, and a pull-up for the sensor Data pin)
- 2 x 330Ω resistors (for the LEDs)
- 2 x LED Diodes (choose your preferred colours)
- 2 x DHT22 temperature and humidity sensors
- 2 x NRF24L01+ 2.4GHz Wireless RF Transceiver
- A breadboard-friendly momentary button
- Mini breadboard
- Jumper wires

The HAT PCB

I designed a tiny HAT that I invite you to use in your project. You can see a photograph this HAT in Figure 3.2.

I found using this Raspberry Pi HAT instead of the breadboard made my work with the application faster. I did not have to worry about loose jumper wires, and it also looks great. You can download a copy of its Gerber files here, and order your boards from your preferred manufacturer.

Alternatively, you can go to this URL and order your PCBs from my shared project:

> https://www.pcbway.com/project/shareproject/Raspberry_Pi_Full_Stack_RF24_and_DHT22_HAT.html

If you choose to work with the custom PCB instead of the breadboard, you will not need to source the through-hole components from the list, numbered 4 to 11. Instead consult the components list in my shared project for the SMD components you will need to solder on the PCB.

Chapter 4 • How to get help

At Tech Explorations, we support our students through our community spaces.

Raspberry Pi Full Stack has its own space, which you can reach at https://community.techexplorations.com/c/raspberry-pi-full-stack.

If you purchased this book directly from the Tech Explorations website, you should have already received an invitation to join our community. Please look for it in your inbox (also check your spam folder). In this email, you will find a link to accept my invitation. Click on the link to accept the invitation and fill in the form to create your free community account. If you purchased this book from one of our partners, you can still join our community, but you will need to complete these steps:

Go to our support page at https://txplo.re/support.

Click on the support option "Join the Makers Club".

In the field "Where did you purchase your course", choose the most appropriate option from the list.

In the text box "Comments", indicate that you wish to become a member of the Raspberry Pi Full Stack community space. Important: please copy your book order details.

In the field "Email configuration to", include the email address where you would like us to send you the invitation.

That's it. Please allow us up to 24 hour to process your request.

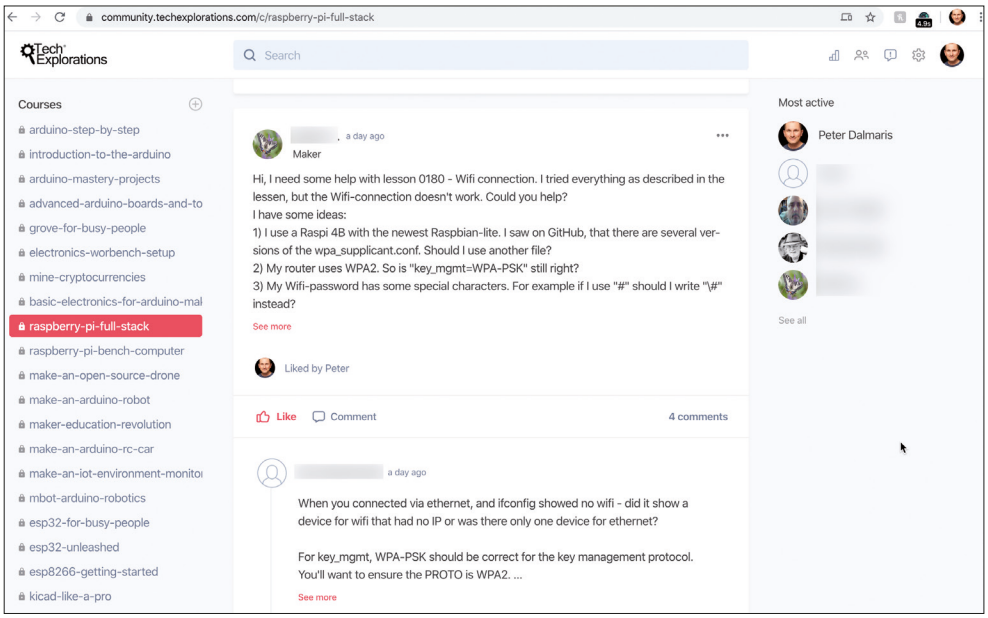


Figure 4.3: The Community Space for the Raspberry Pi Full Stack project.

Chapter 5 • The code repository

You can find the source code for this project in the course code repository on Github.

Bookmark this URL: https://github.com/futureshocked/RaspberryPiFullStack_Raspbian

The repository contains the source code in various stages of development. You can copy this code into your project instead of typing it. This will save you time and frustration that is inevitable with typing.

Apart from Python code, the repository also contains my command line sessions and various configuration files. I have tested each and every one of these files and I am confident that they work.

When you run into a problem, check the source code at Github so that you can trust that it works, and use it to try and identify the problem in your code.

Please note that the code in the repository is "alive". This means that I frequently update it to fix bugs or improve functionality. It is likely that you will find differences between the code I have captured in this book, and the "live" code in the repository.

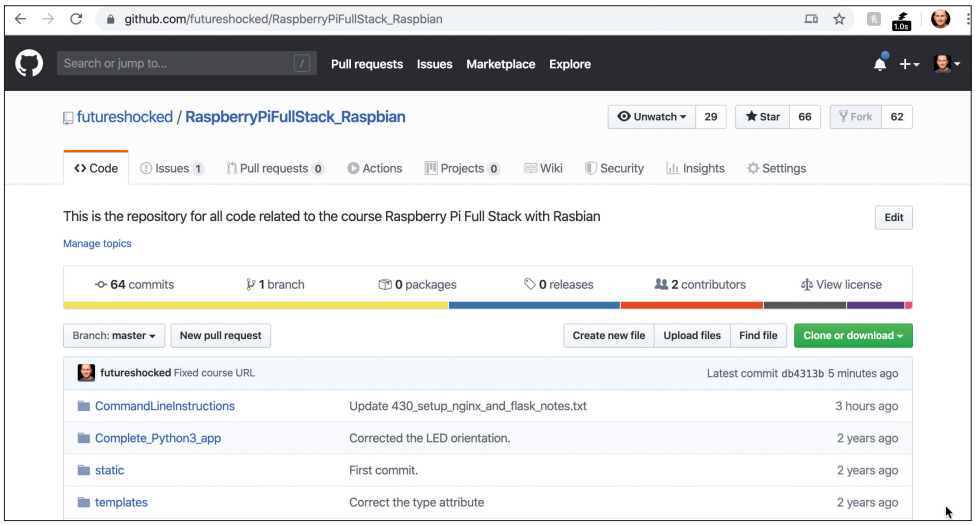


Figure 5.4: The Raspberry Pi Full Stack project code repository on Github.

Part 2: Raspberry Pi, Arduino, and Raspberry Pi Zero W

Chapter 6 • Raspberry Pi vs Arduino high level comparison

For many of us, the Arduino is the technology that introduced us into the world of programmable electronics. It is relatively simple to learn, forgiving of many common beginner mistakes, with a whole universe of documentation, example code, circuits, and projects.

We start exploring other technologies once we feel that we have grasped at least the basics of the Arduino. For many of us, the next logical step is the Raspberry Pi. The Raspberry Pi opens up new possibilities, especially when it comes to interfacing our projects with advanced online or computationally intensive features.

A question I regularly receive is about the differences and similarities between the Raspberry Pi and the Arduino. I think it is a very good question because once you understand the differences and similarities you will be able to devise ways for the two to work together, in a simple project.

The Raspberry Pi and Arduino, or, to be more precise, the Arduino Uno, are complementary technologies. While they share some common characteristics, they are different in almost every aspect that matters. And yet, because of their differences, they are each other's perfect complement.

In this chapter, I present an high-level comparison between the Raspberry Pi and the Arduino, based on the table in Figure 6.5.

Raspberry Pi vs Arduino

Raspberry Pi	Arduino
Microcomputer	Microcontroller
Needs an operating system	Does not need an operating system
Complicated	Simple
Video out, Camera, Ethernet ports, Wifi, Bluetooth, USB, I2C, SPI, UART etc. on board	USB only for power and serial in/out, I2C, SPI, UART
Best for general computer	Best for small tasks that constantly repeat
Capable of performing a huge range of tasks	Optimised for sensing and controlling the world around it
Best for more advanced makers	Best for beginners
Programmed in many languages, including C/C++, Python, Ruby	Programmed in C/C++
Relatively high power consumption	Relatively low power consumption

Raspberry Pi Full Stack Tech Explorations

Figure 6.5: A high-level comparison between the Raspberry Pi and the Arduino Uno.

So, here are the 10 most important differences between the Raspberry Pi and Arduino.

Microcomputer vs Microcontroller

The Raspberry Pi is a regular computer, similar to the one on your desk. It just comes in a small package. The Raspberry Pi has all the usual features and capabilities that you expect a computer to have.

For example:

- You can connect a large, high-definition monitor, capable of playing 1080p video.
- You can connect a keyboard, a mouse, and external hard disks.
- It can connect to the Internet via Ethernet and WiFi.
- It requires an operating system, and can run your choice of Linux or Windows.
- For a tiny computer, it has respectable specifications, including a 64-bit quad core CPU with hardware video acceleration and 4GB RAM.
- It has multiple USB connectors for peripherals.

For the price and size of the Raspberry Pi, these specifications are impressive.

On the other hand, the Arduino is a totally different machine. The most important difference between the Arduino and the Raspberry Pi, is that the latter is not a computer. It is a microcontroller.

Think of a microcontroller as a very small computer without any of the things that we expect from a desktop or a laptop computer.

Take the RAM, for example, and focus on the Arduino Uno. The Arduino Uno has tiny amount of RAM, just 2 kilobytes, and can't to connect to a network like the Internet for example, without additional components. Compare this to the 4 gigabytes of RAM that are available on the Raspberry Pi 4!

The Arduino Uno only has 32 kilobytes of flash memory for storing your program, whereas the Raspberry Pi can use very large amounts of storage space on removable SD cards (and you can also connect terabytes of external disk space via USB).

Another big difference is performance. The Arduino Uno is powered by an 8-bit, 16Mhz microcontroller that contains a single-core CPU. The Raspberry Pi is powered by a 64-bit quad core ARM CPU, clocked at 1.5GHz. The comparison is far from fair. The Raspberry Pi can do real-time video processing on applications like object recognition and artificial intelligence, while the Arduino Uno would be stretched to it limits trying to identify a loud noise.

The Arduino Uno does not use an operating system. Instead, your programs are compiled to run directly on the hardware, and that is why their compiled version is called "firmware".

As you can see, the two devices that are totally different. That's because a microcontroller is designed for doing tasks that are small, constantly and repeatedly. A microcomputer is designed as a general-purpose device, with a very large repertoire of possible applications.

For example, if you want a gadget that measures the temperature and humidity in a room, and then depending on those single two values to turn on a fan or an air conditioner, then something like the Arduino or a microcontroller in general would be the best technology on which you can build such a gadget.

If you want something that can recognize a person's face from a video stream and then sent out an alert or turn on an alarm and log that event in an online database, play a movie on Netflix, and check your email, then you will need a Raspberry Pi.

The Raspberry Pi is great for general computer tasks, whereas the Arduino is great for things that are small in complexity, small in programming size, and repeat constantly.

A great aspect of the Arduino is all of its input and output pins and ability to connect them to actuators and sensors, so things such as relays, switches and motors, then the Arduino is really a good choice, really optimized for sensing and controlling the world around it.

Just like the Arduino, the Raspberry Pi does have a bunch of general-purpose input output pins that you can connect to actuators and sensors, just like the Arduino.

So there is definitely a good case to be put forward that says that the Raspberry Pi can actually do everything that the Arduino can, and therefore you don't really need an Arduino. But there is a flip side to that: complexity and robustness.

The great thing about the Arduino is that it's simple and forgiving.

The fact that there is no operating system to worry about means that you don't have to learn how to use one just to turn on a relay.

For beginners, I always say that the best way to get started with electronics and with control applications is through the Arduino, gain an understanding of its simple hardware, and build-up skill using the bare minimal infrastructure that a microcontroller provides before moving onto something more complicated.

When a new person comes into the world of electronics and control, their objectives tend to be fairly simple; just blinking an LCD very often is a great start. A simple start like that is a significant achievement that can lead to much greater things.

You can achieve wins like this very quickly quickly with the Arduino.

Programming

Another big difference between the Arduino and the Raspberry Pi is programming.

The Arduino is programmed in C++. Normally, C++ is not a language that I would recommend to a beginner. It is fairly complicated, and many concepts and techniques that are not intuitive. A language like Python or Ruby is much more suitable as a first programming language. However, the Arduino did not become so popular despite the difficulty in

learning C++.

The people that created Arduino were keenly aware of this, so they created a collection of libraries that hide most of the complexity of C++. It makes programming so simple that learners don't realise they are using C++ until well into their learning journey. We often refer to these libraries as the "Arduino language", even though we are really using the hard-core C++ that programmers use to create the Linux operating system.

In addition to the language, the Arduino developer team created a very simple development environment that hides all of the complexity of "regular" C++ development environments and compilers that, again, can confuse even those of us with many years experience of using them.

In summary, the Arduino is an excellent choice for beginner because it is based on relatively simple hardware, and has a very simple programming interface and environment.
On the other hand the Raspberry Pi, being a full microcomputer, has a wide range of languages that you can use to program it.

Any language that you can use in Linux is also available for Raspberry Pi.

By default it comes with support for C and C++ and Python. You can also install other languages, like Ruby, PHP, JavaScript, Lua, etc. Virtually any programming language is available on the Raspberry Pi. If you're proficient in any language, chances are that you'll be able to use it with Raspberry Pi.

Keep in mind that most of the Raspberry Pi documentation is written for Python. Python is, by many accounts, the most widely used high level language in the world.

Compared to C++, it has a very short learning curve. Even if you've never used Python, you'll be able to pick it up fairly quickly.

Hardware
Hardware-wise, the Raspberry Pi is more complicated than the Arduino Uno because of the amount of peripheral hardware that is present on its circuit board.

The Raspberry Pi 4 comes with one or two HDMI video output ports, a camera port (so you can do things like face or object recognition), a 1 gigabit Ethernet port, Wi-Fi and Bluetooth, and multiple USB ports (so you can connect your keyboards, mice, external video cameras, and anything else that pretty much has a compatible driver for the Raspberry Pi running linux).

The Raspberry Pi is unique among computers because it exposes general-purpose input/output pins that you can use to interface with other external modules using communications protocols such as I^2C, SPI, and UART serial. Sensors motor controllers, LCD displays, relays, and many other devices can be connected directly to the Raspberry Pi via GPIOs.

In contrast, the Arduino Uno looks like it's lacking in almost every respect. It doesn't have a USB host suitable for things like a keyboard or external storage. It's single USB port allows you to connect your Arduino to your computer for power and serial communication only. However, the Arduino Uno does contain a small number of GPIO pins, that you can use to connect sensor, actuators, displays and other peripherals using the I^2C, SPI and UART protocols.

Power

Finally, I'd like to mention a few things about power consumption.

As a computer with a lot more resources and hardware to power, the Raspberry Pi consumes much more current compared than the Arduino. The Raspberry Pi consumes around 2.8 W of power when it's idle and not doing anything useful. It requires more than that if you have connected things such as touch screens and cameras. It requires a power supply with at least 3A at 5V.

The Arduino, on the other hand, has very low power consumption needs. This makes it suitable for low-power, battery applications. It can operate with as little as 20 milli amps, for example, versus 700 milli amps for the Raspberry Pi. For a microcontroller, 20 milli amps is not negligible, but with the power management features built into the chip, it can be reduced significantly.

What is power management? It is one of the advantages of microcontrollers over regular computers. Microcontrollers are built to be able to programmatically control the power consumption of their various components when not needed in order to reduce power draw. It is possible to program a custom version of an Arduino to operate for many months on a small alkaline battery by turning of its subsystems and "waking" them up when needed. When the job is done, the Arduino can go back to sleep and conserve power.

The ability to programmatically control the power consumption of the microcontroller is a very important advantage of the Arduino and is particularly useful when you want to build a gadget that needs to be deployed out in the field, away from power sockets.

Chapter 7 • Need for efficiency: the Raspberry Pi Zero W

The Raspberry Pi Foundation brought its first computer to the market in 2012. This was the original Raspberry Pi Model B, with a 26-pin GPIO header, 85.60 mm x 56.5 mm in size, which created a precedent for the Raspberry Pis that followed.

The foundation has created several newer models since the original board design: the Raspberry Pi 2 (2015), the Raspberry Pi 3 (2016), and the Raspberry Pi 4 (2019).

In 2015, the Raspberry Pi Foundation released the Raspberry Pi Zero, which had the same CPU as the original Raspberry Pi, a 46-pi GPIO header, but a much smaller footprint: just 65 × 30 mm. Perhaps the most amazing aspect of the Raspberry Pi Zero was its pricing: just $5. This model was so popular that I remember purchasing unit limits and long delays in deliveries. The Zero was perfect for applications that needed a small footprint, and many creators embedded it into their designs without any modifications. Prior to the Zero, people had to manually remove unwanted components, such as video and USB ports in order to reduce the size of the board.

The Raspberry Pi Zero had one major weakness: it had no built-in radios. This meant no WiFi, and or Bluetooth. This was corrected in 2017, with the introduction of the Raspberry Pi Zero W. The Zero W looked the same as and was like the Zero in every aspect: same CPU, same RAM, same GPIO header. But it had one important difference: integrated WiFi and Bluetooth. The price went from $5 to $10, still much cheaper than all other Raspberry Pi's (except for the Zero), and even cheaper then all official Arduinos.

For many makers, the Raspberry Pi Zero W was a game changer because it made it possible to create tiny Internet of Things applications, at a small cost.

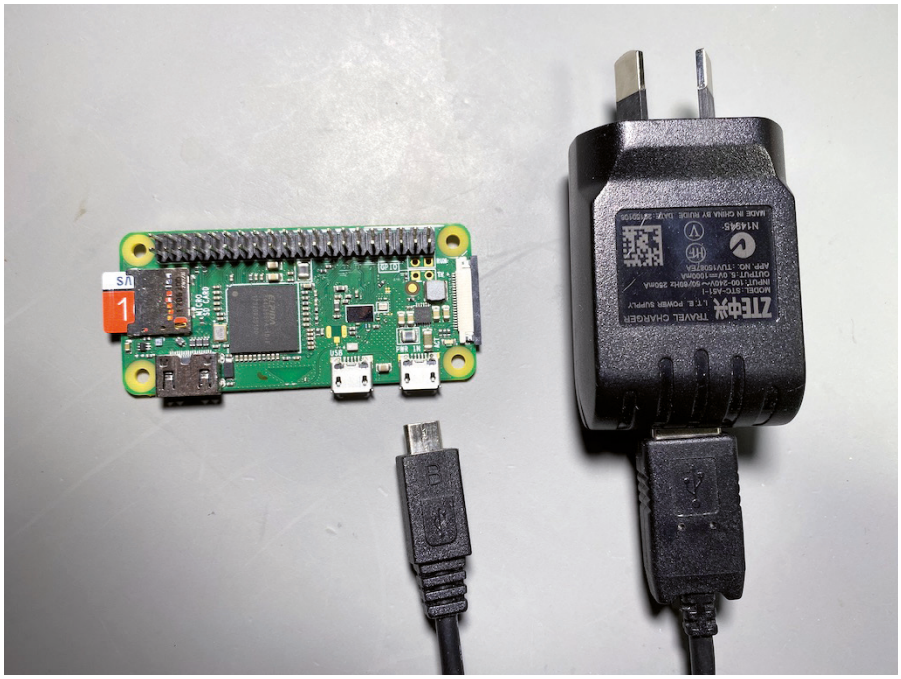
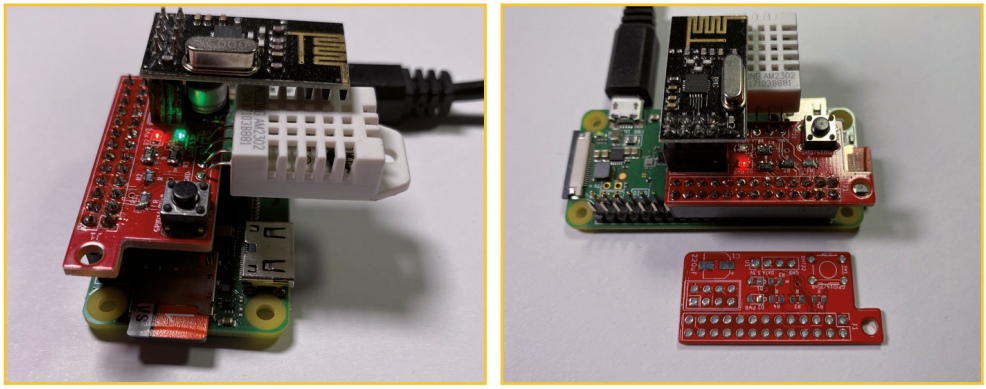


Figure 7.6: My Raspberry Pi Zero W, powered by a USB power supply that I found in a drawer.

When I created the first edition of the Raspberry Pi Full Stack course, I used an original Raspberry Pi Model B and tested it against the Raspberry Pi 2. I have since created the second edition of the course on a Raspberry Pi 3, and tested it with a Raspberry Pi 4. Working with the Raspberry Pi 4 is amazing because of its speed. It is, however, very expensive, and "fiddly". The Raspberry Pi 4, with a recommended power supply, huge heatsink and SD card, can cost around $150.

With the Raspberry Pi Zero W, you don't need a beefed-up and expensive power supply or heatsink. For $10 plus an SD card, you can power it from any USB port on your computer or a regular USB charger, and you can start working. You can see my Raspberry Pi Zero W with the regular USB power supply I use for it in Figure 7.6.

Of course, the Raspberry Pi Zero W will not be as fast as the Raspberry Pi 2, 3 or 4. But for the purposes of this project, this is not a problem. Only a few operations during this project can take advantage of high CPU throughput. One is the compilation of the Python interpreter and the C++ source code of the RF24 modules. Another is when the user retrieves a large number of records from a database. For everything else, it is hard to "feel" what kind of Raspberry Pi you are working with.



Side view Front view

Figure 7.7: My Raspberry Pi Zero W, with the project HAT attached.

Another aspect of the Raspberry Pi Zero W that I appreciate, is that it provides an excellent physical framework on which to implement the project hardware. In Figure 7.7, you can see that the HAT I have designed for the project, with all its components populated, fits perfectly over the Raspberry Pi. There is very little wasted space.

If you are looking for the most efficient hardware platform on which to build your Raspberry Pi Full Stack application, I recommend the Raspberry Pi Zero W.

If you are looking for the highest possible performance specifications on a Raspberry Pi, read the next chapter.

Chapter 8 • Need for speed: the Raspberry Pi 4 (and 3)

The Raspberry Pi 4 is the complete opposite of the Raspberry Pi Zero. It is crazy fast, large, with a full array of ports and connectors. It is a total energy hog. For the price of one Raspberry Pi 4 with 4 GBytes of RAM, you can buy 5x Raspberry Pi Zero W.

The Raspberry Pi 4 is a full computer on a single board. It can run a 64-bit operating system, including Windows, can support two high-definition displays, has a true Gigabit Ethernet port, dual-band WiFi, and even supports USB 3 devices (great for connecting external storage).

Of course, you can use a Raspberry Pi 4 to complete this project. Because the requirements of the project are minimal, it is true that using a Raspberry Pi 4 can be seen as an overkill. However, it is not if you consider the possibility of using a single Raspberry Pi for multiple projects.

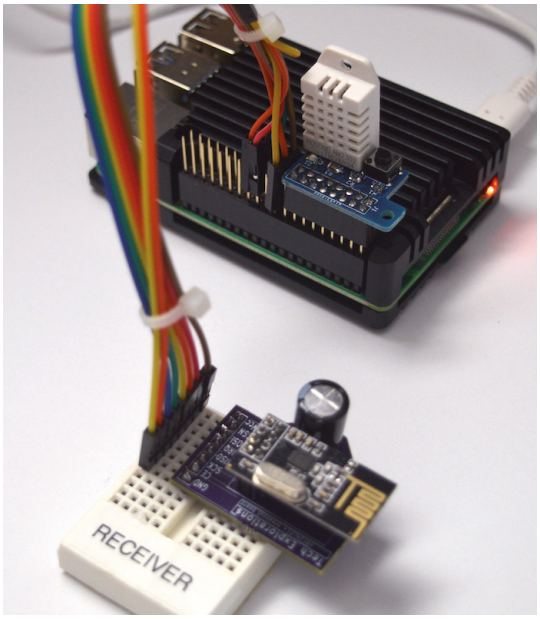
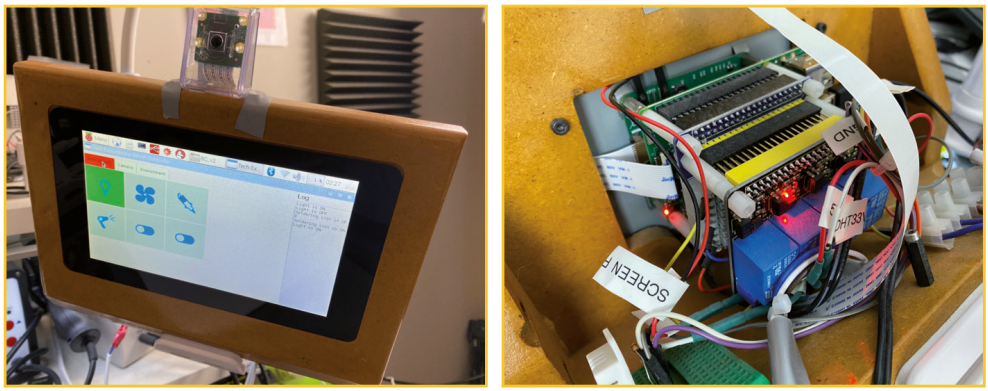


Figure 8.8: My Raspberry Pi 4, with a huge heat sink, running the full-stack application.

For example, I have used my Raspberry Pi 4, 4 GB RAM, to implement the full-stack project, but also as a desktop computer for experimenting with Mathematica, and browsing the web. You can see my Raspberry Pi 4, with its large heatsink, running the full stack application in in Figure 8.8.

In a similar way, I am using a Raspberry Pi 3 to implement the full stack project as well as a workbench computer. The workbench computer project is one of my most popular projects, in which you use a Raspberry Pi with a TFT screen to build a graphical user interface (using Python) to control bench appliances (like a soldering iron and a fan), as well as take photos

and videos of my work on the workbench, and record environmental data. You can learn more about this project on the Tech Explorations website[1].



Front, the user interface **Under the “hood”**

Figure 8.9: My Raspberry Pi 3, as a Bench Computer, also running an early version of the Full Stack application.

You can see my Raspberry Pi 3 as the Bench Computer, in Figure 8.9. In the front view, you can see the large touch screen which I used to implement a touch-based user interface, programmed in Python. Under the "hood", you can see the Raspberry Pi, connected to a wiring HAT and a relay HAT, plus part of the DHT22 sensor. An important component of the hardware setup that is not visible in these photographs is the external mains power box which is controlled by a relay connected to the Raspberry Pi.

If you choose to use a Raspberry Pi 3 or 4 for this project, you will enjoy a significant performance boost compared to the Raspberry Pi Zero W. Because these models are so powerful, you will be able to use the same board for more than one project which then makes this a good choice.

1. Learn more about the project course "Raspberry Pi: Make a Workbench Automation Computer": https://techexplorations.com/so/raspberry-pi-workbench-computer/

Part 3: How to set up the operating system

Chapter 9 • Operating systems for the Raspberry Pi

As you learned in the previous section, one of the most important differences between the Raspberry Pi and Arduino is that the Raspberry Pi (as with any computer) needs an operating system, while the Arduino does not.

The operating system is the software that allows the end-user software (such as our Python scripts or C++ programs) to use the hardware resources of the computer. While it is possible to write software that uses these resources directly (usually low-level assembly code), that would be a very tedious and slow job, that requires us to have detailed knowledge of the hardware and its architecture. Only specialist programmers go to this kind of effort and they have good reasons for doing so.

For example, if you wanted to write a specialized BIOS (Basic Input/Output System), you would need to write low-level C or assembly code. This code would be executed by the microprocessor to prepare the computer for operation before it initializes the operating system.

Anyway, I digress. If you want to learn more about the BIOS, have a look at this article [2]. The Raspberry Pi, especially the last two iterations, are powerful enough to be fully capable of using a variety of modern operating systems. You can see several of these Operating Systems in Figure 9.10.

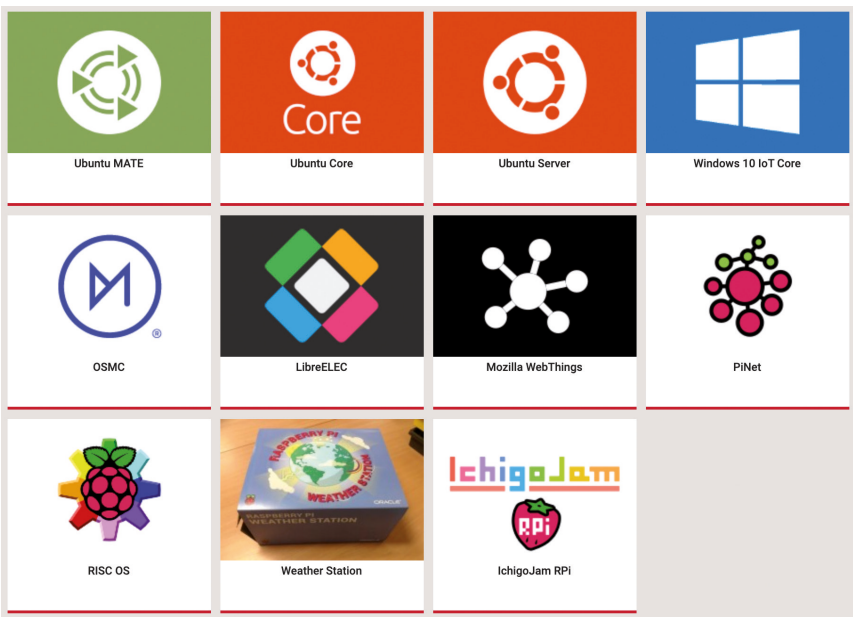


Figure 9.10: Example operating systems that work with the Raspberry Pi.

2. https://en.wikipedia.org/wiki/BIOS

These operating systems are (almost) all derivatives of Linux, and tend to emphasize particular features. For example, Ubuntu MATE emphasises a clean and easy to use Linux desktop environment , while PiNet emphasises classroom and education-friendly features such as centralised network accounts for students and shared folders.

In this project, you will use the Raspbian operating system that is developed and maintained by the Raspberry Pi Foundation. Raspbian is perfect for the Raspberry Pi because it is tuned and tested for the specific hardware that exists on a Raspberry Pi board. The documentation that comes with it is extensive and specifically written for the Raspberry Pi. With most other operating systems that are compatible with the Raspberry Pi, the documentation is common among different hardware platforms which makes it often hard to find instructions for common things, such as how to configure the build-in WiFi module.

Within Raspbian, the Raspberry Pi Foundation offers three variants (Figure 9.11).

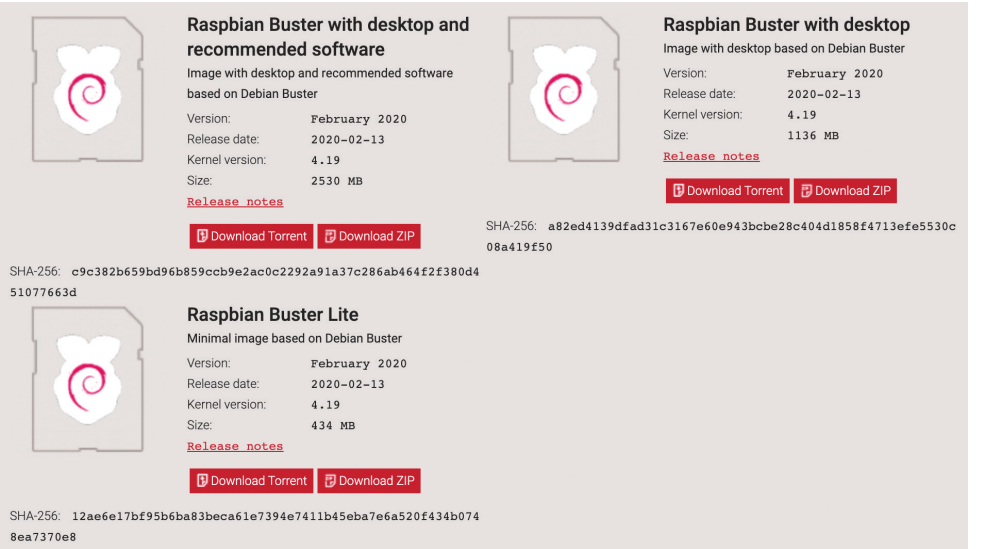


Figure 9.11: The Raspbian operating systems has three variants.

At the time I am writing these lines, the latest version of Raspbian is codenamed "Buster". Buster comes in three variants:

The first variant is the full graphical desktop with all recommended options, such as development modules for writing programs and end-user applications such as web browsers, a Scratch programming editor and compiler, NodeRed, Mathematica, and lots more. This option is loaded with software that you will probably never use, and, for the purposes of this project, is useless. In addition, this variant has a very large download size and you will need a large SD card to install it on. I don't recommend you use this variant.

The second variant contains the graphical desktop with only the necessary software options, such as a web browser and a text editor. Nothing fancy. It is half the download size of the fully-loaded variant. If you are working on the project with a Raspberry Pi 3 or 4, then you consider using this variant. I am using this variant to implement the project on my Raspberry Pi 4, and it is working very well with a 16 GByte SD card.

The third variant is "Lite". It does not contain the graphical components of the operating system, and none of the software options that require a GUI interface. It is a plain operating system that depends on the command-line interface for interaction with the user. As a result, it is very light on resources, and only a fraction of the download size of the other two variants. If you are using a Raspberry Pi Zero W, this is the recommended operating system. This is the operating system I will be using in this book and I encourage you to do the same.

With Raspbian Lite, we have a basic operating system with a very small footprint, so we can use a small, low cost SD card. Another advantage is that it does not contain components that we don't need, including things such as background services that might consume precious resources on our Raspberry Pi Zero W. Also, this means that we don't have to maintain software that we don't need by downloading updates and patches. On the other hand, we will be able to install anything we need, and know exactly what it is.

As I mentioned earlier, Raspbian Lite is a base operating system without a graphical user interface. There are no "windows", drop-down menus or pointer. We interact with Raspbian Lite using a keyboard and terminal (no mouse!).

This kind of operating system is known as "headless".

Want to know more about headless OS? Jump to the next chapter.

Chapter 10 • What is a 'headless' operating system

In this project you will be using Raspbian Lite which, as you know, is a Linux-based operating system with no graphical user interface and only the bare-minimum pre-installed software.

In addition, you will set up your Raspberry Pi Zero W without a monitor, keyboard or mouse. The only cable you will connect to it is a mini-USB power cable.

This type of setup is known as "headless".

A headless computer is one that is configured without typical input and output interfaces (monitor, keyboard, mouse). A user will interact with a headless computer through a network interface and a seperate, client computer.

In this project, you will set up your regular work computer with a terminal program, like Putty, iTerm or Windows Console and SSH (Secure SHell) protocol to connect to your Raspberry Pi via your WiFi network. You will issue text commands on the command line, as in the example provided in Figure 10.12.

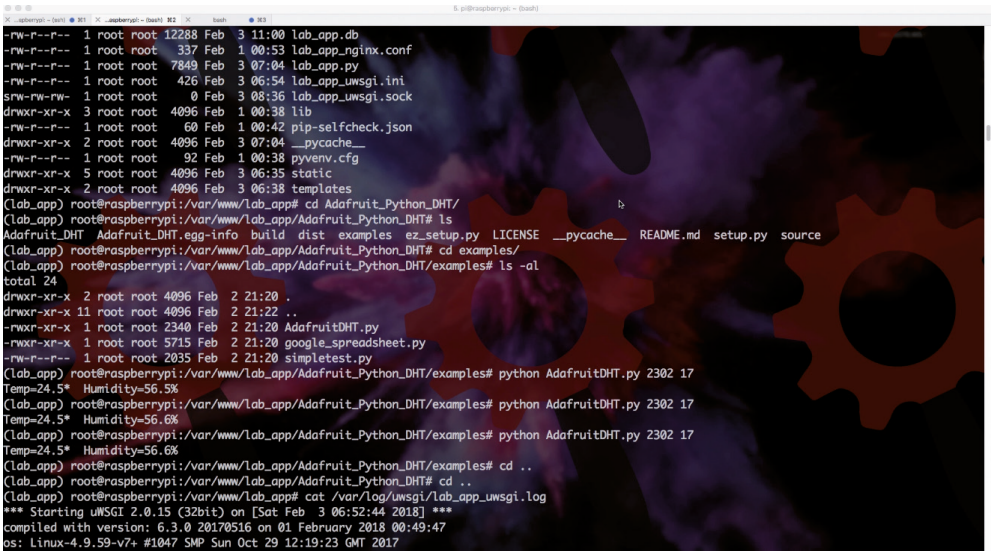


Figure 10.12: Example of an SSH session using the iTerm terminal emulator.

If you have not used a terminal emulator before, or used text commands to interact with a computer, beware that there is a small learning curve. This book will help get you started, and by the end of the project you will be a command line wizard.

There are a few more differences between a headless computer system and a GUI-based one that I want to highlight. I have summarised these differences in Table 1.

Headless	GUI
No monitor, mouse, keyboard.	Requires monitor, mouse, keyboard.
OS occupies much less disk space.	OS is larger to accomodate for the graphical components and applications.
Operated using a seperate computer via a remote connection.	Can be operated directly, no need for seperate computer.
Best for server or remote applications.	Best for desktop applications.
Faster to backup and update (due to its smaller footprint).	Longer to backup and update (due to its larger footprint).
Improved reliability (fewer things to break).	Acceptable reliability for desktop use.
Improved performance that is memory-bound, RAM is dedicated to applications.	RAM has to be shared with video and applications.
User must be familiar with the command line.	User can use point and click interface.

Table 1: The main differences between a headless and GUI computer.

Apart from the differences I have already mentioned, the above table also highlights a few other important differences.

A headless operating system without GUI components and applications has a very small footprint on the disk. This means maintenance and backup/restore operations are faster simply because there are fewer bytes to work with. In this project, I will show you how to backup and restore your work using an SD card. Backing up an 8GByte SD card (which can easily accomodate Raspbian Lite) takes half the amount of time that you need for the same thing when you use a 16GByte SD card (which is the smallest recommended size for the full Raspbian).

Another advantage is reliability. Headless computers typically run minimal operating systems so that the risk software bugs and conflicts between services is minimised. These computers can run for years without a reboot, reliably servicing end user requests. This is impossible to ask from a full desktop operating system. Even the best of these must be restarted regularly.

In this project, you will create an application that operates as an automated service, reliably, non-stop, forever. My Raspberry Pi 1 is still running the original Full Stack application and Raspbian Wheezy (the first version of Raspbian from 2013) from 2014. I admit that I had to restart it a few times due to power outages. The longest uninterrupted stretch was 9 months, which is amazing.

Another advantage of headless computers and Lite operating systems is performance. A regular GUI operating systems contains hundreds of background services that are constantly running. These services do things such as checking for mouse and keyboard events, redrawing the screen and handling user notifications. All these things take up valuable CPU

and memory resources. In a modern computer like a PC with multiple cores, and lots of RAM, this is not a big consideration. After all, these computers are designed for GUI interfaces from the ground up. However, for a Raspberry Pi Zero, with a single processing core and 512MB RAM, useless services quickly become performance bottlenecks. A Lite and headless operation releases resources that we can better use to develop and operate our web application.