

Community Experience Distilled

Raspberry Pi Robotics Projects

Second Edition

Get the most out of Raspberry Pi to build enthralling robotics projects

Richard Grimmett

[PACKT] open source*
PUBLISHING community experience distilled

www.allitebooks.com

Raspberry Pi Robotics Projects

Second Edition

Get the most out of Raspberry Pi to build enthralling robotics projects

Richard Grimmett



BIRMINGHAM - MUMBAI

Raspberry Pi Robotics Projects

Second Edition

Copyright © 2015 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: February 2014

Second edition: April 2015

Production reference: 1270415

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78528-014-6

www.packtpub.com

Credits

Author

Richard Grimmett

Project Coordinator

Kranti Berde

Reviewer

Werner Ziegelwanger

Proofreaders

Safis Editing

Maria Gould

Commissioning Editor

Neil Alexander

Indexer

Monica Ajmera Mehta

Acquisition Editor

Tushar Gupta

Production Coordinator

Conidon Miranda

Content Development Editor

Vaibhav Pawar

Cover Work

Conidon Miranda

Technical Editor

Saurabh Malhotra

Copy Editors

Dipti Kapadia

Tani Kothari

Sonia Mathur

Karuna Narayanan

Kriti Sharma

Alpha Singh

About the Author

Richard Grimmett has been fascinated by computers and electronics since his very first programming project that used Fortran on punched cards. He has a bachelor's degree and a master's degree in electrical engineering, and a PhD in leadership studies. He has 26 years of experience in the radar and telecommunications industries and even has one of the original Brick phones. He now teaches computer science and electrical engineering at Brigham Young University-Idaho, where his office is filled with his many robotics projects.

This book is the result of working with many wonderful students at Brigham Young University-Idaho. Also, it wouldn't have been possible without the help of my wonderful wife, Jeanne.

About the Reviewer

Werner Ziegelwanger, MSc, has studied game engineering and simulation. He got his master's degree in 2011, and his master's thesis was titled *Terrain Rendering with Geometry Clipmaps*, Diplomica Verlag. His hobbies are programming, playing games, and all kinds of technical gadgets.

Werner has worked as a self-employed programmer for a few years, where he mainly did web projects. During this time, he started his own blog (<http://developer-blog.net>), which is about the Raspberry Pi, Linux, and open source.

Since 2013, Werner has been working as a Magento developer and is the head of programming at mStage GmbH, an e-commerce company that focuses on Magento.

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit www.PacktPub.com.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<https://www2.packtpub.com/books/subscription/packtlib>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface	v
Chapter 1: Getting Started with Raspberry Pi	1
Getting started	1
The unboxing	3
Powering your board	5
Hooking up a keyboard, mouse, and display	6
Installing the operating system	8
Accessing the board remotely	18
Establishing Internet access on Raspberry Pi B+	18
Establishing Internet access on Raspberry Pi A+	18
Accessing your Raspberry Pi from your host PC	20
Summary	34
Chapter 2: Programming Raspberry Pi	35
Basic Linux commands on Raspberry Pi	36
Creating, editing, and saving files on Raspberry Pi	41
Creating and running Python programs	43
Basic programming constructs on Raspberry Pi	46
The if statement	46
The while statement	48
Working with functions	49
Libraries/modules in Python	51
Object-oriented code	52
Introduction to the C/C++ programming language	55
Summary	59

Chapter 3: Providing Speech Input and Output	61
Hooking up the hardware to make and input sound	63
Using Espeak to allow our projects to respond in a robotic voice	71
Using PocketSphinx to accept your voice commands	73
Interpreting commands and initiating actions	80
Summary	83
Chapter 4: Adding Vision to Raspberry Pi	85
Connecting the USB camera to Raspberry Pi and viewing the images	86
Connecting the Raspberry Pi camera board and viewing the images	89
Downloading and installing OpenCV – a fully featured vision library	93
Using the vision library to detect colored objects	98
Summary	104
Chapter 5: Creating Mobile Robots on Wheels	105
Gathering the required hardware	105
Using the Raspberry Pi GPIO to control a DC motor	108
Controlling your mobile platform programmatically using Raspberry Pi	111
Controlling the speed of your motors with PWM	114
Adding program arguments to control your platform	117
Making your platform truly mobile by issuing voice commands	119
Summary	122
Chapter 6: Controlling the Movement of a Robot with Legs	123
Gathering the hardware	124
Connecting Raspberry Pi to the mobile platform using a servo controller	127
Connecting the hardware	127
Configuring the software	129
Creating a program in Linux to control the mobile platform	135
Making your mobile platform truly mobile by issuing voice commands	138
Summary	140
Chapter 7: Avoiding Obstacles Using Sensors	141
Connecting Raspberry Pi to an infrared sensor using USB	146
Connecting a sensor using the USB interface	147
Connecting the IR sensor using the GPIO ADC	155

Connecting Raspberry Pi to a USB sonar sensor	162
Connecting the hardware	164
Using a servo to move a single sensor	169
Summary	173
Chapter 8: Going Truly Mobile – The Remote Control of Your Robot	175
<hr/>	
Gathering the hardware	176
Connecting Raspberry Pi to a wireless USB keyboard	183
Using the keyboard to control your project	183
Working remotely with your Raspberry Pi through a wireless LAN	189
Working remotely with your Raspberry Pi through ZigBee	194
Summary	203
Chapter 9: Using a GPS Receiver to Locate Your Robot	205
<hr/>	
Connecting Raspberry Pi to a USB GPS device	207
Accessing the USB GPS programmatically	218
Connecting Raspberry Pi to an RX/TX (UART) GPS device	224
Communicating with the RX/TX GPS programmatically	225
Taking a look at the GPS data	229
Summary	238
Chapter 10: System Dynamics	239
<hr/>	
Creating a general control structure	241
Using the structure of the Robot Operating System to enable complex functionalities	253
Summary	256
Chapter 11: By Land, Sea, and Air	257
<hr/>	
Using Raspberry Pi to sail	258
Getting started	258
Using Raspberry Pi to fly robots	265
Using Raspberry Pi to make the robot swim underwater	275
Summary	276
Index	277

Preface

Robots seem to be everywhere these days. Not only in movies but also in our day-to-day lives. They vacuum floors, play with children, and build automobiles. These new machines are quickly migrating from university and government labs to our homes, offices, schools, and playgrounds. A similar migration has occurred in recent years with computers.

A good part of this migration, as in the case of computers, is that the untrained but interested general population has been able to take part in the development of these new machines. A big reason for this is the introduction of inexpensive hardware and free, open source hardware. Initially, it was Arduino, an inexpensive processor developed for the do-it-yourself crowd, which enabled normal people to create robotics projects with complex functionality. More recently, Raspberry Pi has extended this capability by providing an inexpensive, small, but powerful Linux computer for these projects.

This second edition is designed to allow you to take full advantage of the capabilities of Raspberry Pi in your robotics projects. It will not only teach you how to use the USB port to add additional hardware capabilities, but it will also show you how to connect the hardware to the GPIO as well. It will provide a step-by-step guide to get you well on your way to building your very own amazing robotics projects.

What this book covers

Chapter 1, Getting Started with Raspberry Pi, helps you to power up your Raspberry Pi, connect it to a keyboard, mouse, display, and remote computer, and begin to access all that potential computing power.

Chapter 2, Programming Raspberry Pi, teaches you the basics of how to program the Raspberry Pi, both in Python and in the C programming languages.

Chapter 3, Providing Speech Input and Output, teaches your Raspberry Pi to both speak and listen.

Chapter 4, Adding Vision to Raspberry Pi, shows you how to use standard USB and Raspberry Pi cameras to allow your robotics projects to see.

Chapter 5, Creating Mobile Robots on Wheels, shows you how to connect the Raspberry Pi to a mobile wheeled platform and control its motors, so your robots can be mobile.

Chapter 6, Controlling the Movement of a Robot with Legs, teaches you how to make your robot walk.

Chapter 7, Avoiding Obstacles Using Sensors, shows you how to sense the world around you. Now that your robot is mobile, you'll want to avoid or find objects.

Chapter 8, Going Truly Mobile – The Remote Control of Your Robot, shows you how to control your robot wirelessly – you'll want your robot to move around untethered by cables.

Chapter 9, Using a GPS Receiver to Locate Your Robot, shows you how to use a GPS receiver so that your robot knows where it is – if your robot is mobile, it might get lost.

Chapter 10, System Dynamics, focuses on how to bring it all together to make complex robots since you've got lots of capability.

Chapter 11, By Land, Sea, and Air, shows you how to add capabilities to robots that sail, fly, and even go under the water.

What you need for this book

Here's a partial list of the software you will need for the book:

- 7-Zip: This is a utility to archive and unarchive software
- Image Writer for Windows: This is used to write images to an SD card
- WinSCP: This provides the ability to transfer files to/from a PC
- PuTTY: This allows the user remote access to the Raspberry Pi
- VNC Server/VNC Viewer: This allows the user remote access to the graphical interface of the Raspberry Pi

Who this book is for

This book is for anyone who is keen on using the Raspberry Pi to create robotics projects that have previously been the domain of the research labs of major universities or defense departments. Some programming background is useful, but if you know how to use a personal computer you can, with the aid of the step-by-step instructions in this book, construct complex robotics projects that can move, talk, listen, see, swim, or fly.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Go to the `/home/Raspbian/Desktop` directory."

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in **bold**:

```
a = input("Input value: ")
b = input("Input second value: ")
c = a + b
print c
```

Any command-line input or output is written as follows:

```
cd /home/Raspbian/Desktop
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Go to the **Raspbian** section and select the `.zip` file just to the right of the image identifier."



Warnings or important notes appear in a box like this.



Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book — what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from http://www.packtpub.com/sites/default/files/downloads/01460S_ColoredImages.pdf.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books — maybe a mistake in the text or the code — we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Getting Started with Raspberry Pi

Raspberry Pi, with its low cost and amazing package of functionality, has taken the robotic hobbyist community by Storm. Unfortunately, many, especially those new to embedded systems and programming, can end up so discouraged that the board can end up on the shelf, gathering dust next to the floppy disks and your old CRT.

Getting started

There is nothing as exciting as ordering, and finally receiving a new piece of hardware. Yet things can go south quickly, even in the first few minutes. This chapter will, hopefully, help you avoid the pitfalls that normally accompany unpacking and configuring your Raspberry Pi. We'll walk through the process, answer many of the different questions you might have, and help you understand what is going on. If you don't go through this chapter, you'll not be successful at any of the others, and your hardware will go unused, which would be a real tragedy. So, let's get started.

One of the most challenging aspects of writing this guide is to decide the level at which I should describe each step. Some of you are beginners, some have limited experience, and others know significantly more in some of these areas. I'll try to be brief but thorough, trying to detail the steps to take in order to be successful. So, for this chapter, here are your objectives:

- Unbox and connect the board to power
- Connect a display, keyboard, and mouse
- Load and configure the operating system
- Access the board remotely

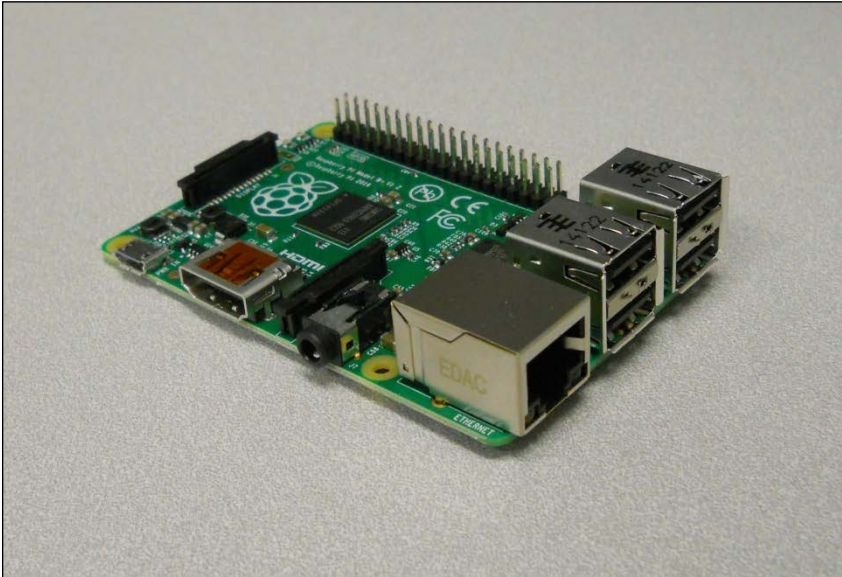
The Raspberry Pi comes in several flavors: the original A and B models, and the new and improved A+ and B+ models. The B+ flavor is the most popular. It comes with additional input/output capability, four USB connections, more memory, and will be the flavor you'll focus on in this book. That does not mean that many, if not most, of the projects here require the extra capability of the B+. As we go, I'll try and point out when you'll need the additional capability of the B+, and when the Raspberry Pi A+ might be enough. The Raspberry Pi now also comes in a B2 version. We'll not talk about that specifically here, but there is no reason why the projects shouldn't work with that version as well.

Here are the items you'll need for this chapter's projects:

- A Raspberry Pi, Model B+
- The USB cable to provide power to the board
- A display with a proper video input
- A keyboard, a mouse, and a powered USB hub
- A microSD card – at least 4 GB capacity
- A microSD card writer
- Another computer that is connected to the Internet
- An Internet connection for the board
- A LAN cable (if you are using the Raspberry Pi A+ you'll need a powered USB hub, a wireless LAN connection, and a Wireless LAN device; you'll learn how to configure this later in the chapter)

The unboxing

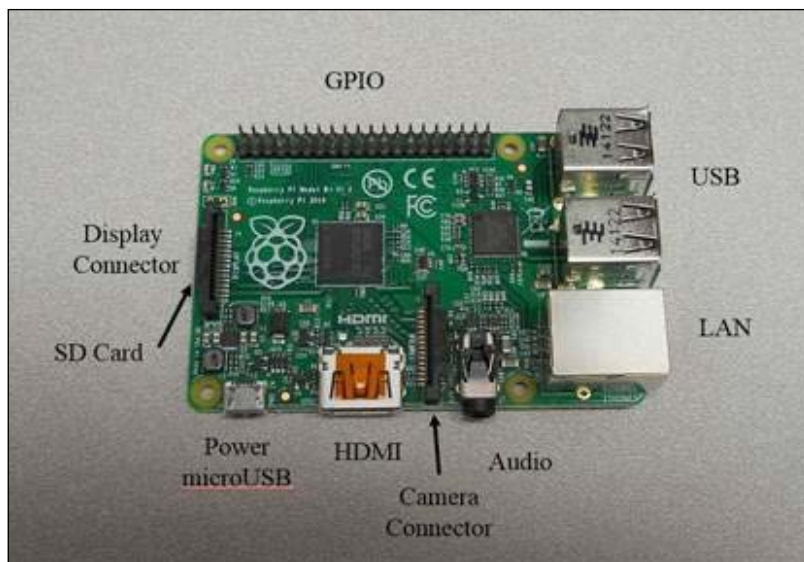
The first step to building any project with Raspberry Pi is to become familiar with Raspberry Pi itself. The Raspberry Pi comes in a box with a power cable. The following image shows what the Raspberry Pi B+ board looks like:



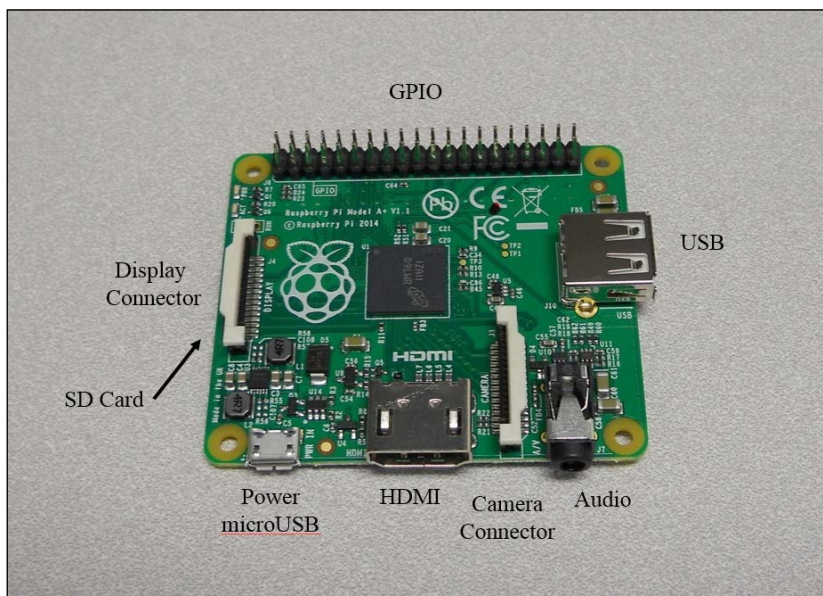
And the next image is that of the Raspberry Pi A+ board:



Before plugging anything in, inspect the board for any issues that might have occurred during shipping. This is normally not a problem, but it is always good to do a quick visual inspection. You should also familiarize yourself with the different connections on the board. Here is the B+ board, labeled for your information:



The labels for the A+ board are shown in the following image:



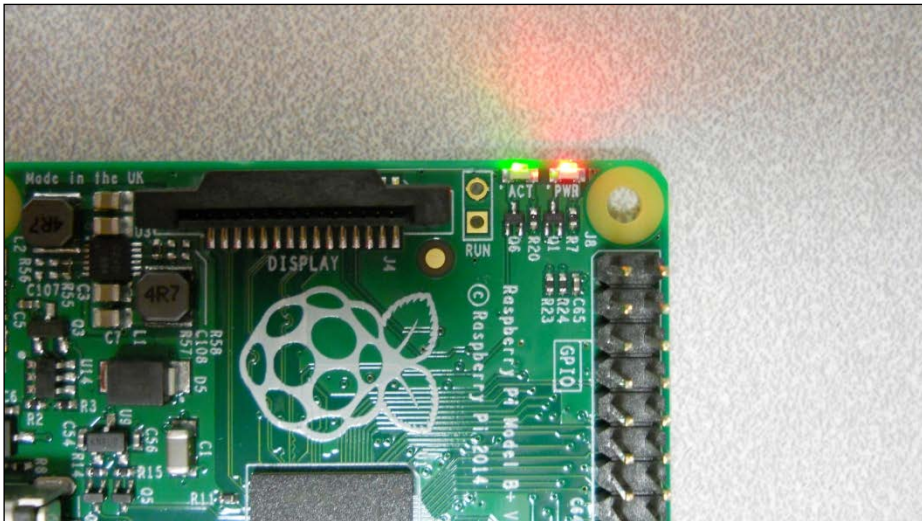
Powering your board

Let's first power the board. To do this, you'll need to go through the USB client connection. This is done by performing the following steps:

1. Connecting the microUSB connector end of the cable to the board.
2. Connecting the standard sized USB connector to either a PC, or a compatible DC power source that has a USB connection. If you are connecting it to a PC, the USB 2.0 standard requires only 500 mA be available. The USB 3.0 standard requires only 900 mA be available. This means that some computers may not supply enough current for the Raspberry Pi to work correctly.

If you are going to use a DC power source at the standard USB connector end, make sure that the unit can supply enough current. You'll need a supply that can provide at least 1000 mA at 5 volts.

When you plug the board in, the **PWR** LED should be red. Here is a close up of the LED locations, just so you're certain which one to look for:



If you've reached this point, congratulations! You're ready for the next step.

Hooking up a keyboard, mouse, and display

Now that you know your board works, you're going to add peripherals so that it can operate as a standalone computer system. This step is optional, as in the future, your projects will often be in systems where you won't connect directly to the board with a keyboard, mouse, and display. However, this can be a great learning step, and is especially useful if you need to do some debugging on the system.

You'll need the following peripherals:

- A USB mouse
- A USB keyboard (this can be wireless, and can contain a built-in mouse pad)
- A display that accepts HDMI or DVI-Video, although using a DVI input will require an adapter
- A powered USB hub (you will need this if you are going to use the Raspberry Pi A+ version for the projects)

You may have most of this stuff already, but if you don't, there are some things to consider before buying additional equipment. Let's start with the keyboard and mouse. Most mice and keyboards have separate USB connectors. This normally works fine for the Raspberry Pi model B+, as it has four USB ports. If you are going to use the Raspberry Pi A+ model, you may want to choose a keyboard that has a mouse pad built-in.

If you are using the Raspberry Pi A+ model, you will want to consider purchasing a powered USB hub. Before deciding on the hub to connect to your board, you need to understand the difference between a powered USB hub and one that gets its power from the USB port itself. Almost all USB hubs are not powered, that is, you don't plug in the USB hub separately. The reason for this is that almost all of these hubs are hooked up to computers with very large power supplies and powering USB devices from the computer is not a problem. This is not the case for your board. The USB port on your board has very limited power capabilities, so if you are going to hook up devices that require significant power—a WAN adapter or webcam for instance—you're going to need a powered USB hub, one that provides power to the devices through a separate power source. Here is an image of such a device, available at www.amazon.com and other online retailers:



Notice that on this hub, there are two connections. The one to the far right is a power connection, and it will be plugged into a battery with a USB port. The connection to the left is the USB connection, which will be plugged into the Raspberry Pi.

Now, you'll also need a display. Fortunately, your Raspberry Pi offers lots of choices here. There are a number of different video standards; here is an image of some of the most prevalent ones, for reference:



There is an HDMI connector on the Raspberry Pi A+ and B+. To use this connector, simply connect your cable with regular HDMI connections to Raspberry Pi and your TV or monitor that has an HDMI input connector. HDMI monitors are relatively new, but if you have a monitor that has a DVI input, you can buy relatively inexpensive adapters that provide an interface between DVI and HDMI.

Don't be fooled by adapters that claim that they go from HDMI or DVI to VGA, or HDMI or DVI to S-video. These are two different kinds of signals: HDMI and DVI are digital standards, and VGA and S-video are analog standards. There are adapters that can do this, but they must contain circuitry and require power, and are significantly more expensive than any simple adapter.

You are almost ready to plug in the Raspberry Pi. Make sure you connect all your devices before you power on the unit. Most operating systems support the hot-swap of devices, which means you are able to connect a device after the system has been powered, but this is a bit shaky. You should always cycle power when you connect new hardware.

Even though your hardware configuration is complete, you'll still need to complete the next section to power on the device. So, let's figure out how to install an operating system.

Installing the operating system

Now that your hardware is ready, you need to install an operating system. You are going to install Linux, an open-source version of Unix, on your Raspberry Pi. Now Linux, unlike Windows, Android, or iOS, is not tightly controlled by a single company. It is a group effort, mostly open-source, and while it is available for free, it grows and develops a bit more chaotically.

Thus, a number of distributions have emerged, each built on a core set of similar capabilities referred to as the Linux kernel. These core capabilities are all based on the Linux specification. However, they are packaged slightly differently, and developed, supported, and packaged by different organizations. Debian, Arch, and Fedora are the names of some of the versions. There are others as well, but these are the main choices for the distribution that you might put on your card.

I choose to use the Raspbian, a Debian distribution of Linux, on my Raspberry Pi projects for a couple of reasons. First, the Debian distribution is used as the basis for another distribution, Ubuntu, and Ubuntu is arguably the most popular distribution of Linux, which makes it a good choice because of the community support that it offers. Also, I personally use Ubuntu when I need to run Linux on my personal computer. It provides a complete set of features, is well organized, and generally supports the latest sets of hardware and software. Having roughly the same version on both my personal computer and my Raspberry Pi makes it easier for me to use both as they operate, at least to a certain degree, the same way. That lets me try some things on my computer before trying them on the Raspberry Pi. I've also found that Ubuntu/Debian has excellent support for new hardware, and this can be very important for your projects.

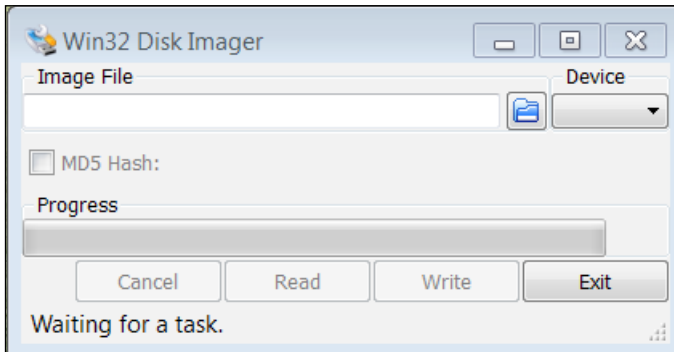
So, you are going to install and run Raspbian, on your Raspberry Pi.

There are two approaches to getting Raspbian on your board. The board is getting popular enough for you to buy an SD card that already has Raspbian installed, or you can download it onto your personal computer and then install it on the card. I'll assume you don't need any directions if you want to purchase a card – simply do an Internet search for companies selling such a product.

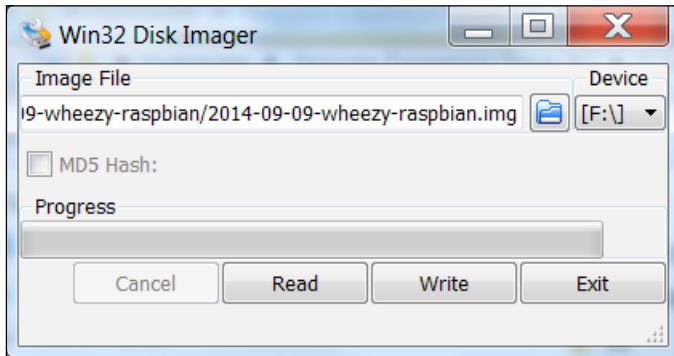
If you are going to download a distribution, you need to decide if you are going to use a Windows computer to download and create an SD card, a MAC OS X, or a Linux machine. I'll give brief directions for the Windows computer and Linux machines here. For directions on the MAC OS X, go to: <http://www.raspberrypi.org/documentation/installation/installing-images/mac.md>.

First, you'll need to download an image. This part of the process is similar for both Windows and Linux. Open a browser window. Go to Raspberry Pi organization's website, www.raspberrypi.org and select the **Downloads** selection at the top of the page. This will give you a variety of download choices. Go to the **Raspbian** section, and select the .zip file just to the right of the image identifier. This will download an archived file that has the image for your Raspbian operating system. Note the default user name and password; you'll need that later.

If you're using Windows, you'll need to unzip the file using an archiving program like 7-Zip. This will leave you with a file that has the `.img` extension, a file that can be imaged on to your card. Next, you'll need a program that can write the image to the card. I use the Image Writer for Windows program. You can find a link to this program at the top of the download section on the www.raspberrypi.org website. Plug your card into the PC, run this program, and you should see this:

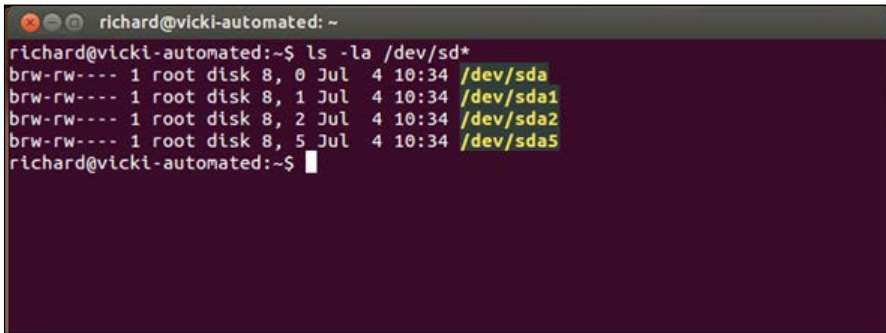


Select the correct card and image; it should look something like this:



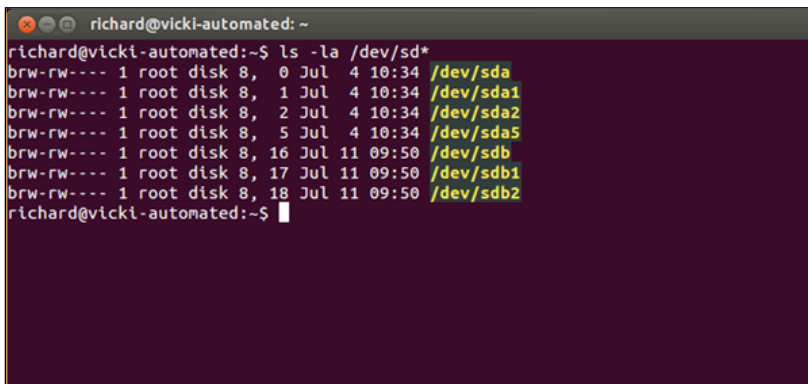
Then click on the **Write** button. This will take some time, but when it is complete, eject the card from the PC.

If you are using Linux, you'll need to un-archive the file and then write it to the card. You can do all of this with one command. However, you do need to find the `/dev` device label for your card. You can do this with the `ls -la /dev/sd*` command. If you run this before you plug in your card, you might see something like the following screenshot:



```
richard@vicki-automated: ~  
richard@vicki-automated:~$ ls -la /dev/sd*  
brw-rw---- 1 root disk 8, 0 Jul  4 10:34 /dev/sda  
brw-rw---- 1 root disk 8, 1 Jul  4 10:34 /dev/sda1  
brw-rw---- 1 root disk 8, 2 Jul  4 10:34 /dev/sda2  
brw-rw---- 1 root disk 8, 5 Jul  4 10:34 /dev/sda5  
richard@vicki-automated:~$
```

After plugging in your card, you might see something like the following screenshot:



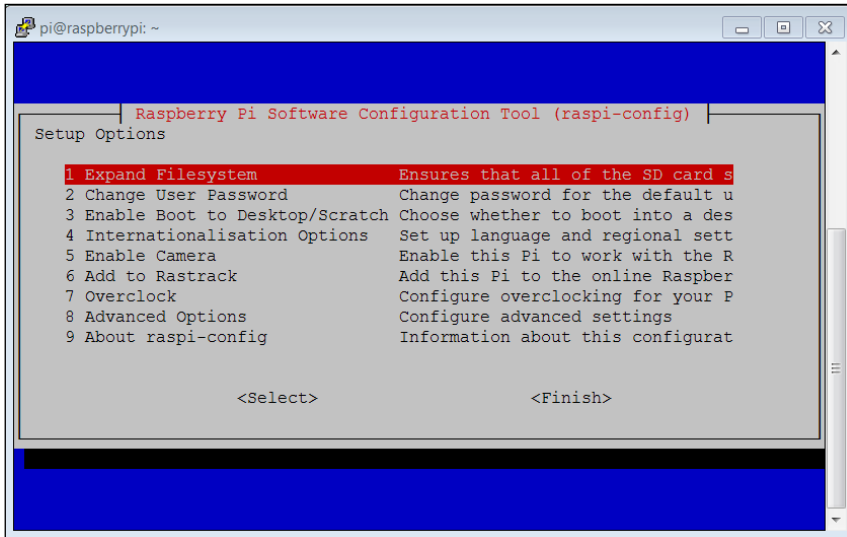
```
richard@vicki-automated: ~  
richard@vicki-automated:~$ ls -la /dev/sd*  
brw-rw---- 1 root disk 8, 0 Jul  4 10:34 /dev/sda  
brw-rw---- 1 root disk 8, 1 Jul  4 10:34 /dev/sda1  
brw-rw---- 1 root disk 8, 2 Jul  4 10:34 /dev/sda2  
brw-rw---- 1 root disk 8, 5 Jul  4 10:34 /dev/sda5  
brw-rw---- 1 root disk 8, 16 Jul 11 09:50 /dev/sdb  
brw-rw---- 1 root disk 8, 17 Jul 11 09:50 /dev/sdb1  
brw-rw---- 1 root disk 8, 18 Jul 11 09:50 /dev/sdb2  
richard@vicki-automated:~$
```

Note that your card is at `sdb`. Now, go to the directory where you downloaded the archived image file, and use the following command:

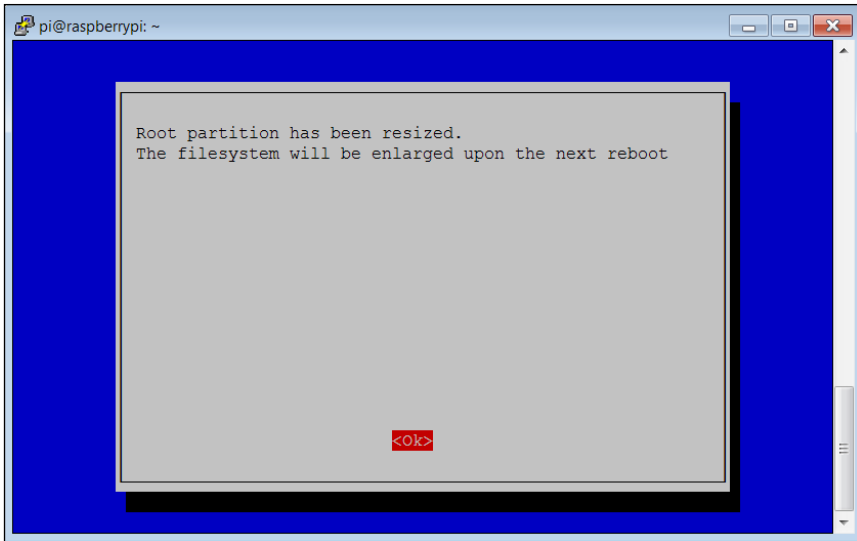
```
sudo dd if=2014-09-09-wheezy-raspbian.img of=/dev/sdX
```

The `2014-09-09-wheezy-raspbian.img` command will be replaced by the image file that you downloaded, and `/dev/sdX` will be replaced by your card ID in this example, `/dev/sdb`. Be careful to specify the correct device as this can overwrite the data on your hard drive. Also, this may take a few minutes. Once the file is written, eject the card and you are ready to plug it into the board and apply power.

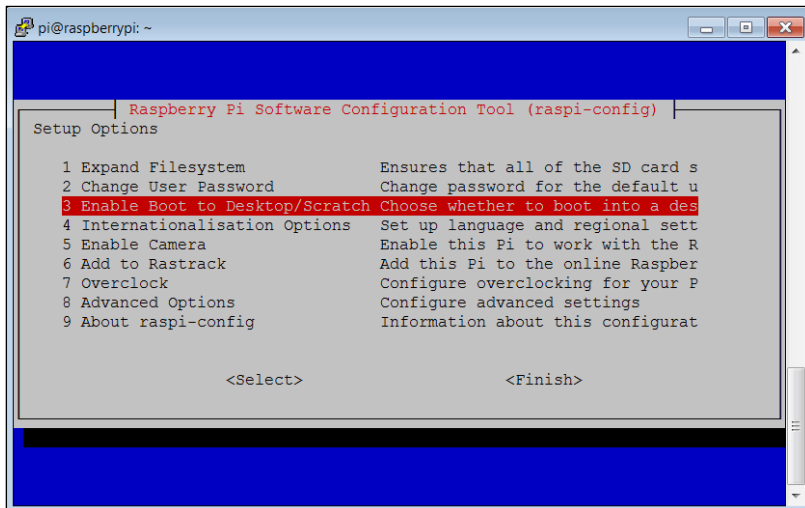
Make sure your Raspberry Pi is unplugged and install the SD card into the slot. Then power the device. After the device boots, you should get the following screenshot:



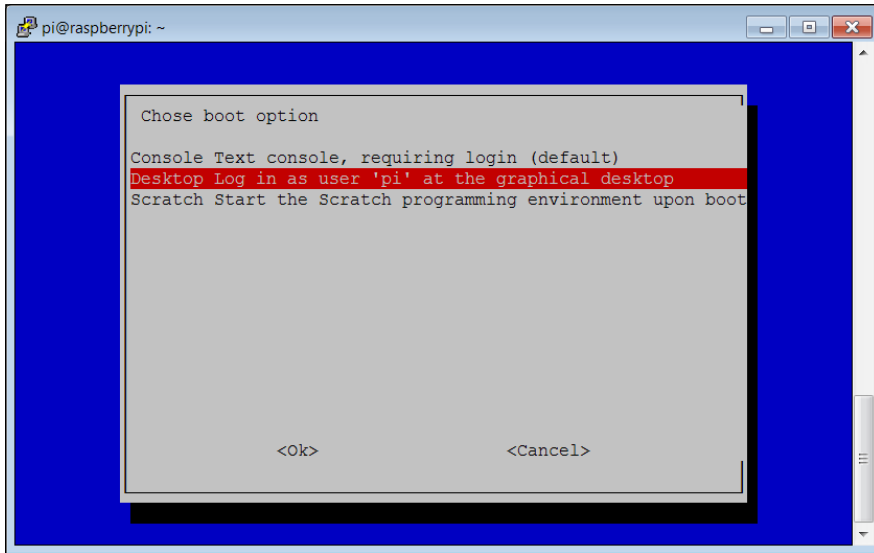
You are going to do two things, and perhaps another, based on your personal preference. First, you'll want to expand the file system to take up the entire card. So, hit the *Enter* key, and you'll see the following screenshot:



Hit *Enter* once again and you'll go back to the main configuration screen. Now select the **Enable Boot to Desktop/Scratch** option, as shown in the following screenshot:



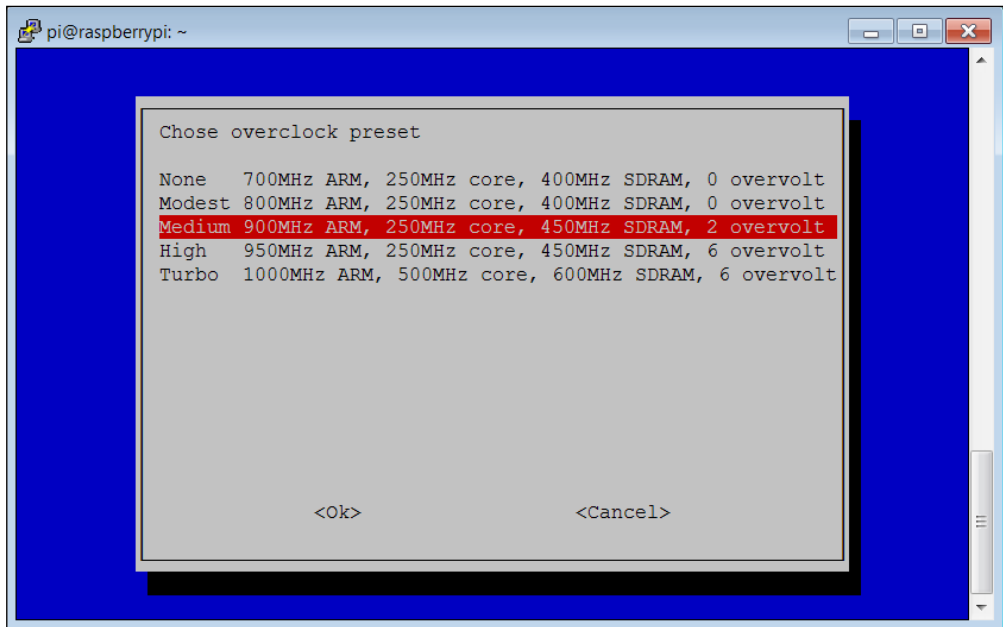
When you hit *Enter* you'll see the following screenshot:



I prefer to select the middle option, **Desktop Log in as user 'pi' at the graphical desktop**. It normally sets up the system the way I like to have it booted up. You could also choose **Console text console, require login (default)**. However, you will need to log in whenever you want to access the graphical environment, for example, when using the VNC server, which we will cover later.

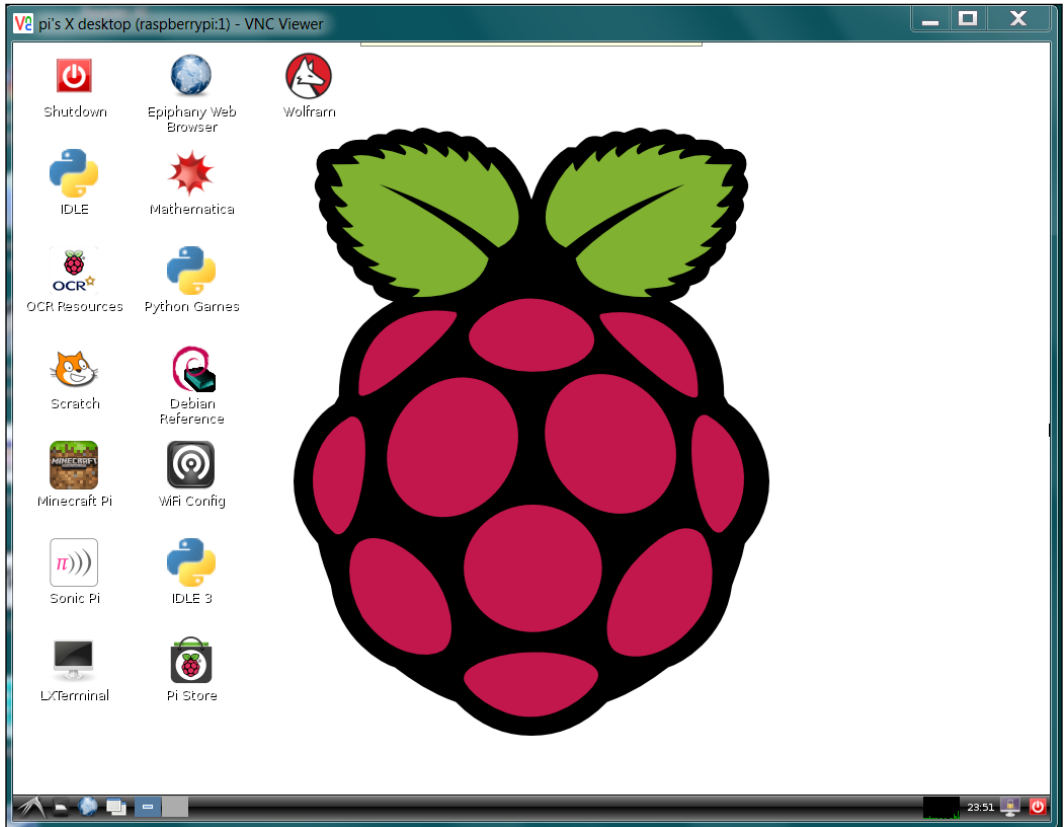
The final choice you can make is to change the over clocking on the Raspberry Pi. This is a way for you to get a higher performance from your system. However, there is a risk that you can end up with a system that has reliability problems.

I normally do a bit of over clocking, I'll select the **Medium** setting, as shown in the following screenshot:

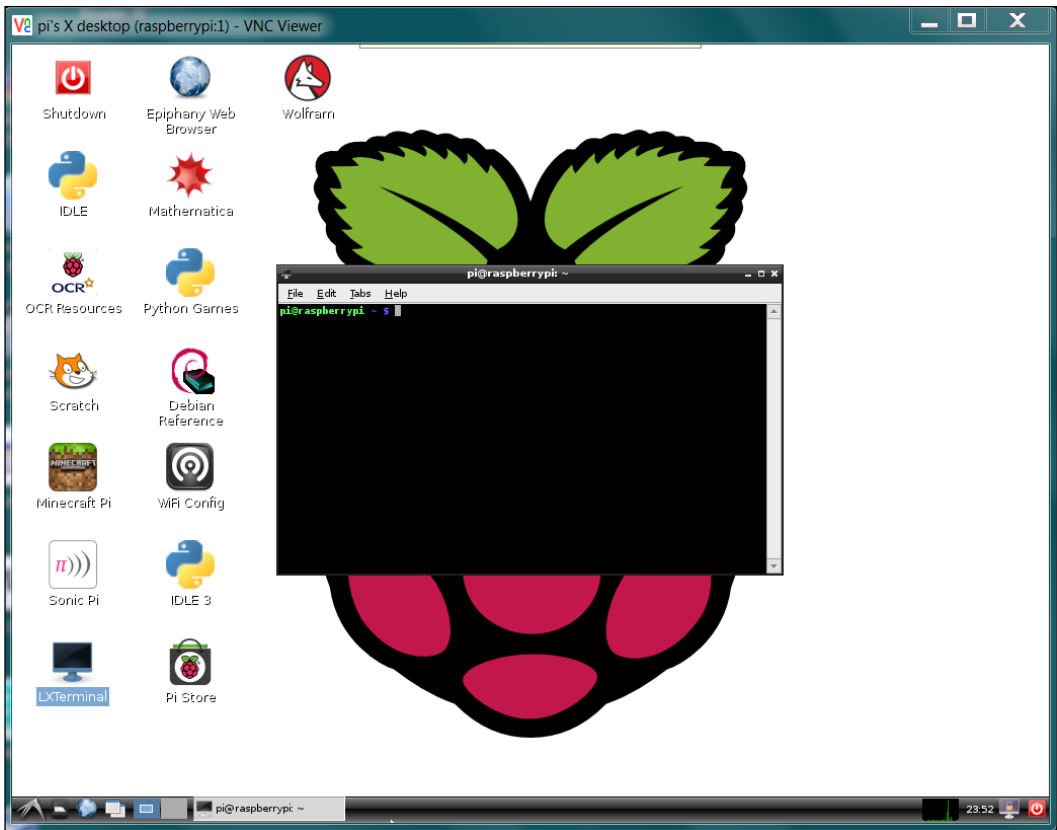


Once you are done, and are back to the main configuration menu, hit the *Tab* key until you are positioned over the **<Finish>** selection, then hit *Enter*. Then, hit *Enter* again so that you can reboot your Raspberry Pi. Now when you boot, your system will take you all the way into the Windows screen.

The Raspberry Pi uses the **Lightweight X11 Desktop Environment (LXDE)** Windows system, and should look like the following screenshot:



Now, when the graphical desktop system is up and running, you can bring up a terminal by double clicking on the **LXTerminal** icon on the screen. You should end up with a terminal window that looks like the following screenshot:



Now you are ready to start interacting with the system!

Two questions arise: do you need an external computer during the creation of your projects? and, what sort of a computer do you need? The answer to the first question is a resounding 'yes'. Most of your projects are going to be self-contained robots with very limited connections and display space; you will be using an external computer to issue commands and see what is going on inside your robotic projects. The answer to the second question is a bit more difficult. Because your Raspberry Pi is working in Linux, most notably a version of Debian that is very closely related to Ubuntu, there are some advantages to having an Ubuntu system available as your remote system. You can then try some things on your computer before trying them in your embedded system. You'll also be working with similar commands for both, which will help your learning curve. You can even use a second Raspberry Pi for this purpose.

However, the bulk of personal computers today run some sort of Windows operating system, so that is what will normally be available. Even with a Windows machine, you can issue commands and display information, so either way, it will work. I'll try to give examples for both, as long as it is practical.

Accessing the board remotely

You now have a very usable Debian computer system. You can use it to access the Internet, write riveting novels, balance your accounts—just about anything you could do with a standard personal computer. However, that is not your purpose; you want to use your embedded system to power your delightfully inventive projects. In most cases, you will not want to connect a keyboard, mouse, and display to your projects. However, you still need to communicate with your device, program it, and have it tell you what is going on, and when things don't work right. You'll spend some time in this section establishing remote access to your device.

Establishing Internet access on Raspberry Pi B+

This section depends on an Internet connection with the Raspberry Pi. For the B+, this is relatively easy. Simply connect the device to the LAN through the Ethernet connection. If you don't have a wired LAN connection, or would prefer to connect your device wirelessly, follow the instructions in the next section, *Establishing Internet access on the Raspberry Pi A+*.

Establishing Internet access on Raspberry Pi A+

The Raspberry Pi A+ does not have a LAN connection. To connect the Raspberry Pi A+ to the Internet, you'll need to establish a wireless LAN connection. First, make sure you have a wireless access point configured. You'll also need a wireless device. See http://elinux.org/RPi_USB_Wi-Fi_Adapters to identify wireless devices that have been verified to work with the Raspberry Pi. Here is one that is available at many online electronics outlets:



You'll also need to connect a powered USB hub for this process so that you can access the wireless keyboard, as well as the USB wireless LAN device. Now, connect the hub to the Raspberry Pi, and then connect your keyboard and the device to the powered hub.

Boot the system, and then edit the network file by typing `sudo nano /etc/network/interfaces`. You'll want to change it to look like this:

```
pi@raspberrypi: ~  
GNU nano 2.2.6      File: /etc/network/interfaces  
  
auto lo  
  
iface lo inet loopback  
iface eth0 inet dhcp  
  
allow-hotplug wlan0  
iface wlan0 inet dhcp  
    wpa-ssid "walkPi"  
    wpa-psk "12345678"  
  
[ Read 9 lines (Warning: No write permission) ]  
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos  
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text ^T To Spell
```

The `wpa-ssid` and `wpa-psk` values here must, of course, match what your wireless access point requires. Reboot, and your device should be connected to your wireless network.



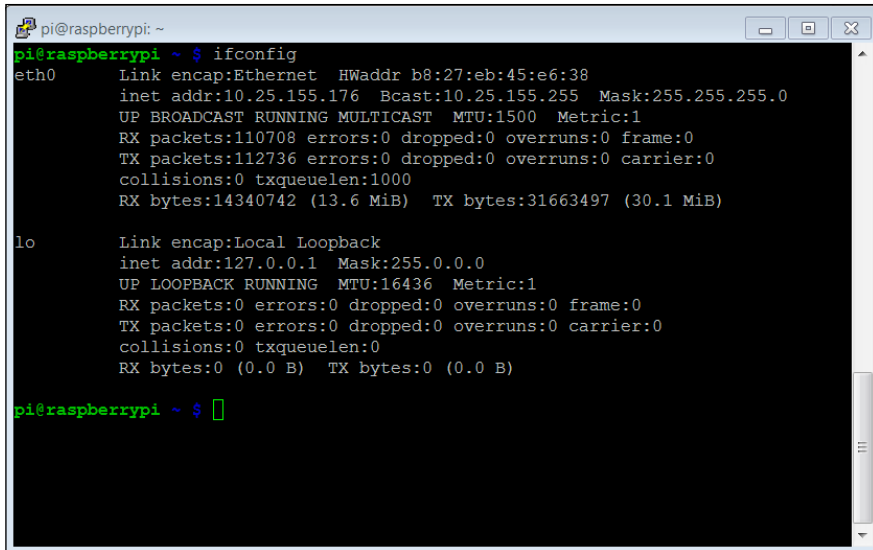
If you are using a US keyboard, you may need to edit the keyboard file for your keyboard to use nano effectively. To do this, type `sudo nano /etc/default/keyboard` and change the `XKBLAYOUT="gb"` to `XKBLAYOUT="us"`.

Accessing your Raspberry Pi from your host PC

Once you have established an Internet network connection with your device, you can access it from your host computer. There are three ways you can access your system from your remote computer:

- The first is through a terminal interface called SSH.
- The second way is by using a program called VNC Server. This allows you to open a graphical user interface remotely that mirrors the graphical user interface on the Raspberry Pi.
- Finally, you can transfer files through a program called WinSCP, which is custom made for this purpose. From Linux, you can use a program called `scp`.

So, first, make sure your basic system is up and working. Open a terminal window, and check the IP address of your unit. You're going to need this no matter how you want to communicate with the system. Do this by using the `ifconfig` command. It should look like the following screenshot:



```

pi@raspberrypi ~ $ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:45:e6:38
          inet addr:10.25.155.176  Bcast:10.25.155.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:110708 errors:0 dropped:0 overruns:0 frame:0
          TX packets:112736 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:14340742 (13.6 MiB)  TX bytes:31663497 (30.1 MiB)

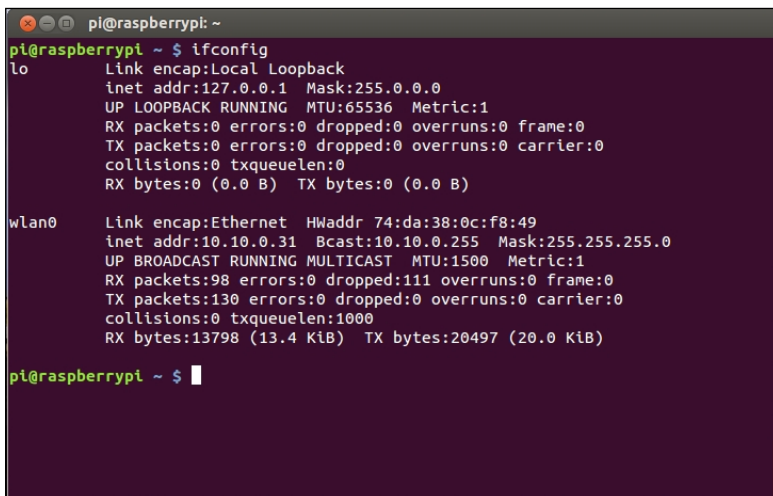
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

pi@raspberrypi ~ $

```

You'll need the `inet addr` shown in the second line of the preceding screenshot to contact your board through the Ethernet.

If you are using the wireless device to gain access to the Internet, your `ifconfig` will look like this:



```

pi@raspberrypi ~ $ ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr 74:da:38:0c:f8:49
          inet addr:10.10.0.31  Bcast:10.10.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:98 errors:0 dropped:111 overruns:0 frame:0
          TX packets:130 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:13798 (13.4 KiB)  TX bytes:20497 (20.0 KiB)

pi@raspberrypi ~ $

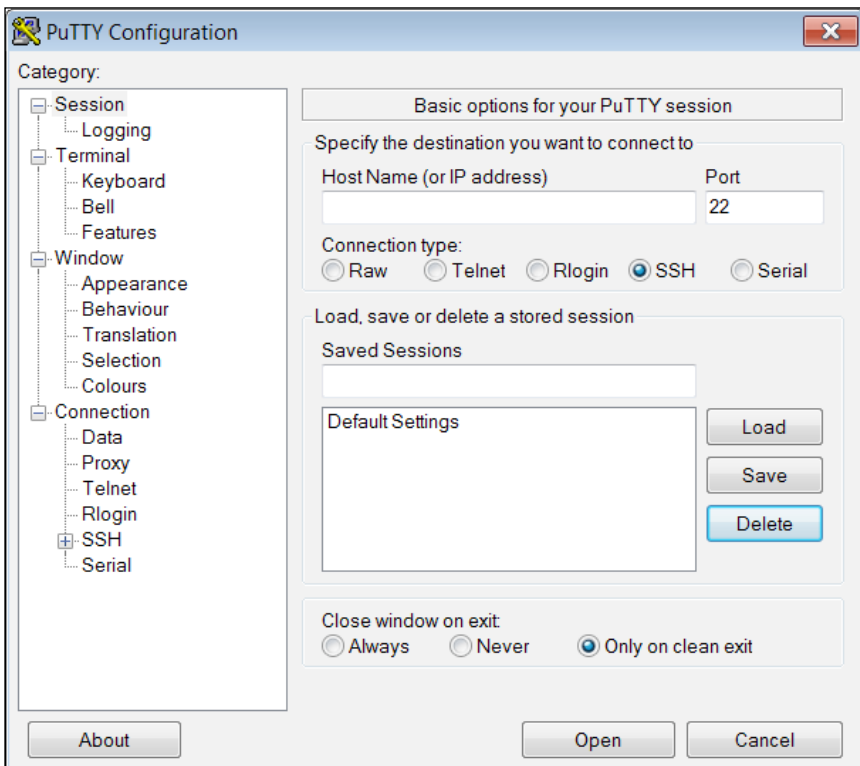
```

The `inet addr` associated with the `wlan0` connection, in this case `10.10.0.31`, is the address you will use to access your Raspberry Pi.

You'll also need an SSH terminal program running on your remote computer. An SSH terminal is a **Secure Shell Hypterminal (SSH)** connection, which simply means you'll be able to access your board and give it commands by typing them into your remote computer. The response from the Pi will appear in the remote computer terminal window. If you'd like to know more about SSH, try <https://www.siteground.com/tutorials/ssh/>.

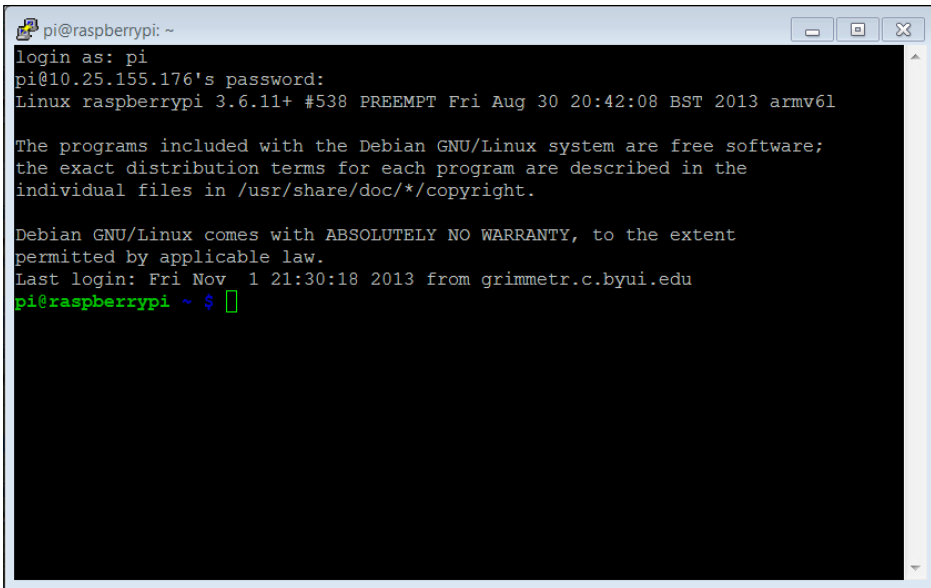
If you are running Microsoft Windows, you can download such an application. My personal favorite is PuTTY. It is free and does a very good job of allowing you to save your configuration so you don't have to type it in each time. Type `putty` in a search window, and you'll soon come to a page that supports a download, or you can go to www.putty.org.

Download PuTTY to your Microsoft Windows machine. Then, run `putty.exe`. You should see a configuration window which looks something like the following screenshot:



Type the `inet addr` from the previous page in the **Host Name** space, and make sure the SSH selection is selected. You may want to save this configuration under Raspberry Pi so you can reload it each time.

When you click on **Open**, the system will try to open a terminal window onto your Raspberry Pi through the LAN connection. The first time you do this, you will get a warning about an RSA key as the two computers don't know about each other. Windows complains that a computer it doesn't know is about to be connected in a fairly intimate way. Simply click on **OK**, and you should get a terminal with a login prompt, like the following screenshot:



```
pi@raspberrypi: ~
login as: pi
pi@10.25.155.176's password:
Linux raspberrypi 3.6.11+ #538 PREEMPT Fri Aug 30 20:42:08 BST 2013 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

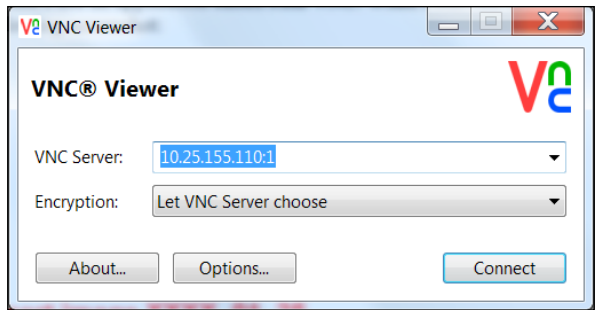
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Nov 1 21:30:18 2013 from grimmetr.c.byui.edu
pi@raspberrypi ~ $
```

Now you can log in and issue commands to your Raspberry Pi. If you'd like to do this from a Linux machine, the process is even simpler. Bring up a terminal window and then type `ssh pi@xxx.xxx.xxx.xxx` where the `xxx.xxx.xxx.xxx` is the `inet addr` of your device. This will then bring you to the login screen of your Raspberry Pi, which should look similar to the preceding screenshot.

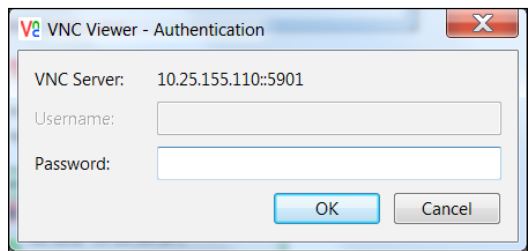
SSH is a really useful tool to communicate with your Raspberry Pi. However, sometimes you need a graphical look at your system, and you don't necessarily want to connect a display. You can get this on your remote computer using an application called `vncserver`. You'll need to install a version of this on your Raspberry Pi by typing `sudo apt-get install tightvncserver` in a terminal window on your Raspberry Pi. This is a perfect opportunity to use SSH, by the way.

Tightvncserver is an application that will allow you to remotely view your complete Windows system. Once you have it installed, you'll need to start the server by typing `vncserver` in a terminal window on the Raspberry Pi. You will then be prompted for a password, to verify the password, and then asked if you'd like to have a view only password. Remember the password you entered, you'll need it to remotely log in via a VNC Viewer.

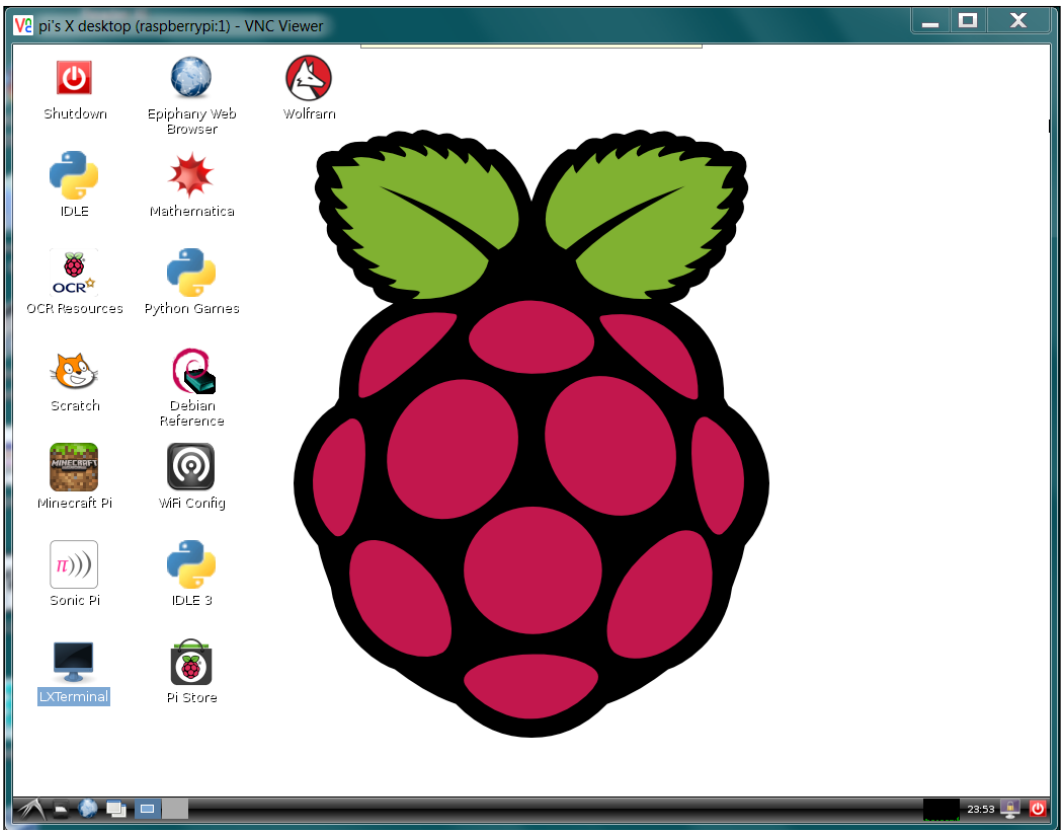
You'll need a **VNC Viewer** application for your remote computer. On my Windows system, I use an application called **RealVNC**. When I start the application, it gives me the following screenshot:



Enter the VNC Server address, which is the IP address of your Raspberry Pi, and click on **Connect**. You will get this pop-up window, as shown in the following screenshot:



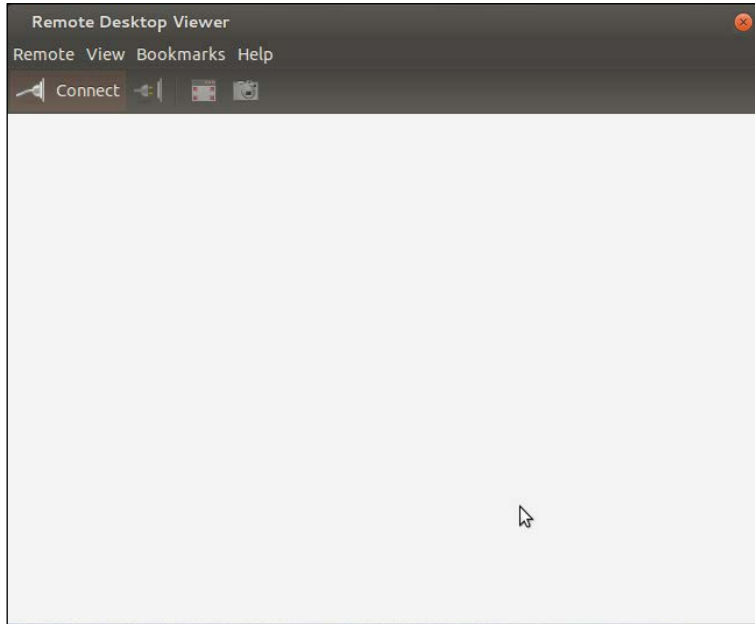
Type in the password you just entered while starting the `vncserver`, and you should then get a graphical view of your Raspberry Pi that looks like the following screenshot:



You can now access all the capabilities of your system, although they may be slower if you are doing graphics-intensive data transfer. Just a note: there are ways to make your `vncserver` start automatically on boot. I have not used them; I choose to type the `vncserver` command from an SSH application when I want the application running. This keeps your running applications to a minimum and, more importantly, provides for fewer security risks. If you'd like to start yours each time you boot, there are several places on the Internet that will show you how to configure this. Try the following website: <http://www.havetheknowhow.com/Configure-the-server/Run-VNC-on-boot.html>.

To view this Raspberry Pi desktop from a remote Linux computer, running Ubuntu for example, you can type `sudo apt-get install xtightvncviewer` and then start it using `xtightvncviewer 10.25.155.110:1` and supplying the chosen password.

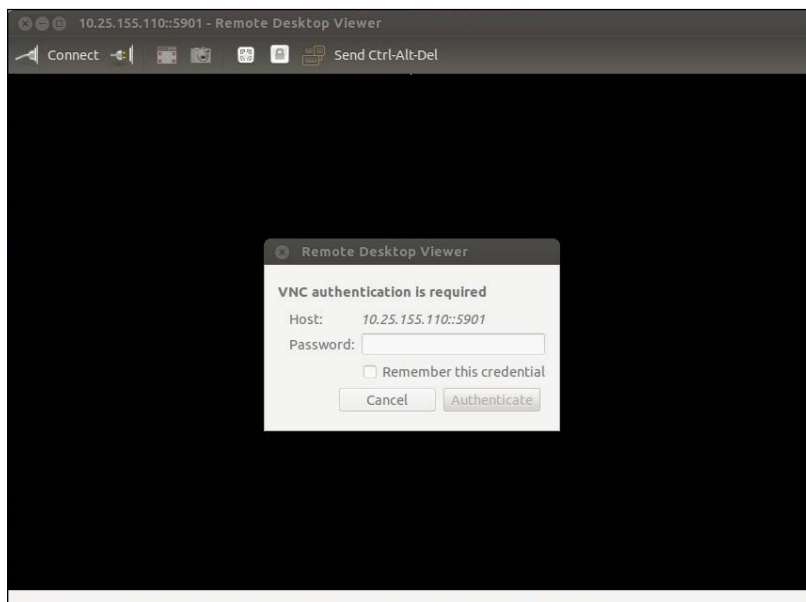
Linux has viewers with graphical interfaces such as **Remmina Remote Desktop Client** (select **VNC-Virtual Network Computing** protocol), which might be used instead of `xtightvncviewer`. Here is a screenshot of the **Remote Desktop Viewer**:



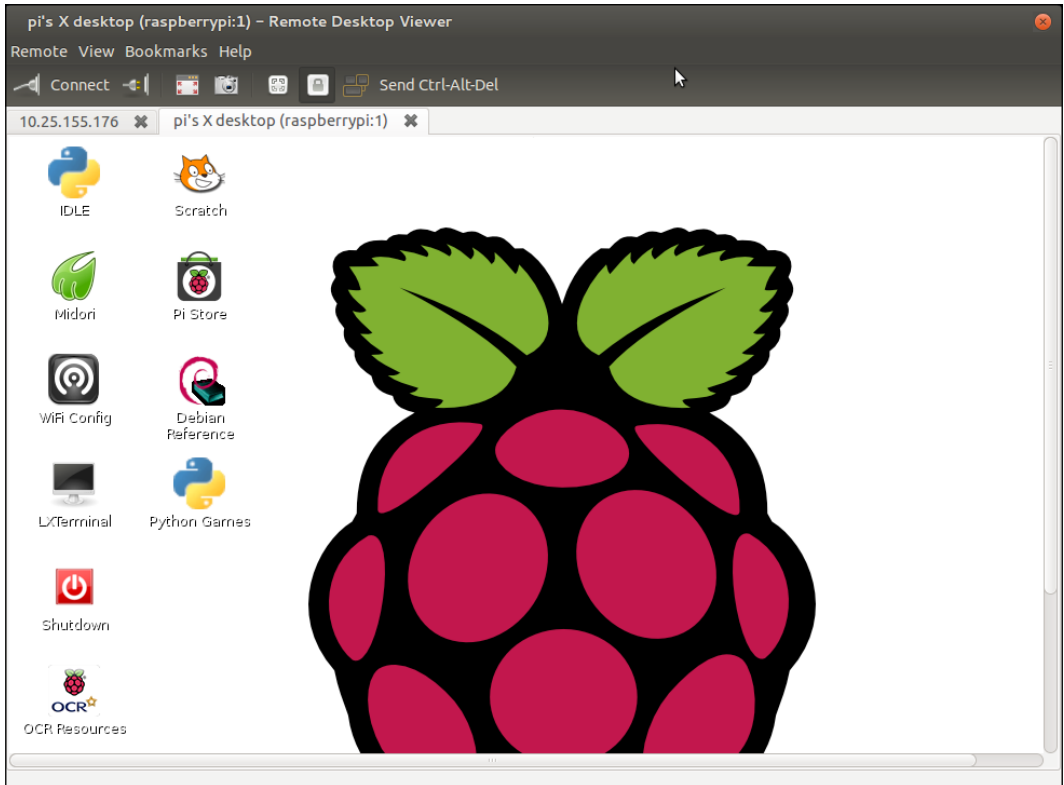
Make sure `vncserver` is running on the Raspberry Pi. The easiest way to do this is to log in using SSH and run `vncserver` at the prompt. Now, click on **Connect** on the **Remote Desktop Viewer**. Fill in the screen as follows; under the **Protocol** selection, choose **VNC**, and you should see the following screenshot:



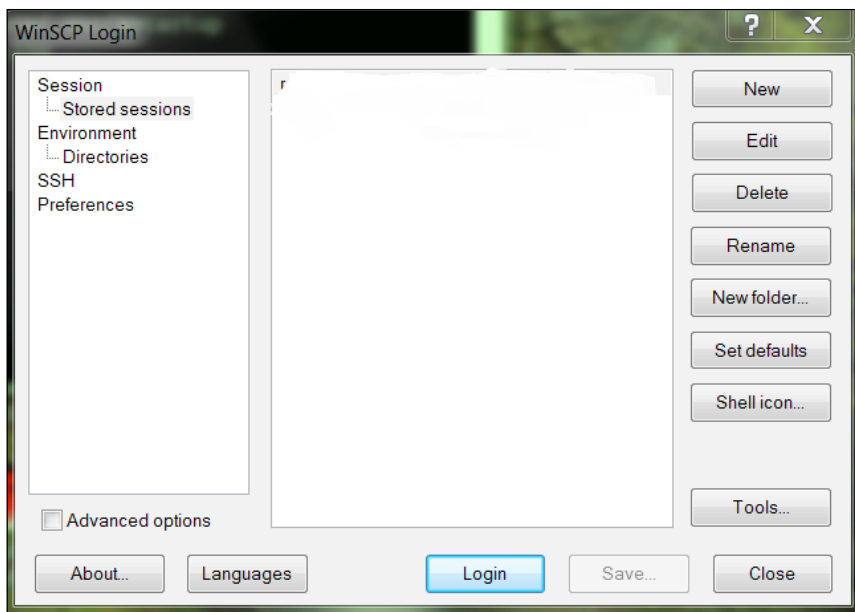
Now enter the **Host** inet address, making sure you include a : 1 at the end, and then click on **Connect**. You'll need to enter the vncserver password you set up, like the following screenshot:



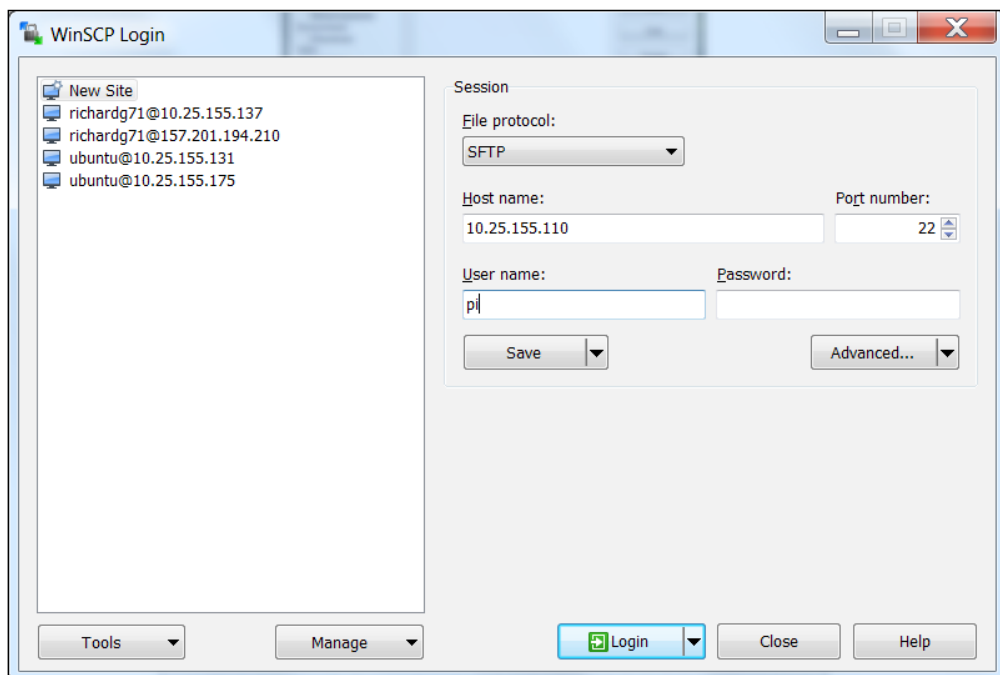
Now you should be able to see the graphical screen of the Raspberry Pi, like this:



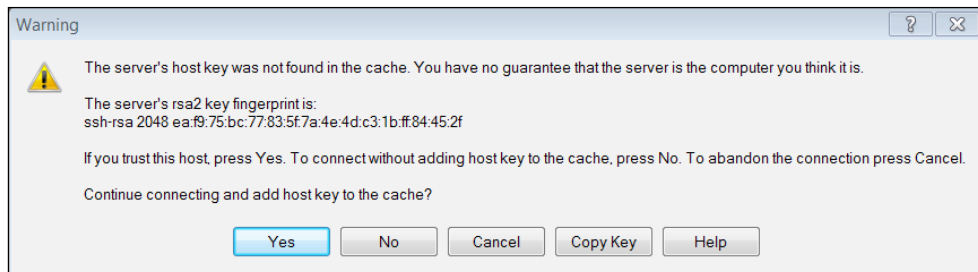
The final piece of software that I like to use with my Windows system is a free application called WinSCP. To download and install this piece of software, simply search the web for WinSCP and follow the instructions. Once installed, run the program. It will open the following dialog box:



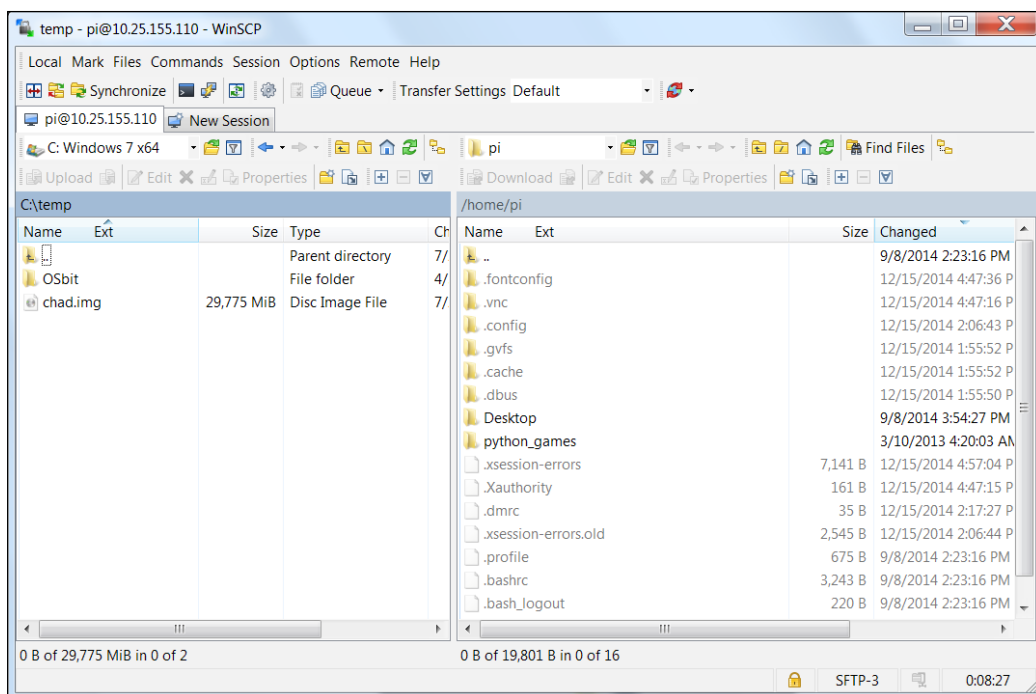
Click on **New**, and you will get the following screenshot:



Here you fill in the IP address in the **host name** tab, pi in the **user name** tab, and the password (not the vncserver password) in the **password** space. Click on **Login** and you should see the following warning displayed, as shown in the following screenshot:

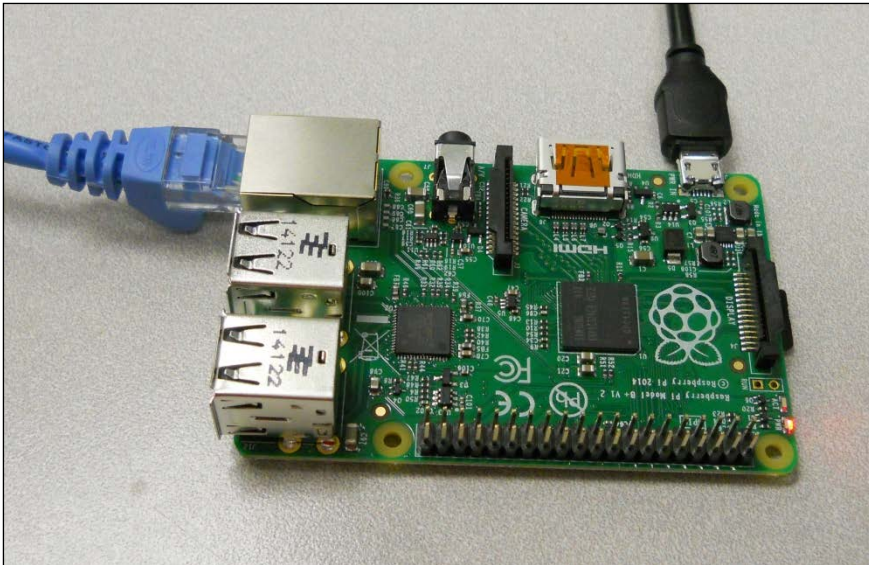


The host computer, again, doesn't know the remote computer. Click on **Yes**, and then the application will display the following screenshot:



Now you can drag and drop files from one system to the other. You can also do similar things on Linux using the command line. To transfer a file to the remote Raspberry Pi, you can use the `scp file user@host.domain:path` command, where `file` is the file name, and `user@host.domain:path` is the location you want to copy it to. For example, if you wanted to copy `robot.py` from your Linux system to the Raspberry Pi, you would type `scp robot.py pi@10.25.155.176:/home/pi/`. The system will ask you for the remote password; this is the login for the Raspberry Pi. Enter the password, and the file will be transferred.

Now that you know how to use SSH, `tightvncserver`, and `scp`, you can access your Raspberry Pi remotely without having a display, keyboard, or mouse connected to it! Now your system looks like the following image (if you are using the Raspberry Pi B+):

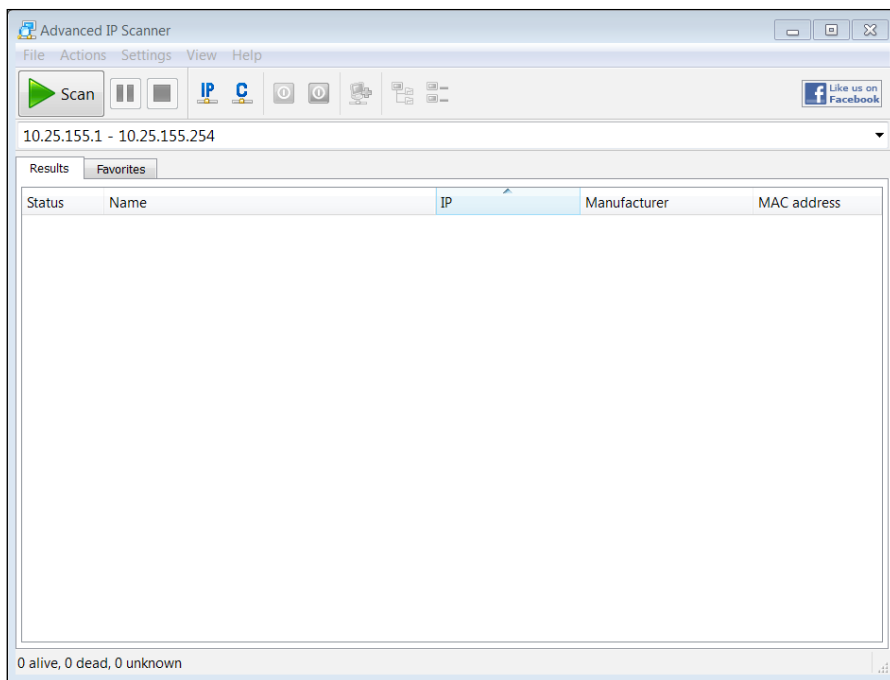


Or this (if you are using the Raspberry Pi A+):



You only need to connect the power and the LAN, either with a cable, or through wireless LAN. If you need to issue simple commands, connect through SSH. If you need a more complete set of graphical functionality, you can access this through vncserver. Finally, if you want to transfer files back and forth, you can use WinSCP from a Windows computer or scp from a Linux computer. Now you have the toolkit to build your first capabilities.

One of the challenges of accessing the system remotely is that you need to know the IP address of your board. If you have the board connected to a keyboard and display, you can always just run `ifconfig` command to get this info. But you're going to use the board in applications where you don't have this information. There is a way to discover this by using an IP scanner application. There are several available for free; on Windows, I use an application called **Advanced IP Scanner**. When I start the program, it looks like the following screenshot:



Clicking on the **Scan** selector, scans for all the devices connected to the network. You can also do this in Linux; one application for IP scanning in Linux is called Nmap. To install Nmap, type `sudo apt-get install nmap`. To run Nmap, type `sudo nmap -sp 10.25.155.1/154` and the scanner will scan the addresses from 10.25.155.1 to 10.25.155.154. For more information on Nmap, see: <http://www.linux.com/learn/tutorials/290879-beginners-guide-to-nmap>. These scanners can let you know which addresses are being used, and this should then let you find your Raspberry Pi address without typing `ipconfig`.

Your system has lots of capabilities. Feel free to play with the system — try to get an understanding of what is already there, and what you'll want to add from a SW perspective. One advanced possibility is to connect the Raspberry Pi through a wireless LAN connection, so you don't have to connect a LAN connection when you want to communicate with it. There are several good tutorials on the Internet. Try <http://learn.adafruit.com/adafruits-raspberry-pi-lesson-3-network-setup/setting-up-wifi-with-occidentalis> or <http://www.howtogeek.com/167425/how-to-setup-wi-fi-on-your-raspberry-pi-via-the-command-line/>.

Remember, there is limited power on your USB port, so make sure that you are familiar with the power needs of accessories plugged into your Raspberry Pi. You may very well need to use a powered USB hub for many projects.

Summary

Congratulations! You've completed the first stage of your journey. You have your Raspberry Pi up and working. No gathering dust in the bin for this piece of hardware. It is now ready to start connecting to all sorts of interesting devices, in all sorts of interesting ways. You would have, by now, installed a Debian operating system, learned how to connect all the appropriate peripherals, and even mastered how to access the system remotely so that the only connections you need are a power supply cable and a LAN cable.

Now you are ready to start commanding your Raspberry Pi to do something. The next chapter will introduce you to the Linux operating system and the Emacs text editor. It will also show you some basic programming concepts in both the Python and C programming languages. Then you'll be ready to add open source software to inexpensive hardware and start building your robotics projects.

2

Programming Raspberry Pi

Now that your system is up and running, you'll want Raspberry Pi to start working. Almost always, this requires you to either create your own programs, or edit someone else's programs. This chapter will provide a brief introduction to file editing and programming.

While it is fun to build hardware (and you'll spend a good deal of time designing and building your robots), your robots won't get very far without programming. This chapter will help introduce you to editing and programming concepts, so you'll feel comfortable creating some of the fairly simple programs that we'll discuss in this book. You'll also learn how to change the existing programs, which will make your robot do more amazing things.

In this chapter, we will cover the following topics:

- Basic Linux commands and navigating the File System on Raspberry Pi
- Creating, editing, and saving files on Raspberry Pi
- Creating and running Python programs on Raspberry Pi
- Some of the basic programming constructs on Raspberry Pi
- How the C programming language is both similar and different to Python, so you can understand when you need to change the C code files

We're going to use the basic configuration that you created in *Chapter 1, Getting Started with Raspberry Pi*. You can accomplish the tasks in this chapter by connecting a keyboard, a mouse, and a monitor to Raspberry Pi, or remotely logging in to Raspberry Pi using `vncserver` or `SSH`. All of these methods will work for executing the examples in this chapter.

Basic Linux commands on Raspberry Pi

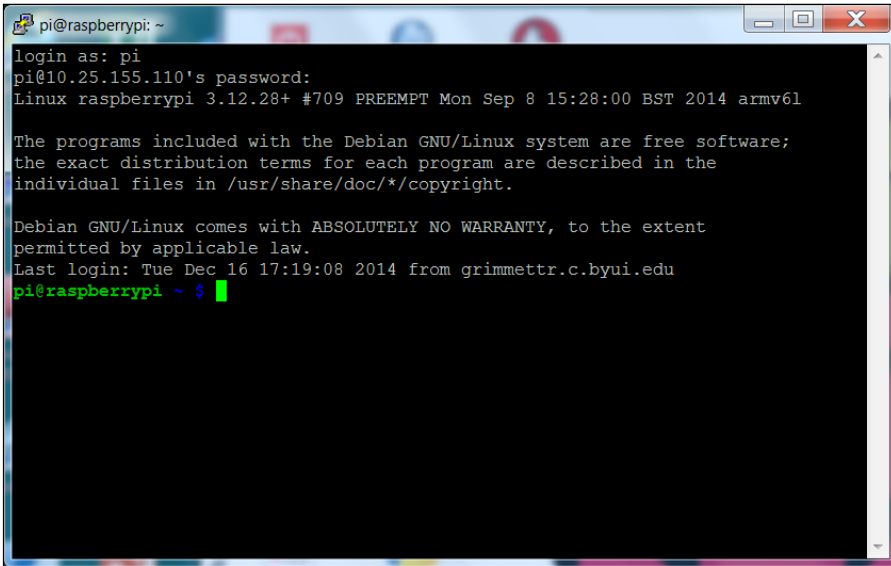
After completing the tasks in *Chapter 1, Getting Started with Raspberry Pi*, you'll have a working Raspberry Pi running a version of Linux called Raspbian. We selected the Raspbian version because it is the most popular version, and thus has the largest range of supported hardware and software. The commands reviewed in this chapter will also work with other versions of Linux, but the examples shown in this chapter use Raspbian.

So, power up your Raspberry Pi and log in, using a valid username and password. If you are going to log in remotely through SSH or vncserver, go ahead and establish the connection now. Next, we will take a quick tour of Linux. This will not be extensive; we will just walk through some of the basic commands.

Once you have logged in, open up a terminal window. If you are logging in using a keyboard, mouse, and monitor, or using vncserver, you'll find the terminal selection by selecting the **LXTerminal** application on the left-hand side of the screen, as shown in the following screenshot:

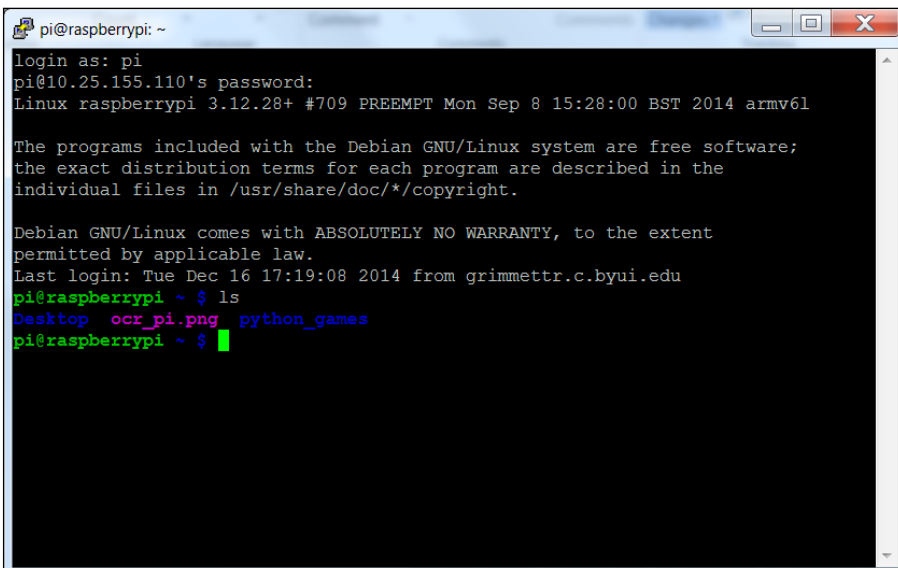


If you are using SSH, you will already be at the terminal emulator program. Either way, the terminal should look, as shown in the following screenshot:



```
pi@raspberrypi: ~  
login as: pi  
pi@10.25.155.110's password:  
Linux raspberrypi 3.12.28+ #709 PREEMPT Mon Sep 8 15:28:00 BST 2014 armv6l  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Tue Dec 16 17:19:08 2014 from grimmettr.c.byui.edu  
pi@raspberrypi ~ $
```

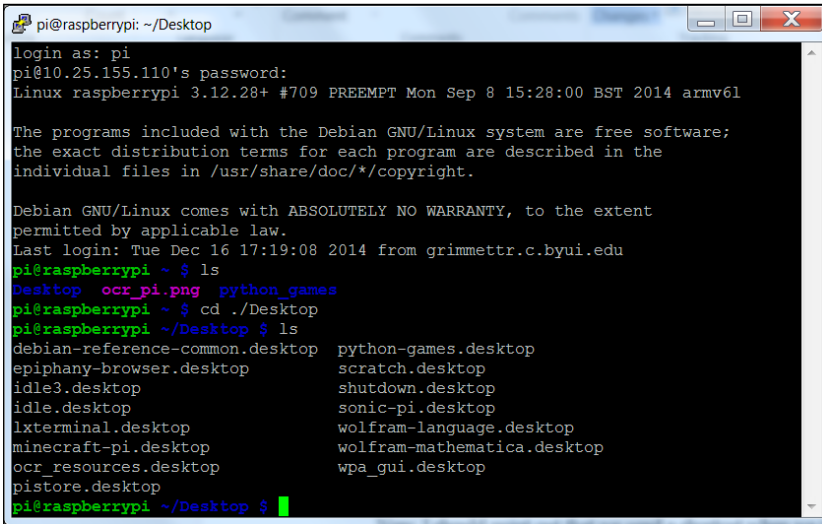
Your cursor is at the Command Prompt. Unlike Microsoft Windows or Apple's OS, with Linux, most of our work will be done by actually typing commands in the command line. So, let's try a few commands. First, type `ls`. The result should be as shown in the following screenshot:



```
pi@raspberrypi: ~  
login as: pi  
pi@10.25.155.110's password:  
Linux raspberrypi 3.12.28+ #709 PREEMPT Mon Sep 8 15:28:00 BST 2014 armv6l  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Tue Dec 16 17:19:08 2014 from grimmettr.c.byui.edu  
pi@raspberrypi ~ $ ls  
Desktop  ocr_pi.png  python_games  
pi@raspberrypi ~ $
```

In Linux, the `ls` command lists all the files and directories in our current directory. You can tell the different file types and directories apart because they are normally in different colors. You can also use `ls -l` to see more information about the files.

You can move around the directory structure by issuing the `cd` (change directory) command. For example, if you want to see what is in the `Desktop` directory, type `cd ./Desktop`. If you issue the `ls` command now, what you should see is shown in the following screenshot:



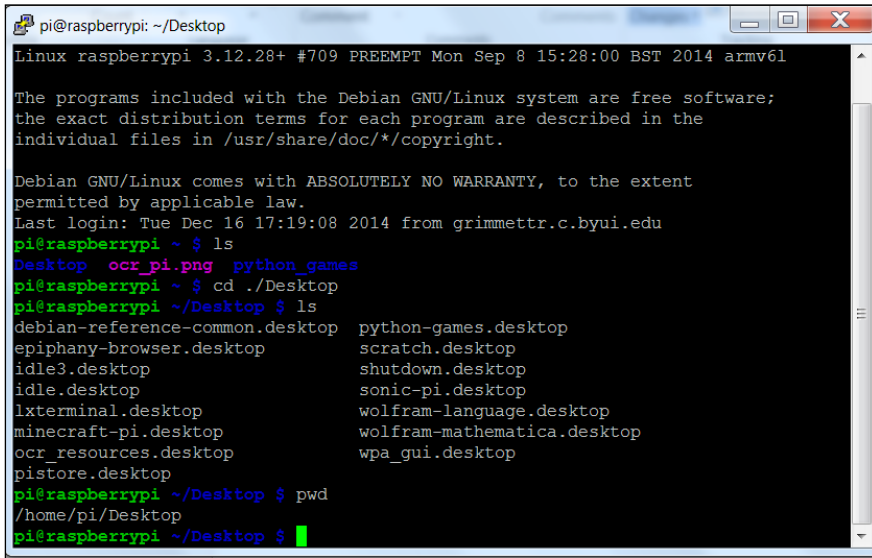
```
pi@raspberrypi: ~/Desktop
login as: pi
pi@10.25.155.110's password:
Linux raspberrypi 3.12.28+ #709 PREEMPT Mon Sep 8 15:28:00 BST 2014 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Dec 16 17:19:08 2014 from grimmettr.c.byui.edu
pi@raspberrypi ~ $ ls
Desktop  ocr_pi.png  python_games
pi@raspberrypi ~ $ cd ./Desktop
pi@raspberrypi ~/Desktop $ ls
debian-reference-common.desktop  python-games.desktop
epiphany-browser.desktop        scratch.desktop
idle3.desktop                    shutdown.desktop
idle.desktop                     sonic-pi.desktop
lxterminal.desktop              wolfram-language.desktop
minecraft-pi.desktop            wolfram-mathematica.desktop
ocr_resources.desktop           wpa_gui.desktop
pistore.desktop
pi@raspberrypi ~/Desktop $
```

This directory mostly has definitions for the behavior of the desktop icons. Now I should point out that we used a shortcut when we typed `cd ./Desktop`. The dot (`.`) character is a shortcut for the current default directory, also called the working directory. The `cd` command changes the directory. You could also have typed `cd /home/pi/Desktop` and received exactly the same result; this is because you were in the `/home/pi` directory, which is the directory in which you always start when you first log in to the system.

If you ever want to see which directory you are in, simply type `pwd`, which stands for print working directory. If you do that, you shall get the result as shown in the following screenshot:



```

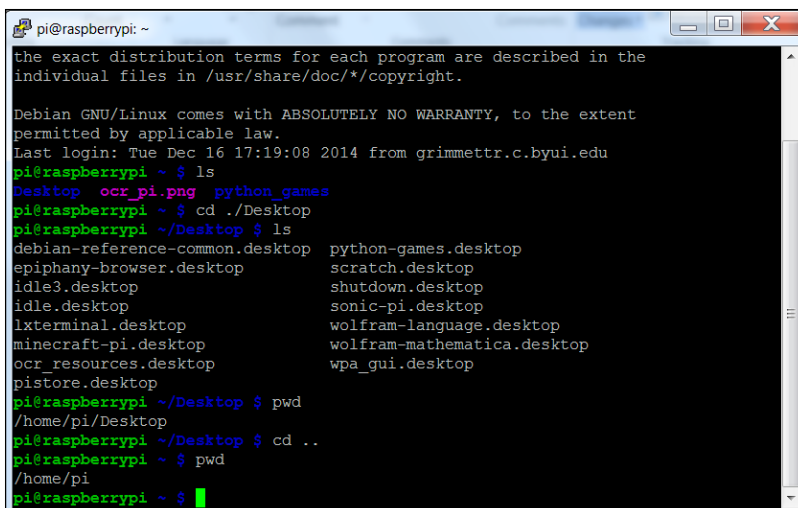
pi@raspberrypi: ~/Desktop
Linux raspberrypi 3.12.28+ #709 PREEMPT Mon Sep 8 15:28:00 BST 2014 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Dec 16 17:19:08 2014 from grimmattr.c.byui.edu
pi@raspberrypi ~ $ ls
Desktop  ocr_pi.png  python_games
pi@raspberrypi ~ $ cd ./Desktop
pi@raspberrypi ~/Desktop $ ls
debian-reference-common.desktop  python-games.desktop
epiphany-browser.desktop        scratch.desktop
idle3.desktop                   shutdown.desktop
idle.desktop                     sonic-pi.desktop
lxterminal.desktop              wolfram-language.desktop
minecraft-pi.desktop            wolfram-mathematica.desktop
ocr_resources.desktop           wpa_gui.desktop
pistore.desktop
pi@raspberrypi ~/Desktop $ pwd
/home/pi/Desktop
pi@raspberrypi ~/Desktop $

```

The result of running the `pwd` command is `/home/pi/Desktop`. Now, you can use two different shortcuts to navigate back to the default directory. The first is to type `cd ..` on the terminal; this will take you to the directory just above the current directory in the hierarchy. Then type `pwd`; you should see the following screenshot as a result:



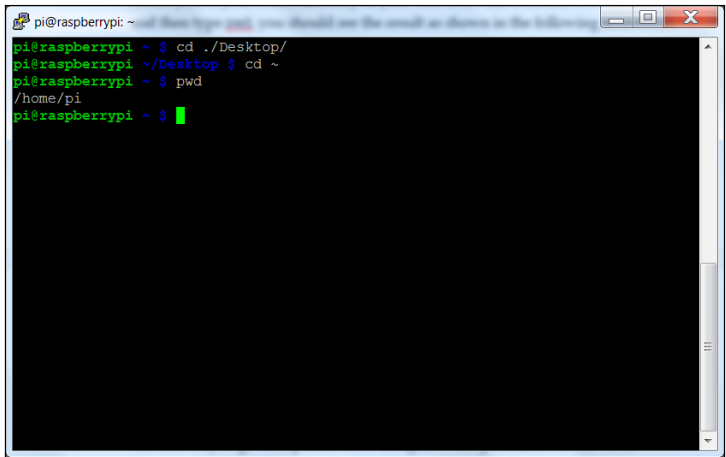
```

pi@raspberrypi: ~
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Dec 16 17:19:08 2014 from grimmattr.c.byui.edu
pi@raspberrypi ~ $ ls
Desktop  ocr_pi.png  python_games
pi@raspberrypi ~ $ cd ./Desktop
pi@raspberrypi ~/Desktop $ ls
debian-reference-common.desktop  python-games.desktop
epiphany-browser.desktop        scratch.desktop
idle3.desktop                   shutdown.desktop
idle.desktop                     sonic-pi.desktop
lxterminal.desktop              wolfram-language.desktop
minecraft-pi.desktop            wolfram-mathematica.desktop
ocr_resources.desktop           wpa_gui.desktop
pistore.desktop
pi@raspberrypi ~/Desktop $ pwd
/home/pi/Desktop
pi@raspberrypi ~/Desktop $ cd ..
pi@raspberrypi ~ $ pwd
/home/pi
pi@raspberrypi ~ $

```


The other way to get back to the home directory is by typing `cd ~`, and this will always return you to your home directory. You can also just type `cd` to return to your home directory. If you were to do this from the `Desktop` directory and then type `pwd`, you would see the result as shown in the following screenshot:

A screenshot of a terminal window titled 'pi@raspberrypi: ~'. The terminal shows a sequence of commands and their outputs: first, 'cd ../Desktop/' is entered, followed by 'cd ~', then 'pwd', which outputs '/home/pi'. The prompt returns to 'pi@raspberrypi ~ \$' with a green cursor.

```
pi@raspberrypi: ~  
pi@raspberrypi ~ $ cd ../Desktop/  
pi@raspberrypi ~/Desktop $ cd ~  
pi@raspberrypi ~ $ pwd  
/home/pi  
pi@raspberrypi ~ $
```

Another way to go to a specific directory is by using its entire pathname. In this case, if you want to go to the `/home/Raspbian/Desktop` directory from anywhere in the file system, simply type `cd /home/Raspbian/Desktop`.

There are a number of other Linux commands that you might find useful as you program your robot. The following is a table with some of the more useful commands:

Linux command	What it does
<code>ls</code>	This command lists all the files and directories in the current directory by just their names.
<code>rm filename</code>	This command deletes the file specified by <code>filename</code> .
<code>mv filename1 filename2</code>	This command renames <code>filename1</code> to <code>filename2</code> .
<code>cp filename1 filename2</code>	This command copies <code>filename1</code> to <code>filename2</code> .
<code>mkdir directoryname</code>	This command creates a directory with the name specified by <code>directoryname</code> ; this will be made in the current directory, unless specified otherwise.
<code>clear</code>	This command clears the current terminal window.

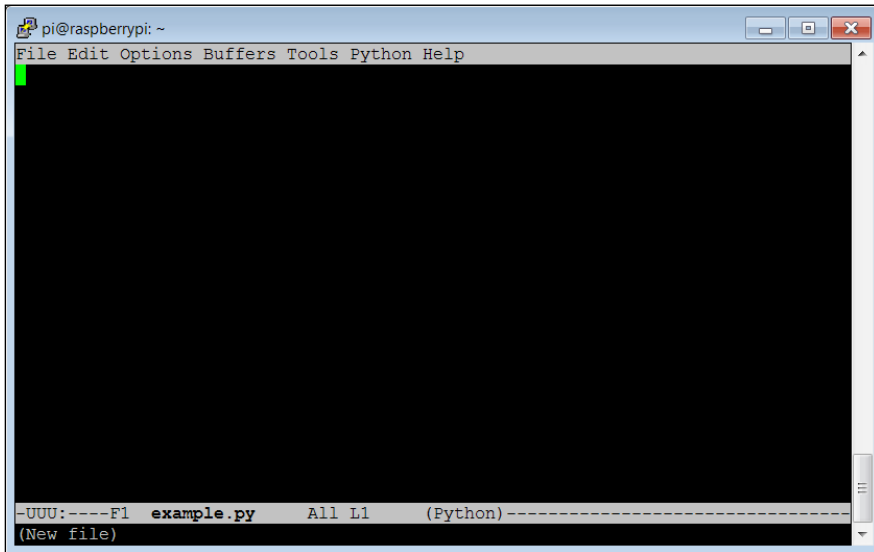
Linux command	What it does
sudo	If you type the sudo command at the beginning of any command, it will execute that command as the super user. This may be required if the command or program you are trying to execute needs super user permissions. If, at any point, you type a command or the name of the program you want to run and the result seems to suggest that the command does not exist or permission is denied, try doing it again with sudo at the beginning.

Now you can play around with the commands and look at your system and the files that are available to you. But, be a bit careful! Linux is not like Windows: the default behavior is to not warn you if you try to delete or replace a file.

Creating, editing, and saving files on Raspberry Pi

Now that you can log in and move easily between directories and see the files in them, you'll want to be able to edit those files. To do this, you'll need a program that allows you to edit the characters in a file. If you are used to working on Microsoft Windows, you have probably used programs such as Microsoft Notepad, WordPad, or Word to do this. As you know, these programs are not available in Linux. But, there are several other choices for editors, all of which are free. In this chapter, we will use an editor program called **Emacs**. Other possibilities are programs such as nano, vi, vim, and gedit. Programmers have strong feelings about which editor to use, so if you already have a favorite, you can skip this section.

If you want to use Emacs, download and install it by typing `sudo apt-get install emacs`. Once installed, you can run Emacs simply by typing `emacs filename`, where `filename` is the name of the file you want to edit. If the file does not exist, Emacs will create it. The following screenshot shows what you will see if you type `emacs example.py` at the prompt:



Notice that unlike Windows, Linux doesn't automatically assign file extensions; it is up to us to specify the kind of file we want to create. Notice also that in the lower left corner of the screen, the Emacs editor indicates that you have opened a new file. Now, if you are using Emacs in the LXDE Windows interface, either because you have a monitor, keyboard, and mouse hooked up or because you are running `vncserver`, you can use the mouse in much the same way as you do in Microsoft Word.

However, if you are running Emacs from SSH, you won't have the mouse available. So you'll need to navigate the file by using the cursor keys. You'll also have to use some keystroke commands to save your file, as well as accomplish a number of other tasks that you would normally use the mouse for. For example, when you are ready to save the file, you must press `Ctrl + X` and `Ctrl + S`, and that will save the file under the current filename. When you want to quit Emacs, you must press `Ctrl + X` and `Ctrl + C`. This will stop Emacs and return you to the command prompt. If you are going to use Emacs, the following are a number of keystroke commands you might find useful:

The Emacs command	What it does
<i>Ctrl + X</i> and <i>Ctrl + S</i>	Save: This command saves the current file.
<i>Ctrl + X</i> and <i>Ctrl + C</i>	Quit: This command makes you exit Emacs and return to the command prompt.
<i>Ctrl + K</i>	Kill: This command erases the current line.
<i>Ctrl + _</i>	Undo: This command undoes the last action.
Left-click and text selection followed by cursor placement and right-click	Cut and paste: If you select the text you want to paste by clicking the mouse, move the cursor to where you want to paste the code and then right-click on it; the code will be pasted in that location.

Now that you have the capability to edit files, in the next section, you'll use this capability to create programs.

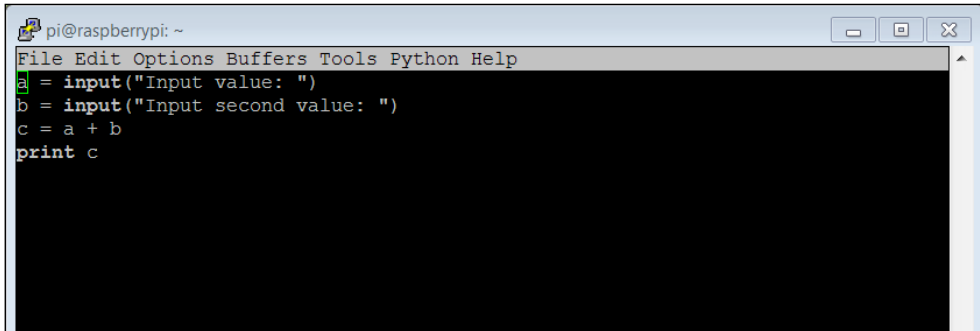
Creating and running Python programs

Now that you are ready to begin programming, you'll need to choose a language. There are many available: C, C++, Java, Python, Perl, and a great deal of other possibilities. I'm going to initially introduce you to Python for two reasons: it is a simple language that is intuitive and very easy to use, and it avails a lot of the open source functionality of the robotics world. We'll also cover a bit of C/C++ in this chapter, as some functionalities are only available in C/C++. But it makes sense to start in Python. To work through the examples in this section, you'll need a version of Python installed. Fortunately, the basic Raspbian system has one already, so you are ready to begin. Python interpreter is what the Pi in Raspberry Pi stands for.

We are only going to cover some of the very basic concepts here. If you are new to programming, there are a number of different websites that provide interactive tutorials. If you'd like to practice some of the basic programming concepts in Python using these tutorials, visit www.codecademy.com or <http://www.learnpython.org/> or <https://docs.python.org> and give it a try.

In this section, we'll cover how to create and run a Python file. It turns out that Python can be used interactively, so you can run it and then type in the commands, one at a time. Using it interactively is extremely helpful when first getting acquainted with the language features and modules. But we want to use Python to create programs, so we are going to type in our commands using Emacs and then run them in the command line by invoking Python. Let's get started.


Open an example Python file by typing `emacs example.py`. Now, let's put some code in the file. Start with the code, as shown in the following screenshot:



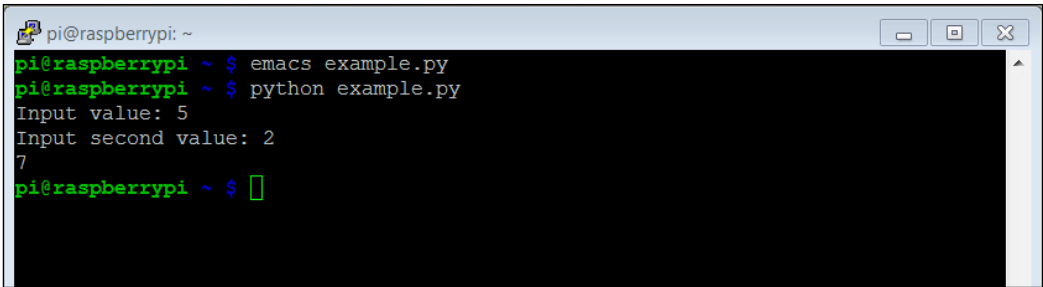
```
pi@raspberrypi: ~
File Edit Options Buffers Tools Python Help
a = input("Input value: ")
b = input("Input second value: ")
c = a + b
print c
```

Let's go through the code to see what is happening. The code lines are as follows:

- `a = input("Input value: ")`: One of the basic purposes of a program is to get input from the user. `input` allows us to do that. The data will be input by the user and stored in `a`. The prompt `Input value:` will be shown to the user.
- `b = input("Input second value: ")`: This data will also be input by the user and stored in `b`. The prompt `Input second value:` will be shown to the user.
- `c = a + b`: This is an example of something you can do with the data; in this example, you can add `a` and `b`.
- `print c`: Another basic purpose of our program is to print out results. The `print` command prints out the value of `c`.

 This code is written using Python 2. If you are using Python 3, you will need to change your `print` to `print(c)`. For other changes that might be required, go to <http://learntocodewith.me/programming/python/python-2-vs-python-3/>.

Once you have created your program, save it (using `Ctrl + X Ctrl + S`) and quit Emacs (using `Ctrl + X Ctrl + C`). Now, from the command line, run your program by typing `python example.py`. You should see the result shown in the following screenshot:

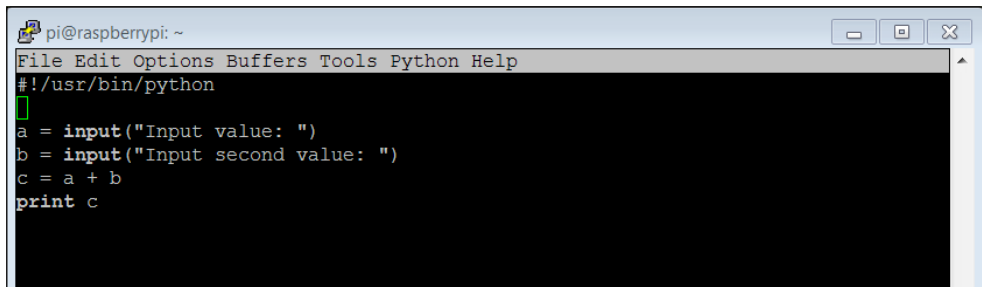


```

pi@raspberrypi: ~
pi@raspberrypi ~ $ emacs example.py
pi@raspberrypi ~ $ python example.py
Input value: 5
Input second value: 2
7
pi@raspberrypi ~ $ 

```

You can also run the program from the command line without typing `python example.py` by adding the line `#!/usr/bin/python` to the program. Then the program will look, as shown in the following screenshot:

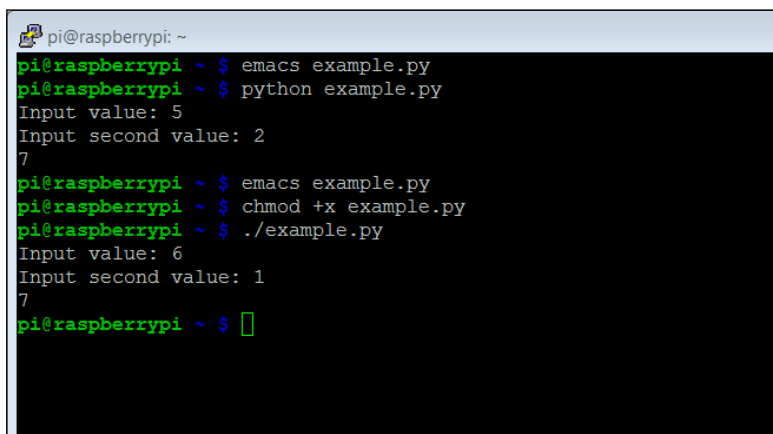


```

pi@raspberrypi: ~
File Edit Options Buffers Tools Python Help
#!/usr/bin/python
a = input("Input value: ")
b = input("Input second value: ")
c = a + b
print c

```

Adding `#!/usr/bin/python` as the first line simply makes this file available for us to execute from the command line. Once you have saved the file and exited Emacs, type `chmod +x example.py`. This will change the file's execution permissions, so the computer will now accept and execute it. You should be able to simply type `./example.py` and see the program run, as shown in the following screenshot:



```

pi@raspberrypi: ~
pi@raspberrypi ~ $ emacs example.py
pi@raspberrypi ~ $ python example.py
Input value: 5
Input second value: 2
7
pi@raspberrypi ~ $ emacs example.py
pi@raspberrypi ~ $ chmod +x example.py
pi@raspberrypi ~ $ ./example.py
Input value: 6
Input second value: 1
7
pi@raspberrypi ~ $ 

```

Notice that if you simply type `example.py`, the system will not find the executable file. Here, the file has not been registered with the system, so you have to give the system a path to it. In this case, `./` is the current directory.

Basic programming constructs on Raspberry Pi

Now that you know how to enter and run a simple Python program on Raspberry Pi, let's look at some more complex programming tools. Specifically, we'll cover what to do when we want to determine the instructions to execute and how to loop our code to do that more than once. I'll give a brief introduction on how to use libraries in the Python version 2.7 code and how to organize statements into functions. Finally, we'll very briefly cover object-oriented code organization.



Indentation in Python is very important; it will specify which group of statements are associated with a given loop or decision set, so watch your indentation carefully.

The if statement

As you have seen in previous examples, your programs normally start by executing the first line of code and then continue with the following lines, until the program runs out of code. But what if you want to decide between two different courses of action? We can do this in Python by using an `if` statement. The following screenshot shows some example code:

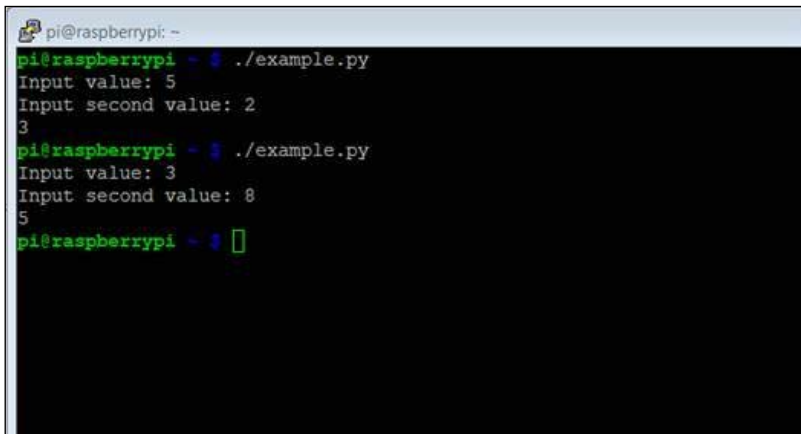
```
pi@raspberrypi: ~
File Edit Options Buffers Tools Python Help
#!/usr/bin/python

a = input("Input value: ")
b = input("Input second value: ")
if a > b:
    c = a - b
else:
    c = b - a
print c
```

The following are the details of the code shown in the previous screenshot, line by line:

- `#!/usr/bin/python`: This is included so you can make your program executable.
- `a = input("Input value: ")`: One of the basic needs of a program is to get input from the user. `input` allows us to do that. The data will be input by the user and stored in `a`. The prompt `Input value:` will be shown to the user.
- `b = input("Input second value: ")`: This data will also be input by the user and stored in `b`. The prompt `Input second value:` will be shown to the user.
- `if a > b:`: This is an `if` statement. The expression evaluated in this case is `a > b`. If it is `true`, the program will execute the next one or more statements that is indented; in this case, `c = a - b`. If not, it will skip that statement.
- `else:`: The `else` statement is an optional part of the command. If the expression in the `if` statement is evaluated as `false`, the indented one or more statements after the `else:` statement will be executed; in this case, `c = b - a`.
- `print c`: Another basic purpose of our program is to print out results. The `print` command prints out the value of `c`.

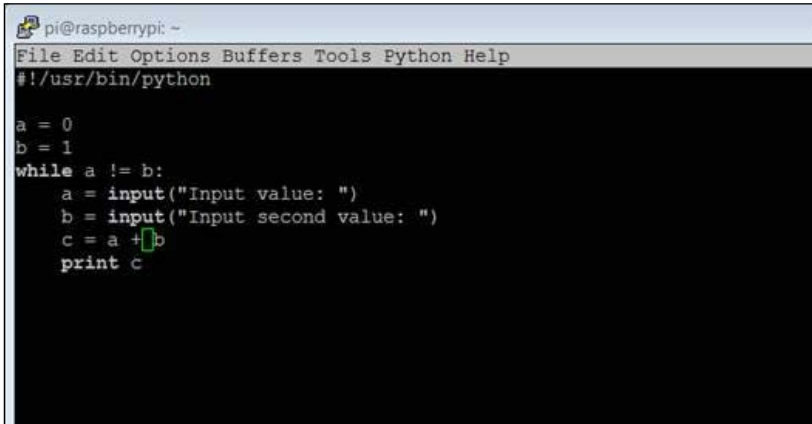
You can run the previous program a couple of times, checking both the `true` and `false` possibilities of the `if` expression, as shown in the following screenshot:



```
pi@raspberrypi: ~  
pi@raspberrypi ~$ ./example.py  
Input value: 5  
Input second value: 2  
3  
pi@raspberrypi ~$ ./example.py  
Input value: 3  
Input second value: 8  
5  
pi@raspberrypi ~$
```


The while statement

Another useful construct is the `while` construct; it allows us to execute a set of statements over and over again, until a specific condition has been met. The following screenshot shows a set of code that uses this construct:

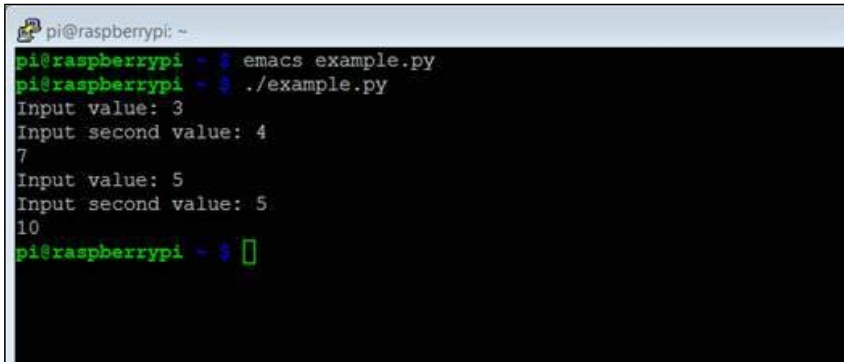


```
pi@raspberrypi: ~  
File Edit Options Buffers Tools Python Help  
#!/usr/bin/python  
  
a = 0  
b = 1  
while a != b:  
    a = input("Input value: ")  
    b = input("Input second value: ")  
    c = a + b  
    print c
```

The following are the details of the code shown in the previous screenshot:

- `#!/usr/bin/python`: This is included so you can make your program executable.
- `a = 0`: This line sets the value of variable `a` to 0. We'll need this only to make sure that we execute the loop at least once.
- `b = 1`: This line sets the value of the variable `b` to 1. We'll need this only to make sure that we execute the loop at least once.
- `while a != b`:: The expression `a != b` (in this case, `!=` means not equal to) is verified. If it is `true`, the indented statements is executed. When the statement is evaluated as `false`, the program jumps to the statements (none in this example) after the indented section.
- `a = input("Input value: ")`: One of the basic purposes of a program is to get input from the user. `input` allows us to do that. The data will be input by the user and stored in `a`. The prompt `Input value:` will be shown to the user.
- `b = input("Input second value: ")`: This data will also be input by the user and stored in `b`. The prompt `Input second value:` will be shown to the user.
- `c = a + b`: The variable `c` is loaded with the sum of `a` and `b`.
- `print c`: The `print` command prints out the value of `c`.

Now you can run the program. Notice that when you enter the same value for *a* and *b*, the program stops, as shown in the following screenshot:



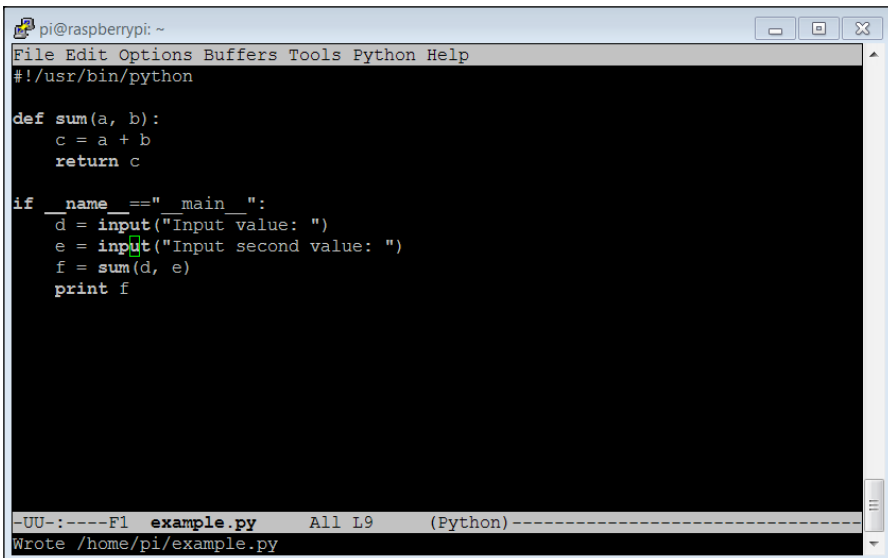
```

pi@raspberrypi: ~
pi@raspberrypi ~$ emacs example.py
pi@raspberrypi ~$ ./example.py
Input value: 3
Input second value: 4
7
Input value: 5
Input second value: 5
10
pi@raspberrypi ~$ 

```

Working with functions

The next concept we need to cover is how to put a set of statements into a function. We use functions to organize code, grouping sets of statements together when it makes sense that they be organized and be in the same location. For example, if we have a specific calculation that we might want to perform many times, instead of copying the set of statements each time we want to perform it, we group them into a function. I'll use a fairly simple example here, but if the calculation takes a significant number of programming statements, you can see how that would make our code significantly more efficient. The following screenshot shows the code:



```

pi@raspberrypi: ~
File Edit Options Buffers Tools Python Help
#!/usr/bin/python

def sum(a, b):
    c = a + b
    return c

if __name__ == "__main__":
    d = input("Input value: ")
    e = input("Input second value: ")
    f = sum(d, e)
    print f

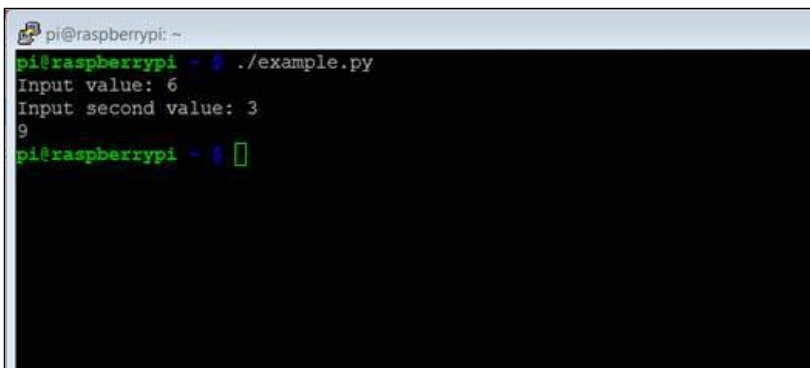
-UU-:----F1 example.py All L9 (Python)-----
Wrote /home/pi/example.py

```

The following is the explanation of the code from our previous example:

- `#!/usr/bin/python`: This is included so you can make your program executable.
- `def sum(a, b) :` This line defines a function named `sum`. The `sum` function takes `a` and `b` as arguments.
- `c = a + b`: Whenever this function is called, it will add the values in the variable `a` to the values in variable `b`.
- `return c`: When the function is executed, it will return the variable `c` to the calling expression.
- `if __name__ == "__main__" :` In this particular case, you don't want your program to start executing each statement from the top of the file; you would rather it started at a particular point. This line tells the program to begin its execution at this particular point.
- `d = input("Input value: ")`: This data will also be input by the user and will be stored in `d`. The prompt `Input second value:` will be shown to the user.
- `e = input("Input second value: ")`: This data will also be input by the user and stored in `e`. The prompt `Input second value:` will be shown to the user.
- `f = sum(d, e)`: The function `sum` is called. In the `sum` function, the value in variable `d` is copied to the variable `a` and the value in the variable `e` is copied to the variable `b`. The program then goes to the `sum` function and executes it. The return value is then stored in the variable `f`.
- `print f`: The `print` command prints out the value of `f`.

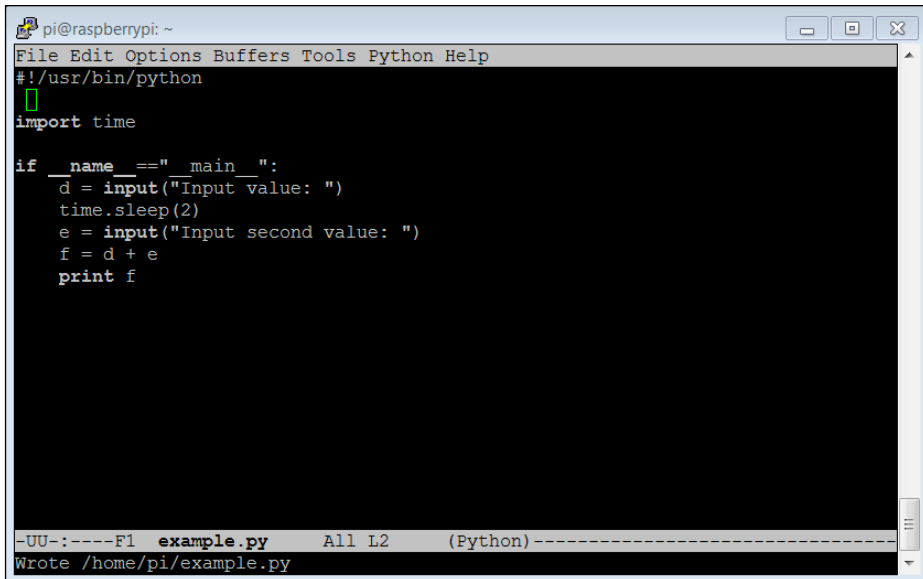
The following screenshot is the result received when you run the code:



```
pi@raspberrypi: ~
pi@raspberrypi ~$ ./example.py
Input value: 6
Input second value: 3
9
pi@raspberrypi ~$
```

Libraries/modules in Python

The next topic we need to cover is how to add functionality to our programs using libraries/ modules. Libraries, or modules as they are sometimes called in Python, include a functionality that someone else has created and that you want to add to your code. As long as the functionality exists and your system knows about it, you can include the library in the code. So, let's modify our code again by adding the library, as shown in the following screenshot:



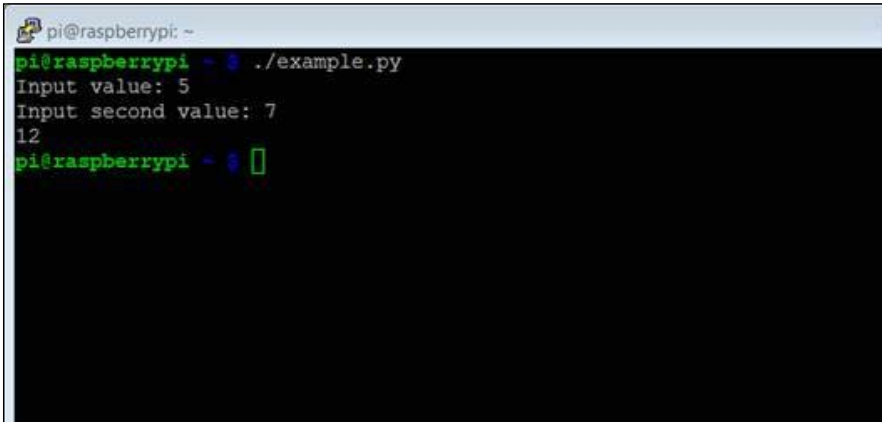
```
pi@raspberrypi: ~  
File Edit Options Buffers Tools Python Help  
#!/usr/bin/python  
  
import time  
  
if __name__=="__main__":  
    d = input("Input value: ")  
    time.sleep(2)  
    e = input("Input second value: ")  
    f = d + e  
    print f  
  
-UU-:----F1 example.py All L2 (Python)-----  
Wrote /home/pi/example.py
```

The following is a line-by-line description of the code:

- `#!/usr/bin/python`: This is included so you can make your program executable.
- `import time`: This includes the `time` library. The `time` library includes a function that allows you to pause for a specified number of seconds.
- `if __name__=="__main__":`: In this particular case, you don't want your program to start executing each statement from the top of the file; you would rather it started from a particular line. This line tells the program to begin its execution at this specified point.

- `d = input("Input value: ")`: This data will also be input by the user and will be stored in `b`. The prompt `Input second value:` will be shown to the user.
- `time.sleep(2)`: This line calls the `sleep` function in the `time` library, which will cause a 2 second delay.
- `e = input("Input second value: ")`: This data will also be input by the user and will be stored in `b`. The prompt `Input second value:` will be shown to the user.
- `f = d + e`: The `f` variable is loaded with the value `d + e`.
- `print f`: The `print` command prints out the value of `f`.

The following screenshot shows the result after running the previous example code:

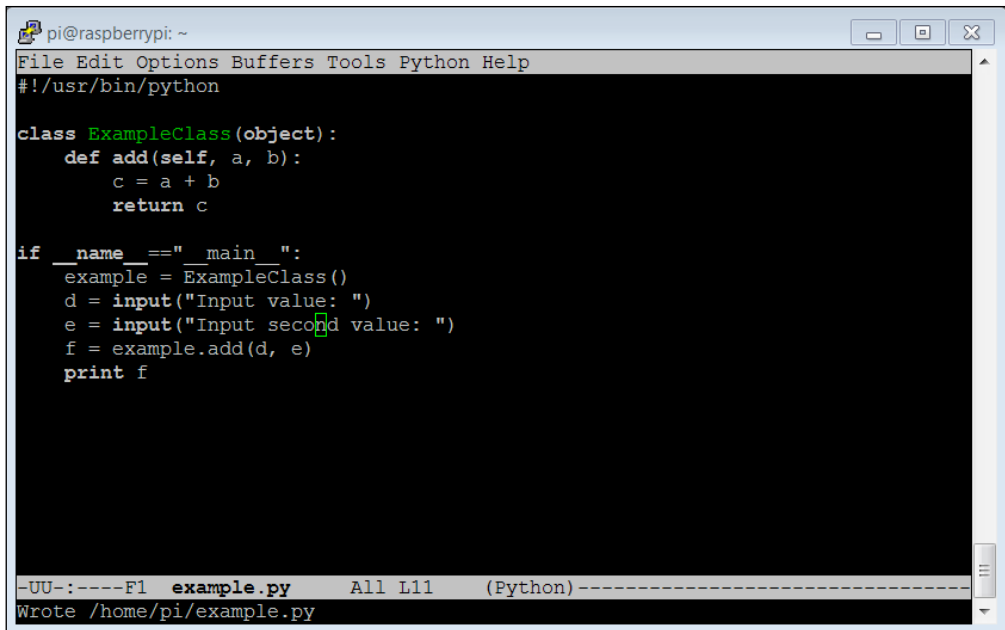


```
pi@raspberrypi ~  
pi@raspberrypi ~$ ./example.py  
Input value: 5  
Input second value: 7  
12  
pi@raspberrypi ~$
```

Of course, this looks very similar to other results. But you will notice a pause between your entering the first value and the appearance of the second value.

Object-oriented code

The final topic that we need to cover is object-oriented organization in our code. In object-oriented code, we organize a set of related functions in an object. If, for example, we have a set of functions that are all related, we can place them in the same class and then call them by associating them with the specified class. This is a complex and difficult topic, but let me just show you a simple example in the following screenshot:



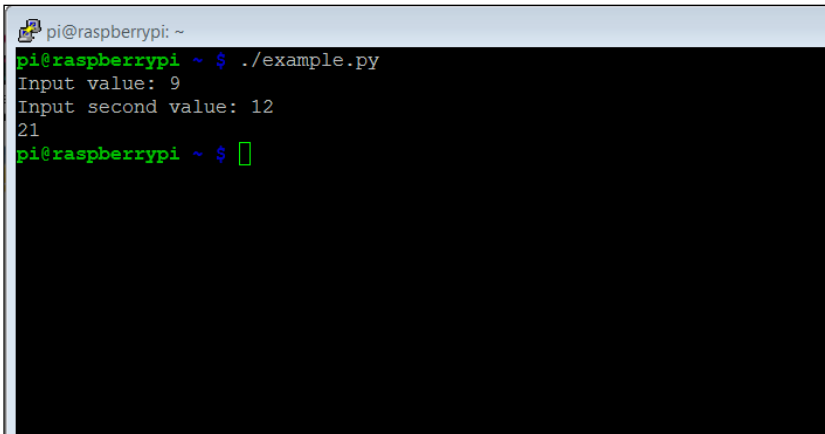
```
pi@raspberrypi: ~  
File Edit Options Buffers Tools Python Help  
#!/usr/bin/python  
  
class ExampleClass(object):  
    def add(self, a, b):  
        c = a + b  
        return c  
  
if __name__ == "__main__":  
    example = ExampleClass()  
    d = input("Input value: ")  
    e = input("Input second value: ")  
    f = example.add(d, e)  
    print f  
  
-UU-:----F1  example.py  All L11  (Python)-----  
Wrote /home/pi/example.py
```

The following is an explanation of the code in the previous screenshot:

- `#!/usr/bin/python`: This is included so you can make your program executable.
- `class ExampleClass(object) ::` This defines a class named `ExampleClass`. This class can have any number of functions associated with it.
- `def add(self, a, b) ::` This defines the function `add` as part of `ExampleClass`. We can have functions that have the same names as long as they belong to different classes. This function takes two arguments: `a` and `b`.
- `c = a + b`: This statement indicates the simple addition of two values.
- `return c`: This function returns the result of the addition.
- `if __name__ == "__main__" ::` In this particular case, you don't want your program to start executing each statement from the top of the file; you would rather it started from a specific line. This line tells the program to begin its execution at the specified point.

- `example = ExampleClass()`: This defines a variable named `example`, the type of which is `ExampleClass`. This variable now has access to all the functions and variables associated with the `ExampleClass` class.
- `d = input("Input value: ")`: This data will be input by the user and stored in the variable `b`. The prompt `Input second value:` will be shown to the user.
- `e = input("Input second value: ")`: This data will also be input by the user and stored in `b`. The prompt `Input second value:` will be shown to the user.
- `f = example.add(d,e)`: The instance of `ExampleClass` is called and its function `add` is executed by sending `d` and `e` to that function. The result is returned and stored in `f`.
- `print f`: The `print` command prints out the value of the variable `f`.

The result after running the previous example code is shown in the following screenshot:



```
pi@raspberrypi: ~  
pi@raspberrypi ~ $ ./example.py  
Input value: 9  
Input second value: 12  
21  
pi@raspberrypi ~ $
```

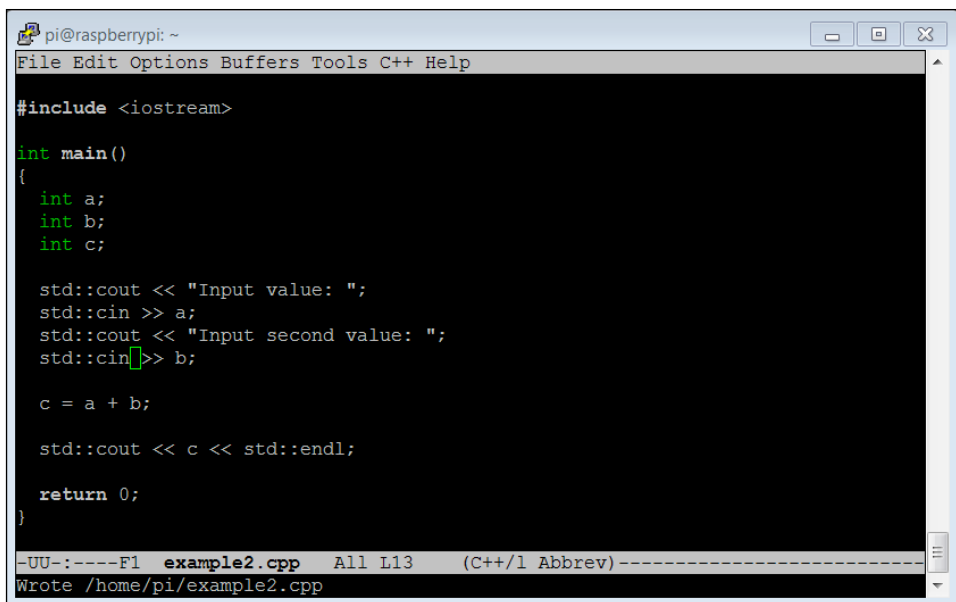
The result shown in the preceding screenshot is the same as the other codes discussed earlier, and there is no functionality difference. However, object-oriented techniques have been used to keep similar functions organized together to make the code easier to maintain. This also makes it easy for others to use your code.

Introduction to the C/C++ programming language

Now that you've been introduced to a simple programming language, Python, we need to spend a bit of time talking about a more complex but powerful language called C. C is the original language of Linux and has been around for many decades, but it is still widely used by open source developers. It is similar to Python, but is also a bit different. Since you may need to understand and make changes to C code, you should be familiar with it and its usage. C++ is a greatly expanded version of C that adds object-oriented programming features. The following code is actually C++ code.

As with Python, you will need to have access to the language capabilities of C. These come in the form of a compiler and build system, which turns your text files that contain programs into machine code that the processor can actually execute. To do this, type `sudo apt-get install build-essential`. This will install the programs you need to turn your code into executables for the system.

Now that the tools required for C/C++ are installed, let's walk through some simple examples. The following screenshot shows the first C/C++ code example (named `example2.cpp`):



```
pi@raspberrypi: ~  
File Edit Options Buffers Tools C++ Help  
#include <iostream>  
  
int main()  
{  
    int a;  
    int b;  
    int c;  
  
    std::cout << "Input value: ";  
    std::cin >> a;  
    std::cout << "Input second value: ";  
    std::cin >> b;  
  
    c = a + b;  
  
    std::cout << c << std::endl;  
  
    return 0;  
}
```

--UU--:----F1 example2.cpp All L13 (C++/1 Abbrev)-----
Wrote /home/pi/example2.cpp

The following is an explanation of the code shown in the previous screenshot:

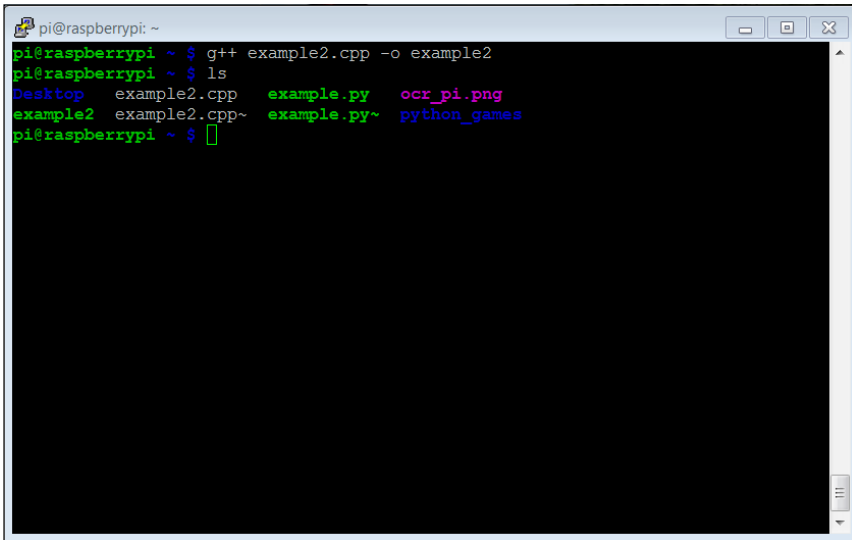
- `#include <iostream>`: This is a library that is included so your program can input data using the keyboard and send output information to the screen.
- `int main()`: As with Python, we can place functions and classes in the file, but you will always want to start a program's execution at a known point; C defines this known point as the main function.
- `int a;`: This defines a variable named `a`, which is of the type `int`. C is what we call a strongly typed language, which means we need to declare the type of the variable we are defining. The normal types are as follows:
 - `int`: This is used for numbers that have no decimal points
 - `float`: This is used for numbers that require decimal points
 - `char`: This is used for a character of text
 - `bool`: This is used for a true or false value

Also note that every line in C ends with the `;` character.

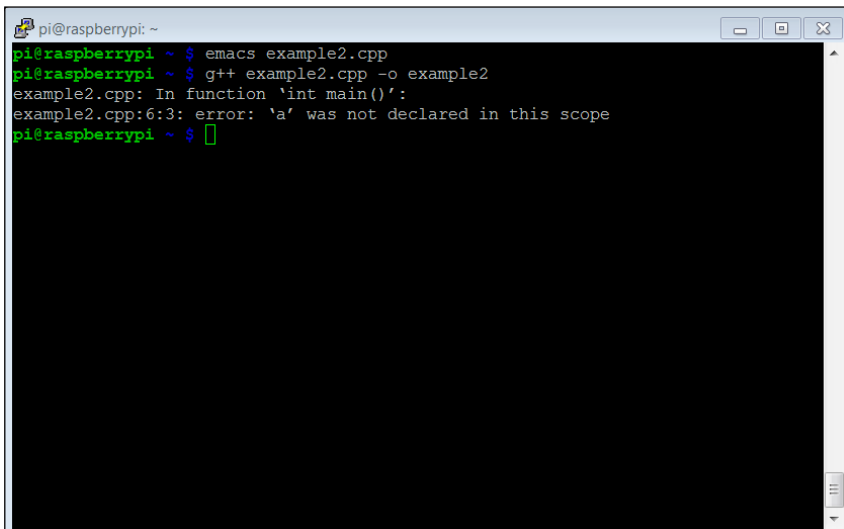
- `int b;`: This defines a variable named `b`, which is of the type `int`.
- `int c;`: This defines a variable named `c`, which is of the type `int`.
- `std::cout << "Input value: ";`: This will display the string "Input value: " on the screen.
- `std::cin >> a;`: The input that the user types will be stored in the variable `a`.
- `std::cout << "Input second value: ";`: This will display the string "Input second value: " on the screen.
- `std::cin >> b;`: The input that the user types will go into the variable `b`.
- `c = a + b;`: The statement is the simple addition of two values.
- `std::cout << c << std::endl;`: The `cout` command prints out the value of `c`. The `endl` command at the end of this line prints out a carriage return so that the next character appears on the next line.
- `return 0;`: The main function ends and returns 0.

To run this program, you'll need to run a compile process to turn it into an executable program. To do this, after you have created the program, type `g++ example2.cpp -o example2`. This will then process your program, turning it into a file that the computer can execute. The name of the executable program will be `example2` (specified as the name after the `-o` option).

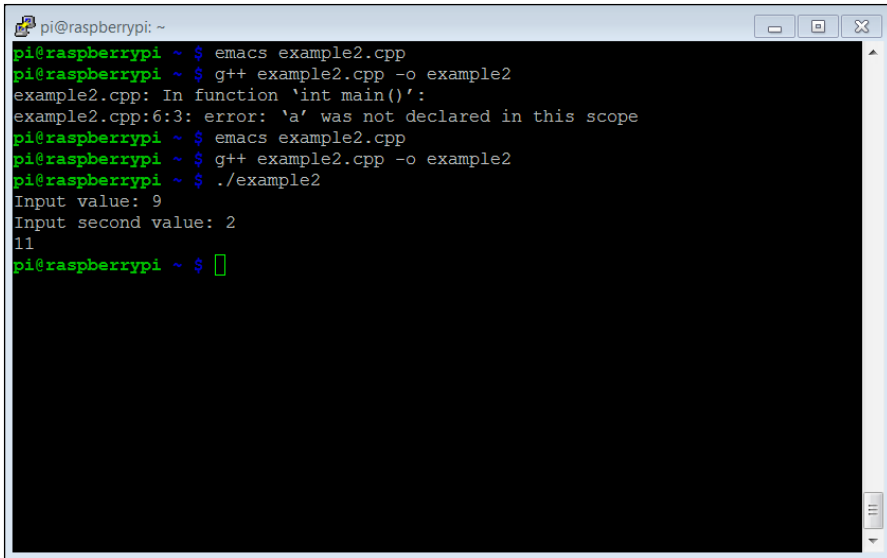
If you run an `ls` command on your directory, after you have compiled this program, you should see the `example2` file in your directory, as shown in the following screenshot:

A terminal window on a Raspberry Pi. The prompt is 'pi@raspberrypi: ~'. The user enters 'g++ example2.cpp -o example2'. The prompt changes to 'pi@raspberrypi ~ \$'. The user enters 'ls'. The output shows a directory listing: 'Desktop example2.cpp example.py ocr_pi.png example2 example2.cpp~ example.py~ python_games'. The prompt returns to 'pi@raspberrypi ~ \$'.

By the way, if you run into a problem, the compiler will try to help you figure it out. If, for example, you were to forget to include the `int` type before the `int a;` declaration, you would get the error, as shown in the following screenshot, when you try to compile the previous code:

A terminal window on a Raspberry Pi. The prompt is 'pi@raspberrypi: ~'. The user enters 'emacs example2.cpp'. The prompt changes to 'pi@raspberrypi ~ \$'. The user enters 'g++ example2.cpp -o example2'. The output shows an error: 'example2.cpp: In function 'int main()': example2.cpp:6:3: error: 'a' was not declared in this scope'. The prompt returns to 'pi@raspberrypi ~ \$'.

The error message indicates a problem in the `int main()` function and tells you that the variable `a` was not successfully declared. Once you have the file compiled, to run the executable, type `./example2` and you should be able to obtain the following result:



```
pi@raspberrypi: ~  
pi@raspberrypi ~$ emacs example2.cpp  
pi@raspberrypi ~$ g++ example2.cpp -o example2  
example2.cpp: In function 'int main()':  
example2.cpp:6:3: error: 'a' was not declared in this scope  
pi@raspberrypi ~$ emacs example2.cpp  
pi@raspberrypi ~$ g++ example2.cpp -o example2  
pi@raspberrypi ~$ ./example2  
Input value: 9  
Input second value: 2  
11  
pi@raspberrypi ~$
```

We will not provide a tutorial for C in this book; there are several good tutorials on the Internet that can help, for example, <http://www.cprogramming.com/tutorial/c-tutorial.html> and <http://thenewboston.org/list.php?cat=14>. There is one more aspect of C you will need to know about. The compile process that you just encountered seems fairly straightforward. However, if you have your functionality distributed among a lot of files, or need lots of libraries for your programs, the command-line approach to executing a compile can get unwieldy.

The C development environment provides a way to automate the compile process; this is called the *make* process. When performing this process, you create a text program named *makefile* that defines the files you want to include and compile. Instead of typing a long command or a set of commands, you simply type *make* and the system will execute a compile based on the definitions in the *makefile* program. There are several good tutorials that discuss this system more; try visiting the these websites <http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/> or <http://mrbook.org/tutorials/make/>.

Now you are equipped to edit and create your own programming files. The forthcoming chapters will provide you with lots of opportunities to practice your skills as you translate lines of code into cool robotic capabilities.

If you are going to do a significant amount of coding, you'll want to install an **Integrated Development Environment (IDE)**. These environments make it much easier to see, edit, compile, and debug your programs. The most popular of these programs in the Linux world is called **Eclipse**. If you'd like to know more, start with a Google search or go to <http://www.eclipse.org/>.

Summary

In this chapter, you've learned how to interact with the Raspbian operating system using the command line and also create and edit files using Emacs. You have also been exposed to both the Python and C programming languages. If this is your first experience with programming, don't be surprised if you are still very uneasy with programing in general, and `if` and `while` statements in particular. You probably felt just as uncomfortable during your first introduction to the English language, although you may not remember it.

It is always a bit difficult to try new things. However, I will try to give you explicit instructions on what to type so that you can be successful. There is one major challenge in working with computers. They always do exactly what you tell them to do and not necessarily what you want them to. So if you encounter problems, check several times to make sure that your code matches the example exactly. Now, on to some actual coding!

In the next chapter, you'll start adding additional functionality that will enable you to create amazing robotics projects. You'll start by providing your system with the capability to speak and also listen to your commands.

3

Providing Speech Input and Output

Now that your Raspberry Pi is up and operating, let's start giving your projects some basic functionality. You'll use this functionality in later chapters as you build wheeled robots, tracked robots, robots that can walk, and even sail or fly. We're going to start with speech; it is a good basic project and offers several examples of adding capability terms of both hardware and software. So, buckle up and get ready to learn the basics of interfacing with your board by facilitating speech.

You'll be adding a microphone and a speaker to your Raspberry Pi. You'll also be adding functionality so that the robot can recognize voice commands, and respond through the speaker. Additionally, you'll be able to issue voice commands and make the robot respond with an action. When you're free from typing in commands, you can interact with your robotic projects in an impressive way. This project will require adding both hardware and software.

Interfacing with your projects through speech is more fun than typing in commands, and it allows interaction with your project without using a keyboard or a mouse. Besides, what self-respecting robot wants to carry around a keyboard? No, you want to interact in natural ways with your projects, and this chapter will teach you how. Interfacing via speech also helps you find your way around the Raspberry Pi, learn how to use the available free, open source software functionality, and become familiar with the community of open source developers. This chapter covers the capabilities needed before you can add vision or motors to your project.

In this chapter, we'll specifically cover the following points:

- Hooking up the hardware to make and input sound
- Using **Espeak** to allow your projects to respond in a robotic voice
- Using **PocketSphinx** to interpret your commands
- Providing the capability to interpret your commands, and having your robot initiate action

Before beginning this project, you'll need a working Raspberry Pi and a LAN connection (refer to *Chapter 1, Getting Started with Raspberry Pi*, for instructions). Additionally, this project requires a USB microphone and a speaker adapter. The Raspberry Pi itself has an audio output, but does not have an audio input. The HDMI output does support audio, but most of your robotics projects will not be connected to HDMI monitors with speaker capability.

You'll need the following three pieces of hardware:

- A USB device that supports microphone input and speaker output.



- A microphone that can plug into the USB device.

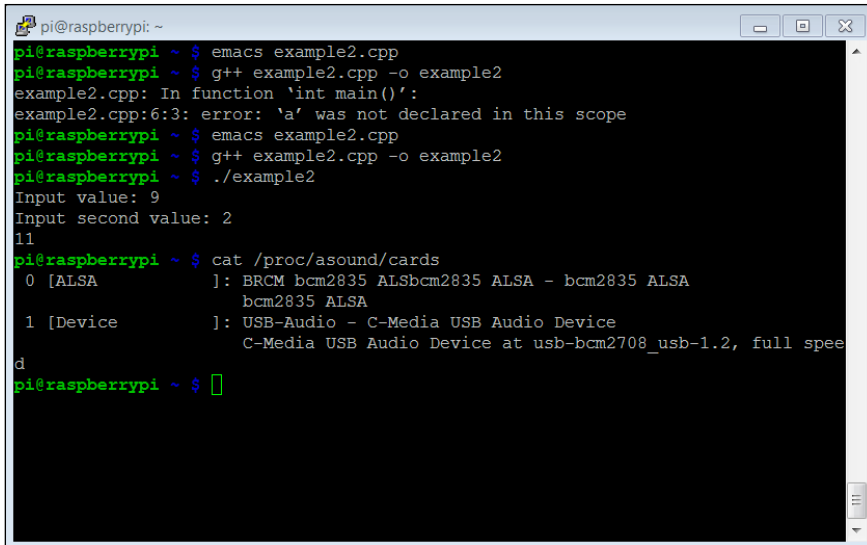


If you are using Raspberry Pi A+ board, then your entire system will look like the following image:



Plug in the power. You can execute all of the following instructions in one of the following ways:

- If you are still connected to the display, keyboard, and mouse, log into the system, and use the Windows system by opening an LXTerminal window.
- If you are only connected through LAN, you can do all of this using an SSH terminal window; so as soon as your board flashes that it has power, open up an SSH terminal window using PuTTY, or some similar terminal emulator. Once the terminal window comes up, log in with your username and password. Now, type in `cat /proc/asound/cards`. You should see the response, as shown in the following screenshot:



```

pi@raspberrypi: ~
pi@raspberrypi ~ $ emacs example2.cpp
pi@raspberrypi ~ $ g++ example2.cpp -o example2
example2.cpp: In function 'int main()':
example2.cpp:6:3: error: 'a' was not declared in this scope
pi@raspberrypi ~ $ emacs example2.cpp
pi@raspberrypi ~ $ g++ example2.cpp -o example2
pi@raspberrypi ~ $ ./example2
Input value: 9
Input second value: 2
11
pi@raspberrypi ~ $ cat /proc/asound/cards
 0 [ALSA          ]: BRCM bcm2835 ALSbcm2835 ALSA - bcm2835 ALSA
                        bcm2835 ALSA
 1 [Device        ]: USB-Audio - C-Media USB Audio Device
                        C-Media USB Audio Device at usb-bcm2708_usb-1.2, full speed
pi@raspberrypi ~ $

```

Notice that the system thinks that there are two possible audio devices. The first is the internal Raspberry Pi audio that is connected to the audio port, and the second is your USB audio plugin. Although you could use the USB audio plugin to record sound and the Raspberry Pi audio out to play the sound, it is easier to just use the USB audio plugin to both create and record sound.

First, let's play some music to test if the USB sound device is working. You'll need to configure your system to look for your USB audio plugin, and use it as the default plugin to play and record sound. To do this, you'll need to add a couple of libraries to your system. The first of these are some **Advanced Linux Sound Architecture (ALSA)** libraries. It will enable your sound system on Raspberry Pi by performing the following steps:

1. Firstly, install two libraries associated with ALSA by typing `sudo apt-get install alsa-base alsa-utils`.
2. Then, install some files that help provide the sound library by typing `sudo apt-get install libasound2-dev`.

If your system already contains these libraries, Linux will simply tell you that they are already installed, or that they are up to date. After installing both libraries, reboot your Raspberry Pi. It takes time, but the system needs a reboot after new libraries or hardware is installed.

Now we'll use an application named **alsamixer** to control the volume of both, the input and the output, of our USB sound card. To do this, perform the following steps:

1. Type `alsamixer` at the Command Prompt. You should see a screen that will look like the following screenshot:

[illegible]

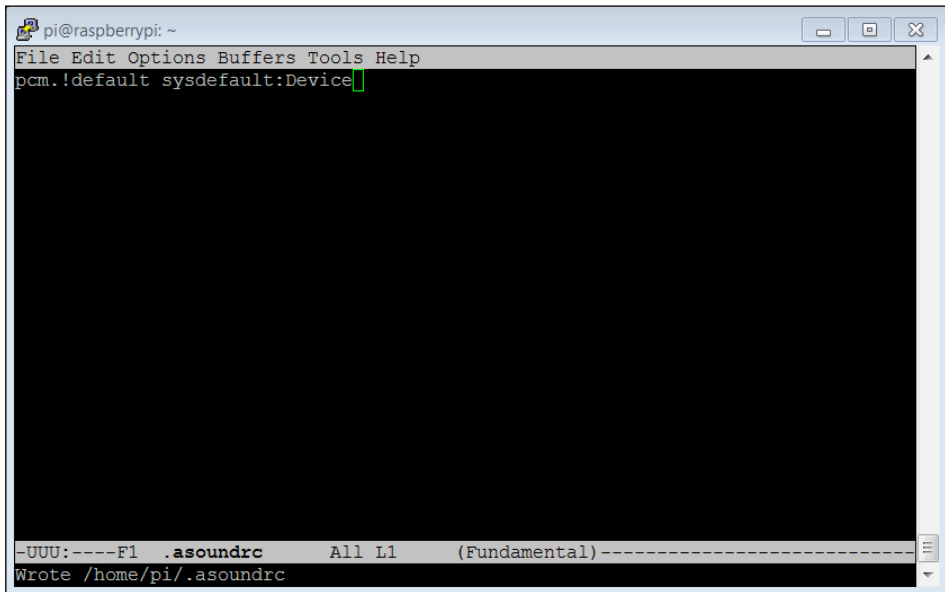
2. Press *F6* and select your USB sound device using the arrow keys. For example, refer to the following screenshot:

[illegible]

If this did not work, try `sudo aplay -l`. Once you have added the libraries, you'll need to create a file for your system. You are going to add a file in your home directory with the name `.asoundrc`. This will be read by your system and used to set your default configuration. To do this, perform the following steps:

1. Open the file named `.asoundrc` using your favorite editor.
2. Type in `pcm.!default sysdefault:Device`.
3. Save the file.

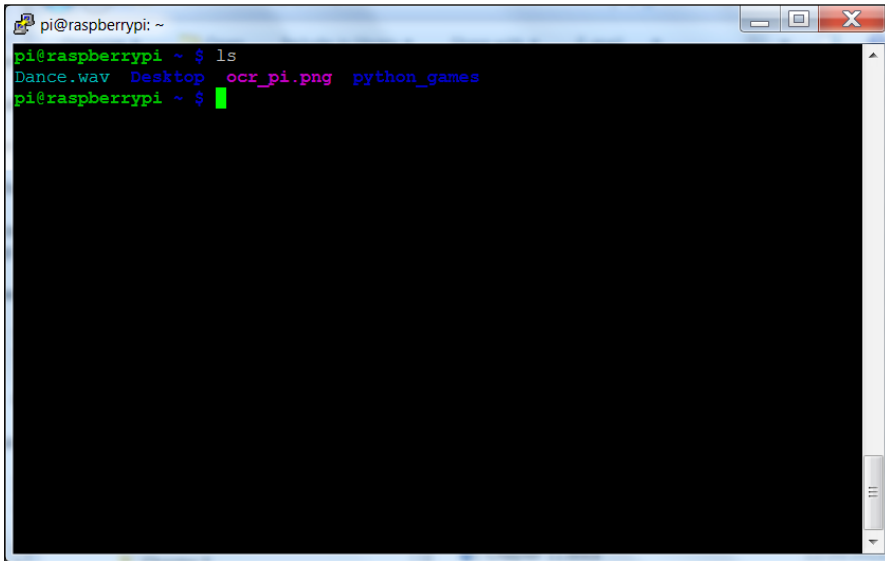
The file should look as follows:



This will tell the system to use your USB device as default. Once you have completed this, reboot your system again.

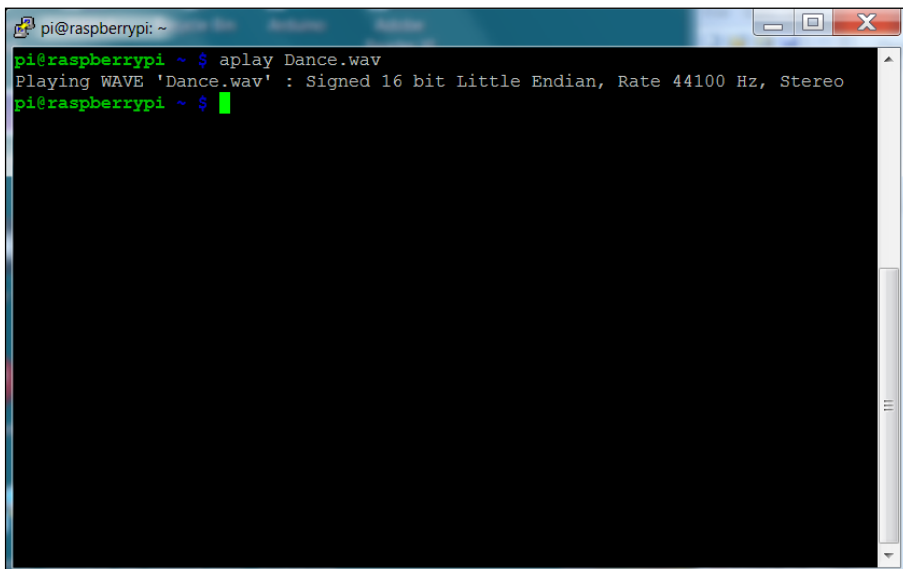
Now, we'll play some music. To do this, you need a sound file and a device to play it. I used **WinSCP** from my Windows machine to transfer a simple `.wav` file to my Raspberry Pi. If you are using a Linux machine as your host, you can also use `scp` from the command line to transfer the file. You could also just download some music to Raspberry Pi using a web browser if you have a keyboard, mouse, and display connected.

You can use an application named `aplay` to play your sound. You should see the music file by simply typing `ls`, which lists the files in this directory, as shown in the following screenshot:



```
pi@raspberrypi: ~  
pi@raspberrypi ~ $ ls  
Dance.wav Desktop ocr_pi.png python_games  
pi@raspberrypi ~ $
```

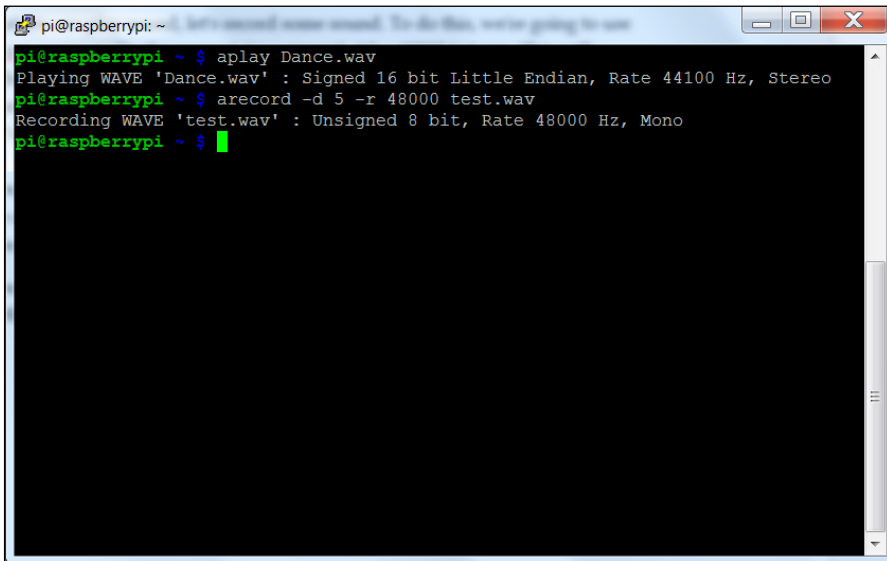
Now, type `aplay Dance.wav` to see if you can play music using the `aplay` music player. You will see the result (and hopefully hear it), as shown in the following screenshot:



```
pi@raspberrypi: ~  
pi@raspberrypi ~ $ aplay Dance.wav  
Playing WAVE 'Dance.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo  
pi@raspberrypi ~ $
```

If you don't hear any music, check the volume you set with `alsamixer` and the power cable of your speaker. Also, `aplay` can be a bit finicky about the type of files it accepts, so you may have to try different `.wav` files until `aplay` accepts one. One more thing to try if the system doesn't seem to know about the program is to type `sudo aplay Dance.wav`.

Now that we can play sound, let's record some sound. To do this, we're going to use the `arecord` program. At the prompt, type `arecord -d 5 -r 48000 test.wav`. This will record the sound at a sample rate of 48000 Hz per 5 seconds. Once you have typed the command, either speak into the microphone or make some other recognizable sound. You should see the following output on the terminal:

A screenshot of a terminal window titled 'pi@raspberrypi: ~'. The window shows the following commands and their outputs:

```
pi@raspberrypi ~ $ aplay Dance.wav
Playing WAVE 'Dance.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
pi@raspberrypi ~ $ arecord -d 5 -r 48000 test.wav
Recording WAVE 'test.wav' : Unsigned 8 bit, Rate 48000 Hz, Mono
pi@raspberrypi ~ $
```

The terminal has a black background with green text. The window has standard Linux window controls (minimize, maximize, close) in the top right corner.

Once you have created the file, play it with `aplay`. Type `aplay test.wav` and you should hear the recording. If you can't hear your recording, check `alsamixer` to make sure your speakers and microphone are both unmuted.

Now you can play music or other sound files using your Raspberry Pi. You can change the volume of your speaker, and record your voice or other sounds on the system. You're now ready for the next step.

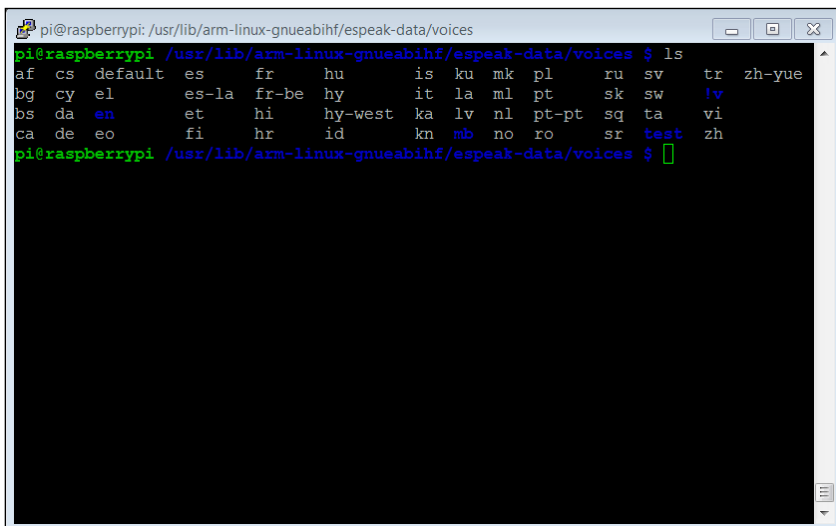
Using Espeak to allow our projects to respond in a robotic voice

Sound is an important tool in our robotic toolkit, but you will want to do more than just play music. Let's make our robot speak. You're going to start by enabling Espeak, an open source application that provides us with a computer voice. Espeak is an open source voice generation application. To get this free functionality, download the Espeak library by typing `sudo apt-get install espeak` at the prompt. The download may take a while, but the prompt will reappear when it is complete.

Now, let's see if Raspberry Pi has a voice. Type the `espeak "hello"` command. The speaker should emit a computer-voiced hello. If it does not, check the speakers and the volume level.

Now that we have a computer voice, you may want to customize it. Espeak offers a fairly complete set of customization features, including a large number of languages, voices, and other options. To access these, you can type in the options at the command-line prompt. For example, type in `espeak -v+f3 "hello"` and you should hear a female voice. You can add a Scottish accent by typing `espeak -ven-sc+f3 "hello"`. Once you have selected the kind of voice you'd like for your projects, you can make it the default setting so that you don't always have to include it in the command line.

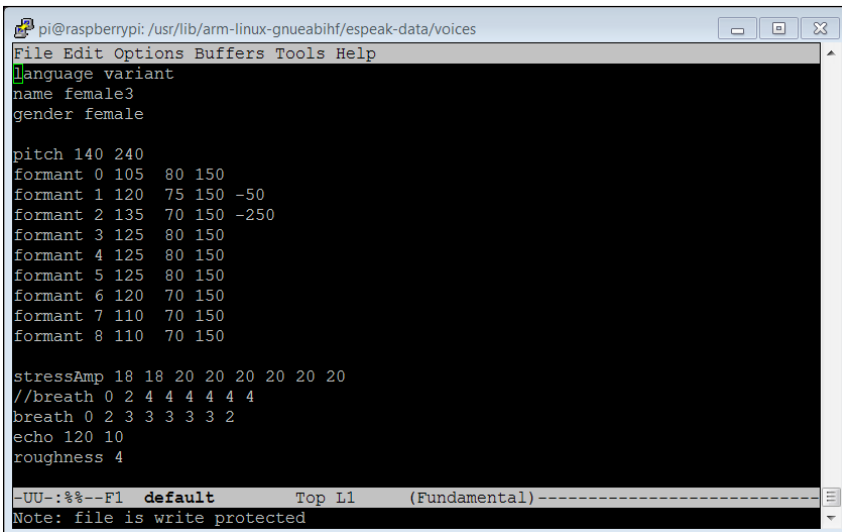
To create the default settings, go to the default file definition for espeak, which is in the `/usr/lib/arm-linux-gnueabi/hf/espeak-data/voices` directory. You should see something as shown in the following screenshot:



```

pi@raspberrypi: /usr/lib/arm-linux-gnueabi/hf/espeak-data/voices
pi@raspberrypi /usr/lib/arm-linux-gnueabi/hf/espeak-data/voices $ ls
af  cs  default  es   fr   hu   is  ku  mk  pl   ru  sv   tr  zh-yue
bg  cy  el       es-la fr-be hy   it  la  ml  pt   sk  sw   !v
bs  da  en       et   hi   hy-west ka  lv  nl  pt-pt sq  ta   vi
ca  de  eo       fi   hr   id   kn  mb  no  ro   sr  test  zh
pi@raspberrypi /usr/lib/arm-linux-gnueabi/hf/espeak-data/voices $
  
```


The default file is the one that Espeak uses to choose a voice. To get your desired voice, say one with a female tone, you need to copy a file into the default file. The file, that is, the female tone, is in the `!v` directory. Type `\!v` whenever you want to specify this directory. We need to type the `\` character because the `!` character is a special character in Linux, and if we want to use it as a regular old character, we need to put a `\` character before it. Before starting the process, copy the current default into a file named `default.old`, so that it can be retrieved later if needed. The next step is to copy the `f3` voice as your default file. Type the `sudo cp ./\!v/f3 default` command. If you open this default file, it will look as follows:



```
pi@raspberrypi: /usr/lib/arm-linux-gnueabi/hf/espeak-data/voices
File Edit Options Buffers Tools Help
language variant
name female3
gender female

pitch 140 240
formant 0 105 80 150
formant 1 120 75 150 -50
formant 2 135 70 150 -250
formant 3 125 80 150
formant 4 125 80 150
formant 5 125 80 150
formant 6 120 70 150
formant 7 110 70 150
formant 8 110 70 150

stressAmp 18 18 20 20 20 20 20 20
//breath 0 2 4 4 4 4 4 4
breath 0 2 3 3 3 3 3 2
echo 120 10
roughness 4

-UU-:%%--F1 default Top L1 (Fundamental)-----
Note: file is write protected
```

This has all the settings for your female voice. Now you can simply type `espeak` and the desired text. You will now get your female computer voice.

Now your project can speak. Simply type `espeak` followed by the text you want to speak in quotes and out comes your speech. If you want to read an entire text file, you can do that as well using the `-f` option and then typing the name of the file. Try this by using your editor to create a text file called `speak`; then type the `espeak -f speak.txt` command.

There are a lot of choices with respect to the voices you might use with `espeak`. Feel free to play around and choose your favorite. Then, edit the default file to set it to that voice. However, don't expect that you'll get the kind of voices that you hear from computers in the movies; those are actors and not computers. Although one day we will hopefully reach a stage where computers will sound a lot more like real people.

Using PocketSphinx to accept your voice commands

Sound is cool and speech is even cooler, but you'll also want to be able to communicate with your projects through voice commands. This section will show you how to add speech recognition to your robotic projects. This isn't nearly as simple as the speaking part, but thankfully, you have some significant help from the open source development community. You are going to download a set of capabilities named PocketSphinx, which will allow our project to listen to our commands.

The first step is downloading the PocketSphinx capabilities. Unfortunately, this is not quite as user-friendly as the `espeak` process, so follow along carefully. There are two possible ways to do this. If you have a keyboard, mouse, and display connected or want to connect through `vncserver`, you can do this graphically by performing the following steps:

1. Go to the Sphinx website hosted by **Carnegie Mellon University (CMU)** at <http://cmusphinx.sourceforge.net/>. This is an open source project that provides you with the speech recognition software. With our smaller, embedded system, we will be using the PocketSphinx version of this code.
2. You will need to download two pieces of software modules — `sphinxbase` and `PocketSphinx`. Select the **Download** option at the top of the page and then find the latest version of both of these packages. Download the `.tar.gz` version of these and move them to the `/home/pi` directory of your Raspberry Pi.

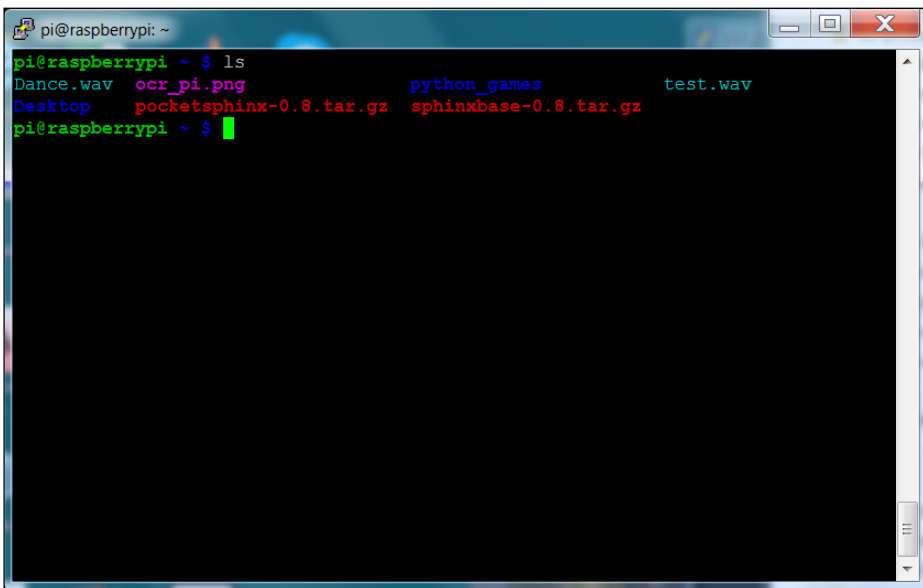
Another way to accomplish this is to use `wget` directly from the command prompt of Raspberry Pi. If you want to do it this way, perform the following steps:

1. To use `wget` on your host machine, find the link to the file you wish to download. In this case, go to the Sphinx website hosted by CMU at <http://cmusphinx.sourceforge.net/>. This is an open source project that provides you with the speech recognition software. With your smaller, embedded system, you will be using the PocketSphinx version of this code.
2. You will need to download two pieces of software modules, namely `sphinxbase` and `PocketSphinx`. Select the **Download** option at the top of the page and then find the latest version of both these packages. Right-click on the `sphinxbase-0.8.tar.gz` file (if 0.8 is the latest version) and select **Copy Link Location**. Now open a PuTTY window in Raspberry Pi, and after logging in, type `wget` and paste the link you just copied. This will download the `.tar.gz` version of `sphinxbase`. Now follow the same procedure with the latest version of `PocketSphinx`.

Before you build these, you need two libraries. The first library is `libasound2-dev`. If you skipped the first two objectives of this chapter, you'll need to download it now, using `sudo apt-get install libasound2-dev`. If you're unsure whether or not it's installed, try it again. The system will let you know if it's already installed.

The second of these libraries is called **Bison**. This is a general purpose, open source parser that will be used by PocketSphinx. To get this package, type `sudo apt-get install bison`.

Once everything is installed and downloaded, you can build PocketSphinx. Firstly, your home directory, with the `tar.gz` files of both, PocketSphinx and sphinxbase, should look like as the following screenshot:



To unpack and build the sphinxbase module, type `sudo tar -xzf sphinx-base-0.y.tar.gz`, where `y` is the version number; in our example, it is 8. This should unpack all the files from the archive into a directory named `sphinxbase-0.8`. Now type `cd sphinxbase-0.8`. The listing of the files should look something like the following screenshot:

```

pi@raspberrypi: ~/sphinxbase-0.8
sphinxbase-0.8/win32/sphinxbase/
sphinxbase-0.8/win32/sphinxbase/sphinxbase.vcxproj
sphinxbase-0.8/win32/sphinxbase/sphinxbase.vcxproj.filters
sphinxbase-0.8/win32/sphinx_jsgf2fsg/
sphinxbase-0.8/win32/sphinx_jsgf2fsg/sphinx_jsgf2fsg.vcxproj.filters
sphinxbase-0.8/win32/sphinx_jsgf2fsg/sphinx_jsgf2fsg.vcxproj
sphinxbase-0.8/win32/sphinx_pitch/
sphinxbase-0.8/win32/sphinx_pitch/sphinx_pitch.vcxproj
sphinxbase-0.8/win32/sphinx_pitch/sphinx_pitch.vcxproj.filters
sphinxbase-0.8/win32/sphinx_cepview/
sphinxbase-0.8/win32/sphinx_cepview/sphinx_cepview.vcxproj
sphinxbase-0.8/win32/sphinx_cepview/sphinx_cepview.vcxproj.filters
sphinxbase-0.8/win32/sphinx_lm_convert/
sphinxbase-0.8/win32/sphinx_lm_convert/sphinx_lm_convert.vcxproj.filters
sphinxbase-0.8/win32/sphinx_lm_convert/sphinx_lm_convert.vcxproj
pi@raspberrypi ~ $ cd sphinxbase-0.8/
pi@raspberrypi ~/sphinxbase-0.8 $ ls
aclocal.m4      config.sub      group           Makefile.am    sphinxbase.pc.in
AUTHORS         configure      include        Makefile.in    sphinxbase.sln
autogen.sh     configure.in   INSTALL        missing        src
ChangeLog      COPYING       install-sh     NEWS           test
config.guess   depcomp       ltmain.sh     python         win32
config.rpath   doc           m4            README         yllwrap
pi@raspberrypi ~/sphinxbase-0.8 $

```

To build the application, start by issuing the `sudo ./configure --enable-fixed` command. This command will check that everything is okay with the system, and then configure a build.

Now you are ready to actually build the `sphinxbase` code base. This is a two-step process, which is as follows:

1. Type `sudo make`, and the system will build all the executable files.
2. Type `sudo make install`, and this will install all the executables onto the system.

Now we need to make the second part of the system — the `PocketSphinx` code itself. Go to the home directory, and decompress and unarchive the code by typing `tar -xvzf pocketsphinx-0.8.tar.gz`. The files should now be unarchived, and we can now build the code. Installing these files is a three-step process as follows:

1. Type `cd pocketsphinx-0.8` to go to the `PocketSphinx` directory, and then type `sudo ./configure` to see if we are ready to build the files.
2. Type `sudo make` and wait for a while for everything to build.
3. Type `sudo make install`.



Several possible additions to our library installations will be useful later if you are going to use your PocketSphinx capability with Python as a coding language. You can install Python-Dev using `sudo apt-get install python-dev`. Similarly, you can get Cython using `sudo apt-get install cython`. You can also choose to install `pkg-config`, a utility that can sometimes help deal with complex compiles. Install it using `sudo apt-get install pkg-config`.

Once the installation is complete, you'll need to let the system know where our files are. To do this, you will need to edit the `/etc/ld.so.conf` path as the root by typing `sudo emacs /etc/ld.so.conf`. You will add the last line to the file, so it should now look like the following screenshot:

```
pi@raspberrypi: ~  
File Edit Options Buffers Tools Conf Help  
include /etc/ld.so.conf.d/*.conf  
/usr/local/lib
```

Now type `sudo /sbin/ldconfig`, and the system will now be aware of your PocketSphinx libraries. You may want to reboot at this point, just to make sure everything is installed and set up.

Now that everything is installed, you can try our speech recognition. Type `cd /home/pi/pocketsphinx-0.8/src/programs` to go to a directory to try a demo program; then type `./pocketsphinx_continuous`. This program takes input from the microphone and turns it into speech. After running the command, you'll get a lot of irrelevant information, and then you will see the following screenshot:

```

pi@raspberrypi: ~/pocketsphinx-0.8/src/programs
INFO: ngram_model_dmp.c(288): 436879 = LM.bigrams(+trailer) read
INFO: ngram_model_dmp.c(314): 418286 = LM.trigrams read
INFO: ngram_model_dmp.c(339): 37293 = LM.prob2 entries read
INFO: ngram_model_dmp.c(359): 14370 = LM.bo wt2 entries read
INFO: ngram_model_dmp.c(379): 36094 = LM.prob3 entries read
INFO: ngram_model_dmp.c(407): 854 = LM.tseg_base entries read
INFO: ngram_model_dmp.c(463): 5001 = ascii word strings read
INFO: ngram_search_fwdtree.c(99): 788 unique initial diphones
INFO: ngram_search_fwdtree.c(147): 0 root, 0 non-root channels, 60 single-phone
words
INFO: ngram_search_fwdtree.c(186): Creating search tree
INFO: ngram_search_fwdtree.c(191): before: 0 root, 0 non-root channels, 60 singl
e-phone words
INFO: ngram_search_fwdtree.c(326): after: max nonroot chan increased to 13428
INFO: ngram_search_fwdtree.c(338): after: 457 root, 13300 non-root channels, 26
single-phone words
INFO: ngram_search_fwdflat.c(156): fwdflat: min_ef width = 4, max_sf_win = 25
INFO: continuous.c(371): /home/pi/pocketsphinx-0.8/src/programs/.libs/lt-pockets
phinx_continuous COMPILED ON: Nov 8 2013, AT: 18:29:54

Warning: Could not find Mic element
Warning: Could not find Capture element
READY....

```

The INFO and Warning statements come from the C or C++ code and are there for debugging purposes. Initially, they will warn you that they cannot find your Mic and Capture elements, but when the Raspberry Pi finds them, it will print out READY.... If you have set things up as previously described, you should be ready to give your Raspberry Pi a command. Say *hello* into the microphone. When it senses that you have stopped speaking, it will process your speech and give lots of irrelevant information again, but it should eventually show the commands, as in the following screenshot:

```

pi@raspberrypi: ~/pocketsphinx-0.8/src/programs
INFO: ngram_search_fwdtree.c(1557): 2844 words for which last channels evalu
ated (40/fr)
INFO: ngram_search_fwdtree.c(1560): 25308 candidate words for entering last p
hone (361/fr)
INFO: ngram_search_fwdtree.c(1562): fwdtree 1.44 CPU 2.057 xRT
INFO: ngram_search_fwdtree.c(1565): fwdtree 2.68 wall 3.823 xRT
INFO: ngram_search_fwdflat.c(302): Utterance vocabulary contains 45 words
INFO: ngram_search_fwdflat.c(937): 457 words recognized (7/fr)
INFO: ngram_search_fwdflat.c(939): 39459 senones evaluated (564/fr)
INFO: ngram_search_fwdflat.c(941): 46877 channels searched (669/fr)
INFO: ngram_search_fwdflat.c(943): 2577 words searched (36/fr)
INFO: ngram_search_fwdflat.c(945): 2051 word transitions (29/fr)
INFO: ngram_search_fwdflat.c(948): fwdflat 0.22 CPU 0.314 xRT
INFO: ngram_search_fwdflat.c(951): fwdflat 0.22 wall 0.318 xRT
INFO: ngram_search.c(1266): lattice start node <s>.0 end node </s>.61
INFO: ngram_search.c(1294): Eliminated 0 nodes before end node
INFO: ngram_search.c(1399): Lattice has 63 nodes, 10 links
INFO: ps_lattice.c(1365): Normalizer P(O) = alpha(</s>:61:68) = -418369
INFO: ps_lattice.c(1403): Joint P(O,S) = -422552 P(S|O) = -4183
INFO: ngram_search.c(888): bestpath 0.00 CPU 0.000 xRT
INFO: ngram_search.c(891): bestpath 0.01 wall 0.010 xRT
000000000: hello
READY....

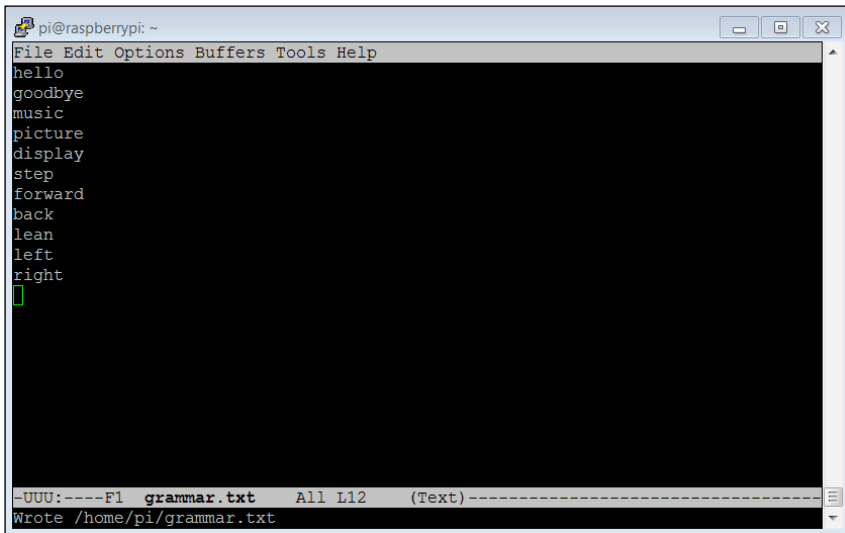
```

Notice the 000000000: hello command. It recognized your speech! You can try other words and phrases too. The system is very sensitive, so it may pick up background noise. You are also going to find that it is not very accurate. We'll deal with that in a moment. To stop the program, type `cntrl-c`.

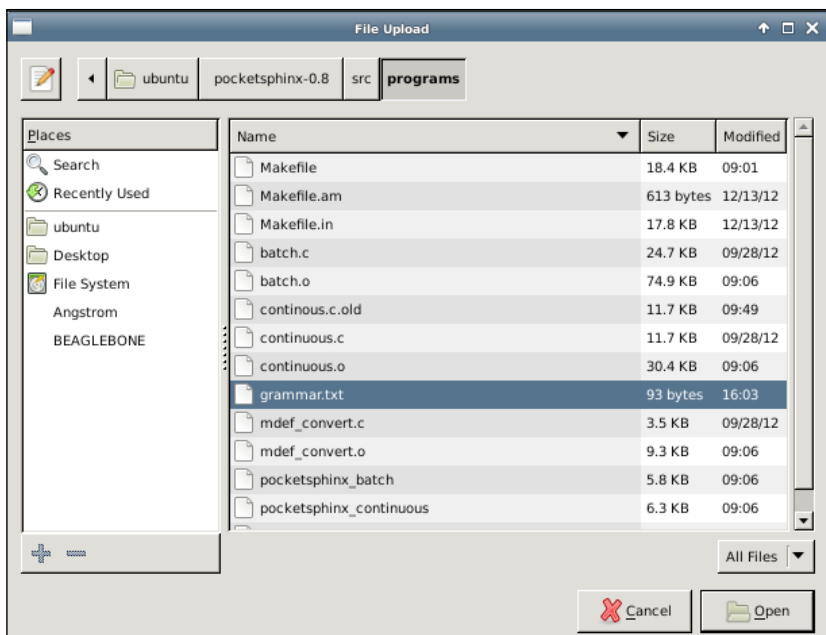
There are two ways to make your voice recognition more accurate. One is to train the system to understand your voice more accurately. This is a bit complex yet, if you want to know more, go to the PocketSphinx website of CMU.

The second way to improve accuracy is to limit the number of words that your system uses to determine what you are saying. The default has literally thousands of word possibilities, so if two words are close, PocketSphinx may choose the wrong word. To avoid this, you can make your own dictionary to restrict the words it has to choose from.

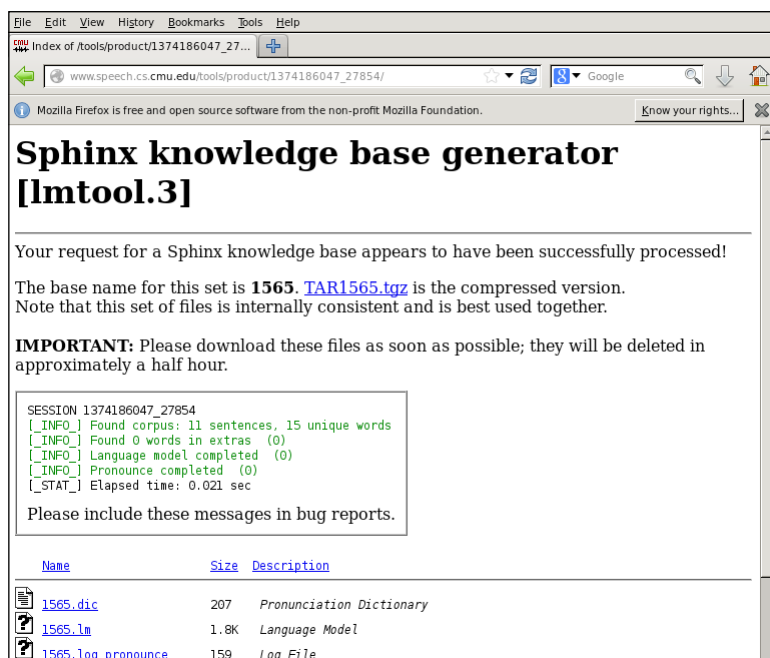
The first step is to create a file with the words or phrases that you want the system to recognize. Then, you use a web tool to create two files that the system will use to define your grammar. I'll do this through the `vncserver` command because I'll need to use a web browser on Raspberry Pi to turn a text file into a set of grammar files. Begin by editing a file; type `emacs grammar.txt` and insert the text, as shown in the following screenshot:



Now you must use the CMU web tool to turn this file into two files that the system can use to define its dictionary. Open a web browser window (the Epiphany web browser is part of the default debian package) and go to <http://www.speech.cs.cmu.edu/tools/lmtool-new.html>. If you hit the **Browse** button, you can find and select the file. It should look something like the following screenshot:



Open the `grammar.txt` file; then, on the web page, select **COMPILE KNOWLEDGE BASE**, and a window should pop up, as shown in the following screenshot:



You need to download the `.tgz` file created; in this case, the `TAR1565.tgz` file. This will download into your `/home/pi/` directory. Move it to the `/home/pi/pocketsphinx-0.8/src/programs` directory and unarchive it by using `tar -xzf` and the filename.

Now you can invoke the `pocketsphinx_continuous` program to use this dictionary by typing `./pocketsphinx_continuous -lm 1565.lm -dict 1565.dic`, and it will look in that directory to find matches to your commands.

You can also do this on your remote computer using Windows or Linux, by creating the file in a text editor such as WordPad or Emacs. Once you have created the required grammar files, you can download them to your Raspberry Pi using WinSCP, if you are using Windows or `scp` from the command line, if you are using Linux.

Your system can now understand your specific set of commands! In the next section of this chapter, you'll learn how to use this input to have the project respond.

Interpreting commands and initiating actions

Now that the system can both hear and speak, you'll want to provide the capability to respond to your speech and execute some commands based on the speech input. Next, you're going to configure the system to respond to simple commands.

In order to respond, we're going to edit the `continuous.c` code in the `/home/pi/pocketsphinx-0.8/src/programs` directory. We could create our own C file, but this file is already set up in the `makefile` system and is an excellent starting spot. You can save a copy of the current file in `continuous.c.old` so that you can always get back to the starting program, if required. Then, you will need to edit the `continuous.c` file. It is very long and a bit complicated, but you are specifically looking for the section in the code, which is shown in the following screenshot. Look for the comment line `/* Exit if the first word spoken was GOODBYE */`:

```

pi@raspberrypi: ~/pocketsphinx-0.8/src/programs
File Edit Options Buffers Tools C Help
ps_end_utt(ps);
hyp = ps_get_hyp(ps, NULL, &uttid);
printf("%s: %s\n", uttid, hyp);
fflush(stdout);

/* Exit if the first word spoken was GOODBYE */
if (hyp) {
    sscanf(hyp, "%s", word);
    if (strcmp(word, "goodbye") == 0)
        break;
}

/* Resume A/D recording for next utterance */
if (ad_start_rec(ad) < 0)
    E_FATAL("Failed to start recording\n");
}

cont_ad_close(cont);
ad_close(ad);
}

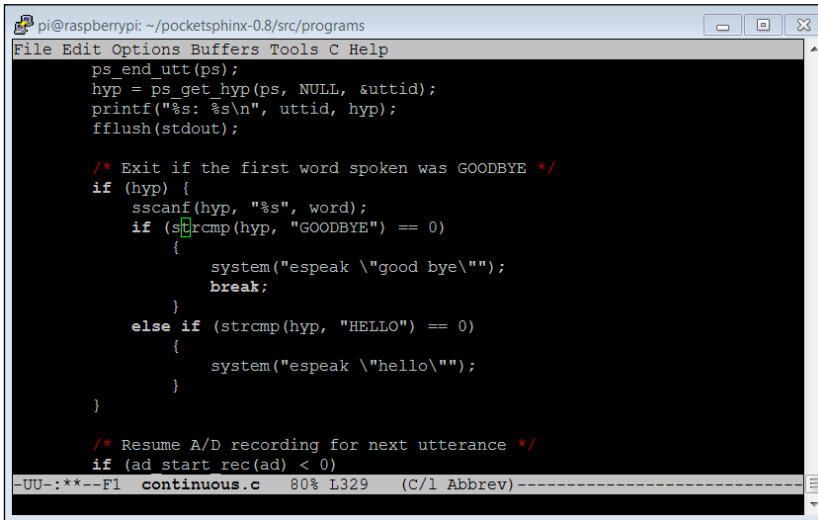
-UU-:----F1 continuous.c 82% L331 (C/l Abbrev)-----

```

In this section of the code, the word has already been decoded and is held in the `hyp` variable. You can add code here to make your system do things, based on the value associated with the word that we have decoded. First, let's try adding the capability to respond to *hello* and *goodbye* to see if we can get the program to stop. Make changes to the code in the following manner:

1. Find the comment `/* Exit if the first word spoken was GOODBYE */`.
2. In the statement `if (strcmp(hyp, "good bye") == 0)`, change `word` to `hyp` and `good bye` to `GOODBYE`.
3. Insert brackets around the `break;` statement and add the system `("espeak" \ "good bye\");` statement just before the `break;` statement.
4. Add the other `else if` statement to the clause by typing `else if (strcmp(hyp, "HELLO") == 0)`. Add brackets after the `else if` statement, and inside the brackets, type `system ("espeak" \ "good bye\");`.

The file should now look as follows:



```
pi@raspberrypi: ~/pocketsphinx-0.8/src/programs
File Edit Options Buffers Tools C Help
ps_end_utt(ps);
hyp = ps_get_hyp(ps, NULL, &uttid);
printf("%s: %s\n", uttid, hyp);
fflush(stdout);

/* Exit if the first word spoken was GOODBYE */
if (hyp) {
    sscanf(hyp, "%s", word);
    if (strcmp(hyp, "GOODBYE") == 0)
    {
        system("espeak \"good bye\"");
        break;
    }
    else if (strcmp(hyp, "HELLO") == 0)
    {
        system("espeak \"hello\"");
    }
}

/* Resume A/D recording for next utterance */
if (ad_start_rec(ad) < 0)
```

Now you need to rebuild your code. As the make system already knows how to build the program `pocketsphinx_continuous`, it will rebuild the application if you make a change to the `continuous.c` file at any point of time. Simply type `make`, and the file will compile and create a new version of `pocketsphinx_continuous`. To run your new version, type `./pocketsphinx_continuous -lm 1565.lm -dict 1565.dic`. Make sure you type the `./` command at the start.

If you haven't created your own dictionary, or would like to use the default dictionary, you can still have your robot respond. You'll just need to change the `GOODBYE` and `HELLO` in the `if (strcmp(hyp, "GOODBYE") == 0)` and `else if (strcmp(hyp, "HELLO") == 0)` statements to words that your system currently recognizes. Simply say your command, see what the system is printing for the word it is recognizing, and replace `"GOODBYE"` or `"HELLO"` with that word.

If everything is set correctly, saying *hello* should result in a response of *hello* from your Raspberry Pi. Saying *good bye* should elicit a response of *good bye* and also shut down the program. Notice that the system command can be used to run any program that runs with a command line. Now you can use this program to start and run other programs based on the commands.

Your Raspberry Pi will now listen, respond to, and execute the commands that you give it. By the way, you may be tempted to use Python to do this, but the reason that we did not use Python is that to get Python to recognize real-time speech for PocketSphinx, you would need a way to stream the data to the application. There is an example of this in the `pocketsphinx-0.8/src/gst-plugin` directory titled `livedemo.py`. If you would like to try this, keep in mind that you will need to install the gtk using `sudo apt-get install libgtk-3-dev` and `sudo apt-get install python-gtk2`. You will also need to install the gstreamer tools using `sudo apt-get install gstreamer1.0` and `sudo apt-get install python-gst0.10` onto Raspberry Pi. This requires a significant amount of disk space, as it does not come with the default release.

Summary

Now your project can both hear and speak. You can use these functions later when you want to interact with your project without typing commands or using a display. You should also feel more comfortable with installing new hardware and software into your system. We'll be using this skill throughout the book, as we look at more complex projects.

In the next chapter, we'll look at adding a capability that will allow your robots to see and use vision to track objects, motion, or whatever else your robot needs to track.

4

Adding Vision to Raspberry Pi

In the previous chapter, you learned how to communicate with Raspberry Pi through voice. In this chapter, you are going to add vision with a webcam; you'll use this functionality in lots of different applications. Fortunately, adding hardware and software for vision is both easy and inexpensive.

To do this, you'll have to add a USB webcam to your system. Having the standard USB interface on your board opens a wide range of amazing possibilities. Furthermore, there are several amazing open source libraries that offer complex capabilities, which you can use in your projects without spending months in coding them.

Vision will open a set of possibilities for your project. These can range from simple motion detection to advanced capabilities, such as facial recognition, object identification, and even object tracking. The robot can also use vision to detect its surroundings and avoid obstacles.

In this chapter, we will cover the following topics:

- Connecting your USB camera to your Raspberry Pi and viewing the images
- Connecting Raspberry Pi Camera board to your Raspberry Pi and viewing the images
- Downloading and installing OpenCV, a full-featured vision library
- Using the vision library to detect colored objects

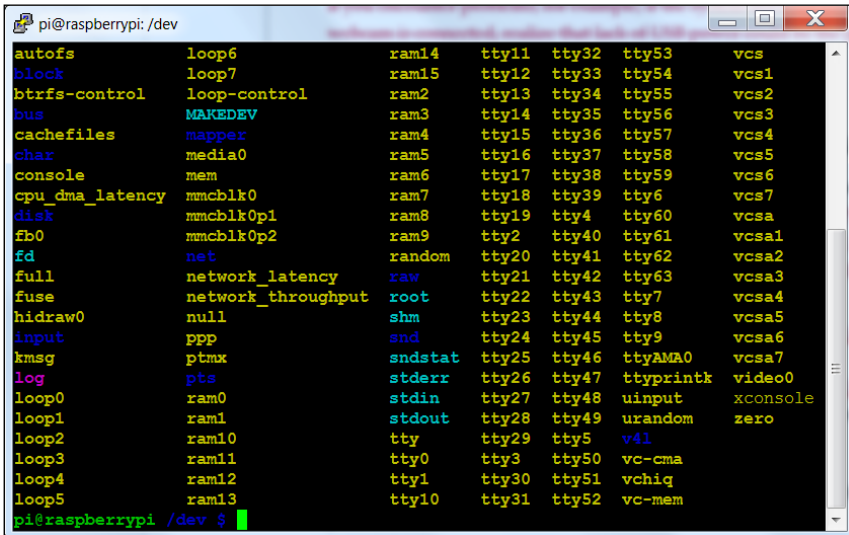
To add vision to your projects, you'll need to add a USB webcam; try to find a recently manufactured one. You may have an older webcam sitting on your project shelf, but it will probably cause problems as Linux may not have the driver support for these devices, and the money you save will not be worth the frustration you might experience later. You should stick to webcams from major players, such as Logitech or Creative Labs.

If you are using Raspberry Pi B+, you can connect your webcam directly to the Raspberry Pi. However, if you are using the Raspberry Pi A+, you'll want to connect this device through your powered USB hub.

Connecting the USB camera to Raspberry Pi and viewing the images

The first step in enabling computer vision is connecting the USB camera to the USB port. This example will use a Logitech model HD 720P. To access the USB webcam directly on Raspberry Pi, you can use a Linux program called `gview`. Install this by powering up the Raspberry Pi, logging in, and entering the `sudo apt-get install gview` command.

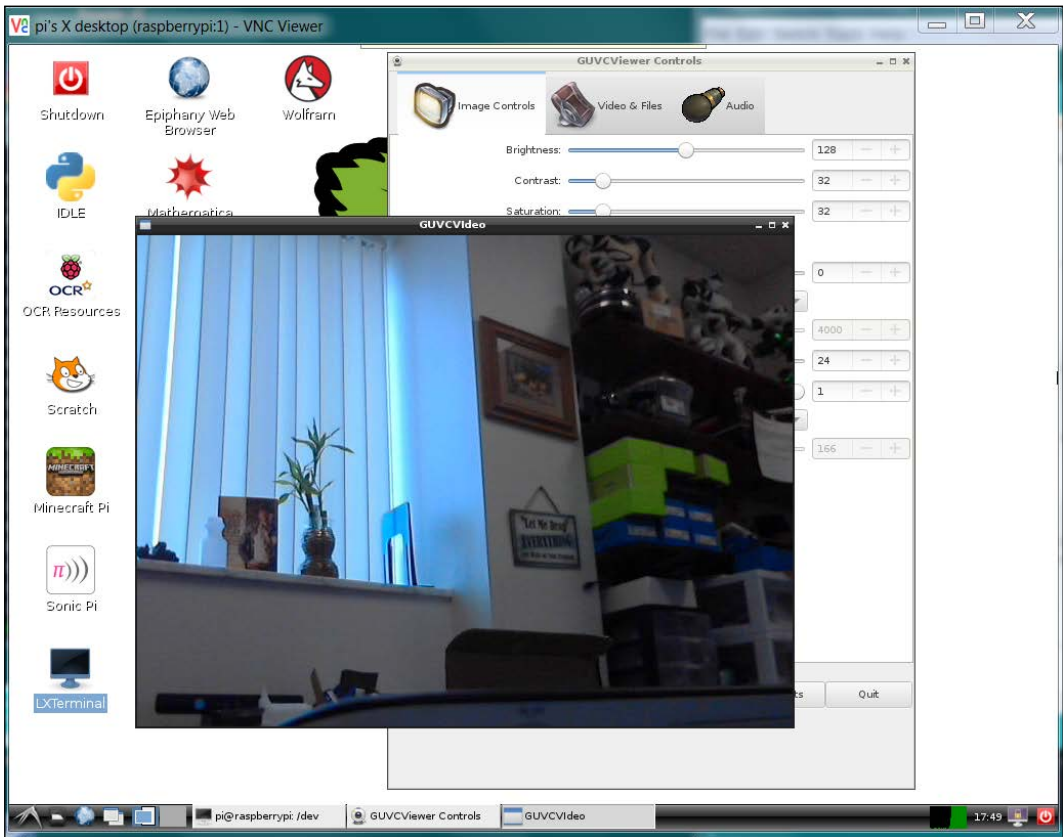
To try your USB camera, connect it and reboot your Raspberry Pi. To check if Raspberry Pi has found your USB camera, go to the `/dev` directory and type `ls`. What you should see is shown in the following screenshot:



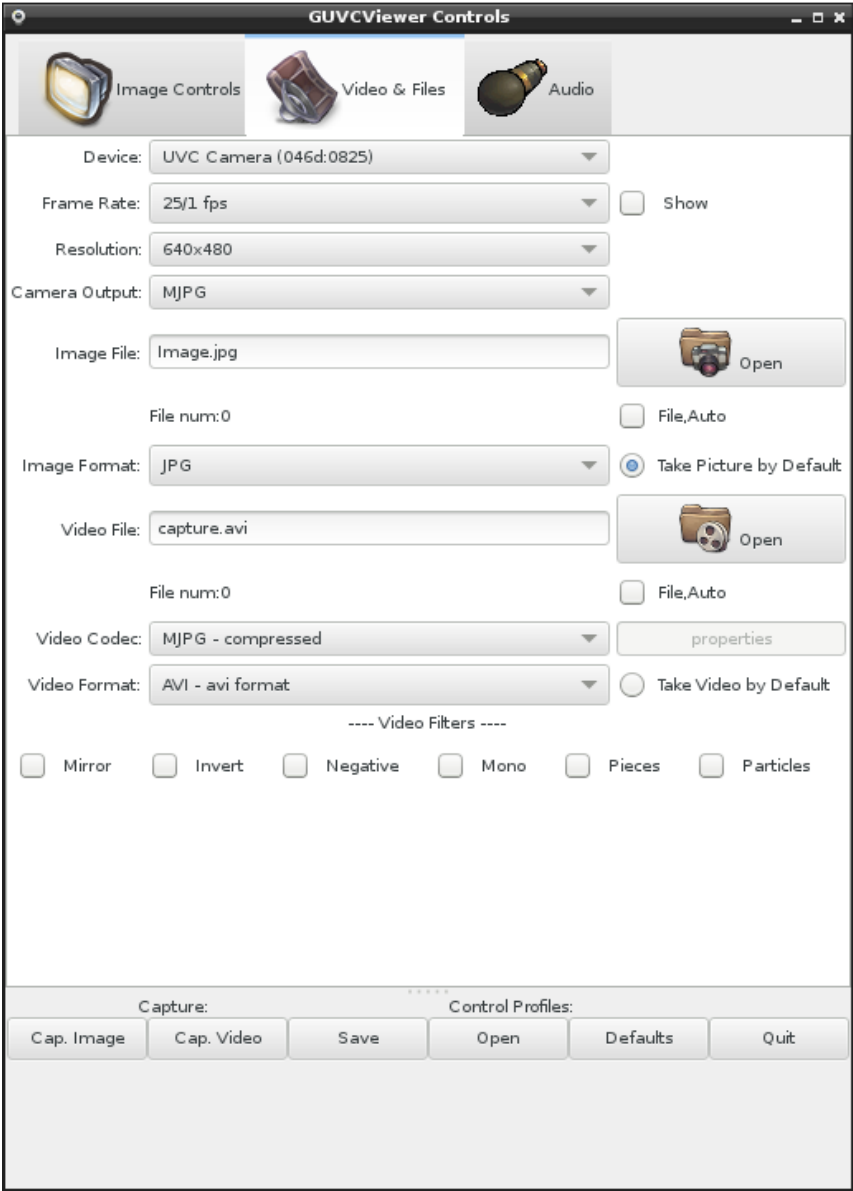
```
pi@raspberrypi: /dev
autofs      loop6       ram14       tty11       tty32       tty53       vcs
block       loop7       ram15       tty12       tty33       tty54       vcs1
btrfs-control loop-control ram2        tty13       tty34       tty55       vcs2
bus         MAKEDEV     ram3        tty14       tty35       tty56       vcs3
cachefiles  mapper      ram4        tty15       tty36       tty57       vcs4
char        media0      ram5        tty16       tty37       tty58       vcs5
console     mem         ram6        tty17       tty38       tty59       vcs6
cpu_dma_latency mmcblk0     ram7        tty18       tty39       tty6        vcs7
disk        mmcblk0p1  ram8        tty19       tty4        tty60       vcsa
fb0         mmcblk0p2  ram9        tty2        tty40       tty61       vcsa1
fd          net         random      tty20       tty41       tty62       vcsa2
full        network_latency raw          tty21       tty42       tty63       vcsa3
fuse        network_throughput root         tty22       tty43       tty7        vcsa4
hidraw0     null        shm         tty23       tty44       tty8        vcsa5
input       ppp         snd         tty24       tty45       tty9        vcsa6
kmsg        ptmx        sndstat     tty25       tty46       ttyAMA0     vcsa7
log         pts         stderr      tty26       tty47       ttyprintk   video0
loop0       ram0        stdin       tty27       tty48       uinput      xconsole
loop1       ram1        stdout      tty28       tty49       urandom     zero
loop2       ram10       tty         tty29       tty5       v4l
loop3       ram11       tty0        tty3        tty50       vc-cma
loop4       ram12       tty1        tty30       tty51       vchiq
loop5       ram13       tty10       tty31       tty52       vc-mem
```

Look for `video0`, as this is the entry for your webcam. If you see it, the system knows your camera is there.

Now, let's use `guvcview` to see the output of the camera. Since it will need to output some graphics, you either need to use a monitor connected to the board, as well as a keyboard and mouse, or you can use `vncserver`, as described in *Chapter 1, Getting Started with Raspberry Pi*. If you are going to use `vncserver`, make sure you start the server on Raspberry Pi by typing `vncserver` through SSH. Then, start up VNC Viewer, as described in *Chapter 1, Getting Started with Raspberry Pi*. Open a terminal window and type `sudo guvcview`. You should see something, as shown in the following screenshot:



The video window displays what the webcam sees, and the **GUVCViewer Controls** window controls the different characteristics of the camera. The default settings of the Logitech 720HD camera work fine. However, if you get a black screen for the camera, you may need to adjust the settings. Select the **GUVCViewer Controls** window and the **Video & Files** tab. You will see a window where you can adjust the settings for your camera, as shown in the following screenshot:



The most important setting is **Resolution**. If you see a black screen, lower the resolution; this will often resolve the issue. This window will also tell you what resolutions are supported by your camera. Also, you can display the frame rate by checking the box to the right of the **Frame Rate** setting. Be aware, however, that if you are going through vncviewer, the refresh rate (how quickly the video window will update itself) will be much slower than if you're using Raspberry Pi and a monitor directly.

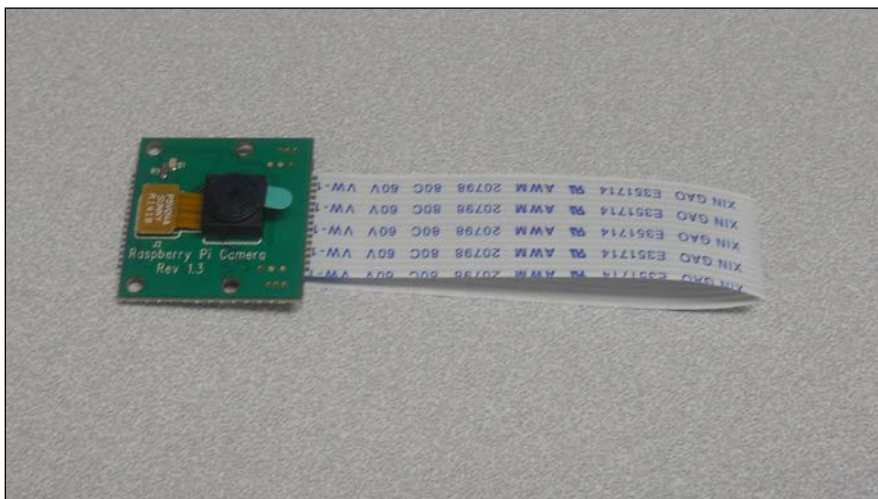
Once you have the camera up and running and the desired resolution set, you can go on to download and install OpenCV.



You can connect more than one webcam to the system. Follow the same steps, but connect to the cameras through a USB hub. List the devices in the /dev directory. Use gvcview to see the different images. One challenge, however, is that connecting too many cameras can overwhelm the bandwidth of the USB port.

Connecting the Raspberry Pi camera board and viewing the images

There is another way to input images into the Raspberry Pi. There is a camera specifically designed to use with the Raspberry Pi, the Raspberry Pi Camera Board. Here is a image of this product:



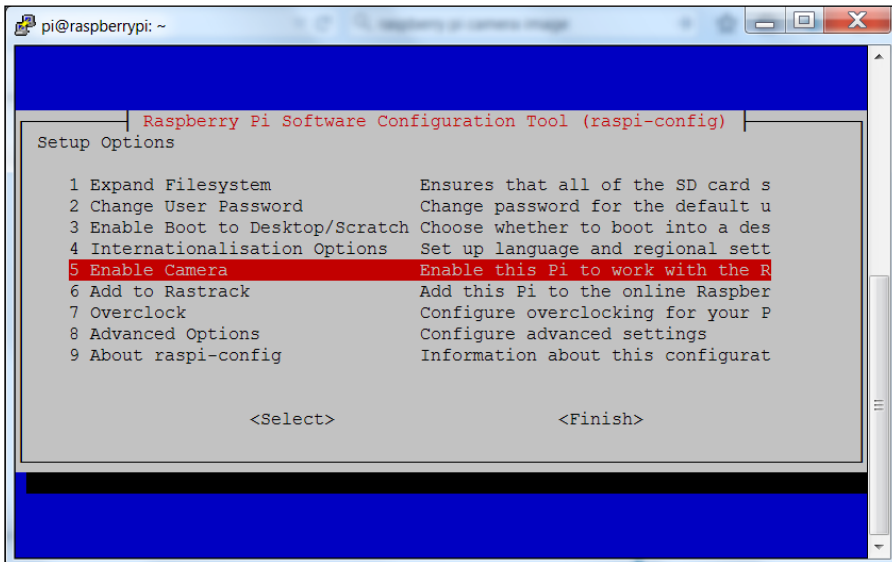
You may also want to purchase a case for the Raspberry Pi camera; this provides protection for the camera and makes it easier to mount into your project. Here is a image of the camera, mounted into its protective casing:



The camera connects to the Raspberry Pi by installing it into the connector marked camera on the Raspberry Pi. To see how this is to be done, see the video at <http://www.raspberrypi.org/help/camera-module-setup/>.

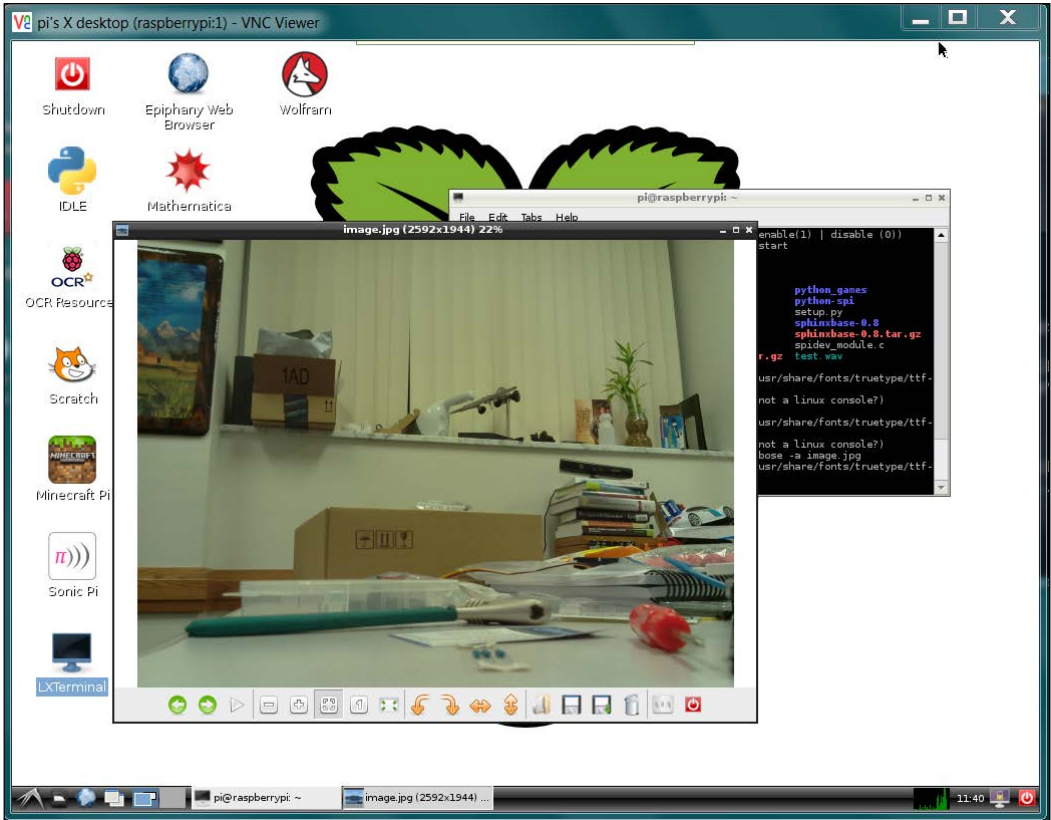
Once the device is connected, you can access the device by enabling it through the configuration utility. To enable the camera, Perform the following steps:

1. Run the configuration utility by typing `sudo raspi-config`.
2. Select the **Enable Camera**, as shown in the following screenshot:



3. Select the enable selection, then exit the utility, and choose to reboot the device.

To take a picture with the camera, simply type `raspistill -o image.jpg`. This will take a picture with the camera, then store the image in the file `image.jpg`. Once you have the picture, you can view it by opening the Raspberry Pi image viewer by selecting the lower left icon for **applications**, then **accessories**, and then **image viewer**. Open the file `image.jpg`, and you should see the result, as shown in the following screenshot:



If you are developing from a remote computer, you will want to open a `vncserver` connection between your computer and the Raspberry Pi. See *Chapter 1, Getting Started with Raspberry Pi* for details.

The last step is to add a Python library that will allow you to access the Raspberry Pi camera from Python. It is called **picamera**; to get this and the required libraries, type `sudo apt-get install python-picamera python3-picamera python-rpi.gpio`.

Downloading and installing OpenCV – a fully featured vision library

Now that you have your camera connected, you can begin to access some amazing capabilities that have been provided by the open source community. The most popular of these for computer vision is OpenCV. To do this, you'll need to install OpenCV. There are several possible ways of doing this; I'm going to suggest the ones that I follow to install it on my system. Once you have booted the system and opened a terminal window, type the following commands in the given order:

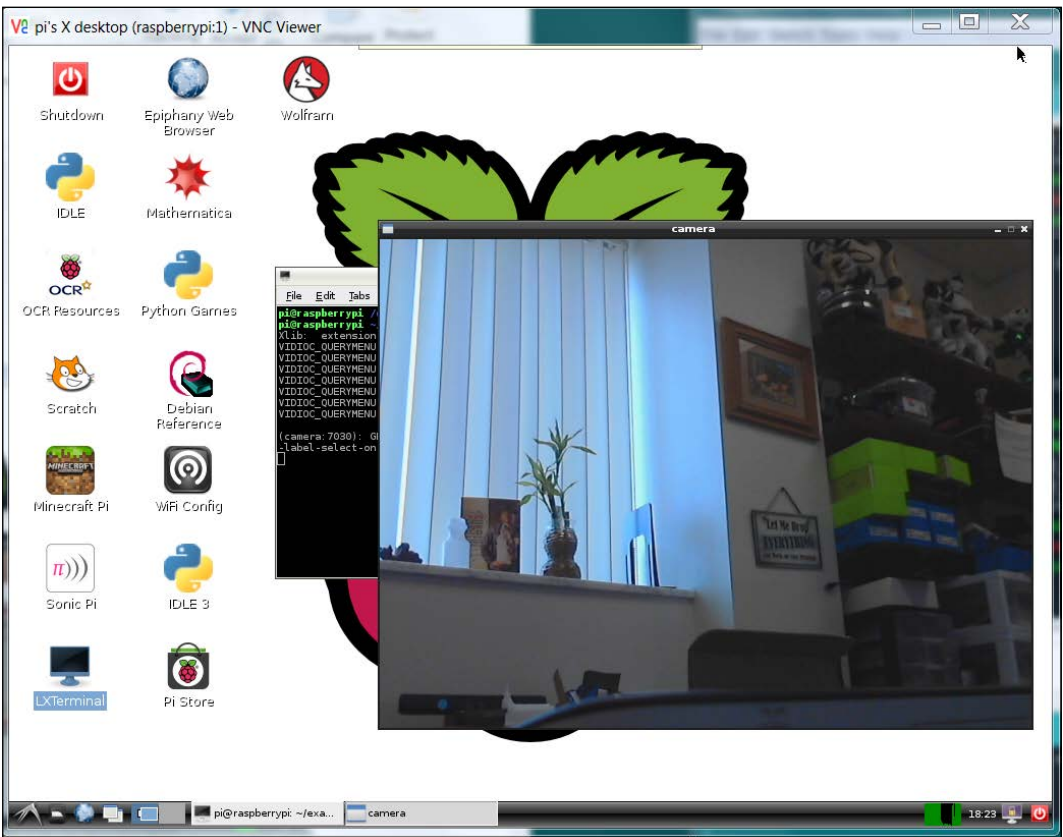
1. `sudo apt-get update`: If you haven't done this in a while, it is a good idea to do this now, before you start. You're going to download a number of new software packages, so it is good to make sure that everything is up-to-date.
2. `sudo apt-get install build-essential`: You should have done this in a previous chapter. In case you skipped that part, you will have to do it now, as you need this package.
3. `sudo apt-get install libavformat-dev`: This library provides a way to code and decode audio and video streams.
4. `sudo apt-get install ffmpeg`: This library provides a way to transcode audio and video streams.
5. `sudo apt-get install libcv2.4 libcvaux2.4 libhighgui2.4`: This command shows the basic OpenCV libraries. Note the number in the command. This will almost certainly change as new versions of OpenCV become available. If 2.4 does not work, you can either try 3.0, or Google for the latest version of OpenCV.
6. `sudo apt-get install python-opencv`: This is the Python development kit needed for OpenCV, as you are going to use Python.
7. `sudo apt-get install opencv-doc`: This command will show the documentation for OpenCV, just in case you need it.
8. `sudo apt-get install libcv-dev`: This command shows the header file and static libraries to compile OpenCV.
9. `sudo apt-get install libcvaux-dev`: This command shows more development tools for compiling OpenCV.
10. `sudo apt-get install libhighgui-dev`: This is another package that provides header files and static libraries to compile OpenCV.

Now type `cp -r /usr/share/doc/opencv-doc/examples /home/pi/`. This will copy all the examples to your home directory.

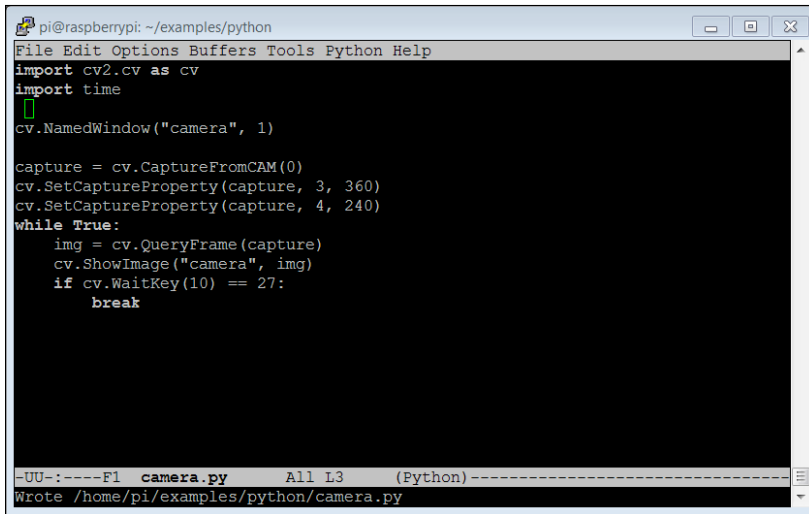
Now you are ready to try out the OpenCV library. I prefer to use Python while programming simple tasks; hence, I'll show the Python examples. If you prefer the C examples, feel free to explore. In order to use the Python examples, you'll need one more library. So type `sudo apt-get install python-numpy`, as you will need this to manipulate the matrices that OpenCV uses to hold images.

Now that you have these, you can try one of the Python examples. Switch to the directory with the Python examples by typing `cd /home/pi/examples/python`. In this directory, you will find a number of useful examples; you'll only look at the most basic, which is called `camera.py`. If `camera.py` is not created, you can create it by typing in the code shown in the next few pages.

You can try running this example; however, to do this, you'll either need to have a display connected to Raspberry Pi, or you can do this over the vncserver connection. If you are doing this with a webcam you can run the standard Python example. For this, bring up the **LXTerminal** window and type `python camera.py`. You should see something, as shown in the following screenshot:



The camera window is quite large; you can change the resolution of the image to a lower one, which will make the update rate faster, and the storage requirement for the image smaller. To do this, edit the `camera.py` file and add two lines, as shown in the following screenshot:

A screenshot of a terminal window on a Raspberry Pi. The window title is 'pi@raspberrypi: ~/examples/python'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Python', and 'Help'. The terminal displays the following Python code:

```
import cv2.cv as cv
import time

cv.NamedWindow("camera", 1)

capture = cv.CaptureFromCAM(0)
cv.SetCaptureProperty(capture, 3, 360)
cv.SetCaptureProperty(capture, 4, 240)
while True:
    img = cv.QueryFrame(capture)
    cv.ShowImage("camera", img)
    if cv.WaitKey(10) == 27:
        break
```

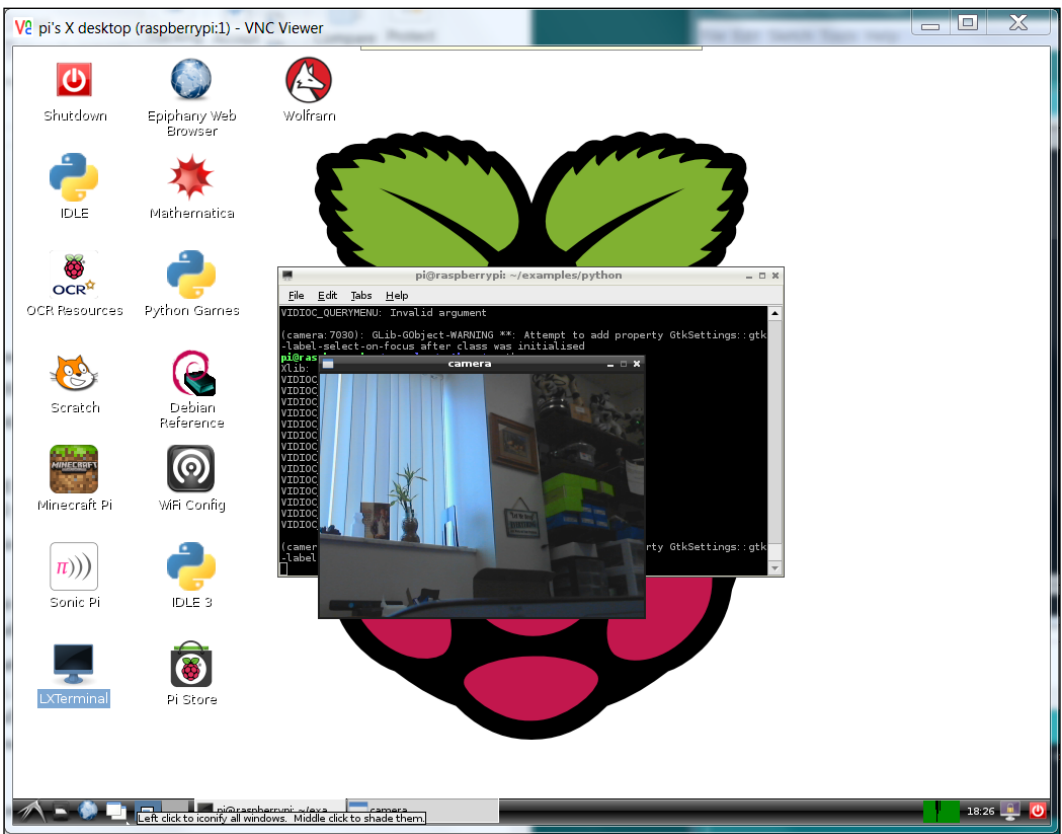
The status bar at the bottom shows '-UU-:----F1 camera.py All L3 (Python)-----' and a message 'Wrote /home/pi/examples/python/camera.py'.

Here is an explanation of the Python code:

- `import cv2.cv as cv`: This line imports the OpenCV library, so you can access its functionality.
- `import time`: This line imports the time library, so you can access the time functionality.
- `cv.NamedWindow("camera", 1)`: This line creates a window that you will use to display your image.
- `capture = cv.CaptureFromCAM(0)`: This line creates a structure that knows how to capture images from the connected webcam.
- `cv.SetCaptureProperty(capture, 3, 360)`: This line sets the image width to 360 pixels.
- `cv.SetCaptureProperty(capture, 4, 240)`: This line sets the image height to 240 pixels.
- `while True::` Here, you are creating a loop that will capture and display the image over and over again, until you press the *Esc* key.

- `img = cv.QueryFrame(capture)`: This line captures the image and stores it in the `img` data structure.
- `cv.ShowImage("camera", img)`: This line maps the `img` variable to the camera window, which you created previously.
- `If cv.WaitKey(10) == 27::` This if statement checks whether a key has been pressed, and if the pressed key is the *Esc* key, it executes the break. This stops the `while` loop, and the program reaches its end and stops. You need this statement in your code because it also signals OpenCV to display the image now.

Now run `camera.py`, and you should see the following screenshot:



You can use OpenCV to do something similar with the Raspberry Pi camera. The following screenshot shows an example program that will allow you to open a window, and display a moving picture of what the camera sees:

```

pi@raspberrypi: ~
File Edit Options Buffers Tools Python Help
import cv2 as cv
import picamera
import picamera.array
import numpy as np

with picamera.PiCamera() as camera:
    with picamera.array.PiRGBArray(camera) as stream:
        camera.resolution = (320, 240)

        while True:
            camera.capture(stream, 'bgr', use_video_port=True)
            cv.imshow('frame', stream.array)
            data = np.fromstring(stream.getvalue(), dtype=np.uint8)
            img = cv.imdecode(data, 1)

            if cv.waitKey(1) & 0xFF == ord('q'):
                break

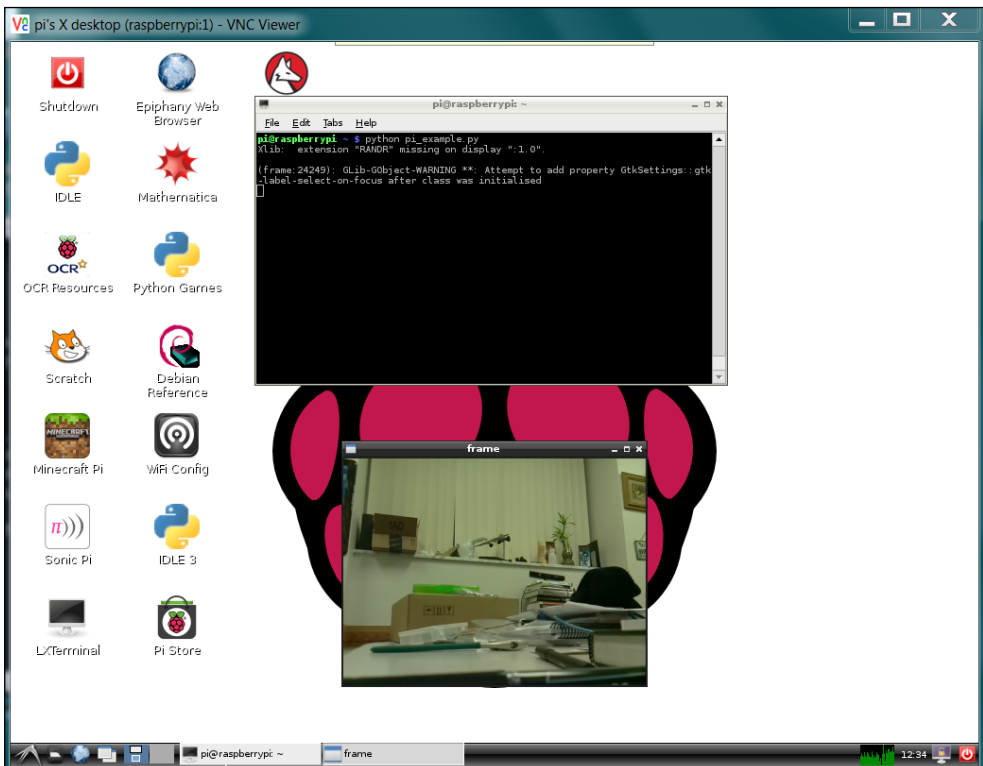
            stream.seek(0)
            stream.truncate()

cv.destroyAllWindows()

--UU-:***--F1 pi example.py All L22 (Python)-----
Closes with picamera.PiCamera() as camera:

```

You can create this program, then run it, and you should see a window on the console that updates in real time. The picture of the display is given in the following screenshot:



Now that you are up and running, you can use OpenCV, an open source set of software, to do some amazing things with your images. You may want to play with the resolution to find the optimum settings for your application. Bigger images are great – they give you a more detailed view of the world – but they also take up significantly more processing power. You'll play with this more as you actually ask your system to do some real image processing. Be careful if you are going to use `vncserver` to understand your system performance, as this will significantly slow down the update rate. An image that is twice the size (width/height) will involve four times more processing.

Your project can now see! You will use this capability to do a number of impressive tasks that will use this vision capability.

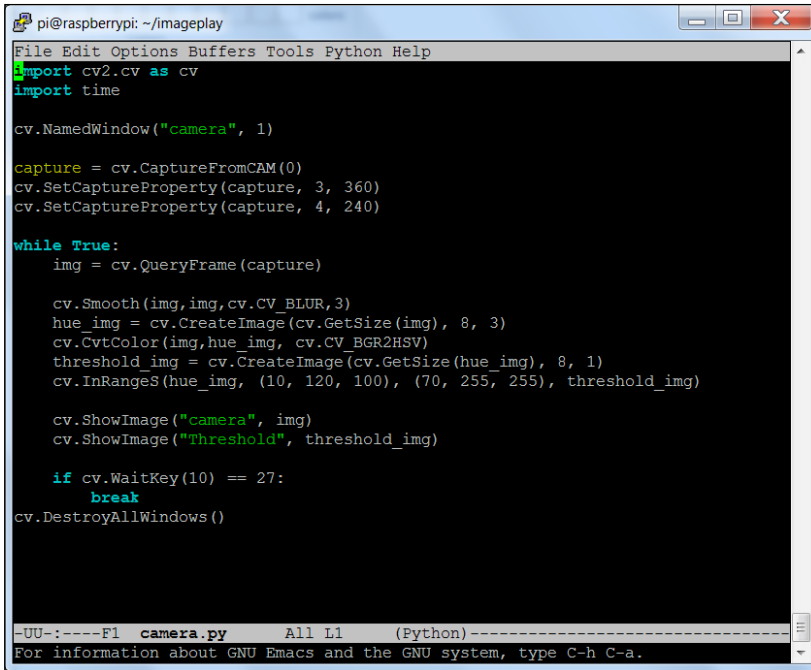
Using the vision library to detect colored objects

OpenCV and your webcam can track objects. This might be useful if you are building a system that needs to track and follow a colored ball. OpenCV makes this amazingly simple by providing some high-level libraries that can help us with this task. I'm going to do this in Python, as I find it much easier to work with than C. If you feel more comfortable with C, these instructions should be fairly easy to translate. Also, performance will be better if implemented in C, so you can create the initial capability in Python and then finalize the code in C.

If you'd like, you can create a directory to hold your image-based work. To do this, perform the following steps:

1. From your home directory, create a directory named `imageplay` by typing `mkdir imageplay` while in your home directory. Then, switch from the home directory to the `imageplay` directory by typing `cd imageplay`.
2. Once there, let's bring over your `camera.py` file as a starting point by typing `cp /home/pi/examples/python/camera.py camera.py`.

Now you are going to edit the file, until it looks something, as shown in the following screenshot:



```

pi@raspberrypi: ~/imageplay
File Edit Options Buffers Tools Python Help
import cv2.cv as cv
import time

cv.NamedWindow("camera", 1)

capture = cv.CaptureFromCAM(0)
cv.SetCaptureProperty(capture, 3, 360)
cv.SetCaptureProperty(capture, 4, 240)

while True:
    img = cv.QueryFrame(capture)

    cv.Smooth(img, img, cv.CV_BLUR, 3)
    hue_img = cv.CreateImage(cv.GetSize(img), 8, 3)
    cv.CvtColor(img, hue_img, cv.CV_BGR2HSV)
    threshold_img = cv.CreateImage(cv.GetSize(hue_img), 8, 1)
    cv.InRangeS(hue_img, (10, 120, 100), (70, 255, 255), threshold_img)

    cv.ShowImage("camera", img)
    cv.ShowImage("Threshold", threshold_img)

    if cv.WaitKey(10) == 27:
        break
cv.DestroyAllWindows()

-UU-:----F1 camera.py All L1 (Python)-----
For information about GNU Emacs and the GNU system, type C-h C-a.

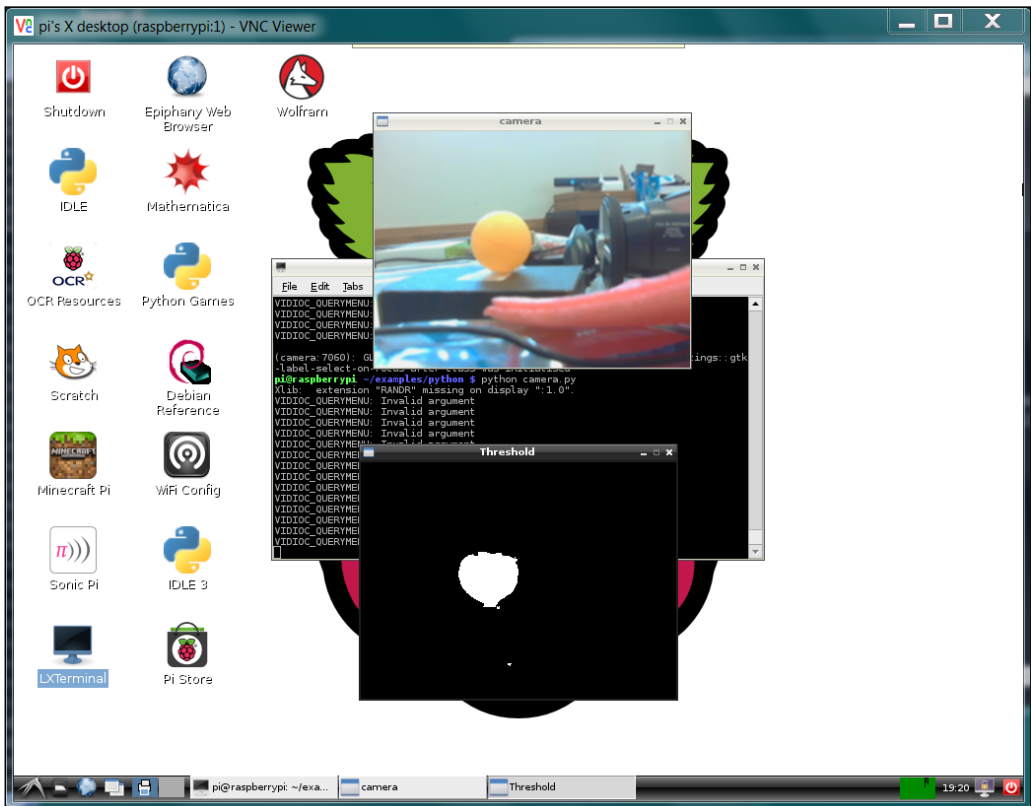
```

Let's look specifically at the following changes that you need to make to `camera.py`:

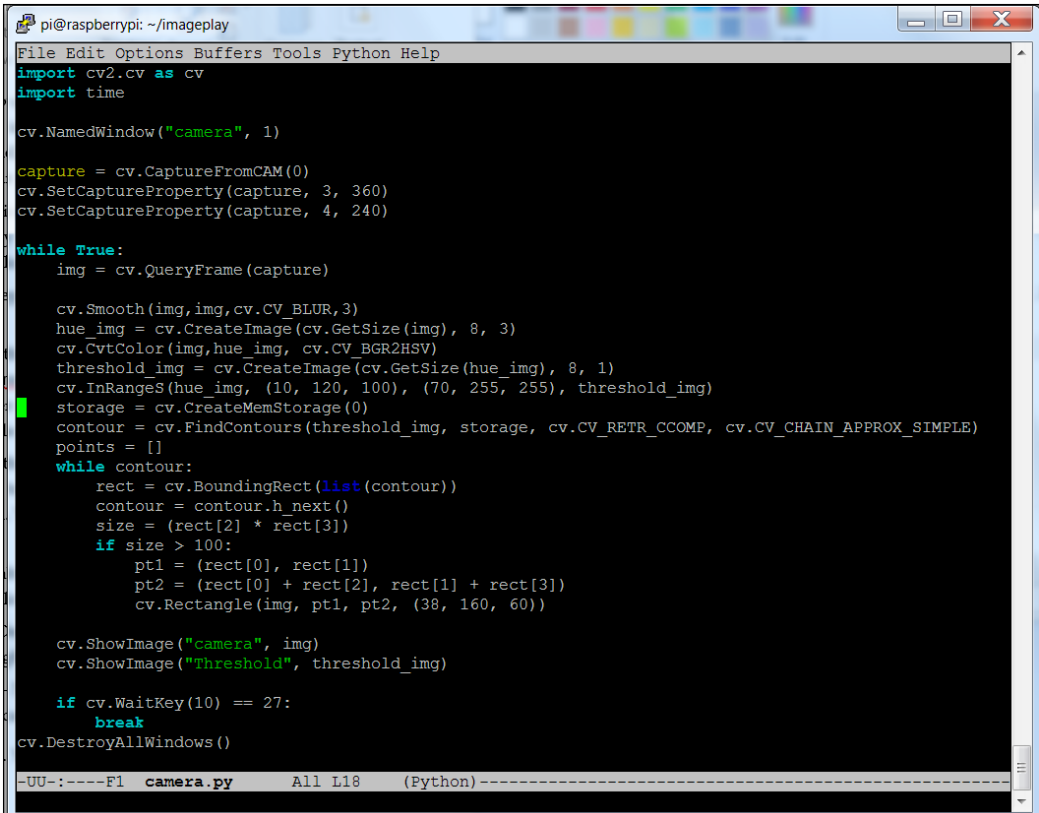
- `cv.Smooth(img, img, cv.CV_BLUR, 3)`: You are going to use the OpenCV library first to smoothen the image, taking out any large deviations.
- `hue_img = cv.CreateImage(cv.GetSize(img), 8, 3)`: This statement creates a default image that can hold the hue image, which you create in the next statement.
- `cv.CvtColor(img, hue_img, cv.CV_BGR2HSV)`: This line creates a new image that stores the image as per the values of hue (color), saturation, and value (HSV) instead of the red, green, and blue (RGB) pixel values of the original image. Converting to HSV focuses our processing more on the color, as opposed to the amount of light hitting it.
- `threshold_img = cv.CreateImage(cv.GetSize(hue_img), 8, 1)`: You are going to create yet another image, this time a black and white image, which is black for any pixel that is not between two certain color values.

- `cv.InRangeS(hue_img, (10,120, 100), (70, 255, 255), threshold_img):` The (10, 120, 100), (75, 255, 255) parameters determine the color range. In this case, I have an orange ball and I want to detect the color orange. For a good tutorial on using hue to specify color, try <http://www.tomjewett.com/colors/hsb.html>. Also, <http://www.shervinemami.info/colorConversion.html> includes a program that you can use to determine your values by selecting a specific color.
- `cv.ShowImage("Camera", img):` This shows a window with the original image in it.
- `cv.ShowImage("Threshold", threshold_img):` This shows a window with just the threshold image.

Now run the program. You'll need to either have a display, keyboard, and mouse connected to the board, or you can run it remotely using vncserver. Run the program by typing `sudo python ./camera.py`. If you see a single black image, move this window and you will expose the original image window, as well. Now take your target (I used my orange ball) and move it into the frame. You should see something as shown in the following screenshot:



Notice the white pixels in our threshold image showing where the ball is located. You can add more OpenCV code that gives the actual location of the ball. In our original image file of the ball's location, you can actually draw a rectangle around the ball as an indicator. Edit the `camera.py` file to look as follows:



```

pi@raspberrypi: ~/imageplay
File Edit Options Buffers Tools Python Help
import cv2 as cv
import time

cv.NamedWindow("camera", 1)

capture = cv.CaptureFromCAM(0)
cv.SetCaptureProperty(capture, 3, 360)
cv.SetCaptureProperty(capture, 4, 240)

while True:
    img = cv.QueryFrame(capture)

    cv.Smooth(img, img, cv.CV_BLUR, 3)
    hue_img = cv.CreateImage(cv.GetSize(img), 8, 3)
    cv.CvtColor(img, hue_img, cv.CV_BGR2HSV)
    threshold_img = cv.CreateImage(cv.GetSize(hue_img), 8, 1)
    cv.InRangeS(hue_img, (10, 120, 100), (70, 255, 255), threshold_img)
    storage = cv.CreateMemStorage(0)
    contour = cv.FindContours(threshold_img, storage, cv.CV_RETR_CCOMP, cv.CV_CHAIN_APPROX_SIMPLE)
    points = []
    while contour:
        rect = cv.BoundingRect(list(contour))
        contour = contour.h_next()
        size = (rect[2] * rect[3])
        if size > 100:
            pt1 = (rect[0], rect[1])
            pt2 = (rect[0] + rect[2], rect[1] + rect[3])
            cv.Rectangle(img, pt1, pt2, (38, 160, 60))

    cv.ShowImage("camera", img)
    cv.ShowImage("Threshold", threshold_img)

    if cv.WaitKey(10) == 27:
        break
cv.DestroyAllWindows()

-UU-:----F1 camera.py All L18 (Python)-----

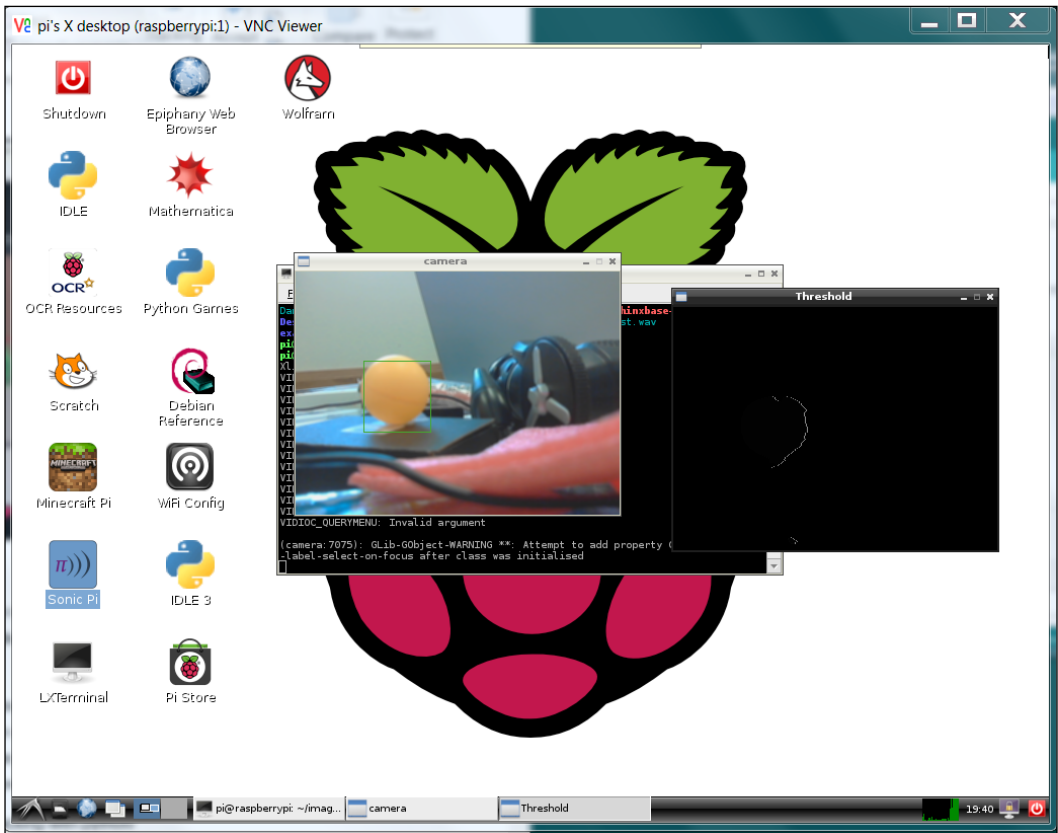
```

Start by editing just below the `cv.InRangeS(hue_img, (10,120, 100), (70, 255, 255), threshold_img)` line. The lines used are as follows:

- `storage = cv.CreateMemStorage(0)`: This line creates some memory for you to manipulate the images in.
- `contour = cv.FindContours(threshold_img, storage, cv.CV_RETR_CCOMP, cv.CV_CHAIN_APPROX_SIMPLE)`: This finds all the areas on your image that are within the threshold. There could be more than one, so you may want to capture them all.

- `points = []`: This creates an array for us to hold all the different possible color points.
- `while contour::` Adding a while loop will let you step through all the possible contours. By the way, it is important to note that if there is another larger orange **binary large object (blob)** in the background, you will *find* that location. Just to keep this simple, you'll assume that your orange ball is unique.
- `rect = cv.BoundingRect(list(contour))`: This gets a bounding rectangle for each area of color. The rectangle is defined by the corners of a rectangle around the blob of color.
- `contour = contour.h_next()`: This will prepare you for the next contour, if one exists.
- `size = (rect[2] * rect[3])`: This calculates the diagonal length of the rectangle that you are evaluating. The data structure `rect` contains four integers 0 and 1 for the pixel values of the lower-left corner of the box, and 2 and 3 for the size in pixels of the rectangle.
- `if size > 100::` Here, you check to see if the area is big enough to be of concern. 100 tells your program to not worry about any rectangles that are less than 100 pixels in area. You may want to vary this, based on the application.
- `pt1 = (rect[0], rect[1])`: Define a `pt1` variable and set its two values to the x and y coordinates of the left side of the blob's rectangular location.
- `pt2 = (rect[0] + rect[2], rect[1] + rect[3])`: Define a `pt2` variable and set its two values to the x and y coordinates of the right side of the blob's rectangular location.
- `cv.Rectangle(img, pt1, pt2, (38, 160, 60))`: Now you add a rectangle to your original image by identifying where it is located.

Now that the code is ready, you can run it. You should see something, as shown in the following screenshot:



You can now track your object.

Now that you have the code, you can modify the color or add more colors. You also have the location of your object, so later you can attempt to follow the object or manipulate it in some way.

OpenCV is an amazing, powerful library of functions. You can do all sorts of incredible things with just a few lines of code. Another common feature that you may want to add to your projects is motion detection. If you'd like to try it, there are several good tutorials; try looking at the following links:

- <http://derek.simkowiak.net/motion-tracking-with-python/>
- <http://stackoverflow.com/questions/3374828/how-do-i-track-motion-using-opencv-in-python>
- <https://www.youtube.com/watch?v=8QouvYMfmQo>
- <https://github.com/RobinDavid/Motion-detection-OpenCV>

Having a webcam connected to your system provides all kinds of complex vision capabilities. You can get 3D vision with OpenCV using two cameras. There are several good places; for example, the code in the `samples/cpp` directory that comes with OpenCV has a sample `stereo_match.cpp`. For more information, refer to <http://code.google.com/p/opencvstereo vision/source/checkout>.

Summary

As you learned in this chapter, your projects can now speak and see! You can issue commands, and your projects can respond to changes in the physical environment sensed by the webcam. In the next chapter, you will add mobility using motors, servos, and other methods.

5

Creating Mobile Robots on Wheels

You can now talk to your Raspberry Pi, and it can talk back. It can even see. Now you will add the capability to move the entire project using wheels. Perhaps the easiest way to make your projects mobile is to use a wheeled platform. In this chapter, you will be introduced to some of the basics of controlling DC motors and using Raspberry Pi to control the speed and direction of your wheeled platform.

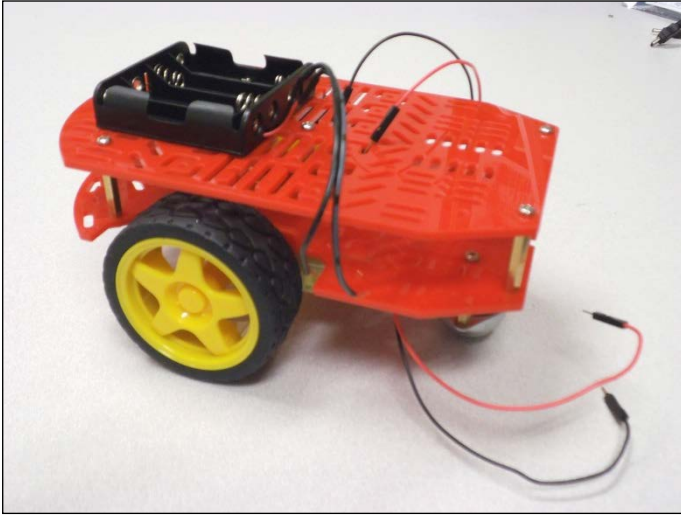
In this chapter, you will learn how to perform the following actions:

- Using the Raspberry Pi GPIO to control a DC Motor
- Controlling your mobile platform programmatically using Raspberry Pi
- Making your platform truly mobile by issuing voice commands

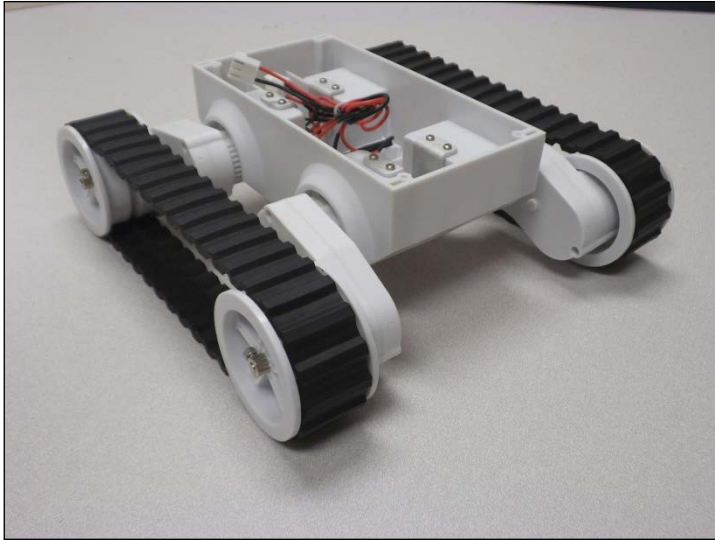
Gathering the required hardware

You'll need to add some hardware, specifically a wheeled or tracked platform, to make your project mobile. You're going to use a platform that uses differential motion to propel and steer the vehicle. This simply means that instead of turning the wheels, you're going to vary the speed and direction of the two motors that drive the wheels or tracks. There are a lot of choices. Some are completely assembled, while others require some assembly; or you can buy the components and construct your own custom mobile platform.

Throughout this book, I'm going to assume that you don't want to do any soldering or mechanical machining yourself. So let's look at a couple of the more popular variants that are available completely assembled or can be assembled with simple tools (screwdriver and/or pliers). The simplest mobile platform is one that has two DC motors and each motor controls a single wheel; the platform has a small ball in the front or at the back. DC motors are very straightforward to control. As you vary the DC voltage, the speed on the motor varies. The following image is an example of a simple wheeled platform, sold by SparkFun, found at www.sparkfun.com:



This one does need to be assembled, but it is fairly straightforward. For more choices in two-wheeled platforms, go to <http://www.robotshop.com/2-wheeled-development-platforms-1.html> or dx.com/es/p/diy-disc-type-2-wheel-smart-car-model-body-black-yellow-184395. You could also choose a tracked platform instead of a wheeled platform. A tracked platform has more traction, but is not as nimble as it takes a longer distance to turn. Again, manufacturers make pre-assembled units. The following image is an example of a pre-assembled tracked platform, made by Dagü. It's called the Dagü Rover 5 Tracked Chassis.



Since you have a mobile platform, you'll need a mobile power supply for Raspberry Pi. I personally like the external 5V rechargeable cell phone batteries that are available from almost any place that sells cell phones. These batteries can be charged using a USB cable connected either through a DC power supply or directly from a computer USB port, as shown in the following image:



You'll also need a USB cable to connect your battery to the Raspberry Pi, but you can just use the cable supplied with the Raspberry Pi.

Now that you have the mobile platform, you'll need some bits and pieces to connect your Raspberry Pi to the DC motors. You'll need a small breadboard and some male-to-female jumper cables. You'll also need an L293D H-bridge part that will drive your DC motors with the voltage and current that will allow your platform to move.

In this project, you'll mount the breadboard to your robot. If you'd like a more permanent solution, consider a prototype shield kit, as such. This kit will not work with Raspberry Pi A+, see <http://www.doctormonk.com/2012/08/review-of-raspberry-pi-prototyping.html> for possible prototyping shields. These shields are nice because it allows you to solder your components on to a more stable final configuration. You may want to try the system with the breadboard, and then take the time to translate that solution to a prototyping shield.

There are also a number of DC motor controllers that can be connected right to the Raspberry Pi GPIO connector. For example, Pololu at www.pololu.com makes the DRV8835 Dual Motor Driver Kit for the Raspberry Pi B+. Another possible choice is the RasPiRobot Board V2 available at www.monkmakes.com. However, since you are interested in learning the specifics, let's walk through a DC motor control example using the breadboard and the basic parts.

Using the Raspberry Pi GPIO to control a DC motor

The first step to make the platform mobile is connecting the Raspberry Pi to your H-bridge. This allows us to control the speed of each wheel (or track) independently. Before you get started, let's spend some time understanding the basics of motor control. Whether you choose the two-wheeled mobile platform or the tracked platform, the basic movement control is the same. The unit moves by engaging the motors. If the desired direction is straight, the motors are run at the same speed. If you want to turn the unit, the motors are run at different speeds. The unit can turn in a circle if you run one motor forward and the other one backward.

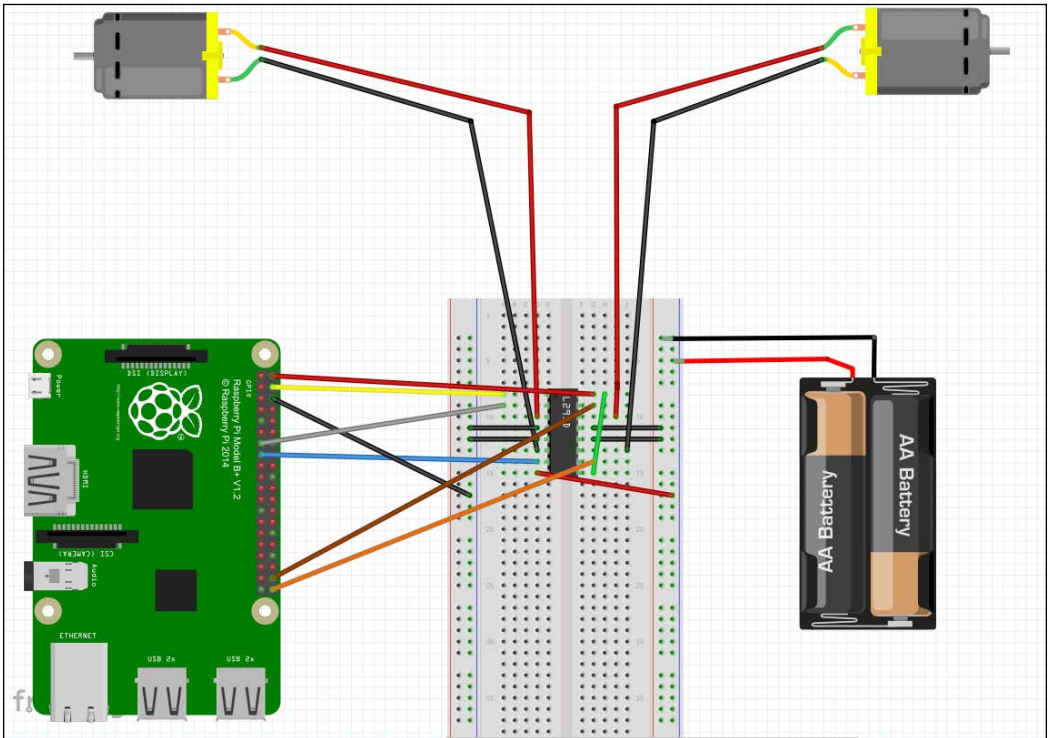
The DC motors are fairly straightforward devices. The speed and direction of the motor is controlled by the magnitude and polarity of the voltage applied to its terminals. The higher the voltage, the faster the motor will turn. If you reverse the polarity of the voltage, you can reverse the direction in which the motor is turning.

The magnitude and polarity of the voltage are not the only important factors when you think about controlling the motors. The power that your motor can apply to moving your platform is also determined by the voltage and the current supplied at its terminals.





















There are GPIO (short for general purpose input-output) pins on the Raspberry Pi that you could use to create the control voltage and drive your motors. These GPIO pins provide direct access to some of the control lines available from the processor itself. However, the unit cannot source enough current and your motors would not be able to generate enough power to move your mobile platform. This can also cause physical damage to your Raspberry Pi board.

You can, however, connect your Raspberry Pi to the DC motors by using an H-bridge DC motor controller. An H-bridge is a fairly simple device. It basically consists of a set of switches and adds the additional functionality of allowing the direction of the current to be reversed, so that the motor can either be run in the forward or the reverse direction.

Let's start this example by building the H-bridge circuit and controlling just one motor. To do this, you'll need to get an H-bridge. One of the most common options is the L293 dual H-bridge chip. This chip will allow you to control the direction of the DC motors. These are available at most electronics stores and online. Once you have your H-bridge, build the circuit, as shown in the following image with the Raspberry Pi, motor, jumper wires, 4AA battery holder, and breadboard:



Also, before you start connecting wires, here is an image of the GPIO pins on the Raspberry Pi Black board:

Pin 1 3.3V		Pin 2 5V
Pin 3 GPIO2		Pin 4 5V
Pin 5 GPIO3		Pin 6 GND
Pin 7 GPIO4		Pin 8 GPIO14
Pin 9 GND		Pin 10 GPIO15
Pin 11 GPIO17		Pin 12 GPIO18
Pin 13 GPIO27		Pin 14 GND
Pin 15 GPIO22		Pin 16 GPIO23
Pin 17 3.3V		Pin 18 GPIO24
Pin 19 GPIO10		Pin 20 GND
Pin 21 GPIO9		Pin 22 GPIO25
Pin 23 GPIO11		Pin 24 GPIO8
Pin 25 GND		Pin 26 GPIO7
Pin 27 ID_SD		Pin 28 ID_SC
Pin 29 GPIO5		Pin 30 GND
Pin 31 GPIO6		Pin 32 GPIO12
Pin 33 GPIO13		Pin 34 GND
Pin 35 GPIO19		Pin 36 GPIO16
Pin 37 GPIO26		Pin 38 GPIO20
Pin 39 GND		Pin 40 GPIO21

Pin 1 on the Raspberry Pi GPIO is the one closest to the power on LED, but if you're not sure, flip the board over and you will see the pin with the square pattern. Specifically, you'll want to connect these pins on the Raspberry Pi GPIO to the pins on the H-bridge, as shown in the following table:

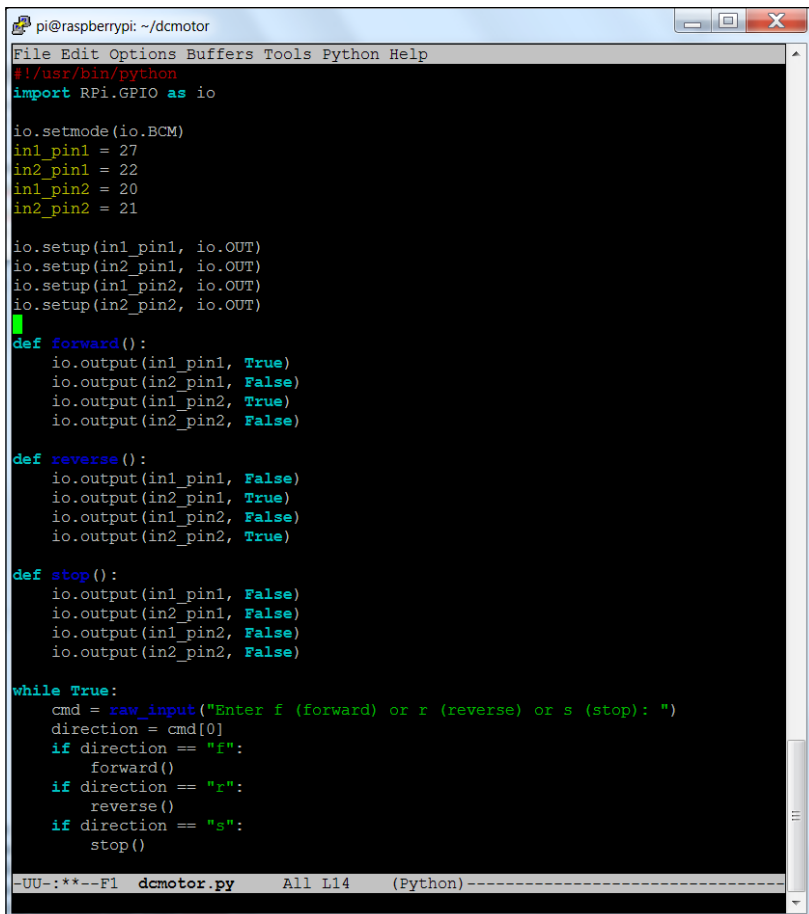
Raspberry Pi GPIO pin	H-Bridge pin
4 (5V)	1 (Enable pin)
13 (GPIO 27)	2 (Forward)
15 (GPIO 22)	7 (Backward)
4 (5V)	11 (Enable 2)
38 (GPIO 6)	10 (Forward)
40 (GPIO 13)	15 (Backward)
6 (GND)	4, 5, 12, 13 (GND)
2 (5 Volts)	16 (VCC)
Battery positive terminal	8 (Vc)
Battery negative terminal	GND (connect to the same GND as previous GND pins)

Once you have made the connections, you can test the system. To do this, you'll need to add some code.

Controlling your mobile platform programmatically using Raspberry Pi

Now that you have your basic motor controller functionality up and running, you need to connect both motor controllers to the Raspberry Pi. This section will cover this, and also show you how to control your entire platform programmatically.

You are going to use Python in your initial attempts to control the motor. It is very straightforward to code, run, and debug your code in Python. The first Python program you are going to create is shown in the following screenshot:



```

pi@raspberrypi: ~/dcmotor
File Edit Options Buffers Tools Python Help
#!/usr/bin/python
import RPi.GPIO as io

io.setmode(io.BCM)
in1_pin1 = 27
in2_pin1 = 22
in1_pin2 = 20
in2_pin2 = 21

io.setup(in1_pin1, io.OUT)
io.setup(in2_pin1, io.OUT)
io.setup(in1_pin2, io.OUT)
io.setup(in2_pin2, io.OUT)

def forward():
    io.output(in1_pin1, True)
    io.output(in2_pin1, False)
    io.output(in1_pin2, True)
    io.output(in2_pin2, False)

def reverse():
    io.output(in1_pin1, False)
    io.output(in2_pin1, True)
    io.output(in1_pin2, False)
    io.output(in2_pin2, True)

def stop():
    io.output(in1_pin1, False)
    io.output(in2_pin1, False)
    io.output(in1_pin2, False)
    io.output(in2_pin2, False)

while True:
    cmd = raw_input("Enter f (forward) or r (reverse) or s (stop): ")
    direction = cmd[0]
    if direction == "f":
        forward()
    if direction == "r":
        reverse()
    if direction == "s":
        stop()

-UU-:***-F1 dcmotor.py All L14 (Python)-----

```


Perform the following steps to create this program:

1. Create a directory called `dcmotor` in your home directory by typing `mkdir dcmotor` and then type `cd dcmotor`.
2. Now open the file by typing `emacs dcmotor.py` (if you are using a different editor, open a new file with the `dcmotor.py` name).
3. Now enter the program. Let's go through the program step by step:
 1. `#!/usr/bin/python`: This line lets you run this program without having to type `python` before the filename. You'll learn how to do this at the end of these instructions.
 2. `import RPi.GPIO as io`: This lets you import the RPi library, which will allow you to control the GPIO pins.
 3. `io.setmode(io.BCM)`: This sets the specification mode of the GPIO pins to **Broadcom SOC channel number (BCM)**. This means you will specify the GPIO numbers of the pins you want to control, instead of the actual physical pin values.
 4. `in1_pin1 = 27`: This assigns the value 27 to the `in1_pin1` variable.
 5. `in2_pin1 = 22`: This assigns the value 22 to the `in1_pin1` variable.
 6. `in1_pin2 = 20`: This assigns the value 20 to the `in1_pin1` variable.
 7. `in2_pin2 = 21`: This assigns the value 21 to the `in1_pin1` variable.
 8. `io.setup(in1_pin1, io.OUT)`: This sets the GPIO pin 27 to an output control.
 9. `io.setup(in2_pin1, io.OUT)`: This sets the GPIO pin 22 to an output control.
 10. `io.setup(in1_pin2, io.OUT)`: This sets the GPIO pin 20 to an output control.
 11. `io.setup(in2_pin2, io.OUT)`: This sets the GPIO pin 21 to an output control.
 12. `def forward() :` This defines the forward function. You'll turn on GPIO27 and GPIO20, and turn off GPIO22 and GPIO21.
 13. `io.output(in1_pin1, True)`: Output a 3.3 volt signal out on `in1_pin1` (this is GPIO 27).
 14. `io.output(in2_pin1, False)`: Output 0 volts out on `in2_pin1` (this is GPIO 22).

15. `io.output(in1_pin2, True)`: Output a high voltage out on `in1_pin2` (this is GPIO 20).
16. `io.output(in2_pin2, False)`: Output 0 volts out on `in2_pin2` (this is GPIO 21).
17. `def reverse()`: This defines the reverse function. You'll turn on GPIO22 and GPIO21, and turn off GPIO27 and GPIO20.
18. `io.output(in1_pin1, False)`: Output 0 volts out on `in1_pin1` (this is GPIO 27).
19. `io.output(in2_pin1, True)`: Output a high voltage out on `in2_pin1` (this is GPIO 22).
20. `io.output(in1_pin2, False)`: Output 0 volts out on `in1_pin2` (this is GPIO 20).
21. `io.output(in2_pin2, True)`: Output a high voltage out on `in2_pin2` (this is GPIO 21).
22. `def stop()`: This defines the stop function. You'll set the level to 0 on pins of GPIO22, GPIO21, GPIO27, and GPIO20.
23. `io.output(in1_pin1, False)`: Output 0 volts out on `in1_pin1` (this is GPIO 27).
24. `io.output(in2_pin1, False)`: Output 0 volts out on `in2_pin1` (this is GPIO 22).
25. `io.output(in1_pin2, False)`: Output 0 volts out on `in1_pin2` (this is GPIO 20).
26. `io.output(in2_pin2, False)`: Output 0 volts out on `in2_pin2` (this is GPIO 21).
27. `while True`: This performs loop over and over. You can stop the program by pressing *Ctrl* + *C*.
28. `cmd = raw_input("Enter f (forward) or r (reverse) or s (stop): ")`: Enter a character for what you want the robot to do.
29. `direction = cmd[0]`: Take just the first character of the input.
30. `if direction == "f"`: If the direction is "f", then execute the next statement.
31. `forward()`: Execute the forward function.
32. `if direction == "r"`: If the direction is "r", then execute the next statement.

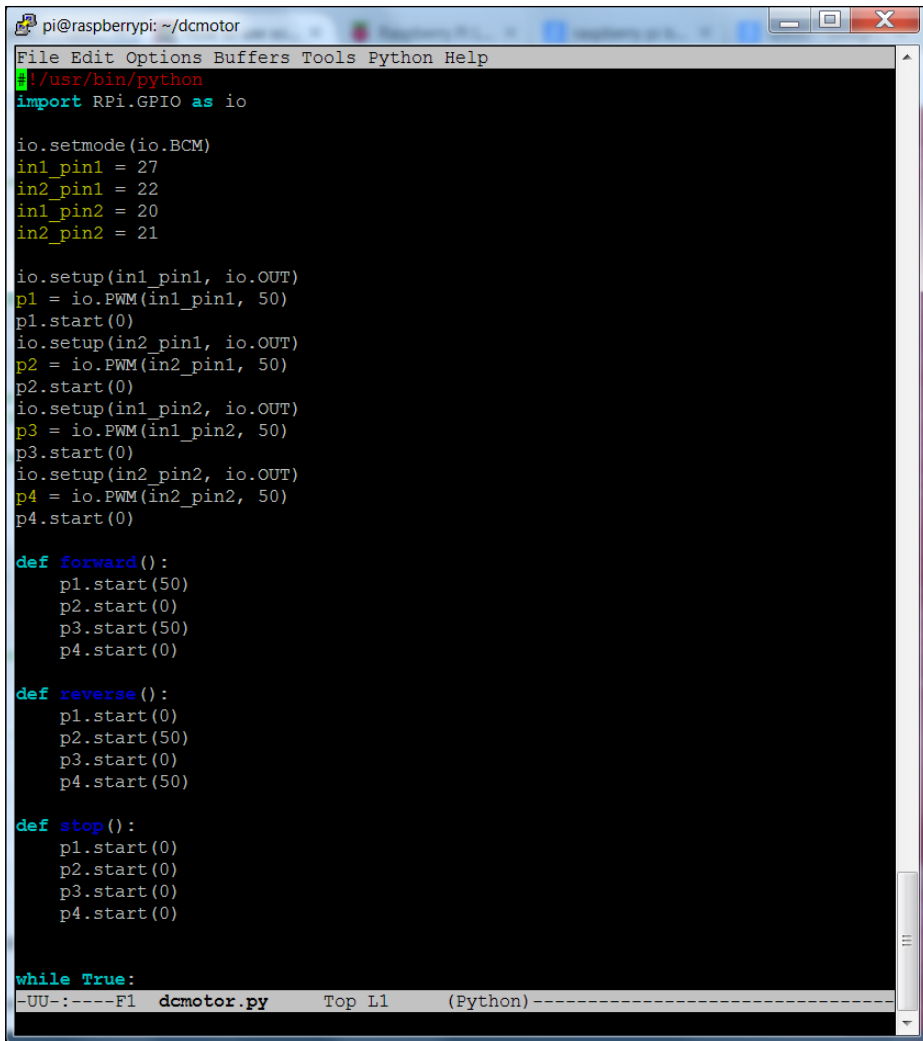
33. `reverse()`: Execute the reverse function.
 34. `if direction == "s"::` If the direction is "f", then execute the next statement.
 35. `stop()`: Execute the stop function.
4. You can now run your program. To do this, type `sudo python ./dcmotor.py`. When you enter `f`, the motors should run forward, with `r` they should run backward, and with `s` they should stop. You can now control the motor through Python. Additionally, you'll want to make this program available to run from the command line. Type `chmod +x dcmotor.py`. If you will now type `ls` (list programs), then you'll see that your program is now green, which means you can execute it directly. Now you can type `sudo ./dcmotor.py` and the program will run.

Now that you know the basics of commanding your mobile platform, feel free to add even more functions and their commands to make your mobile platform move in different ways. Running just one motor will make the platform turn, as will running both motors in opposite directions.

Controlling the speed of your motors with PWM

The previous example either turned on the motors to full speed, or turned them off. You may want to configure your motors to run at different speeds. This can be done by using **Pulse Width Modulation (PWM)** to adjust the speed. PWM simply defines a way of changing the voltage value of the signal by sending a series of pulses of equal value, and changing the width of each pulse. The wider the pulse, the higher the average voltage delivered to the receiver. The DC motors that you are using will respond to this higher average voltage by spinning faster.

The Raspberry Pi GPIO can create PWM signals. The code snippet is shown in the following screenshot:



```
pi@raspberrypi: ~/dcmotor
File Edit Options Buffers Tools Python Help
#!/usr/bin/python
import RPi.GPIO as io

io.setmode(io.BCM)
in1_pin1 = 27
in2_pin1 = 22
in1_pin2 = 20
in2_pin2 = 21

io.setup(in1_pin1, io.OUT)
p1 = io.PWM(in1_pin1, 50)
p1.start(0)
io.setup(in2_pin1, io.OUT)
p2 = io.PWM(in2_pin1, 50)
p2.start(0)
io.setup(in1_pin2, io.OUT)
p3 = io.PWM(in1_pin2, 50)
p3.start(0)
io.setup(in2_pin2, io.OUT)
p4 = io.PWM(in2_pin2, 50)
p4.start(0)

def forward():
    p1.start(50)
    p2.start(0)
    p3.start(50)
    p4.start(0)

def reverse():
    p1.start(0)
    p2.start(50)
    p3.start(0)
    p4.start(50)

def stop():
    p1.start(0)
    p2.start(0)
    p3.start(0)
    p4.start(0)

while True:
--UU-:----F1 dcmotor.py Top L1 (Python)-----
```

Following is an explanation of the lines of code you just added:

- `io.setup(in2_pin1, io.OUT)`: This sets GPIO27 to an output.
- `p1 = io.PWM(in1_pin1, 50)`: Instead of just an on or off setting, this PWM setting allows the programmer to set the relative width of the pulse. This initializes this functionality on GPIO27.
- `p1.start(0)`: This starts the pulses on p1, GPIO27, with a pulse width of 0 percent, or off.

- `io.setup(in2_pin1, io.OUTPUT):` This sets GPIO22 to an output.
- `p2 = io.PWM(in2_pin1, 50):` This initializes this functionality on GPIO22.
- `p2.start(0):` This starts the pulses on p2, GPIO22, with a pulse width of 0 percent, or off.
- `io.setup(in1_pin2, io.OUTPUT):` This sets GPIO20 to an output.
- `p3 = io.PWM(in1_pin2, 50):` This initializes this functionality on GPIO20.
- `p3.start(0):` This starts the pulses on p3, GPIO20, with a pulse width of 0 percent, or off.
- `io.setup(in2_pin2, io.OUTPUT):` This sets GPIO21 to an output.
- `p4 = io.PWM(in2_pin2, 50):` This initializes this functionality on GPIO21.
- `p4.start(0):` This starts the pulses on p3, GPIO21, with a pulse width of 0 percent, or off.
- `def forward(50) ::` This function moves the unit forward by setting the pulse width in the forward direction of 50 percent.
- `p1.start(50):` This sets the value of p1 (GPIO27) to 50 percent on and 50 percent off. This should result in the motor running forward at half speed.
- `p2.start(0):` This sets the value of p2 (GPIO22) to 0 percent. This effectively turns this pin off.
- `p3.start(50):` This sets the value of p3 (GPIO20) to 50 percent on and 50 percent off. This should result in the motor running forward at half speed.
- `p4.start(0):` This sets the value of p4 (GPIO21) to 0 percent. This effectively turns this pin off.
- `def reverse(50) ::` This function moves the unit in reverse by setting the pulse width in the reverse direction of 50 percent.
- `p1.start(0):` This sets the value of p1 (GPIO27) to 0 percent. This effectively turns this pin off.
- `p2.start(50):` This sets the value of p2 (GPIO22) to 50 percent on and 50 percent off. This should result in the motor running in reverse at half speed.
- `p3.start(0):` This sets the value of p3 (GPIO20) to 0 percent. This effectively turns this pin off.
- `p4.start(50):` This sets the value of p4 (GPIO21) to 50 percent on and 50 percent off. This should result in the motor running in reverse at half speed.
- `def stop() ::` This function sets all PWM signals to 0 percent, effectively stopping the motors.
- `p1.start(0):` This sets the value of p1 (GPIO27) to 0 percent. This effectively turns this pin off.

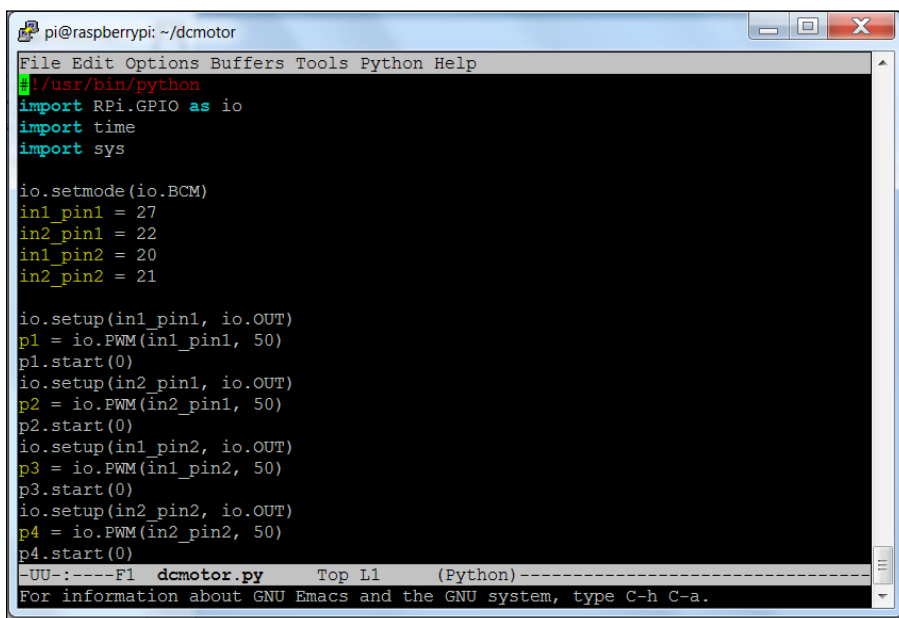
- `p2.start(0)`: This sets the value of `p2` (GPIO22) to 0 percent. This effectively turns this pin off.
- `p3.start(0)`: This sets the value of `p3` (GPIO20) to 0 percent. This effectively turns this pin off.
- `p4.start(0)`: This sets the value of `p4` (GPIO21) to 0 percent. This effectively turns this pin off.

The rest of the program is the same as the first `dcmotor.py` file. Running this program should result in the unit running at half the speed of the first program. You can easily change this speed by changing the value sent to the various start functions.

Adding program arguments to control your platform

In the next section, you will add the ability for your platform to respond to voice commands. To make this work, you will need to call your `dcmotor.py` program, using command line arguments. These are additional commands that you can type in addition to your `dcmotor.py` program name to specify what you want the program to do. In this case, you'll add commands to move your platform forward, backward, or to stop it.

Here is the first part of the code to add this capability:



```
pi@raspberrypi: ~/dcmotor
File Edit Options Buffers Tools Python Help
#!/usr/bin/python
import RPi.GPIO as io
import time
import sys

io.setmode(io.BCM)
in1_pin1 = 27
in2_pin1 = 22
in1_pin2 = 20
in2_pin2 = 21

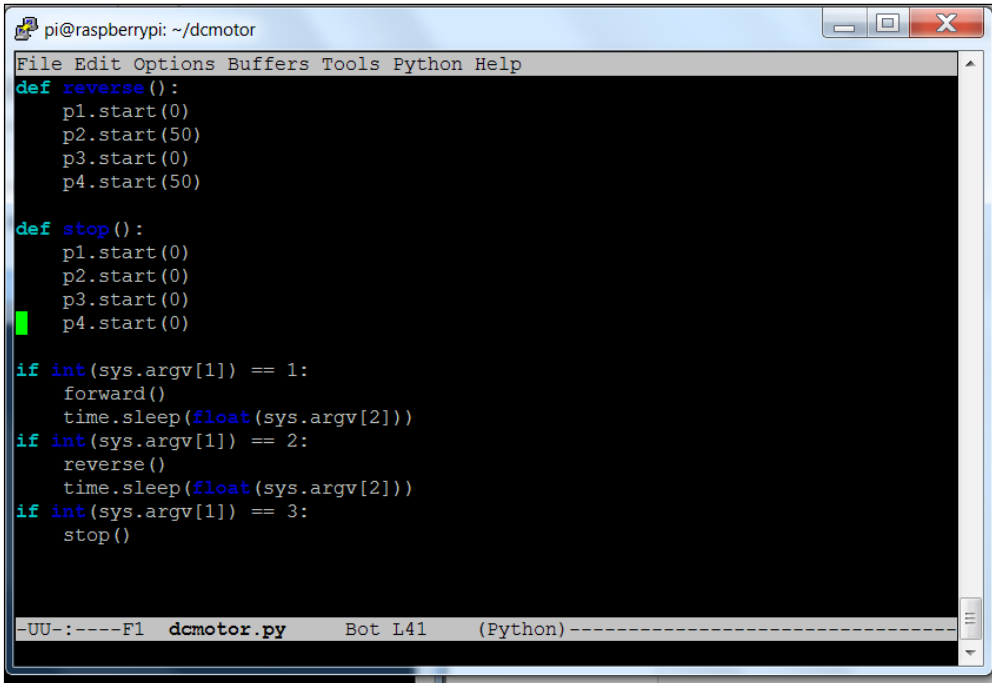
io.setup(in1_pin1, io.OUT)
p1 = io.PWM(in1_pin1, 50)
p1.start(0)
io.setup(in2_pin1, io.OUT)
p2 = io.PWM(in2_pin1, 50)
p2.start(0)
io.setup(in1_pin2, io.OUT)
p3 = io.PWM(in1_pin2, 50)
p3.start(0)
io.setup(in2_pin2, io.OUT)
p4 = io.PWM(in2_pin2, 50)
p4.start(0)

-UU-:----F1 dcmotor.py Top L1 (Python)-----
For information about GNU Emacs and the GNU system, type C-h C-a.
```

In this part of the code, you will have added the following two lines of code:

- `import time`: This imports the `time` library. You'll use this to add a delay command, which waits for a specified length of time.
- `import sys`: This imports the `sys` library, which allows you to access the command line arguments.

The second part of the code is shown in the following screenshot:

A screenshot of a terminal window on a Raspberry Pi. The window title is 'pi@raspberrypi: ~/dcmotor'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Python', and 'Help'. The code is as follows:

```
def reverse():  
    p1.start(0)  
    p2.start(50)  
    p3.start(0)  
    p4.start(50)  
  
def stop():  
    p1.start(0)  
    p2.start(0)  
    p3.start(0)  
    p4.start(0)  
  
if int(sys.argv[1]) == 1:  
    forward()  
    time.sleep(float(sys.argv[2]))  
if int(sys.argv[1]) == 2:  
    reverse()  
    time.sleep(float(sys.argv[2]))  
if int(sys.argv[1]) == 3:  
    stop()
```

The status bar at the bottom shows '-UU-:----F1 dcmotor.py Bot L41 (Python)-----'.

In the second part of the code, you will have added the following lines of code:

- `if int(sys.argv[1]) == 1:: sys.argv[1]` holds the second string that the user has typed on the command line. In this case, a command such as `./dcmotor.py 1 1` would result in the `1` string in the `sys.argv[1]` argument. `int(sys.argv[1])` translates the `1` string to the `1` integer. The `if` statement then simply checks if this value is `1`, and then runs the next two lines.

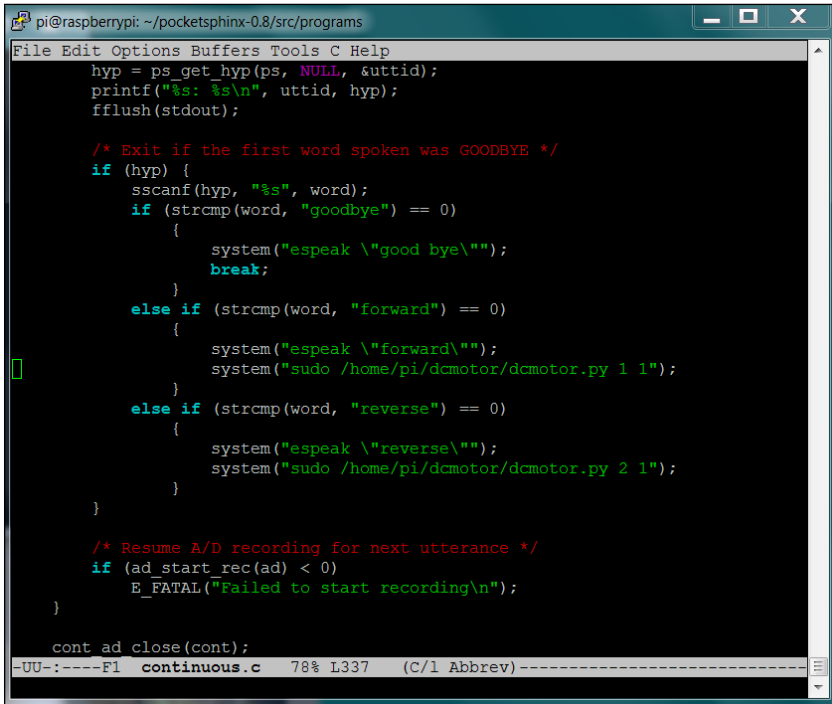
- `forward()`: Run the forward function.
- `time.sleep(float(sys.argv[2]))`: `sys.argv[2]` holds the third string that the user has typed on the command line. In this case, a command such as `./dcmotor.py 1 1` would result in the `1` string in the `sys.argv[2]` argument. `float(sys.argv[2])` translates the `1` string to a `1` float. The results of `time.sleep(float(sys.argv[2]))` is a delay by the number of seconds in the parenthesis of the statement.
- `if int(sys.argv[1]) == 2::` If the second value in the command line arguments is `2`, then execute the next two statements.
- `reverse()`: Call the reverse function.
- `time.sleep(float(sys.argv[2]))`: Sleep for the specified number of seconds.
- `if int(sys.argv[1]) == 3::` If the second value in the command line arguments is `3`, then execute the next statement.
- `stop()`: Call the stop function.

You can now run your program. To do this, type `sudo ./dcmotor.py 1 1`. Your platform should move forward for one second. If you run the program with `sudo ./dcmotor.py 2 1`, your platform should move in reverse for one second. Now you can use the program in conjunction with `pocketSphinx` to respond to voice commands.

Making your platform truly mobile by issuing voice commands

You should now have a mobile platform that you can program to move in any number of ways. Unfortunately, you still have your LAN cable connected, so the platform isn't completely mobile. And once you have begun the program, you can't alter the behavior of your program. In this section, you will use the principles from *Chapter 3, Providing Speech Input and Output*, to issue voice commands and initiate movement.

You'll need to modify your voice recognition program so it will run your Python program when it gets a voice command. If you feel rusty on how this works, review *Chapter 3, Providing Speech Input and Output*. You are going to make a simple modification to the `continuous.c` program in `/home/pipocketsphinx-0.8/src/programs`. To do this, type `cd /home/pi/pocketsphinx-0.8/src/programs` and then type `emacs continuous.c`. The changes will appear in the same section as your other voice commands, and will look like the following screenshot:



```

pi@raspberrypi: ~/pocketsphinx-0.8/src/programs
File Edit Options Buffers Tools C Help
hyp = ps_get_hyp(ps, NULL, &uttid);
printf("%s: %s\n", uttid, hyp);
fflush(stdout);

/* Exit if the first word spoken was GOODBYE */
if (hyp) {
    sscanf(hyp, "%s", word);
    if (strcmp(word, "goodbye") == 0)
    {
        system("espeak \"good bye\"");
        break;
    }
    else if (strcmp(word, "forward") == 0)
    {
        system("espeak \"forward\"");
        system("sudo /home/pi/dcmotor/dcmotor.py 1 1");
    }
    else if (strcmp(word, "reverse") == 0)
    {
        system("espeak \"reverse\"");
        system("sudo /home/pi/dcmotor/dcmotor.py 2 1");
    }
}

/* Resume A/D recording for next utterance */
if (ad_start_rec(ad) < 0)
    E_FATAL("Failed to start recording\n");
}

cont ad close(cont);
-UU-:---Fl continuous.c 78% L337 (C/l Abbrev)-----

```

The additions are pretty straightforward. Let's walk through them; they are as follows:

- `else if (strcmp(word, "forward") == 0)`: This line checks the word as recognized by your voice command program. If it corresponds with the word `forward`, you will execute everything inside the `if` statement. You use `{ }` to group and tell the system which commands go with this `else if` clause.

- `system("espeak \"forward\")`: This line executes `espeak`, which should tell us that you are about to run your robot program. By the way, you need to type `\` because the `"` character is a special character in Linux, and if you want the actual `"` character, you need to precede it with the `\` character.
- `system("/home/pi/track/dcmotor.py 1 1")`: This is the program you will execute. In this case, your mobile platform will do whatever the `dcmotor.py` program tells it to do; in this case, move forward one second.
- `else if (strcmp(word, "reverse") == 0)`: This line checks the word, as recognized by your voice command program. If it corresponds with the `reverse` word, you will execute everything inside the `if` statement. You use `{ }` to group and tell the system which commands go with this `else if` clause.
- `system("/home/pi/track/dcmotor.py 2 1")`: This is the program you will execute. In this case, your mobile platform will do whatever the `dcmotor.py` program tells it to do; in this case, move in reverse for one second.

After doing this, you will need to recompile the program. So, type `sudo make` and the executable `pocketsphinx_continuous` program will be created. Run the program by typing `sudo ./pocketsphinx_continuous`. When the program is running, speak the command *forward* or *reverse* and the mobile platform will now take the voice command and execute your program.

You should now have a completely mobile platform! When you execute your program, the mobile platform can now move around, based on what you have programmed it to do. Now you have your mobile platform up and ready to move around. You can command it using your voice. In the next chapter, you'll be introduced to a different kind of mobile platform, that is, one with legs.

You have already covered how to add vision to your Raspberry Pi project. A great addition to your mobile robot is the ability to follow a colored object attached to a target. Remember how you used OpenCV to find a colored object, and then found out where in your field of view (left or right/up or down) it existed? You can use this to decide whether to move your mobile platform right or left/forward or backward. Try this, and then move the target to see if your mobile robot can follow it.

Summary

This chapter provided you with an opportunity to make your robot mobile. Whether you choose a wheeled or tracked platform, your robot should now be able to move around. In the next chapter, you'll learn how to build a robot with legs; an even more flexible mobile platform.

6

Controlling the Movement of a Robot with Legs

In the previous chapter, we covered wheeled and tracked movement. That's cool enough, but what if you want your robot to navigate uneven ground? Now, you will add the ability to move the entire project using legs. In this chapter, you will be introduced to some of the basics of servo motors and to using Raspberry Pi to control the speed and direction of your legged platform.

Even though you've learned to make your robot mobile by adding wheels or tracks, these platforms will only work well on smooth, flat surfaces. Often, you'll want your robot to work in environments where the path is not smooth or flat; perhaps, you'll even want your robot to go upstairs or over other barriers. In this chapter, you'll learn how to attach your board, both mechanically and electrically, to a platform with legs so that your projects can be mobile in many more environments. Robots that can walk! What could be more amazing than this?

In this chapter, we will cover the following topics:

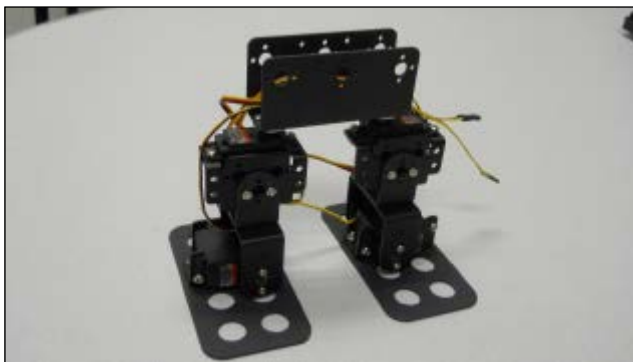
- Connecting Raspberry Pi to a two-legged mobile platform using a servo motor controller
- Creating a program in Linux so that you can control the movement of the two-legged mobile platform
- Making your robot truly mobile by adding voice control

Gathering the hardware

In this chapter, you'll need to add a legged platform to make your project mobile.

For a legged robot, there are a lot of choices for hardware. As seen in *Chapter 5, Creating Mobile Robots on Wheels*, some are completely assembled and others require some assembly; you may even choose to buy the components and construct your own custom mobile platform. Also, I'm going to assume that you don't want to do any soldering or mechanical machining yourself, so let's look at several choices of hardware that are available completely assembled or can be assembled using simple tools (a screwdriver and/or pliers).

One of the simplest legged mobile platforms is one that has two legs and four servo motors. The following is an image of this type of platform:



You'll use this legged mobile platform in this chapter because it is the simplest to program and the least expensive, requiring only four servos. To construct this platform, you must purchase the parts and then assemble them yourself. Find the instructions and parts list at <http://www.lynxmotion.com/images/html/build112.htm>. Another easy way to get all the mechanical parts (except servos) is by purchasing a biped robot kit with six **degrees of freedom (DOFs)**. This will contain the parts needed to construct a six-servo biped, but you can use a subset of the parts for your four-servo biped. These six DOF bipeds can be purchased on eBay or at <http://www.robotshop.com/2-wheeled-development-platforms-1.html>.

You'll also need to purchase the servo motors. Servo motors are similar to the DC motors you may have used in *Chapter 5, Creating Mobile Robots on Wheels*, except that servo motors are designed to move at specific angles based on the control signals that you send. For this type of robot, you can use standard-sized servos. I like Hitec HS-311 for this robot. They are inexpensive but powerful enough for the operations you'll use for this robot. You can get them on Amazon or eBay. The following is an image of an HS-311 servo:



As in the last chapter, you'll need a mobile power supply for Raspberry Pi. I personally like the 5-V cellphone rechargeable batteries that are available at almost any place that supplies cellphones. Choose one that comes with two USB connectors; you can use the second port to power your servo controller. The mobile power supply shown in the following image mounts well on the biped hardware platform:



You'll also need a USB cable to connect your battery to Raspberry Pi. You should already have one of these.

Now that you have the mechanical parts for your legged mobile platform, you'll need some hardware that will turn the control signals from your Raspberry Pi into voltage levels that can control the servo motors. Servo motors are controlled using a signal called PWM. For a good overview of this type of control, see http://pcbheaven.com/wikipages/How_RC_Servos_Works/ or <https://www.ghielectronics.com/docs/18/pwm>.

Although the Raspberry Pi's GPIO pins do support some limited **square-wave pulse width modulation** (SW PWM) signals, unfortunately these signals are not stable enough to accurately control servos. In order to control servos reliably, you should purchase a servo controller that can talk over a USB and control the servo motor. These controllers protect your board and make controlling many servos easy. My personal favorite for this application is a simple servo motor controller utilizing a USB from Pololu that can control six servo motors – Micro Maestro 6-Channel USB Servo Controller (assembled). This is available at www.pololu.com. The following is an image of the unit:



Make sure you order the assembled version. This piece of hardware will turn USB commands into voltage levels that control your servo motors. Pololu makes a number of different versions of this controller, each able to control a certain number of servos. Once you've chosen your legged platform, simply count the number of servos you need to control and choose a controller that can control that many servos. In this book, you will use a two-legged, four-servo robot, so you'll build the robot by using the six-servo version. Since you are going to connect this controller to Raspberry Pi through USB, you'll also need a USB A to mini-B cable.

You'll also need a power cable running from the battery to your servo controller. You'll want to purchase a USB to FTDI cable adapter that has female connectors, for example, the PL2303HX USB to TTL to UART RS232 COM cable available at www.amazon.com. The TTL to UART RS232 cable isn't particularly important; other than that, the cable itself provides individual connectors to each of the four wires in a USB cable. The following is an image of the cable:



Now that you have all the hardware, let's walk through a quick tutorial of how a two-legged system with servos works and then some step-by-step instructions to make your project walk.

Connecting Raspberry Pi to the mobile platform using a servo controller

Now that you have a legged platform and a servo motor controller, you are ready to make your project walk! Before you begin, you'll need some background on servo motors. Servo motors are somewhat similar to DC motors. However, there is an important difference; while DC motors are generally designed to move in a continuous way, rotating 360 degrees at a given speed, servo motors are generally designed to move at angles within a limited set. In other words, in the DC motor world, you generally want your motors to spin at a continuous rotation speed that you control. In the servo world, you want to limit the movement of your motor to a specific position. For more information on how servos work, visit <http://www.seattlerobotics.org/guide/servos.html> or http://www.societyofrobots.com/actuators_servos.shtml.

Connecting the hardware

To make your project walk, you first need to connect the servo motor controller to the servos. There are two connections you need to make, the first is to the servo motors, and the second is to the battery. In this section, before connecting your controller to your Raspberry Pi, you'll first connect your servo controller to your PC or Linux machine to check whether or not everything is working. The steps for doing so are as follows:

1. Connect the servos to the controller. The following is an image of your two-legged robot and the four different servo connections:



2. In order to be consistent, let's connect your four servos to the connections marked from **0** to **3** on the controller by using the following configurations:
 - **0**: Left foot
 - **1**: Left hip
 - **2**: Right foot
 - **3**: Right hip

The following is an image of the back of the controller; it will show you where to connect your servos:



3. Connect these servos to the servo motor controller as follows:
 - The left foot to 0 (the top connector) and the black cable to the outside (-)
 - The left hip to connector 1 and the black cable out
 - The right foot to connector 2 and the black cable out
 - The right hip to connector 3 and the black cable out

See the following image indicating how to connect servos to the controller:



- Now, you need to connect the servo motor controller to your battery. You'll use the USB to the FTDI UART cable; plug the red and black cables into the power connector on the servo controller, as shown in the following image:

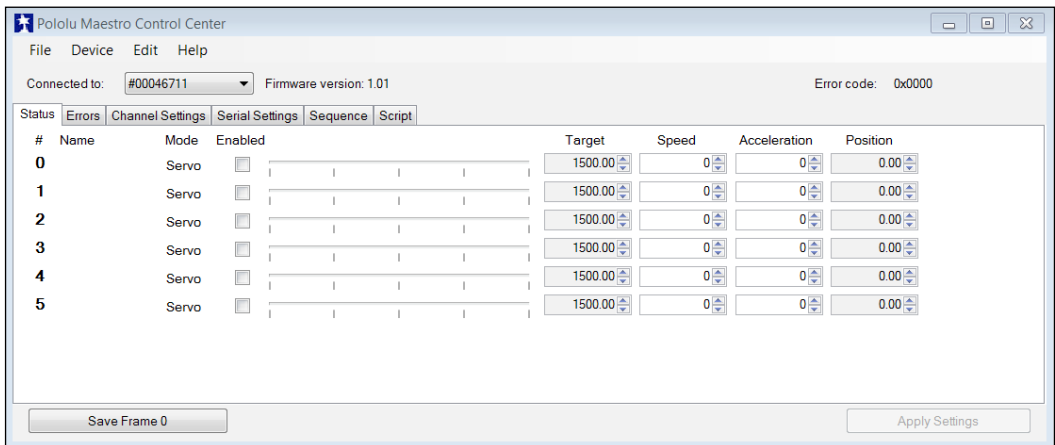


Now, plug the other end of the USB cable into one of the battery outputs.

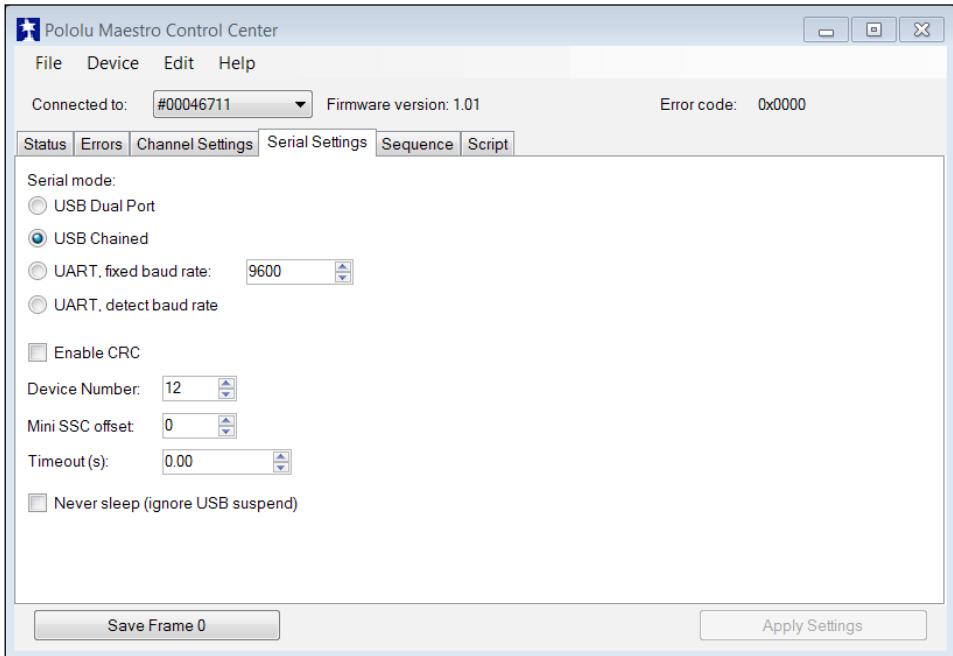
Configuring the software

Now, you can connect the motor controller to your PC or Linux machine to see whether or not you can talk to it. Once the hardware is connected, you will use some of the software provided by Polulu to control the servos. The steps to do so are as follows:

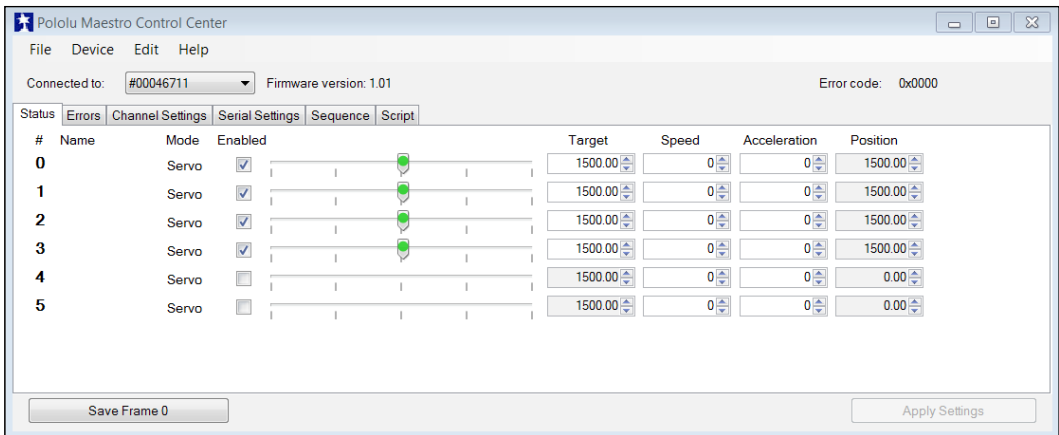
- Download the Polulu software from <http://www.pololu.com/docs/0J40/3.a> and install it using the instructions on the website. Once it is installed, run the software; you should see the window shown in the following screenshot:



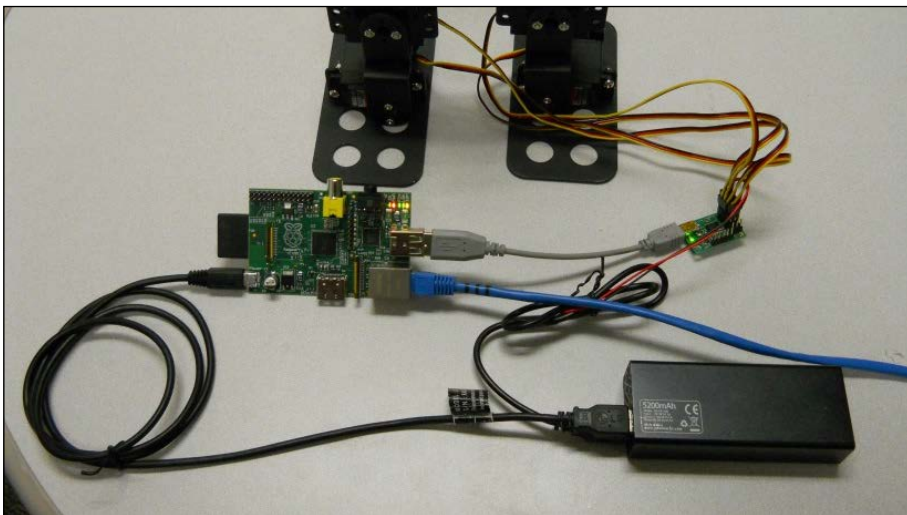
2. You will first need to change the **Serial mode** configuration in **Serial Settings**, so select the **Serial Settings** tab; you should see the window shown in the following screenshot:



3. Make sure that **USB Chained** is selected; this will allow you to connect to and control the motor controller over the USB. Now, go back to the main screen by selecting the **Status** tab; you can now turn on the four servos. The screen should look as shown in the following screenshot:

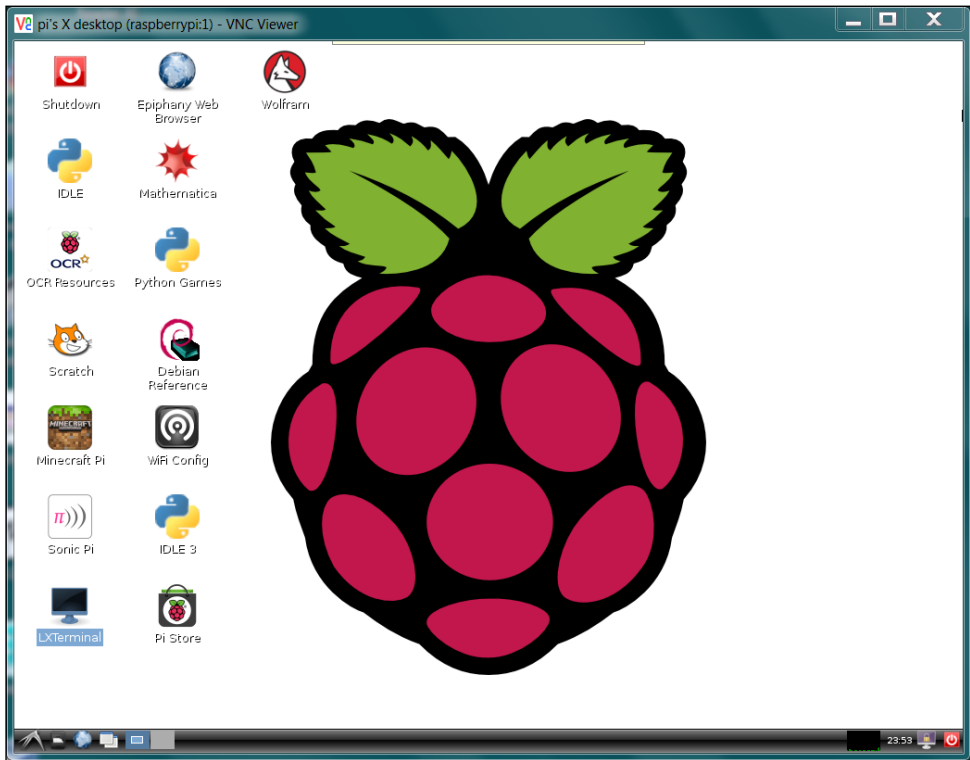


- Now, you can use the sliders to control the servos. Enable the four servos and make sure that servo 0 moves the left foot; 1, the left hip; 2, the right foot; and 3, the right hip.
- You've checked the motor controllers and the servos and you'll now connect the motor controller to Raspberry Pi to control the servos from there. Remove the USB cable from the PC and connect it to Raspberry Pi. The entire system will look as shown in the following image:



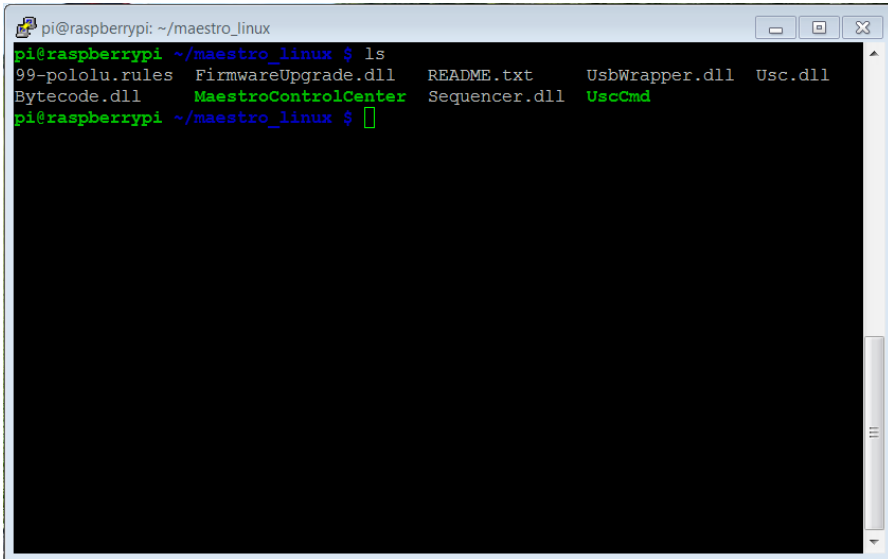
Let's now talk to the motor controller from your Raspberry Pi by downloading the Linux code from Pololu at <http://www.pololu.com/docs/0J40/3.b>. Perhaps the best way to do this is by logging on to Raspberry Pi using vncserver and opening a **VNC Viewer** window on your PC. To do this, log in to your Raspberry Pi by using PuTTY, and then, type `vncserver` at the prompt to make sure vncserver is running. Then, perform the following steps:

1. On your PC, open the **VNC Viewer** application, enter your IP address, and then click on **Connect**. Then, enter the password that you created for the vncserver; you should see the Raspberry Pi viewer screen, which should look as shown in the following screenshot:



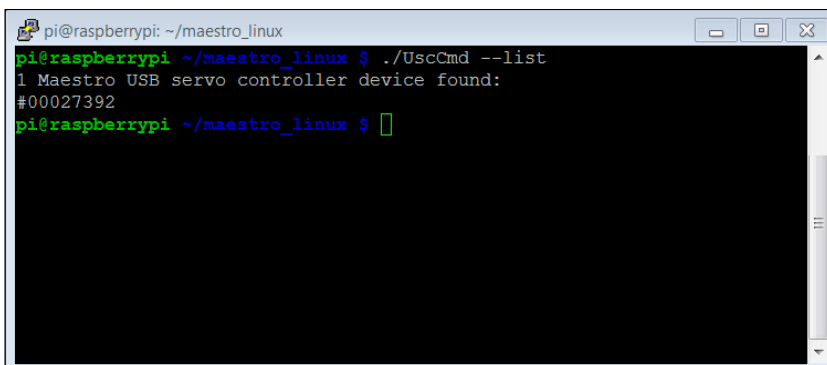
2. Open a browser window and go to <http://www.pololu.com/docs/0J40/3.b>. Click on the **Maestro Servo Controller Linux Software** link. You will need to download the `maestro_linux_100507.tar.gz` file to the Download folder. You can also use `wget` to get this software by typing `wget http://www.pololu.com/file/download/maestro-linux-100507.tar.gz?file_id=0J315` in a terminal window.

3. Go to your Download folder, move it to your home folder by typing `mv maestro_linux_100507.tar.gz ..`, and then go back to your home folder.
4. Unpack the file by typing `tar -xzf maestro_linux_011507.tar.gz`. This will create a folder called `maestro_linux`. Go to this folder by typing `cd maestro_linux` and then, type `ls`. You should see the output as shown in the following screenshot:



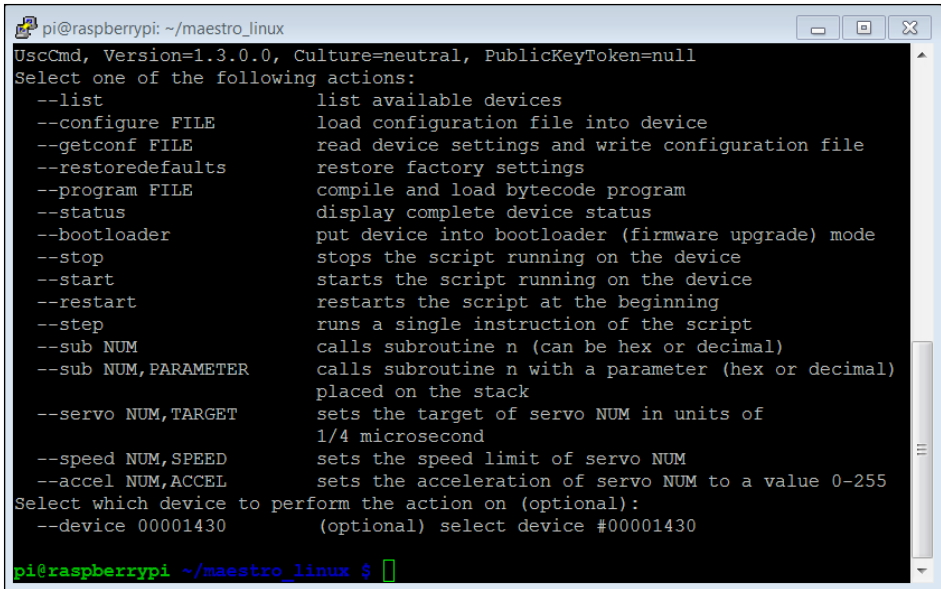
```
pi@raspberrypi: ~/maestro_linux
pi@raspberrypi ~/maestro_linux $ ls
99-pololu.rules  FirmwareUpgrade.dll  README.txt  UsbWrapper.dll  Usc.dll
Bytecode.dll    MaestroControlCenter  Sequencer.dll  UscCmd
pi@raspberrypi ~/maestro_linux $
```

5. The document `README.txt` will give you explicit instructions on how to install the software. Unfortunately, you can't run Maestro Control Center on your Raspberry Pi. The standard version of Maestro Control Center doesn't support the Raspberry Pi graphical system, but you can control your servos by using the `UscCmd` command-line application. First, type `./UscCmd --list`; you should see the following screenshot:



```
pi@raspberrypi: ~/maestro_linux
pi@raspberrypi ~/maestro_linux $ ./UscCmd --list
1 Maestro USB servo controller device found:
#00027392
pi@raspberrypi ~/maestro_linux $
```

6. The software now recognizes that you have a servo controller. If you just type `./UscCmd`, you can see all the commands you could send to your controller. When you run this command, you can see the result as shown in the following screenshot:



```
pi@raspberrypi: ~/maestro_linux
UscCmd, Version=1.3.0.0, Culture=neutral, PublicKeyToken=null
Select one of the following actions:
--list                list available devices
--configure FILE      load configuration file into device
--getconf FILE        read device settings and write configuration file
--restoredefaults      restore factory settings
--program FILE        compile and load bytecode program
--status              display complete device status
--bootloader          put device into bootloader (firmware upgrade) mode
--stop                stops the script running on the device
--start               starts the script running on the device
--restart             restarts the script at the beginning
--step                runs a single instruction of the script
--sub NUM             calls subroutine n (can be hex or decimal)
--sub NUM,PARAMETER   calls subroutine n with a parameter (hex or decimal)
--servo NUM,TARGET    sets the target of servo NUM in units of
                      1/4 microsecond
--speed NUM,SPEED     sets the speed limit of servo NUM
--accel NUM,ACCEL      sets the acceleration of servo NUM to a value 0-255
Select which device to perform the action on (optional):
--device 00001430      (optional) select device #00001430

pi@raspberrypi ~/maestro_linux $
```

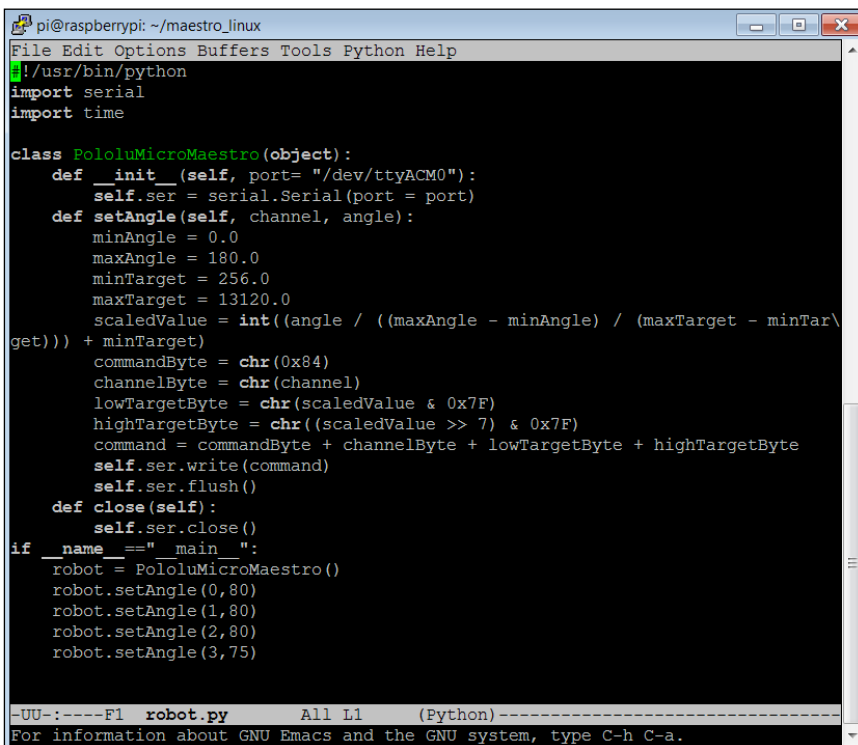
Notice that you can send a servo a specific target angle, although if the target angle is not within range, it makes it a bit difficult to know where you are sending your servo. Try typing `./UscCmd --servo 0, 10`. The servo will most likely move to its full angle position. Type `./UscCmd --servo 0, 0` and it will prevent the servo from trying to move. In the next section, you'll write some software that will translate your angles to the electronic signals that will move the servos.

If you haven't run the Maestro Controller tool and set the **Serial Settings** setting to **USB Chained**, your motor controller may not respond.

Creating a program in Linux to control the mobile platform

Now that you can control your servos by using a basic command-line program, let's control them by programming some movement in Python. In this section, you'll create a Python program that will let you talk to your servos a bit more intuitively. You'll issue commands that tell a servo to go to a specific angle and it will go to that angle. You can then add a set of such commands to allow your legged mobile robot to lean left or right and even take a step forward.

Let's start with a simple program that will make your legged mobile robot's servos turn at 90-degrees; this should be somewhere close to the middle of the 180-degree range you can work within. However, the center, maximum, and minimum values can vary from one servo to another, so you may need to calibrate them. To keep things simple, we will not cover that here. The following screenshot shows the code required for turning the servos:



```

pi@raspberrypi: ~/maestro_linux
File Edit Options Buffers Tools Python Help
#!/usr/bin/python
import serial
import time

class PololuMicroMaestro(object):
    def __init__(self, port= "/dev/ttyACM0"):
        self.ser = serial.Serial(port = port)
    def setAngle(self, channel, angle):
        minAngle = 0.0
        maxAngle = 180.0
        minTarget = 256.0
        maxTarget = 13120.0
        scaledValue = int((angle / ((maxAngle - minAngle) / (maxTarget - minTar\
get))) + minTarget)
        commandByte = chr(0x84)
        channelByte = chr(channel)
        lowTargetByte = chr(scaledValue & 0x7F)
        highTargetByte = chr((scaledValue >> 7) & 0x7F)
        command = commandByte + channelByte + lowTargetByte + highTargetByte
        self.ser.write(command)
        self.ser.flush()
    def close(self):
        self.ser.close()
if __name__ == "__main__":
    robot = PololuMicroMaestro()
    robot.setAngle(0,80)
    robot.setAngle(1,80)
    robot.setAngle(2,80)
    robot.setAngle(3,75)

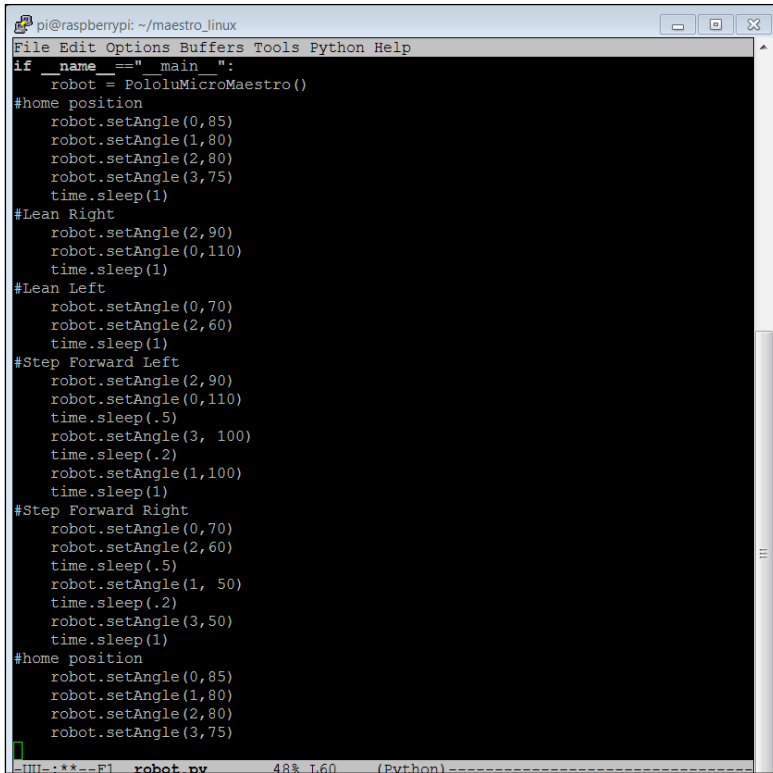
-UU-:----F1  robot.py      All L1      (Python)-----
For information about GNU Emacs and the GNU system, type C-h C-a.

```


The following is an explanation of the code:

- The `#!/user/bin/python` line allows you to make this Python file available for execution from the command line. It will allow you to call this program from your voice command program. We'll talk about this in the next section.
- The `import serial` and `import time` lines include the `serial` and `time` libraries. You need the `serial` library to talk to your unit via USB. If you have not installed this library, type `sudo apt-get install python-serial`. You will use the `time` library later to wait between servo commands.
- The `PololuMicroMaestro` class holds the methods that will allow you to communicate with your motor controller.
- The `__init__` method, opens the USB port associated with your servo motor controller.
- The `setAngle`, method converts your desired settings for the servo and angle to the serial command that the servo motor controller needs. The values, such as `minTarget` and `maxTarget`, and the structure of the communications — `channelByte`, `commandByte`, `lowTargetByte`, and `highTargetByte` — comes from the manufacturer.
- The `close`, method closes the serial port.
- Now that you have the class, the `__main__` statement of the program instantiates an instance of your servo motor controller class so that you can call it.
- Now, you can set each servo to the desired position. The default would be to set each servo to 90-degrees. However, the servos weren't exactly centered, so I found that I needed to set the angle of each servo so that my robot has both feet on the ground and both hips centered.

Once you have the basic home position set, you can ask your robot to do different things; the following screenshot shows some examples in simple Python code:



```

pi@raspberrypi: ~/maestro_linux
File Edit Options Buffers Tools Python Help
if __name__ == "__main__":
    robot = PololuMicroMaestro()
    #home position
    robot.setAngle(0,85)
    robot.setAngle(1,80)
    robot.setAngle(2,80)
    robot.setAngle(3,75)
    time.sleep(1)
    #Lean Right
    robot.setAngle(2,90)
    robot.setAngle(0,110)
    time.sleep(1)
    #Lean Left
    robot.setAngle(0,70)
    robot.setAngle(2,60)
    time.sleep(1)
    #Step Forward Left
    robot.setAngle(2,90)
    robot.setAngle(0,110)
    time.sleep(.5)
    robot.setAngle(3, 100)
    time.sleep(.2)
    robot.setAngle(1,100)
    time.sleep(1)
    #Step Forward Right
    robot.setAngle(0,70)
    robot.setAngle(2,60)
    time.sleep(.5)
    robot.setAngle(1, 50)
    time.sleep(.2)
    robot.setAngle(3,50)
    time.sleep(1)
    #home position
    robot.setAngle(0,85)
    robot.setAngle(1,80)
    robot.setAngle(2,80)
    robot.setAngle(3,75)

```

In this case, you are using your `setAngle` command to set your servos to manipulate your robot. This set of commands first sets your robot to the home position. Then, you can use the feet to lean to the right and then to the left and then you can use a combination of commands to make your robot step forward with the left and then the right foot. Once you have the program working, you'll want to package all your hardware onto the mobile robot.

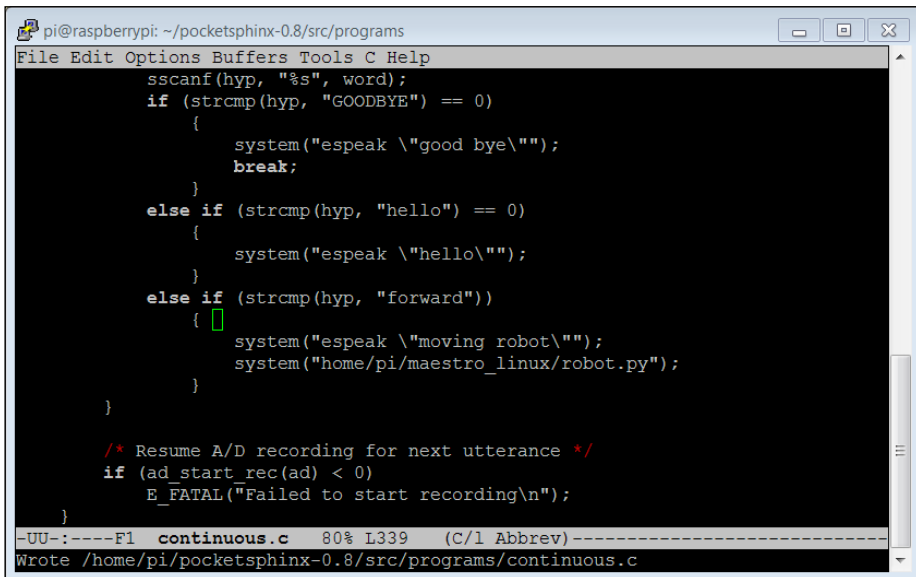
By following these principles, you can make your robot do many amazing things, such as walk forward and backward, dance, and turn around – any number of movements are possible. The best way to learn these movements is to try positioning the servos in new and different ways.

Making your mobile platform truly mobile by issuing voice commands

Now that your robot can move, wouldn't it be neat to have it obey your commands?

You should now have a mobile platform that you can program to move in any number of ways. Unfortunately, you still have your LAN cable connected, so the platform isn't completely mobile. Once you have started executing the program, you can't alter its behavior. In this section, you will use the principles from *Chapter 3, Providing Speech Input and Output*, to issue voice commands to initiate movement.

You'll need to modify your voice recognition program so that it will run your Python program when it gets a voice command. If you feel your knowledge of how this works is rusty, review *Chapter 3, Providing Speech Input and Output*. You are going to make a simple modification to the `continuous.c` program in `/home/pi/pocketsphinx-0.8/src/`. To do this, type `cd /home/pi/pocketsphinx-0.8/src/programs` and then type `emacs continuous.c`. The changes will appear in the same section as your other voice commands and will look as shown in the following screenshot:



```
pi@raspberrypi: ~/pocketsphinx-0.8/src/programs
File Edit Options Buffers Tools C Help
    sscanf(hyp, "%s", word);
    if (strcmp(hyp, "GOODBYE") == 0)
    {
        system("espeak \"good bye\"");
        break;
    }
    else if (strcmp(hyp, "hello") == 0)
    {
        system("espeak \"hello\"");
    }
    else if (strcmp(hyp, "forward"))
    {
        system("espeak \"moving robot\"");
        system("home/pi/maestro_linux/robot.py");
    }
}

/* Resume A/D recording for next utterance */
if (ad_start_rec(ad) < 0)
    E_FATAL("Failed to start recording\n");
}

-UU-:----F1 continuous.c 80% L339 (C/l Abbrev)-----
Wrote /home/pi/pocketsphinx-0.8/src/programs/continuous.c
```

The additions are pretty straightforward. Let's walk through them:

- `else if (strcmp(hyp, "FORWARD") == 0):` This checks the input word as recognized by your voice command program. If it corresponds with the word FORWARD, you will execute everything within the `if` statement. You use `{` and `}` to tell the system which commands go with this `else if` clause.
- `system("espeak \\"moving robot\\"):` This executes Espeak, which should tell you that you are about to run your robot program.
- `system("/home/pi/maestro_linux/robot.py"):` This indicates the name of the program you will execute. In this case, your mobile platform will do whatever the `robot.py` program tells it to.

After doing this, you will need to recompile the program, so type `make` and the `pocketsphinx_continuous` executable will be created. Run the program by typing `./pocketsphinx_continuous`. Disconnect the LAN cable and the mobile platform will now take the forward voice command and execute your program. You should now have a complete mobile platform! When you execute your program, the mobile platform can now move around based on what you have programmed it to do.

You can use the command-line arguments that you learned about in *Chapter 5, Creating Mobile Robots on Wheels*, to make your robot do many different actions. Perhaps one voice command can move your robot forward, a different one can move it backwards, and another can turn it right or left.

Congratulations! Your robot should now be able to move around in any way you program it to move. You can even have the robot dance. You have now built a two-legged robot and you can easily expand on this knowledge to create robots with even more legs. The following is an image of the mechanical structure of a four-legged robot that has eight DOFs and is fairly easy to create by using many of the parts that you have used to create your two-legged robot; this is my personal favorite because it doesn't fall over and break the electronics:



You'll need eight servos and lots of batteries. If you search eBay, you can often find kits for sale for four-legged robots with 12 DOFs, but remember that the battery will need to be much bigger. For this application, you can use an RC (which stands for remote control) battery. RC batteries are nice as they are rechargeable and can provide lots of power, but make sure you either purchase one that is 5 V to 6 V or include a way to regulate the voltage. The following is an image of such a battery, available at most hobby stores:



If you use this type of battery, don't forget its charger. The hobby store can help with choosing an appropriate match.

Summary

Now, you have the ability to build not only wheeled robots but also robots with legs. It is also easy to expand this ability to robots with arms; controlling the servos for an arm is the same as controlling them for legs. In later chapters, you can even use this ability to control the position of your sonar sensors or webcams. In the next chapter, you'll learn how to connect a sonar sensor and avoid or find obstacles.

7

Avoiding Obstacles Using Sensors

In the previous two chapters, we covered wheeled, tracked, and legged movement. Now your robot can move around. But what if you want the robot to sense the outside world, so it doesn't run into things? In this chapter, you'll discover how to add some sensors to help avoid barriers.

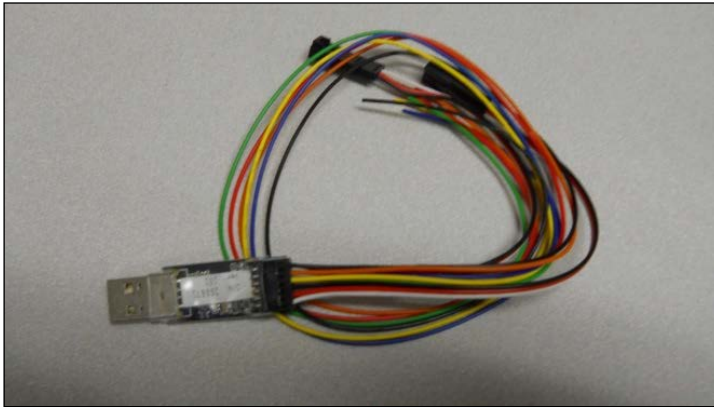
Your robot will take quite a beating if it continually runs into walls, or off the edge of a surface. Let's help your robot avoid these so that it looks intelligent.

In this chapter, you will cover the following:

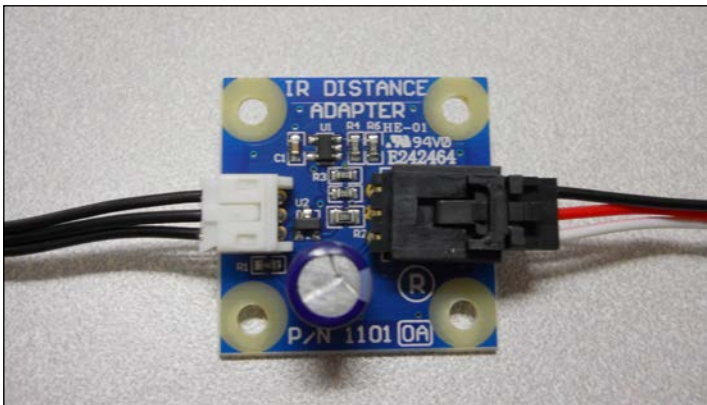
- Gathering the hardware
- Connecting Raspberry Pi to an **infrared (IR)** sensor to detect the world
- Connecting Raspberry Pi to a USB sonar sensor to detect the world
- Using a servo to change the position of your sensor so that a single sensor can view a large field, eliminating the need for additional sensors

To find obstacles, you'll need some sensors. There are two choices and in this chapter I am going to show you how to interface with an infrared sensor and a sonar sensor. It is not an easy choice; both will do an adequate job. If you're not sure which one to use, I suggest you read through this chapter first and then choose which you think will work in your specific application.

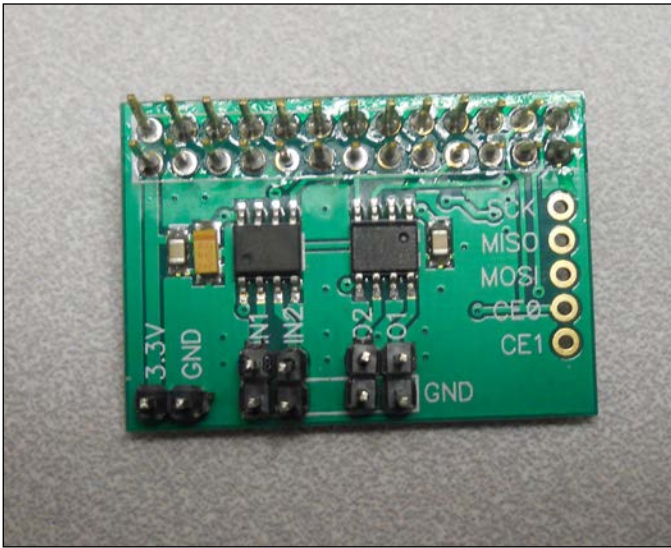
If you are going to choose the infrared sensor, you'll need to add an analog-to-digital converter to your Raspberry Pi, as it does not have one. One way to do this is to use a set of parts that includes not only the sensor, but also an A/D converter that communicates with the Raspberry Pi. This is offered by www.phidgets.com. This board is really quite amazing; it takes the analog signals, turns them into digital numbers using an analog to digital converter, and then makes them available so that they can be read from the USB port. The model number of this part is 1011_0-PhidgetInterfaceKit 2/2/2 and the following is an image of it:



If you want to use the Phidget ADC USB interface, the second item you'll need is an IR distance adapter that can provide the signals to the sensor and condition the signals coming back from the sensor. The model number is 1101_0-IR Distance Adapter, which is shown in the following image:



However, there is also an A/D that can be added to the Raspberry Pi that can use its GPIO pins to communicate with an infrared sensor. The ADC-DAC Pi from www.abelectronics.co.uk, is shown in the following image:



Whether you use an ADC that interfaces through USB or through the GPIO, you'll need the sensor itself. This sensor is made by Sharp and they come in several different distance specifications. The following image is the GP2Y0A02YK, the 20-150 centimeters version:



If you purchase this sensor from www.phidgets.com, you can get the black interface cable as shown in the preceding image, which will be useful if you are going to use their USB ADC. If not, the sensor is available at a number of different electronics online merchants; www.amazon.com and www.adafruit.com, for example.

If you are going to go with the sonar sensor, the following is an image of the USB sonar sensor I like to use on my projects:



It is the USB-ProxSonar-EZ sensor, and can be purchased directly from MaxBotix or Amazon. There are several models, each with a different distance specification; however, they all work in the same way.

Whether you choose an infrared sensor or a sonar sensor, you may want to detect distance in more than just one direction. You have two choices. The first is simply to use a number of these sensors, one in each direction. The second option is to use a single sensor that can be turned into different directions. But in the *Connecting the IR sensor using the GPIO ADC* section of this chapter, you will learn how to use a servo to rotate the sensor. To complete this section, you'll need a servo and a way to mount it on your project. I like the Hitec series of servos, and this is ready-made for an HS-311 servo, which should look as shown in the following image:

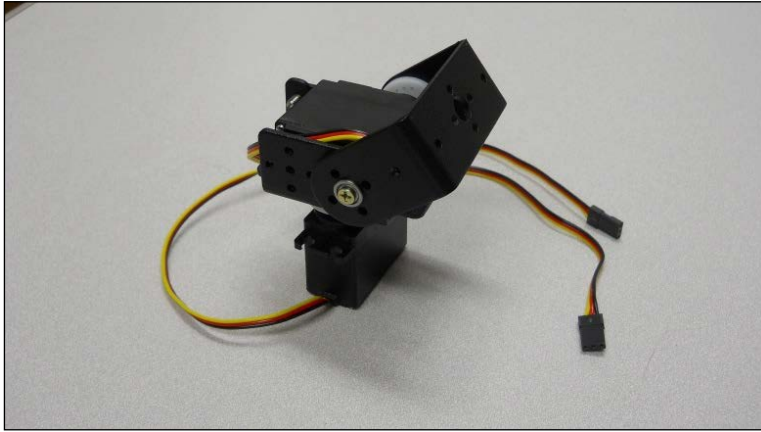


Here is a way to mount the sensor on a 90-degree angle bracket. I used one from a robot kit I purchased on eBay. It can connect to the servo as shown in the following screenshot:



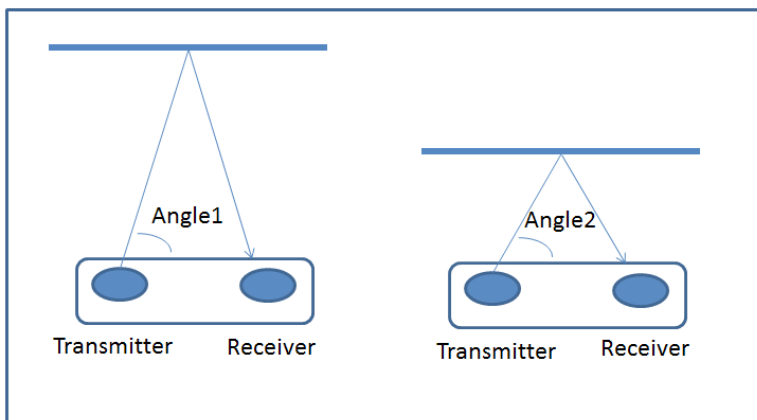
However, if you want to get really fancy, you can purchase a *Pan and Tilt assembly*. These contain two servos and they allow you to rotate your sensor in both the vertical and horizontal axes. They are available from online stores, such as www.robotshop.com. You can also construct a Pan and Tilt assembly out of components that you may have, if you purchased a legged robot kit.

The finished product with servos looks as shown in the following image:



Connecting Raspberry Pi to an infrared sensor using USB

The ability of the robot to move around is impressive, but it won't be if your robot keeps running into barriers. To avoid this, you'll want to be able to sense a barrier or a target. One of the ways to do this is with an IR sensor. Now for a little tutorial on IR sensors. The sensor you are using has both a transmitter and a sensor. The transmitter sends out a narrow beam of light, and the sensor receives this beam of light. The difference in transit ends up as an angle measurement at the sensor, as shown in the following figure:

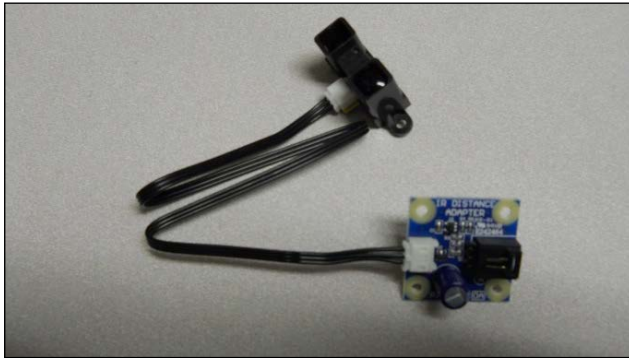


The different angles give you an indication of the distance from the object. Unfortunately, the relationship between the output of the sensor and the distance is not linear, so you'll need to do some calibration to predict the actual distance and its relationship to the output of the sensor.

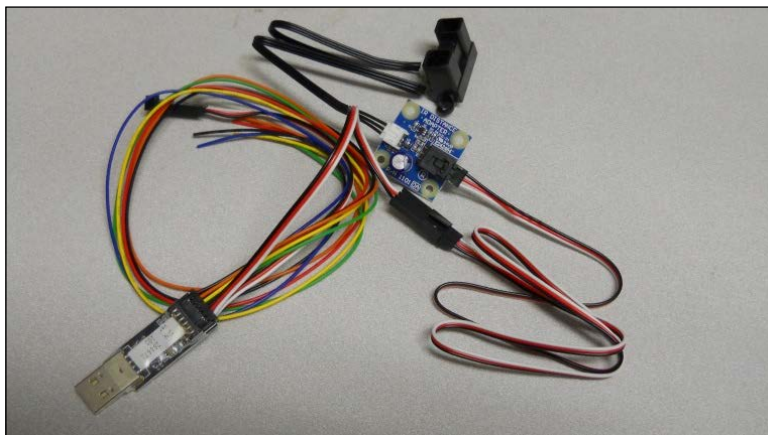
Connecting a sensor using the USB interface

If you have chosen the Phidget USB interface, here are the steps to connect the sensor:

1. The first step is to connect the Sharp IR sensor to the IR Distance Adapter. Simply take the output of the sensor and insert it into the white connector on the adapter board, as shown in the following image:



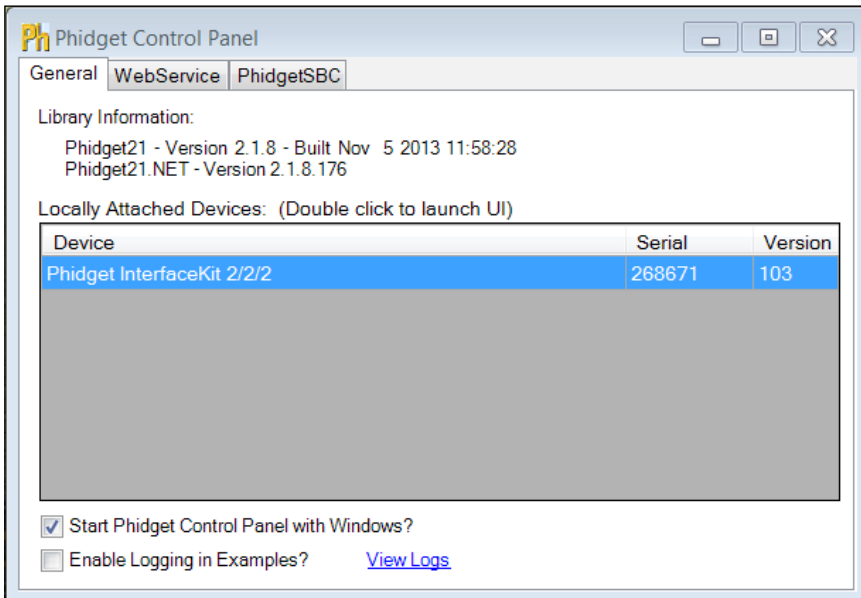
2. Now you need to connect the IR Distance Adapter to the USB interface board. Connect the board using the two cables, as shown in the following image (fortunately, there is only one way to connect them):



Now that everything is connected, you can begin to access the data from Raspberry Pi. To do this, perform the following steps:

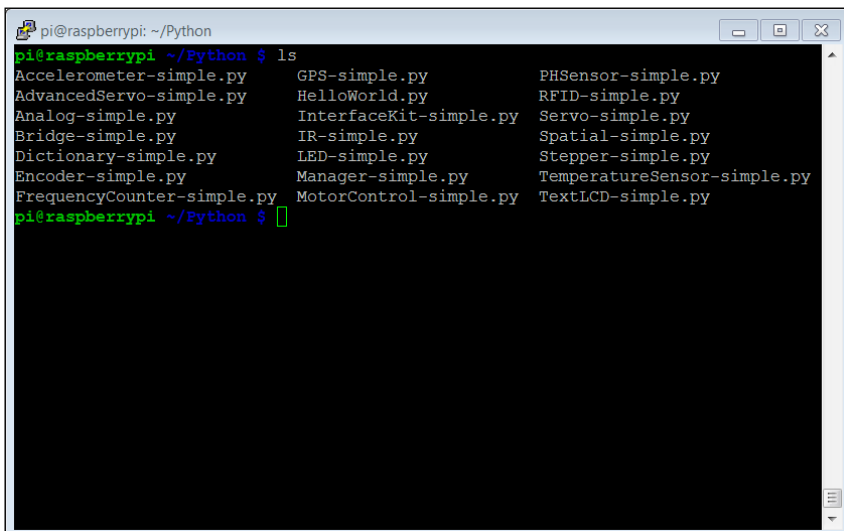
1. The first step is to look at the data from a PC. For this, you'll first need to download some software from http://www.phidgets.com/docs/OS_-_Windows#Quick_Downloads. Go to this website and go to the *Getting Started with Windows* section.
2. After you have downloaded the software, run the downloaded installation software and it will install the **Phidgets Control Panel** application in the Phidgets folder.
3. You can then run the software from the **Start** menu by selecting the Phidgets folder and then the **Phidgets Control Panel** application.

When you run the software with your Phidget USB device plugged in, you should see the following screenshot:



Note that your Phidget device is being recognized by the system. Now you can move the device to Raspberry Pi. Plug the Phidget device into Raspberry Pi. Then follow the directions given at http://www.phidgets.com/docs/OS_-_Linux#Installing.

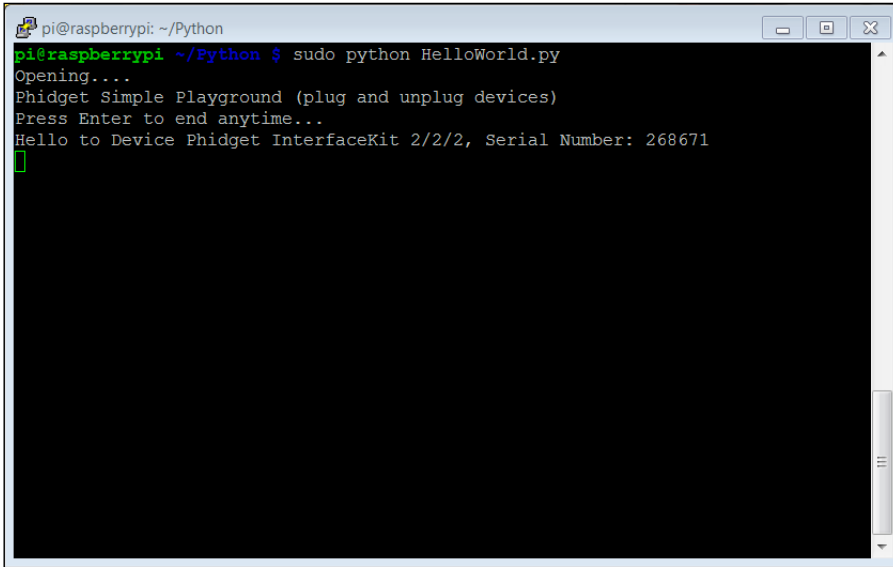
1. Download the Phidgets Libraries from the website. You can either install them directly to your Raspberry Pi, use WinSCP to first download them to your PC and then transfer them to Raspberry Pi, or use `wget` to transfer them by typing `wget http://www.phidgets.com/downloads/libraries/libphidget.tar.gz`. In any case, you'll then need to unzip them, so they are in your Raspberry Pi's home directory. You can use the `tar -xzf` command to accomplish this.
2. Once you have unzipped the files, go to the directory created by the `tar -xzf` command. Type `./configure`, then `make`, and then `sudo make install`.
3. The next step is to install the Python modules. Install them from http://www.phidgets.com/docs/Language_-_Python#Linux by selecting the **Phidget Python Module** option. You will then need to unzip this file using the `unzip` command. Go to the directory created by this unzipping process, by typing `cd PhidgetsPython`. Then type `sudo python setup.py install`.
4. Now install the Python examples by downloading the code from http://www.phidgets.com/docs/Language_-_Python#Use_Our_Examples_5. This will be a ZIP file, so unzip this using the `unzip` command, and the directory `Python` will be created with a number of examples. Go to the directory by typing `cd Python`. You should see the following screenshot:



```

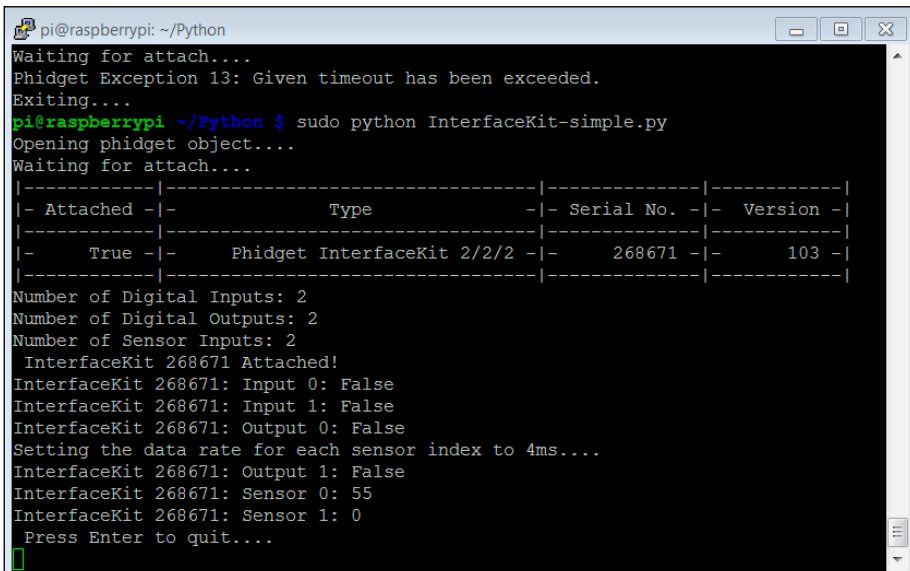
pi@raspberrypi: ~/Python
pi@raspberrypi ~/Python $ ls
Accelerometer-simple.py  GPS-simple.py          PHSensor-simple.py
AdvancedServo-simple.py  HelloWorld.py          RFID-simple.py
Analog-simple.py         InterfaceKit-simple.py  Servo-simple.py
Bridge-simple.py         IR-simple.py           Spatial-simple.py
Dictionary-simple.py     LED-simple.py          Stepper-simple.py
Encoder-simple.py        Manager-simple.py       TemperatureSensor-simple.py
FrequencyCounter-simple.py  MotorControl-simple.py TextLCD-simple.py
pi@raspberrypi ~/Python $
  
```

- Let's see if the system can sense your Phidget device. Type `sudo python HelloWorld.py`. You should see the following screenshot:



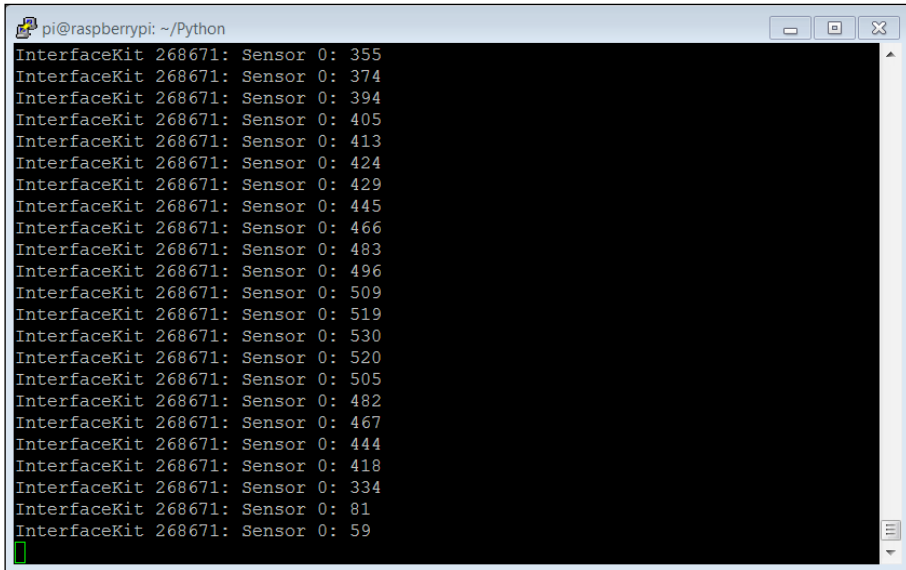
```
pi@raspberrypi: ~/Python
pi@raspberrypi ~/Python $ sudo python HelloWorld.py
Opening....
Phidget Simple Playground (plug and unplug devices)
Press Enter to end anytime...
Hello to Device Phidget InterfaceKit 2/2/2, Serial Number: 268671
█
```

- Now you can run the `InterfaceKit-simple.py` example by typing `sudo python InterfaceKit-simple.py`. This will allow you to sense a target on your sensor. You should see the following screenshot if no targets are in front of the sensor:



```
pi@raspberrypi: ~/Python
Waiting for attach....
Phidget Exception 13: Given timeout has been exceeded.
Exiting....
pi@raspberrypi ~/Python $ sudo python InterfaceKit-simple.py
Opening phidget object....
Waiting for attach....
|-----|-----|-----|-----|
|- Attached -|-          Type          -|- Serial No. -|- Version -|
|-----|-----|-----|-----|
|-   True   -|-  Phidget InterfaceKit 2/2/2 -|-   268671 -|-    103 -|
|-----|-----|-----|-----|
Number of Digital Inputs: 2
Number of Digital Outputs: 2
Number of Sensor Inputs: 2
  InterfaceKit 268671 Attached!
InterfaceKit 268671: Input 0: False
InterfaceKit 268671: Input 1: False
InterfaceKit 268671: Output 0: False
Setting the data rate for each sensor index to 4ms....
InterfaceKit 268671: Output 1: False
InterfaceKit 268671: Sensor 0: 55
InterfaceKit 268671: Sensor 1: 0
  Press Enter to quit....
█
```

7. Place a target in front of the sensor and then remove it; you will see the following screenshot:

A screenshot of a terminal window titled 'pi@raspberrypi: ~/Python'. The window displays a series of 20 lines of text, each representing a sensor reading from an 'InterfaceKit 268671'. The readings are: 355, 374, 394, 405, 413, 424, 429, 445, 466, 483, 496, 509, 519, 530, 520, 505, 482, 467, 444, 418, 334, 81, and 59. The values start at 355, rise to a peak of 530, and then generally decrease to 59. A green cursor is visible at the end of the last line.

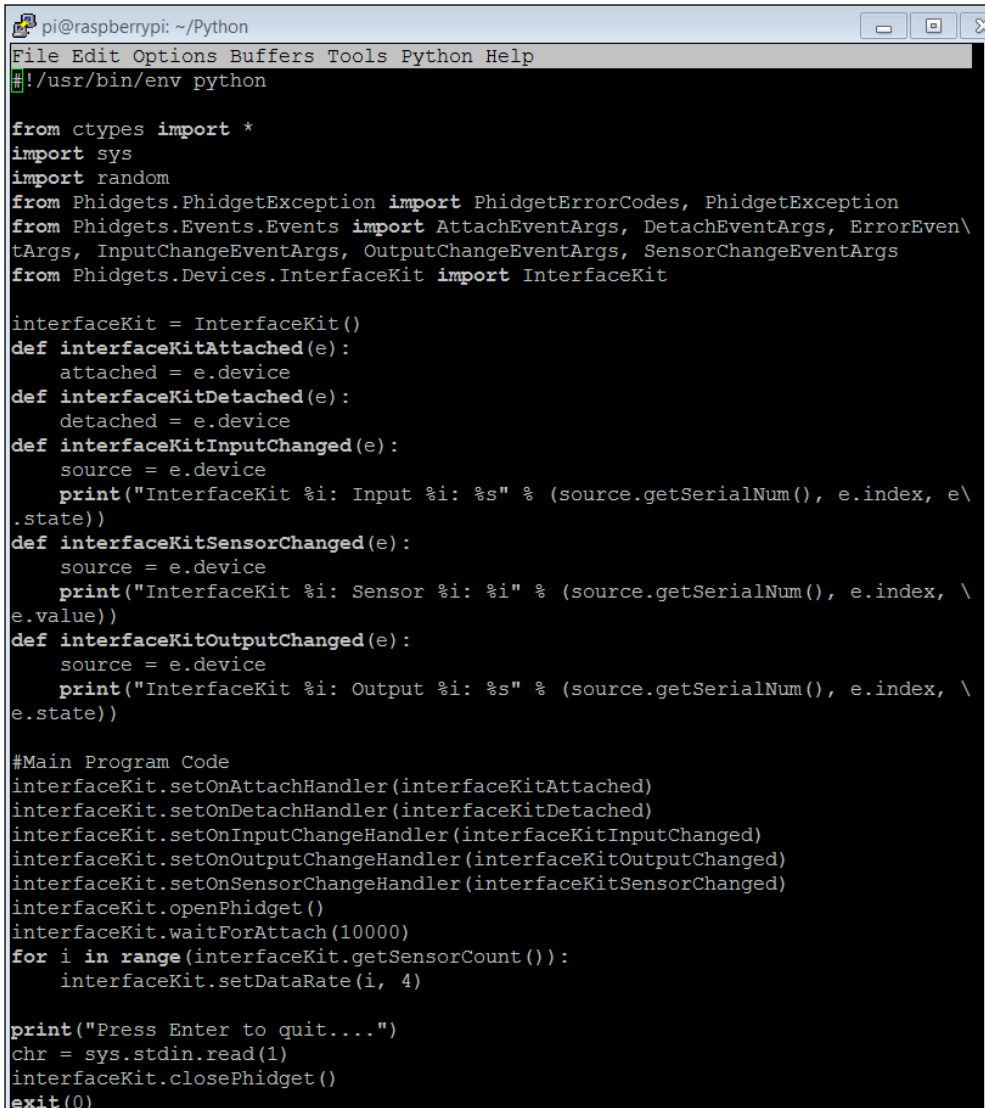
```
pi@raspberrypi: ~/Python
InterfaceKit 268671: Sensor 0: 355
InterfaceKit 268671: Sensor 0: 374
InterfaceKit 268671: Sensor 0: 394
InterfaceKit 268671: Sensor 0: 405
InterfaceKit 268671: Sensor 0: 413
InterfaceKit 268671: Sensor 0: 424
InterfaceKit 268671: Sensor 0: 429
InterfaceKit 268671: Sensor 0: 445
InterfaceKit 268671: Sensor 0: 466
InterfaceKit 268671: Sensor 0: 483
InterfaceKit 268671: Sensor 0: 496
InterfaceKit 268671: Sensor 0: 509
InterfaceKit 268671: Sensor 0: 519
InterfaceKit 268671: Sensor 0: 530
InterfaceKit 268671: Sensor 0: 520
InterfaceKit 268671: Sensor 0: 505
InterfaceKit 268671: Sensor 0: 482
InterfaceKit 268671: Sensor 0: 467
InterfaceKit 268671: Sensor 0: 444
InterfaceKit 268671: Sensor 0: 418
InterfaceKit 268671: Sensor 0: 334
InterfaceKit 268671: Sensor 0: 81
InterfaceKit 268671: Sensor 0: 59
```



Note that the value increases as the target gets closer and then decreases when the target is removed.

You can calibrate your sensor by noting the value returned for a corresponding distance from the target. This can allow your project to know how far the target is. The code for this capability is a bit complicated to detail in this text, but basically it goes out, senses the presence of the Phidget device, sets the device to sense the IR sensor changes, and returns them to the user.

To create your own code, follow the tutorial at http://www.phidgets.com/docs/Language_-_Python#Follow_the_Examples. The following screenshot shows a simpler example of the code that may help you implement your IR capability:



```
pi@raspberrypi: ~/Python
File Edit Options Buffers Tools Python Help
#!/usr/bin/env python

from ctypes import *
import sys
import random
from Phidgets.PhidgetException import PhidgetErrorCodes, PhidgetException
from Phidgets.Events.Events import AttachEventArgs, DetachEventArgs, ErrorEvent\
Args, InputChangeEventArgs, OutputChangeEventArgs, SensorChangeEventArgs
from Phidgets.Devices.InterfaceKit import InterfaceKit

interfaceKit = InterfaceKit()
def interfaceKitAttached(e):
    attached = e.device
def interfaceKitDetached(e):
    detached = e.device
def interfaceKitInputChanged(e):
    source = e.device
    print("InterfaceKit %i: Input %i: %s" % (source.getSerialNum(), e.index, e\
.state))
def interfaceKitSensorChanged(e):
    source = e.device
    print("InterfaceKit %i: Sensor %i: %i" % (source.getSerialNum(), e.index, \
e.value))
def interfaceKitOutputChanged(e):
    source = e.device
    print("InterfaceKit %i: Output %i: %s" % (source.getSerialNum(), e.index, \
e.state))

#Main Program Code
interfaceKit.setOnAttachHandler(interfaceKitAttached)
interfaceKit.setOnDetachHandler(interfaceKitDetached)
interfaceKit.setOnInputChangeHandler(interfaceKitInputChanged)
interfaceKit.setOnOutputChangeHandler(interfaceKitOutputChanged)
interfaceKit.setOnSensorChangeHandler(interfaceKitSensorChanged)
interfaceKit.openPhidget()
interfaceKit.waitForAttach(10000)
for i in range(interfaceKit.getSensorCount()):
    interfaceKit.setDataRate(i, 4)

print("Press Enter to quit...")
chr = sys.stdin.read(1)
interfaceKit.closePhidget()
exit(0)
```

The following is a description of the code used in the preceding screenshot:

- `#!/usr/bin/env python`: This sets up the code so that you can run it from the command line without the Python directory.
- `from ctypes import *`: This imports the `ctypes` library, a library that allows Python to specify C data types.
- `import sys`: This imports the `sys` library.
- `import random`: This imports the `random` library and allows you access to random variables.
- `from Phidgets.PhidgetException import PhidgetErrorCodes, PhidgetException`: This imports libraries for Phidgets capabilities.
- `from Phidgets.Events.Events import AttachEventArgs, DetachEventArgs, ErrorEventArgs, InputChangeEventArgs, OutputChangeEventArgs, SensorChangeEventArgs`: This imports even more capabilities of Phidgets as a library.
- `from Phidgets.Devices.InterfaceKit import InterfaceKit`: This is one last import for the interface functionality from the Phidgets library.
- `interfaceKit = InterfaceKit()`: This creates an instance of an interface to your Phidgets device.
- `def interfaceKitAttached(e) :` The following lines will create callback functions. These are the functions that are called by the device when a certain event occurs. This is the callback function for an attach event.
- `attached = e.device`: This simply attaches a device when asked.
- `def interfaceKitDetached(e) :` This is the callback function for an attach event.
- `detached = e.device`: This detaches the device when asked.
- `def interfaceKitInputChanged(e) :` This is the callback function for an input changed event.
- `source = e.device`: This defines the device that is communicating with you.
- `print("InterfaceKit %i: Input %i: %s" % (source.getSerialNum(), e.index, e.state))`: This prints out the result of the event.

- `def interfaceKitSensorChanged(e) ::` This is the callback function for a sensor changed event. This is the function that is called when the value changes and will get the value from the sensor. It is here that you put additional code to do other things with the sensor reading.
- `source = e.device:` This defines the device that is communicating with you.
- `print("InterfaceKit %i: Sensor %i: %i" % (source.getSerialNum(), e.index, e.value)):` This prints out the result of the event.
- `def interfaceKitOutputChanged(e) ::` This is the callback function for an output changed event.
- `source = e.device:` This defines the device that is communicating with you.
- `print("InterfaceKit %i: Output %i: %s" % (source.getSerialNum(), e.index, e.state)):` This prints out the result of the event.
- **#Main Program Code:** The following lines attach the callback functions to the device:

```
interfaceKit.setOnAttachHandler(interfaceKitAttached)
interfaceKit.setOnDetachHandler(interfaceKitDetached)
interfaceKit.setOnInputChangeHandler(
    interfaceKitInputChanged)
interfaceKit.setOnOutputChangeHandler(
    interfaceKitOutputChanged)
interfaceKit.setOnSensorChangeHandler(
    interfaceKitSensorChanged)
```
- `interfaceKit.openPhidget():` This opens the device and attaches all the callbacks.
- `interfaceKit.waitForAttach(10000):` This tells the program to wait to make sure the device attaches.
- `for i in range(interfaceKit.getSensorCount()) ::` This creates a loop for all the sensors available.
- `interfaceKit.setDataRate(i, 4):` This sets the update rate to 4 milliseconds.
- `print("Press Enter to quit...."):` This tells the user to hit *Enter* to quit the program.

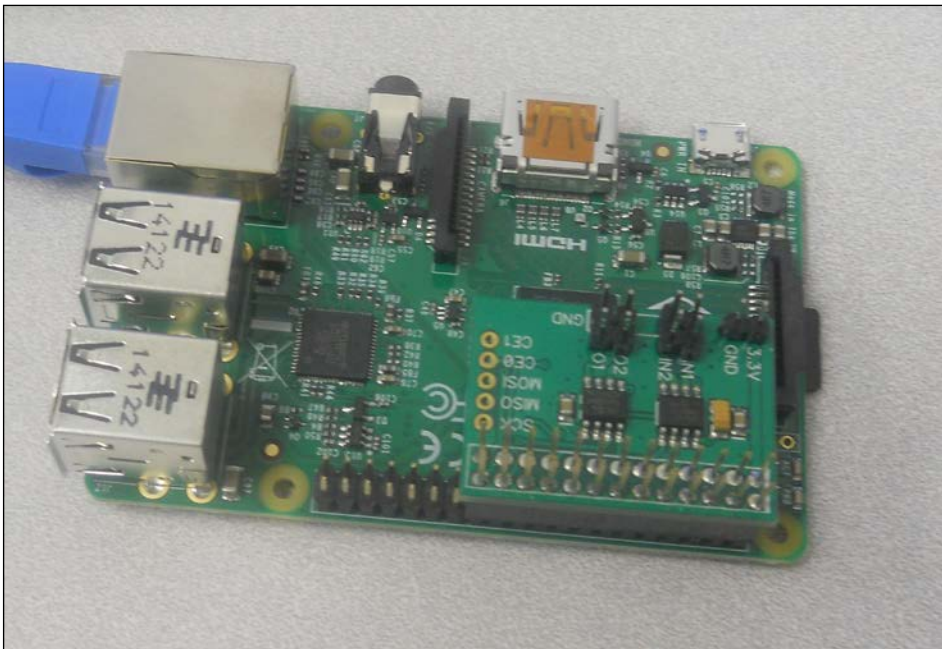
- `chr = sys.stdin.read(1)`: This tells the program to get ready to read in a character. When the character comes, this will close the program.
- `interfaceKit.closePhidget()`: This closes the interface to the object.
- `exit(0)`: This exits the program.

Now your project can sense objects!

Connecting the IR sensor using the GPIO ADC

If you have chosen the ADC-DAC GPIO interface, here are the steps to connect the IR sensor:

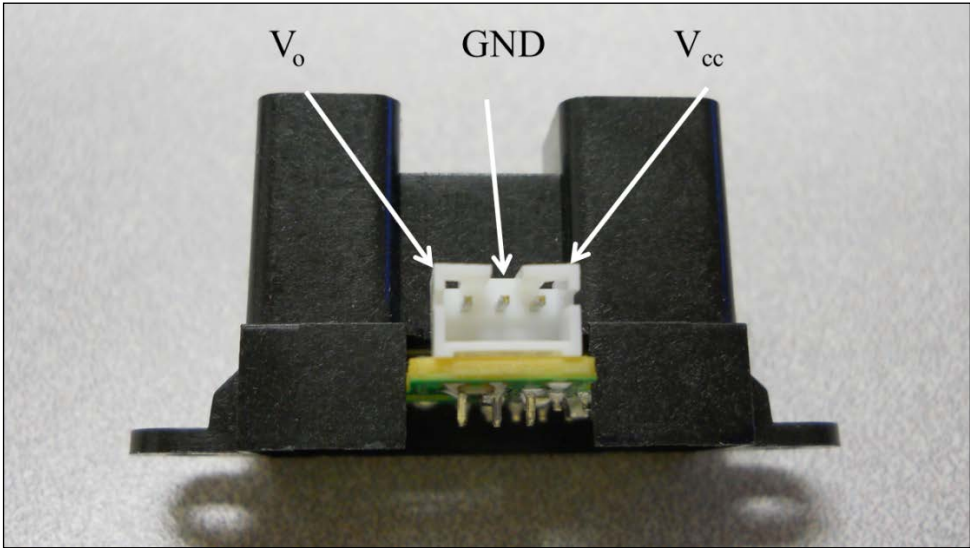
1. Plug the ADC-DAC board into the Raspberry Pi. The following is a picture of the combination:



2. Now you'll connect the IR sensor to the ADC. To connect this unit, you'll connect the three pins that are available at the bottom of the sensor. The following is the connection list:

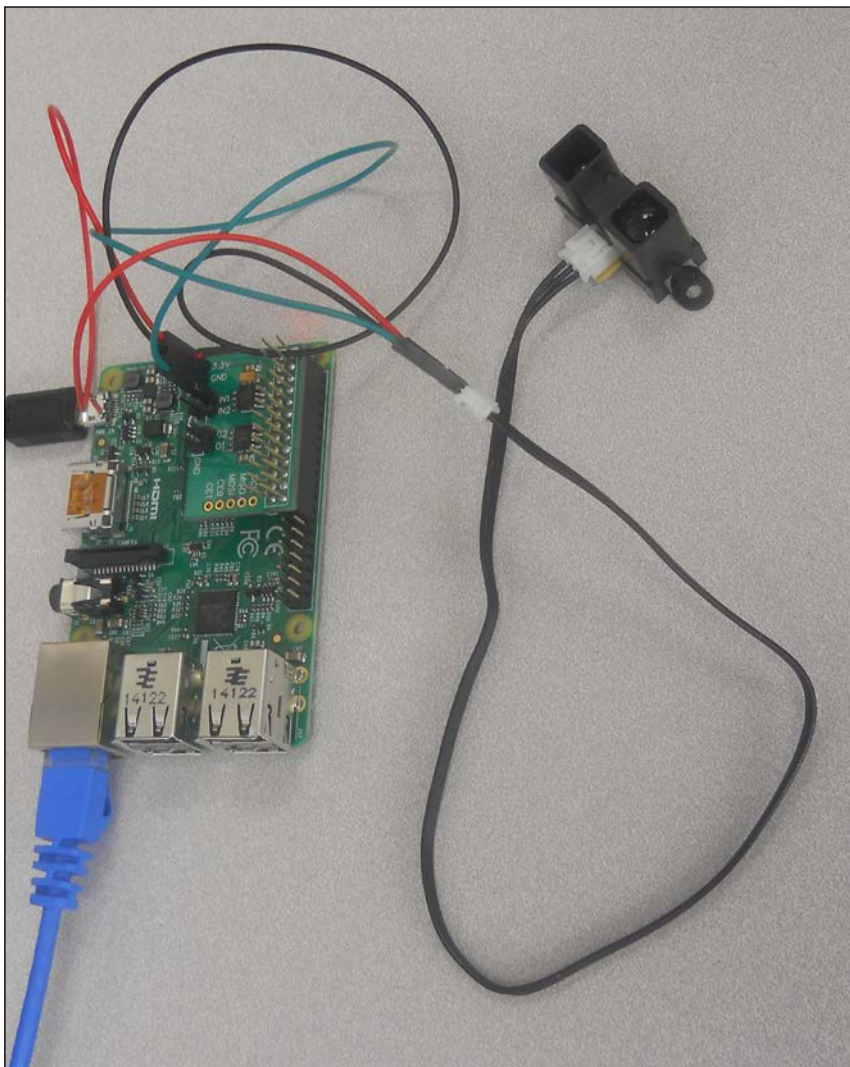
ADC-DAC board	Sensor pin
3.3V	Vcc
GND	Gnd
In1	Vo

Unfortunately, there are no labels on the unit, but the following is the screenshot of the pins you'll connect:



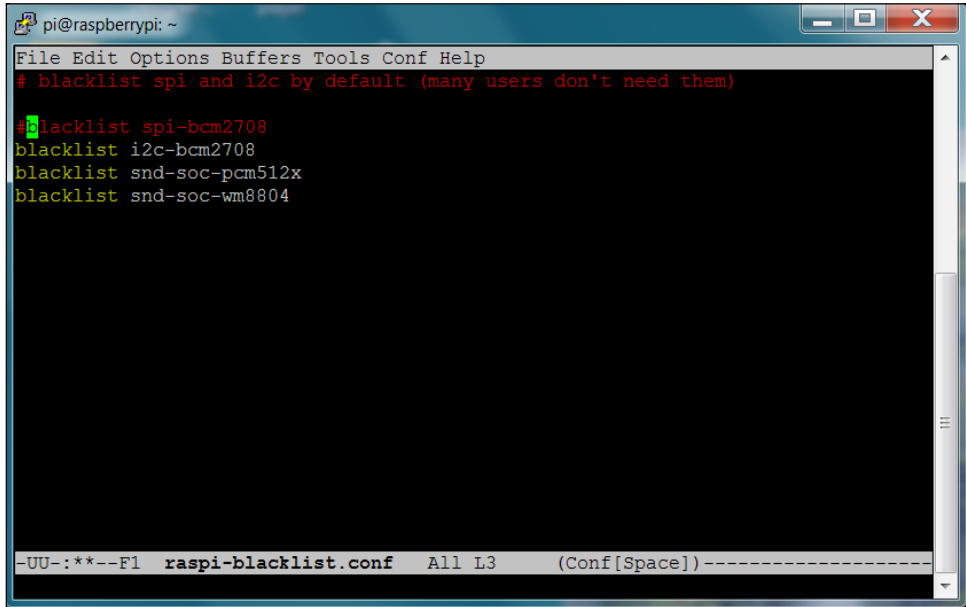
One of the challenges in making this connection is, that the female to male connection jumpers are too big to connect directly to the sensor. You'll have to arrange the three wire cable with connectors along with the sensor, and then you can make the connections between this cable and the Raspberry Pi using the male to male jumper wires.

Once the pins are connected, you are ready to access the data from the sensor, via a python program on the Raspberry Pi. The entire system looks as shown in the next image:



Now you are ready to add some code to read the IR sensor. But you'll need to follow these steps to make the sensor talk to the ADC:

1. The first step is to enable the SPI interface, which you will use to talk to the ADC board. To do this, type `sudo emacs /etc/modprobe.d/raspi-blacklist.conf` and comment out the line `blacklist spi-bcm2708` by adding a `#` at the start of the line. Following is a screenshot depicting this:



```
pi@raspberrypi: ~
File Edit Options Buffers Tools Conf Help
# blacklist spi and i2c by default (many users don't need them)

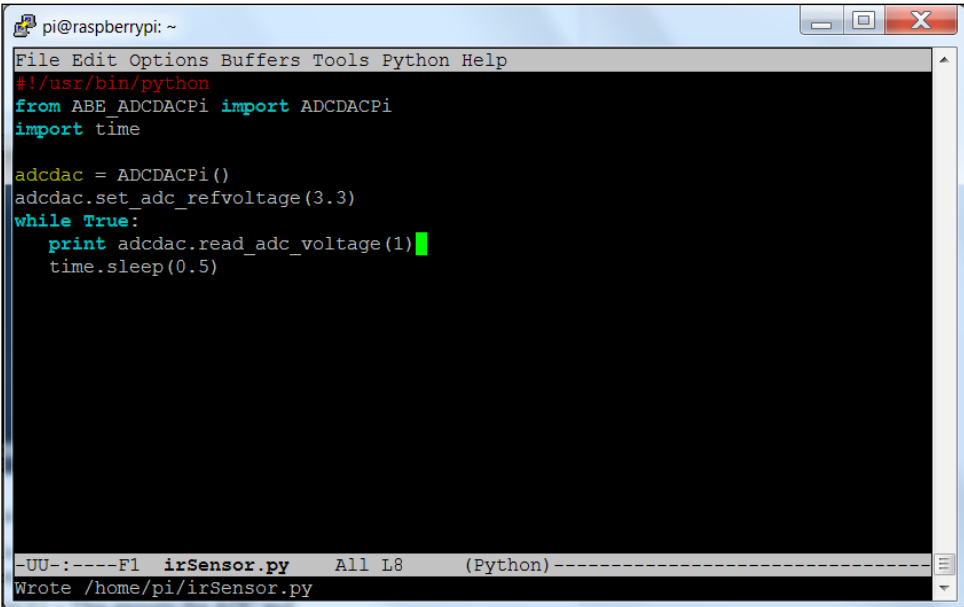
#blacklist spi-bcm2708
blacklist i2c-bcm2708
blacklist snd-soc-pcm512x
blacklist snd-soc-wm8804

-UU-:***-F1 raspi-blacklist.conf All L3 (Conf[Space])
```

2. Now reboot the Raspberry Pi.
3. Log on, and at the command prompt, type `sudo apt-get update`. This will update all the repositories from which you may want to download the code.
4. Now type `mkdir python-spi`, and then `cd python-spi`. This will create a directory to download and create the `python spi` library.
5. Type `wget https://raw.github.com/doceme/py-spidev/master/setup.py`, and then `wget https://raw.github.com/doceme/py-spidev/master/spidev_module.c`. This will bring in two sets of code that will provide the capability to install the `python spi` library.

6. The next step is to install the library by typing `sudo python setup.py install`.
7. Now you'll need to get one more set of library capabilities. Go back to your home directory by typing `cd ~`, and then type `git clone https://github.com/abelectronicsuk/ABElectronics_Python_Libraries.git`.
8. Finally, you'll need to update your path variable, so the system will know where this new capability is, by typing `export PYTHONPATH=$:~/ABElectronics_Python_Libraries/ADCDACPi/`

Now that all this is installed, you can create a program in Python to read the ADC. Following is the screenshot of the program:



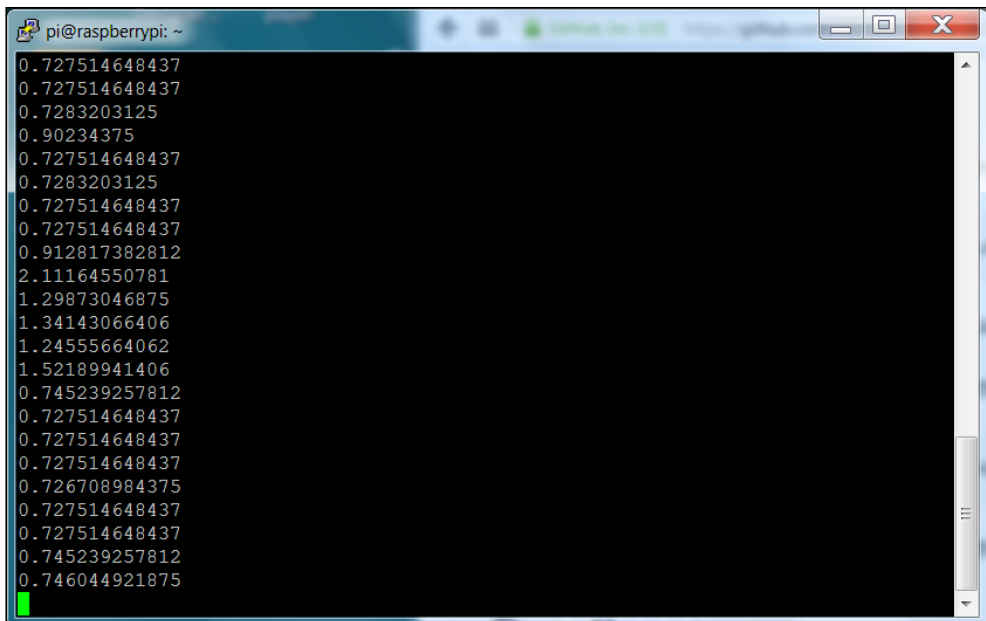
```
pi@raspberrypi: ~  
File Edit Options Buffers Tools Python Help  
#!/usr/bin/python  
from ABE_ADCDACPi import ADCDACPi  
import time  
  
adcdac = ADCDACPi()  
adcdac.set_adc_refvoltage(3.3)  
while True:  
    print adcdac.read_adc_voltage(1)  
    time.sleep(0.5)  
  
-UU-:----F1  irSensor.py  All L8  (Python)-----  
Wrote /home/pi/irSensor.py
```

This program is very short, but here are the details:

- `#!/usr/bin/python`: This line allows you to run the program without invoking Python at the command prompt
- `from ABE_ADCDACPi import ADCDACPi`: This imports the ADC and DAC Pi Python libraries

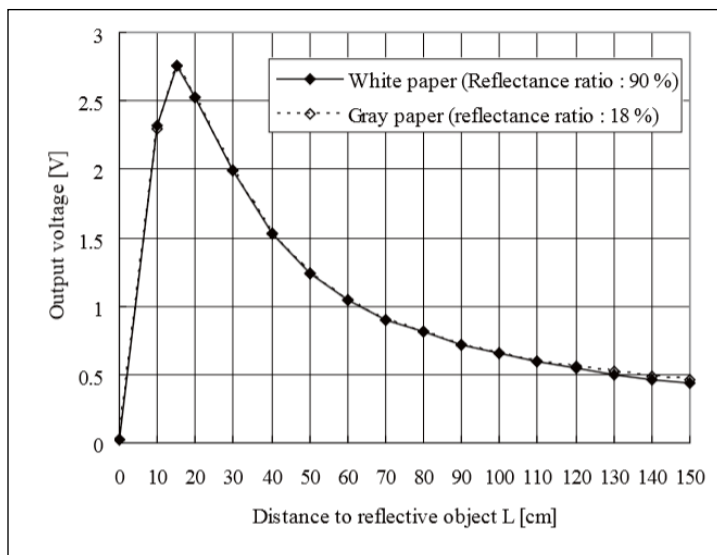
- `import time`: This imports the time library. This will allow you to invoke a time delay in your program
- `adcdac = ADCDACPi()`: This creates an instance of the ADC so you can interact with it
- `adcdac.set_adc_refvoltage(3.3)`: This sets the reference voltage for your ADC to 3.3 volts
- `while True::` This makes the program continue, until you interrupt it with `CTRL + C`
- `print adcdac.read_adc_voltage(1)`: This prints the voltage at Port 1 of the ADC
- `time.sleep(0.5)`: This makes the program sleep for half a second

To run the program, type `python irSensor.py`. You should see something like the following screenshot:



```
pi@raspberrypi: ~  
0.727514648437  
0.727514648437  
0.7283203125  
0.90234375  
0.727514648437  
0.7283203125  
0.727514648437  
0.727514648437  
0.912817382812  
2.11164550781  
1.29873046875  
1.34143066406  
1.24555664062  
1.52189941406  
0.745239257812  
0.727514648437  
0.727514648437  
0.727514648437  
0.726708984375  
0.727514648437  
0.727514648437  
0.745239257812  
0.746044921875
```

These raw readings are great, but you'll want to translate these to distance. To do this, you'll need a graph of the voltage to know the distance readings for your sensor. The following is the graph for the IR sensor in this example:



There are really two parts to the curve; the first is the distance up to about 15 centimeters, then the distance from 15 centimeters out to 150 centimeters. It is the easiest way to build a simple mathematical model that ignores distances closer than 15 centimeters and accurately translates the voltage to the distance from 15 centimeters out.

For more information on how to build this model, see <http://davstott.me.uk/index.php/2013/06/02/raspberry-pi-sharp-infrared/>. The following screenshot shows the program using this model:

```

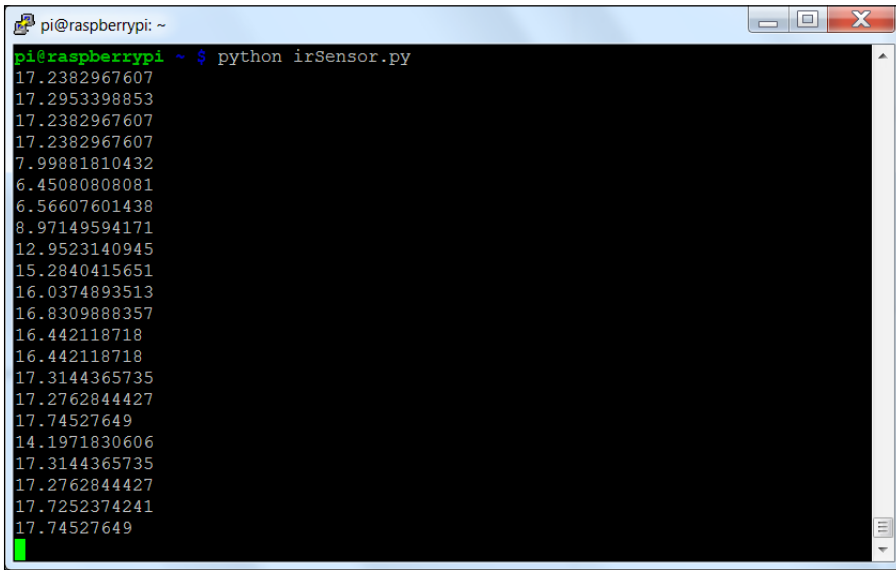
pi@raspberrypi: ~
File Edit Options Buffers Tools Python Help
~/usr/bin/python
from ABE_ADCDACPi import ADCDACPi
import time

adcdac = ADCDACPi()
adcdac.set_adc_refvoltage(3.3)
while True:
    distance = (1.0 / (adcdac.read_adc_voltage(1) / 13.15)) - 0.35
    print distance
    time.sleep(0.5)

-UU-:----F1  irSensor.py  All L1  (Python)
For information about GNU Emacs and the GNU system, type C-h C-a.

```

The only new line of code is the `distance = (1.0 / (adcdac.read_adc_voltage(1) / 13.15)) - 0.35` line, which converts your voltage to distance. You can now run your program and you'll see the results in centimeters, as seen in the following screenshot:

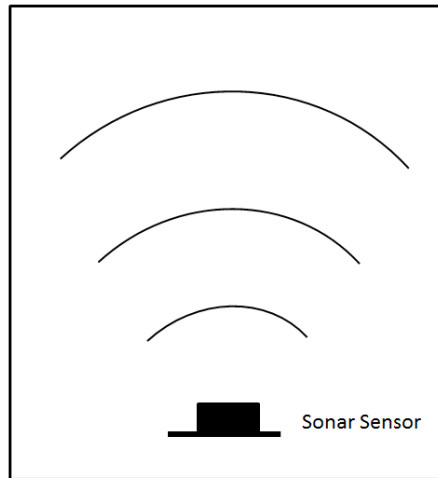


```
pi@raspberrypi: ~  
pi@raspberrypi ~$ python irSensor.py  
17.2382967607  
17.2953398853  
17.2382967607  
17.2382967607  
7.99881810432  
6.45080808081  
6.56607601438  
8.97149594171  
12.9523140945  
15.2840415651  
16.0374893513  
16.8309888357  
16.442118718  
16.442118718  
17.3144365735  
17.2762844427  
17.74527649  
14.1971830606  
17.3144365735  
17.2762844427  
17.7252374241  
17.74527649
```

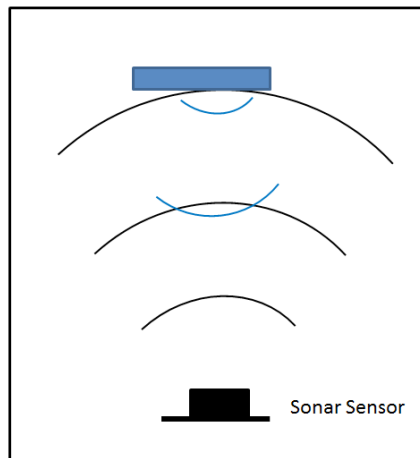
Now, you can measure the distance to the objects!

Connecting Raspberry Pi to a USB sonar sensor

There is yet another way to sense the presence of objects; by using a sonar sensor. But before you add this capability to your system, here's a little tutorial on sonar sensors. This type of sensor uses ultrasonic sound to calculate the distance from an object. The sound wave travels out from the sensor, as illustrated in the following figure:



The device sends out a sound wave 10 times per second. If an object is in the path of these waves, then the waves reflect off the object, sending waves that return to the sensor, as shown in the following figure:



The sensor then measures any return. It uses the time difference between when the sound wave was sent out and when it returned, to measure the distance from the object.

Connecting the hardware

The first thing you'll want to do is connect the USB sonar sensor to your PC, just to make sure everything works well. Perform the following steps to do so:

1. First, download the terminal emulator software from <http://www.maxbotix.com/articles/059.htm> and click on the **Windows Download** button.

MaxBotix
High Performance Ultrasonic Rangefinders

High performance ultrasonic rangefinders

Follow MaxBotix:
Like +1.2k +1

View Cart

Home
Products / Buy Now
Documents & Downloads
Performance Data
Tutorials & Application Notes
Contact
News

SENSORS PORTAL MAGAZINE TOP 10 Products of 2012

We are glad to support
FRC
FIRST Robotics Competition

F.I.R.S.T.® ROBOTICS 2014

Terminal Program Setup Guide
Written By: Tom Bonar | Date Posted: 10-25-2012 | Updated 04-03-2013

This article provides instruction on the easy setup for the MaxBotix® Inc., USB-MaxSonar® ultrasonic sensor lines. This instructional set will help you set up the USB-MaxSonar® ultrasonic sensors with your computer system.

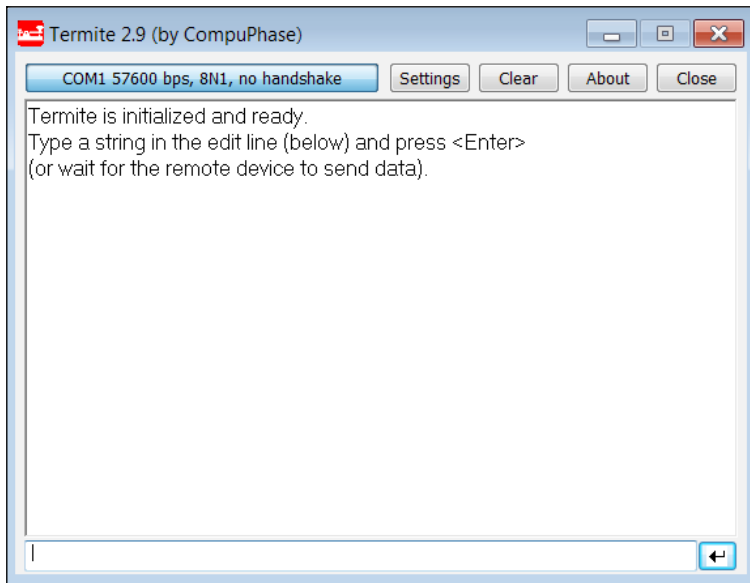
Windows Download
Linux Download
Apple Download

Please use your preferred operating system instruction set:
[Windows](#)
[Linux](#)
[Apple OS](#)

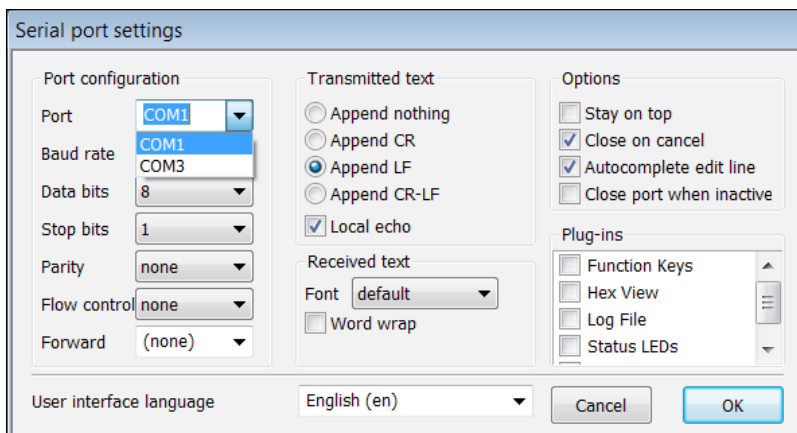
2. Unzip this file. Then, plug the sensor into a USB port on your PC and open the terminal emulator file by selecting this file from the directory.

Name	Date modified	Type	Size
Function_Keys.flt	9/16/2013 5:53 PM	FLT File	36 KB
Hex_View.flt	9/16/2013 5:53 PM	FLT File	36 KB
Log_File.flt	9/16/2013 5:53 PM	FLT File	47 KB
setup.log	9/16/2013 5:53 PM	Text Document	2 KB
Status_LEDs.flt	9/16/2013 5:53 PM	FLT File	33 KB
Termite.exe	9/16/2013 5:53 PM	Application	116 KB
Timestamp.flt	9/16/2013 5:53 PM	FLT File	39 KB
WritingFilters.pdf	9/16/2013 5:53 PM	Adobe Acrobat D...	66 KB

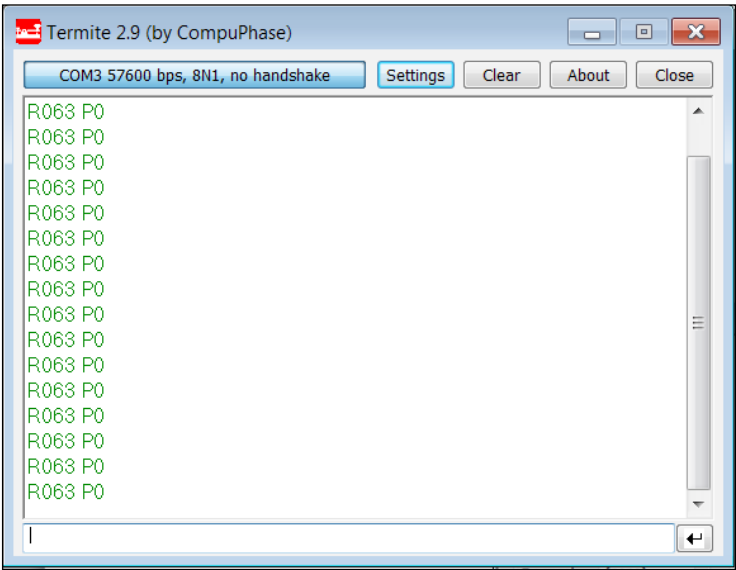
- The following application window should pop up:



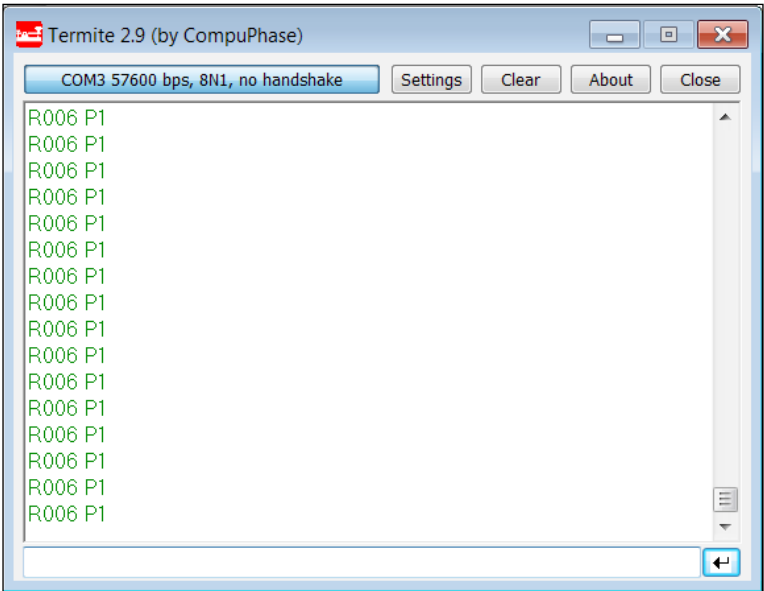
- We'll need to change the setting to find the sensor, so click on the **Settings** button, and you should see the following screenshot:



5. Select the **Port** menu and select the port that is connected to your sensor. In my case, I selected **COM3**, clicked on **OK**, and the following screenshot is what I saw on the main screen:



6. Note the sensor readings. Now place an object in front of the sensor. You should now see something as shown in the following screenshot:

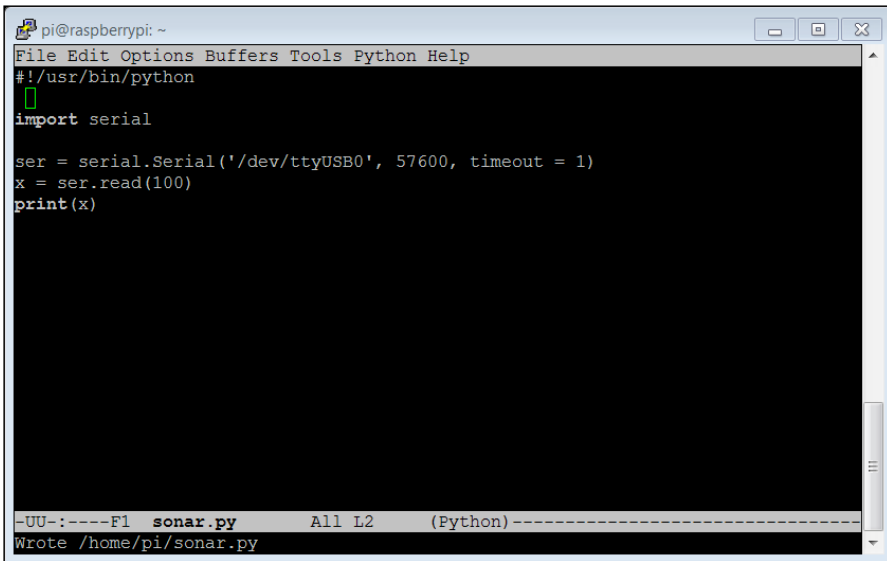


The readings have changed, specifically the value after **R** and the value **P1**, indicating an object in front of the sensor. You'll need to read these values into your program and then you can avoid the object.

Now that you know how the unit works, you'll want to mount the USB sensor on your mobile platform. In this case, I am going to mount the USB sonar sensor on my quadruped robot.

Make sure you plug one end of the USB cable into the sensor and the other end into the USB hub connected to Raspberry Pi.

With all the hardware constructed and the sensor working, you can start talking to your USB sensor using Raspberry Pi. You are going to create a simple Python program that will read the value from the sensor. To do this, using emacs as an editor, type `emacs sonar.py`. A new file called `sonar.py` will be created. Then type the code, as shown in the following screenshot:



```
pi@raspberrypi: ~
File Edit Options Buffers Tools Python Help
#!/usr/bin/python
import serial

ser = serial.Serial('/dev/ttyUSB0', 57600, timeout = 1)
x = ser.read(100)
print(x)

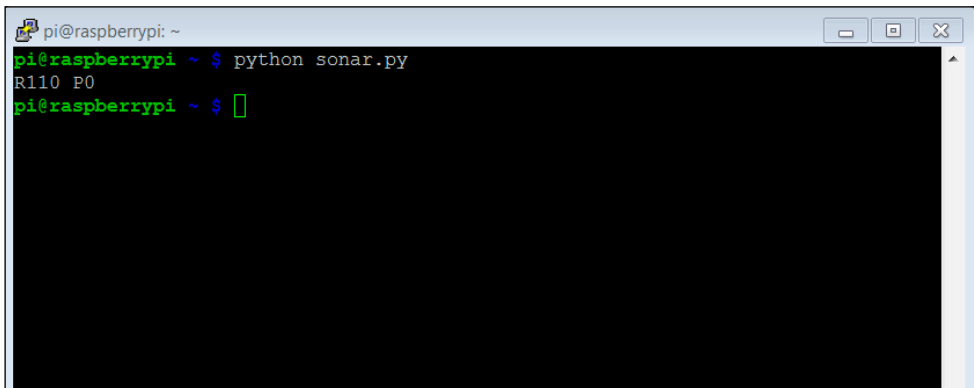
-UU-:----F1 sonar.py All L2 (Python)-----
Wrote /home/pi/sonar.py
```

Let's go through the code to see what is happening:

- `#!/usr/bin/python`: As explained earlier, the first line simply makes this file available for us to execute from the command line
- `import serial`: We again import the `serial` library. This will allow us to interface with the USB sonar sensor

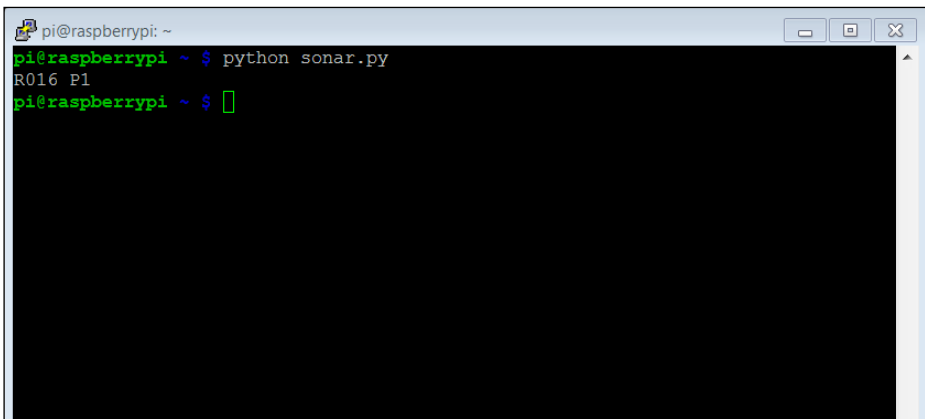
- `ser=serial.Serial('/dev/ttyUSB0', 57600, timeout = 1):` This command sets up the serial port to use the `/dev/ttyUSB0` device, which is the sonar sensor using a baud rate of 57600 and a timeout of 1
- `x = ser.read(100):` This command then reads the next 100 values from the USB port
- `print(x):` This final command then prints out the value

Once you have created this file, you can run the program and talk to the device. Do this by typing `./sonar.py`, and the program will run. I have found that sometimes the device returns no data the first time, so don't be surprised if you print out no values the first time you run your program. The second time, you should receive a valid return string. The following screenshot is my result after running the program:



```
pi@raspberrypi: ~  
pi@raspberrypi ~ $ python sonar.py  
R110 P0  
pi@raspberrypi ~ $
```

The sensor returns 110, which indicates a relative distance to a barrier in millimeters. If you place a good reflector just a few inches in front of the sensor and run the program, you will get the following result:



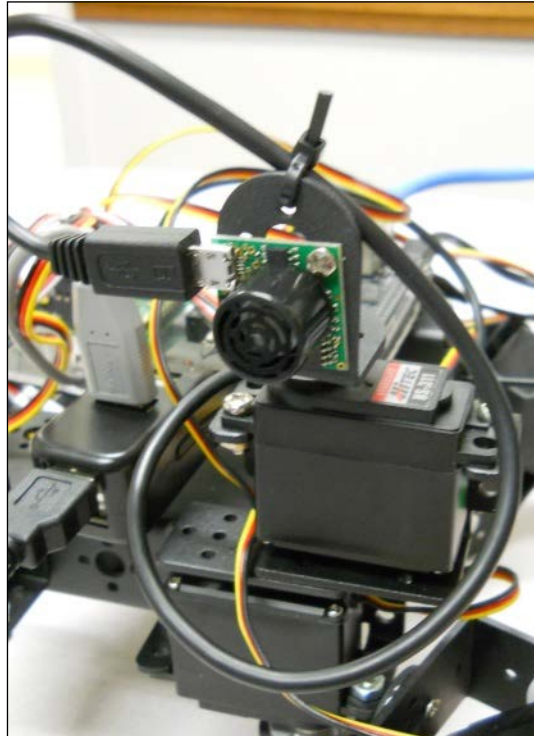
```
pi@raspberrypi: ~  
pi@raspberrypi ~ $ python sonar.py  
R016 P1  
pi@raspberrypi ~ $
```

Now the robot can sense its environment, so you can avoid bumping into walls and other barriers!

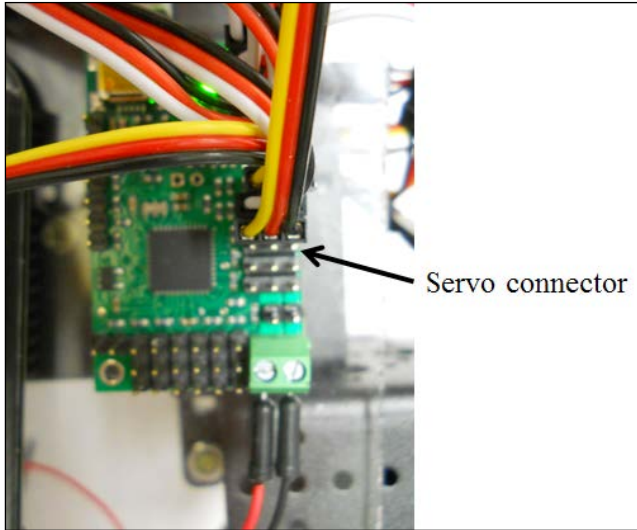
Using a servo to move a single sensor

You now have your sensors and if you want to sense more than just one direction, you can use several sensors, each mounted to a different side of the robot. However, there is a way to use servos to move your sensor, that allows you to use a single sensor to sense in several directions.

The simplest way to avoid having to purchase and configure several sensors is to mount the sensor on a single servo and then use a servo bracket to connect this assembly to the platform. Using the sonar sensor, the assembly will look, as shown in the following image:



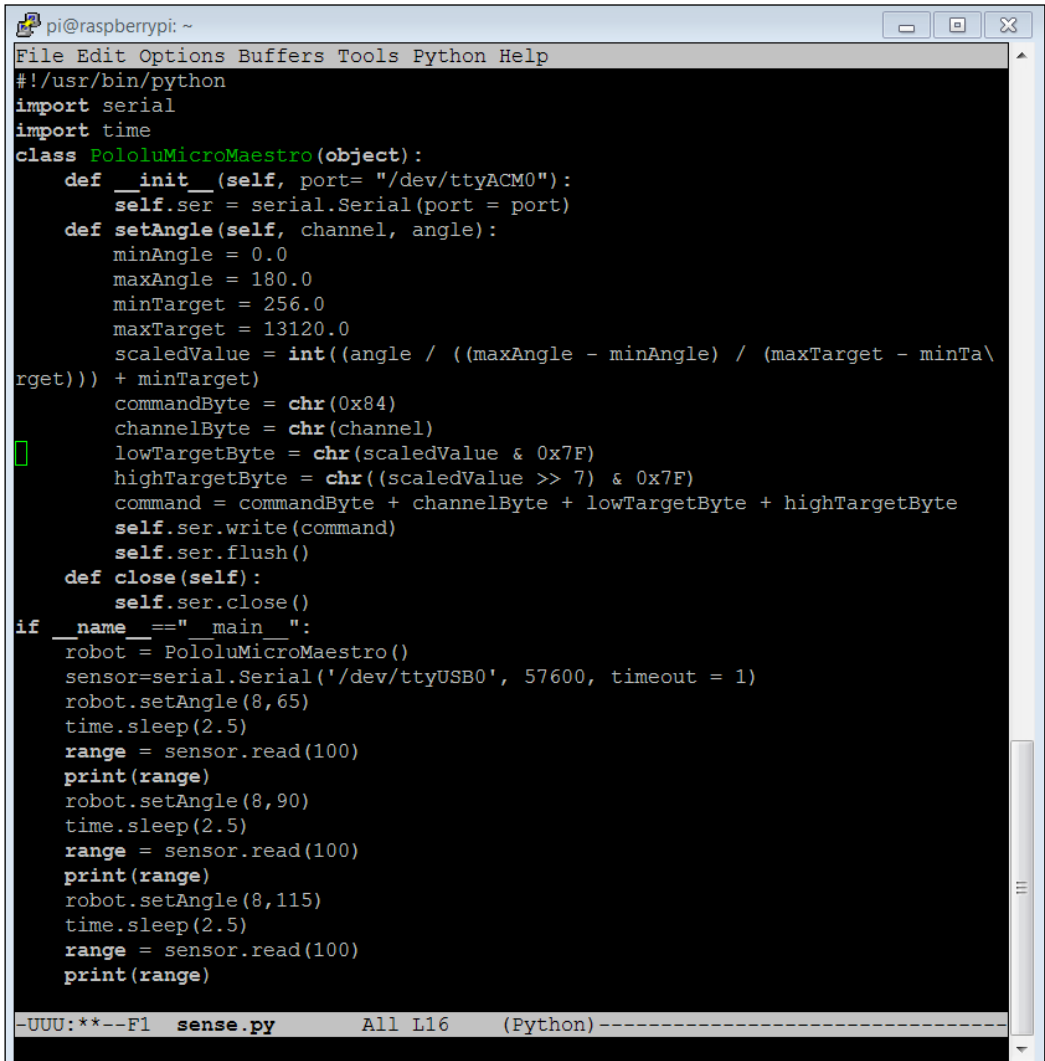
Make sure you connect your servo to the servo controller; it can fit into any open connection. I am connecting mine to my quadruped robot that has eight servos to control, so I have connected mine to the eighth connection on the servo controller board, as follows:



I'll assume you already have your sensor up and working, and know how to read the data. In this section, you will add the ability to move the sensor by communicating with the servo, through the servo controller we configured in the previous chapter.

For the program, you will begin with the `robot.py` program you created in *Chapter 6, Controlling the Movement of a Robot with Legs*, as you are going to need to access the servo controller. However, you may want to keep a copy of this program, just in case you want to use it later. First, go to the directory that contains the `robot.py` program; in my case I placed it in the `maestro_linux` directory, so I would type `cd ./maestro_linux` from my log in or home directory. Now, let's create a copy of this program by typing `cp robot.py sense.py`.

You'll want to edit this program. If you are using the emacs editor, type `emacs sense.py`. The program you want to create will look as shown in the following screenshot:



```

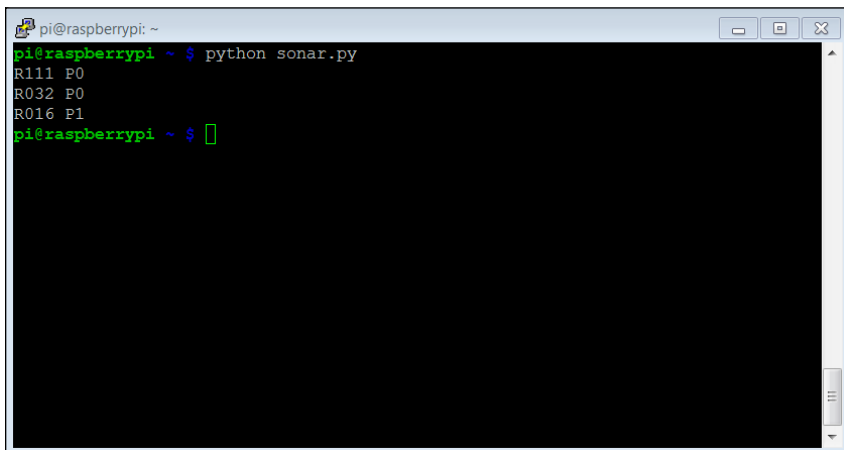
pi@raspberrypi: ~
File Edit Options Buffers Tools Python Help
#!/usr/bin/python
import serial
import time
class PololuMicroMaestro(object):
    def __init__(self, port= "/dev/ttyACM0"):
        self.ser = serial.Serial(port = port)
    def setAngle(self, channel, angle):
        minAngle = 0.0
        maxAngle = 180.0
        minTarget = 256.0
        maxTarget = 13120.0
        scaledValue = int((angle / ((maxAngle - minAngle) / (maxTarget - minTa\
rget))) + minTarget)
        commandByte = chr(0x84)
        channelByte = chr(channel)
        lowTargetByte = chr(scaledValue & 0x7F)
        highTargetByte = chr((scaledValue >> 7) & 0x7F)
        command = commandByte + channelByte + lowTargetByte + highTargetByte
        self.ser.write(command)
        self.ser.flush()
    def close(self):
        self.ser.close()
if __name__=="__main__":
    robot = PololuMicroMaestro()
    sensor=serial.Serial('/dev/ttyUSB0', 57600, timeout = 1)
    robot.setAngle(8,65)
    time.sleep(2.5)
    range = sensor.read(100)
    print(range)
    robot.setAngle(8,90)
    time.sleep(2.5)
    range = sensor.read(100)
    print(range)
    robot.setAngle(8,115)
    time.sleep(2.5)
    range = sensor.read(100)
    print(range)
-UUU:***-F1  sense.py      All L16      (Python)-----

```

Let's walk through the code to see what it does. I will begin with the section that begins with `if __name__=="__main__":`, as everything above this comes to us from the `robot.py` code and was covered in the previous chapter.

- The `robot=PololuMicroMaestro()` line initializes the servo motor controller and connects it to the proper USB port.
- The next line opens a serial port, which we will call **sensor**, that connects you to the USB Sonar sensor at the `/dev/ttyUSB0` port and sets its parameters.
- You can now ask the servo to go to a specific position and then take a reading. In this case, I am doing this for the servo positions at 65 degrees, 90 degrees, and 115 degrees. At each of these locations, you ask for a range reading. Note that, based on the specifications of the manufacturer, you need to wait 2.5 seconds for the sensor to respond, and for the device to deliver a stable reading.

That's it! Now you can sense in front of you and on either side. The following screenshot is an example of what might be displayed as a result of running the program:



```
pi@raspberrypi: ~  
pi@raspberrypi ~ $ python sonar.py  
R111 P0  
R032 P0  
R016 P1  
pi@raspberrypi ~ $
```

If you are adding the sensor/servo combination to your wheeled vehicle, you'll need to add the servo motor controller as well. The motor controllers, the servo controllers, and the USB sonar or IR sensor can all coexist on the same Raspberry Pi. You'll need to merge the `dcmotor.py` and `sense.py` programs, so that you can access each individual capability.

You can also use sensors to find an object by having two sensors determine the distance of each sensor from the object and then triangulate its position. This can help your robot actually find the position of specific obstacles. How this is accomplished is detailed on the MaxBotix website at http://www.maxbotix.com/documents/MaxBotix_Ultrasonic_Sensors_Find_Direction_and_Distance.pdf. You have all the knowledge you need to add this type of capability to your robot.

For more details on using IR and sonar sensors for obstacle avoidance, there are several good places on the web. Try <http://www.intorobotics.com/interfacing-programming-ultrasonic-sensors-tutorials-resources/>, <http://www.geology.smu.edu/~dpa-www/robo/challenge/obstacles.html>, and http://www.societyofrobots.com/member_tutorials/book/export/html/71.

Summary

Congratulations! Your robot can now detect and avoid walls and other barriers. You can also use these sensors to detect objects that you might want to find. In the next chapter, you'll learn how to disconnect your robot from all its wires and control it wirelessly.

8

Going Truly Mobile – The Remote Control of Your Robot

Based on the previous chapters, you now have mobile robots that can move around, accept commands, see, and even avoid obstacles. This chapter will teach you how to electronically communicate with your robot without using any wires.

As you send your device out into the world, you may still want to communicate with it electronically without connecting a cable. If you add this capability, you can change what your mobile robot is doing without any physical contact, but still remain in complete control of your project.

In this chapter, we will cover the following points:

- Connecting Raspberry Pi to a wireless USB input device
- Using the wireless USB input device in order to issue commands to your project
- Connecting to your robot over wireless LAN
- Connecting to your robot over ZigBee

Gathering the hardware

In this chapter, you'll learn how to connect to your device wirelessly. There are several ways to accomplish this. The first way that we'll cover in this chapter is to do this with a standard USB wireless input device. This is perhaps the easiest way to control your robot but only provides basic functionality with a limited range. The second way to connect to your robot that you will learn is via a wireless LAN device; this provides an excellent bandwidth and good range but requires bit more hardware. This will provide you with the opportunity to both control your robot and see what it is seeing.

Finally, in this chapter, we will cover communicating with your robot wirelessly via a dedicated wireless link called ZigBee. It will provide the same sort of control as with the wireless USB device but with a much greater range.

No matter what you choose, you may want to purchase a small LCD display for your Raspberry Pi. This will allow you to monitor what is going on with your project. In the previous chapters, you used a separate computer monitor for this. However, the monitor is just too big and not really designed for mobile use. For the original Raspberry Pi with S-Video output, there are several inexpensive choices for small LCDs. Here is one possible choice:

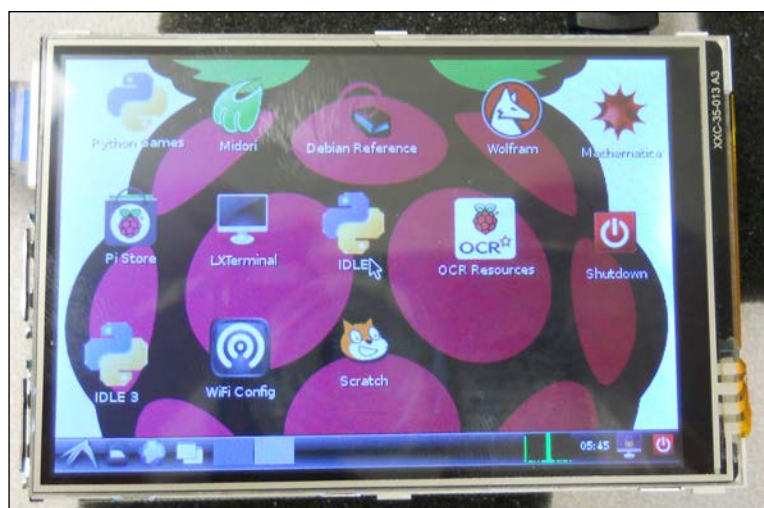


This type of LCD is available on Amazon and other online electronics stores, so you should be able to get it at almost any place. One of the challenges, however, is that most of these are designed for auto applications and may require 12 V to operate. I have been lucky; each one of the devices that I have ordered has worked just fine at 5 V. However, if you want to make sure that you get one that is compatible, you can order it from www.adafruit.com; they have a set of devices that are all compatible with Raspberry Pi.

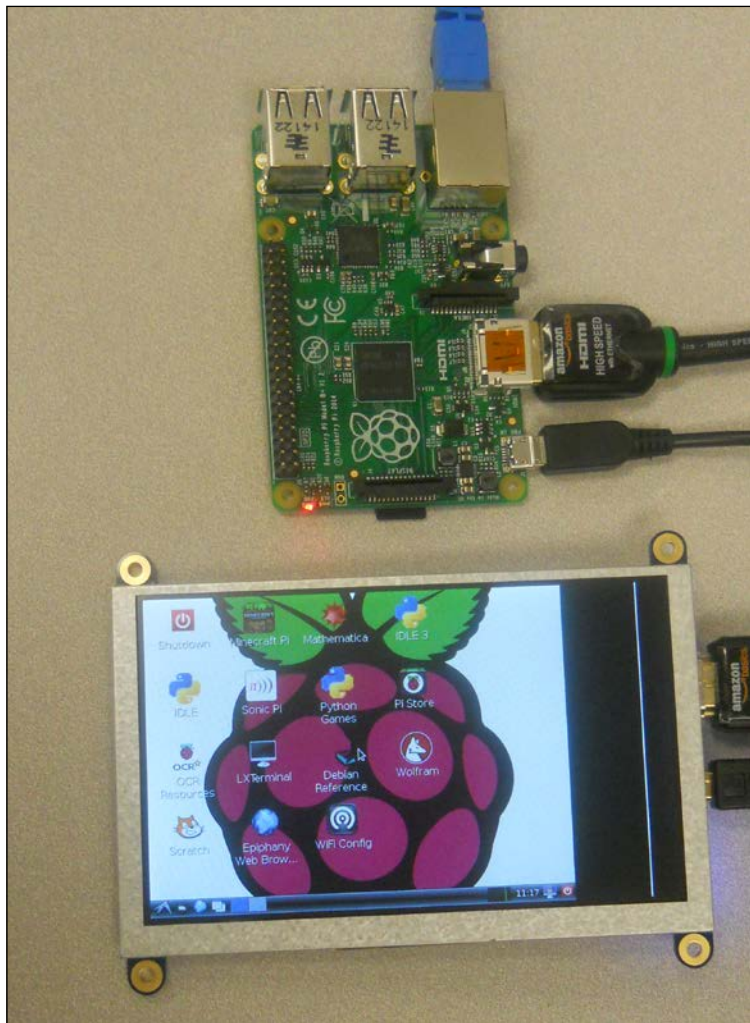
If you are working with the Raspberry Pi A or B+, these don't have an S-Video output. There are two choices here. First, there are several versions of LCDs that are made for Raspberry Pi as a plugin cape. Here is an image of just such a unit, available at www.amazon.com:



The unit simply plugs into the GPIO connector on the Raspberry Pi. Simply download the image from the page shown on Amazon, unzip the file, burn a card, and plug in the display, and you can get a simple display on your Raspberry Pi, as follows:



You can also purchase a display unit with HDMI input; here is one that is available from www.adafruit.com, connected to Raspberry Pi B+:



When you start making your projects mobile, you will also need a battery. One excellent choice is a dual-output USB battery. The following is an image of such a battery:



Just a quick note on battery selection. You'll need to keep in mind two key characteristics when you buy your battery. The first is the size of the battery, normally noted in **mAh** or **milliAmpHours**. This is a measure of the capacity of the battery, which will tell you how long a battery might last while drawing a certain current. For example, if you purchased a 2000-mAh battery and drew an average of 100 mA, this battery would theoretically last for 20 h. However, it won't last that long; as the battery discharges, the voltage will start to drop as well, and eventually you won't be able to power your system. The voltage drop depends on the quality of the battery.

The second key characteristic is the amount of current you can draw at any given time. Most batteries give a C rating and if you divide the mAh rating by this value, you will get the amount of instantaneous current that you can draw. This value is important because this will be required to estimate the amount of power you need to draw from your electronics. For example, Raspberry Pi can draw as much as 500 mA, so you'll need to make sure you get a battery that can supply this kind of current.

Now that you have a screen, you can display the results on the robotic platform itself. No extra programming is needed; the Raspbian release will automatically send signals to the LCD screen and boot with the screen acting as a display.

Now that you can display what is going on inside Raspberry Pi directly, you need to choose the wireless connection that you'd like to use. If you just need to send basic control signals to your device and you are always going to be close, a 2.4-GHz wireless keyboard is the best choice.

The following image is of a standard 2.4-GHz wireless keyboard:



This is a Logitech keyboard. Logitech generally makes very reliable keyboards and these connect well to Raspberry Pi. This keyboard is available online on Amazon and at most electronics or computer stores. You'll notice that this version has a built-in mouse pad.

Another option is a small keyboard that looks more like a game controller. It will make your projects look amazing and will make them easier to control. The following is an image of such a keyboard:



This 2.4-GHz wireless keyboard by HausBell is small, about the size of a game controller, is relatively inexpensive, and is sold online, again by Amazon.

For this application, there are several choices that you could make for the wireless technology to communicate with Raspberry Pi. Bluetooth is quite popular and works well. However, it comes with the added complexity of having to pair the device with the Bluetooth USB dongle and the system. The 2.4-GHz wireless technology comes with the keyboard and the wireless USB receiver already paired. So, the device only works with the USB dongle that is shipped with the device and the system automatically recognizes the device as long as the USB receiver is plugged into the USB port of Raspberry Pi.

The 2.4-GHz wireless devices work with the same frequency range as many 2.4-GHz wireless LAN devices, although they do not use the same modulation or protocol that is used by the standard 2.4-GHz wireless LAN. Rather, they use a proprietary modulation and protocol that is specific to the device and company that manufactures the device. There are more details on the 2.4-GHz wireless keyboards at http://www.logitech.com/images/pdf/emea_business/2.4ghz_white_paper.pdf.

While each device is different, most devices use the same overall approach where they define a number of different channels or small frequency ranges inside the overall range of 2.4 GHz. The keyboard communicates with the USB receiver on one of these frequencies. However, if either the keyboard or the USB receiver senses that some other device is transmitting on that frequency, the device will move to a different channel to try and avoid the interference.

The transmissions between the wireless keyboard and the USB receiver are encrypted, so no device except the paired keyboard and USB receiver can understand the messages that are being sent between the two devices. The range of the keyboard and receiver pair is dependent upon the amount of power both use for transmission; the higher the power, the longer is the range. Unfortunately, the higher the power, the less time the batteries in the wireless device last for. Most wireless keyboards are designed to work for up to 10 m, or around 30 ft.

Sometimes, you may want to have a physical connection with your Raspberry Pi and want to not only control your robot but also connect via VNC Viewer, so you can access the display of Raspberry Pi; in such a case, wireless LAN is the best choice. For this, you'll need to purchase a supported wireless LAN device. The following is an image of such a device:

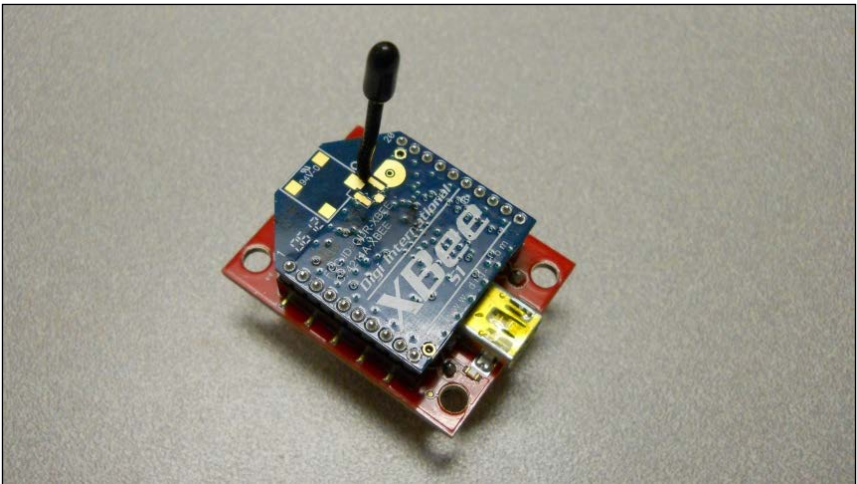


It is best to choose a device that is known to be supported by your Raspbian release. Check the http://elinux.org/RPi_USB_Wi-Fi_Adapters link for a list of these devices. Note that you'll need access to a wireless LAN signal, so you'll need to supply this by connecting to your own wireless LAN. This can come from a wireless LAN router that you have already set up, a spare wireless LAN router that can set up an ad hoc network, or many of today's smart cellphones, which also allow you to turn your device into a hotspot that can provide this signal.

If you'd like to have basic communication with your device, but want to do it at a significant distance, ZigBee provides a possible solution. There are a number of different types of devices; one is a USB stick-type ZigBee device at www.zigbee.org. The following is an image of such a device:



Also, there are devices made to connect with Linux systems such as Raspberry Pi. The following is an image of one of these devices:



This is the device we will use in this chapter. Make sure you purchase an XBee Series 1 device as it is the easiest device to configure and use, and there is a great open source community support for the device too. If you choose a different device, you'll need to follow the directions for this device from the manufacturer. Also, if you want to use this type of point-to-point communication, you'll need two units, one for Raspberry Pi and the other for the host computer.

Connecting Raspberry Pi to a wireless USB keyboard

You've been able to control your projects by using a LAN connection, but you don't always want to have your projects tethered in this manner. In this section, I'll show you how to connect via a wireless keyboard.

Find your USB keyboard. It should come with a USB dongle. Plug the USB dongle into the Raspberry Pi USB port. After some time, the unit should power on to the windowing system. Now, if you move your finger around on the mouse pad, you should see the mouse moving on the screen. You can also select a terminal and type some text into the terminal. You now have keyboard and mouse inputs. Next, you will learn how to accept the key strokes into a program in order to control the robot.

Using the keyboard to control your project

Now that the keyboard is connected, let's figure out how to accept commands on Raspberry Pi. Now that you can enter commands wirelessly, the next step is to create a program that can take these commands and then have your project execute them. There are a couple of options here; you'll see examples of both. The first is to simply include the command interface in your program. Let's take an example of the `dcmotor.py` program you wrote to move your wheeled robot. If you want, you can copy that program by using the `cp dcmotor.py remote.py` command.

In order to add user control, you need two new programming constructs, the while loop and the if statement. Let's add them to the program, and then, we will learn what they do. The following is a listing of the area of code you are going to change:

```
#!/usr/bin/python
import serial
import time
def setSpeed(ser, motor, direction, speed):
    if motor == 0 and direction == 0:
        sendByte = chr(0xC2)
    if motor == 1 and direction == 0:
        sendByte = chr(0xCA)
    if motor == 0 and direction == 1:
        sendByte = chr(0xC1)
    if motor == 1 and direction == 1:
        sendByte = chr(0xC9)
    ser.write(sendByte)
    ser.write(chr(speed))
ser = serial.Serial('/dev/ttyUSB0', 19200, timeout = 1)
var = 'n'
while var != 'q':
    var = raw_input(">")
    if var == '<':
        setSpeed(ser, 0, 0, 100)
        setSpeed(ser, 1, 0, 100)
        time.sleep(.5)
        setSpeed(ser, 0, 0, 0)
        setSpeed(ser, 1, 0, 0)
    if var == '>':
        setSpeed(ser, 0, 1, 100)
        setSpeed(ser, 1, 1, 100)
        time.sleep(.5)
        setSpeed(ser, 0, 0, 0)
        setSpeed(ser, 1, 0, 0)
    if var == 'f':
        setSpeed(ser, 0, 0, 100)
        setSpeed(ser, 1, 1, 100)
        time.sleep(.5)
        setSpeed(ser, 0, 0, 0)
        setSpeed(ser, 1, 0, 0)
    if var == 'r':
        setSpeed(ser, 0, 1, 100)
        setSpeed(ser, 1, 0, 100)
        time.sleep(.5)
        setSpeed(ser, 0, 0, 0)
        setSpeed(ser, 1, 0, 0)
ser.close()
```

--UU-:***--F1 remote.py All L43 (Python)-----

You will edit your program by making some changes. Add the code in the preceding screenshot just below the `ser = serial.Serial('/dev/ttyUBS0', 19200, timeout = 1)` statement. The code can be explained as follows:

- The `var = 'n'` statement will define a variable named `var` and it will be of the type `character`, which you will use in your program to get the input from the user.
- The `while var != 'q':` statement will place your program in a loop. This loop will keep repeating until you or the user enters the letter `q`.
- The `var = raw_input(">")` statement will get the character value from the user. The `>` text is simply the character that will be displayed for the user to enter something.
- The `if var == '<':` statement checks the value that you get from the user. If it is a `<` character, the robot will turn left by running the right DC motor for half a second. You will need to determine how much time is required to run the right DC motor for a left turn. The actual time value, `0.5` in this case, may need to be higher or lower.
- The next few lines send a `Speed` command to the motor, wait for `0.5 s`, and then send a command for the motor to stop.
- The `if var == '>':` statement checks the value that you get from the user. If it is a `>` character, the robot will turn left by running the left DC motor for half a second. You will need to determine how much time is required to run the left DC motor for a right turn. The actual time value, `0.5` in this case, may need to be higher or lower.
- The next few lines send a `Speed` command to the motor, wait for `0.5 s`, and then send a command for the motor to stop.
- The `if var == 'f':` statement checks the value that you get from the user. If it is an `f` character, the robot will run forward by running the right and left DC motors for half a second. You will need to determine the speed to set each motor to follow a forward path.
- The next few lines send a `Speed` command to both motors, wait for `0.5 s`, and then send a command for both motors to stop.
- The `if var == 'r':` statement checks the value that you get from the user. If it is an `r` character, the robot will run backward by running the right and left DC motors for half a second. You will need to determine the speed to set each motor to follow a backward path.
- The next few lines send a `Speed` command to both motors, wait for `0.5 s`, and then send a command for both motors to stop.

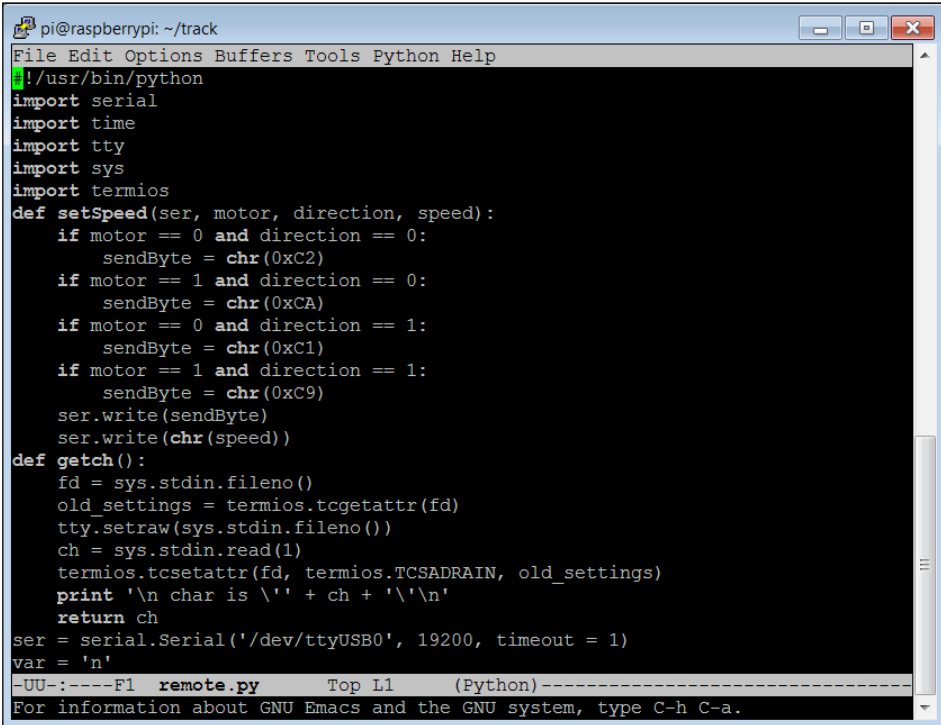
Once you have edited the program, save it and make it executable by typing `chmod +x remote.py`. Now, you can run the program, but you must run it by typing the command using the wireless keyboard. If you are not yet directly logged into Raspberry Pi, make sure you can see the LCD screen and access it via the wireless keyboard. You can now disconnect the LAN cable; you will be able to communicate with Raspberry Pi through the wireless keyboard. The system should look like the following image:



To use this system, type `cd` to go to the folder that holds the `remote.py` program. In my case, this file was in the `/home/pi/track` folder, so I did a `cd track` from my home folder. Now, you can run the program by typing `./remote.py`. The screen will display a prompt and each time you type the appropriate command (`<`, `>`, `f`, and `r`) and press *Enter*, your robot will move. You need to be aware that the range of this technology is at best around 30 ft, so don't let your robot get too far away.

Now, you can move your robot around by using the wireless keyboard! You can even take your robot outside. You don't need the LAN cable to run your programs because you can run them by using the LCD display and keyboard.

There is one more change you can make so that you don't have to hit the *Enter* key after each input character is typed. In order to make this work, you'll need to add some inclusions to your program and then add one function that can get a single character without the *Enter* key. The following screenshot shows the first set of changes you'll need to make:



```

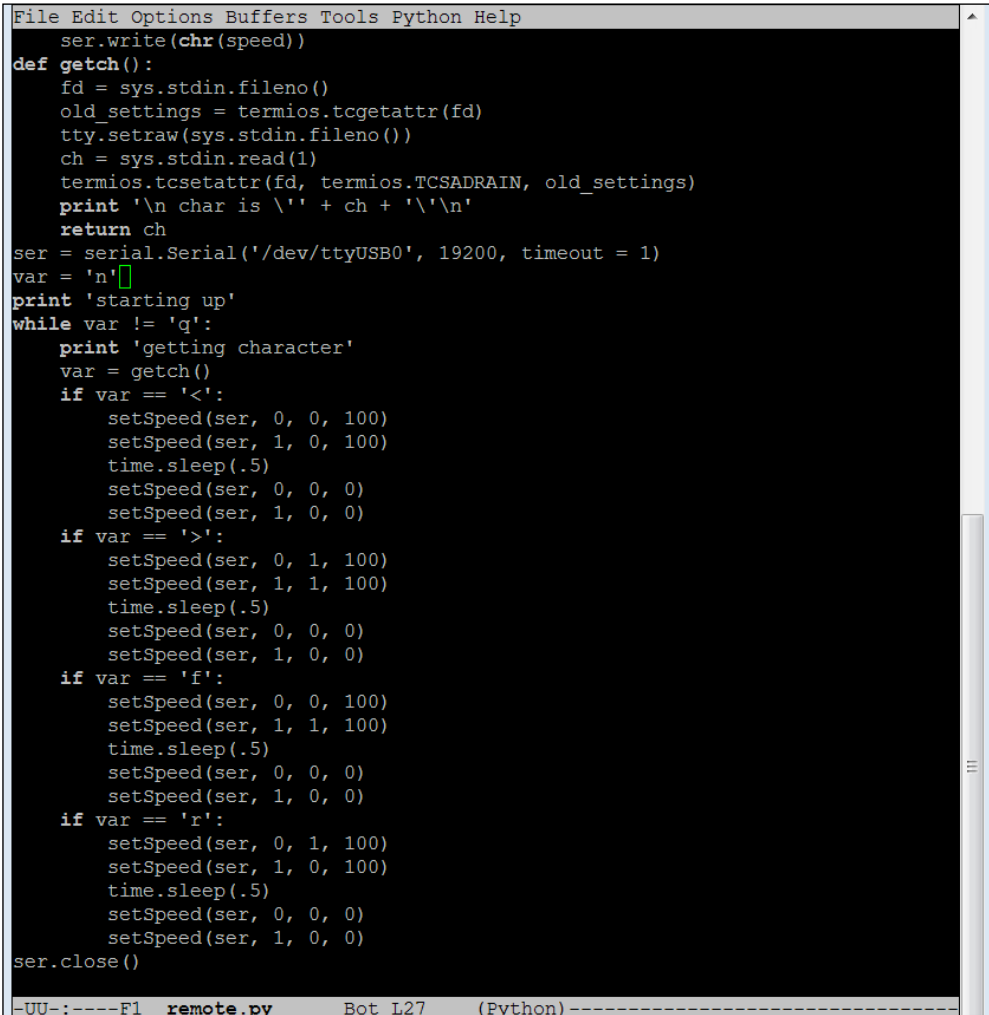
pi@raspberrypi: ~/track
File Edit Options Buffers Tools Python Help
#!/usr/bin/python
import serial
import time
import tty
import sys
import termios
def setSpeed(ser, motor, direction, speed):
    if motor == 0 and direction == 0:
        sendByte = chr(0xC2)
    if motor == 1 and direction == 0:
        sendByte = chr(0xCA)
    if motor == 0 and direction == 1:
        sendByte = chr(0xC1)
    if motor == 1 and direction == 1:
        sendByte = chr(0xC9)
    ser.write(sendByte)
    ser.write(chr(speed))
def getch():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    tty.setraw(sys.stdin.fileno())
    ch = sys.stdin.read(1)
    termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    print '\n char is \'' + ch + '\''
    return ch
ser = serial.Serial('/dev/ttyUSB0', 19200, timeout = 1)
var = 'n'
-UU--:----F1 remote.py Top L1 (Python)-----
For information about GNU Emacs and the GNU system, type C-h C-a.

```

The specifics of the preceding part of the file are listed as follows:

- You'll need to add `import tty`, `import sys`, and `import termios`. All these are the libraries you'll need for your function to work. The `termios` library is the general I/O library used in order to get characters from the keyboard.
- The `setSpeed(ser, motor, direction, speed):` function is unchanged from the `dcmotor.py` code.

- The `def getch() :` function gets a single character without requiring the *Enter* key after each key stroke. The `print` statement in the function is optional; you can use it to map the different keys of the keyboard.
- The next set of changes are shown in the following screenshot:

A screenshot of a text editor window with a dark background. The editor has a menu bar at the top with 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Python', and 'Help'. The code is written in Python and includes a `getch()` function that reads a single character from stdin using `termios`. Below the function, a `serial` object is initialized for a USB device, and a `while` loop runs until the user presses 'q'. Inside the loop, different key presses ('<', '>', 'f', 'r') are mapped to specific `setSpeed` calls for two motors, with a 0.5-second delay between calls. The status bar at the bottom shows the file path `-UU-:----F1 remote.py`, the current line `Bot L27`, and the file type `(Python)`.

```
File Edit Options Buffers Tools Python Help
ser.write(chr(speed))
def getch():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    tty.setraw(sys.stdin.fileno())
    ch = sys.stdin.read(1)
    termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    print '\n char is \'' + ch + '\'\n'
    return ch
ser = serial.Serial('/dev/ttyUSB0', 19200, timeout = 1)
var = 'n'
print 'starting up'
while var != 'q':
    print 'getting character'
    var = getch()
    if var == '<':
        setSpeed(ser, 0, 0, 100)
        setSpeed(ser, 1, 0, 100)
        time.sleep(.5)
        setSpeed(ser, 0, 0, 0)
        setSpeed(ser, 1, 0, 0)
    if var == '>':
        setSpeed(ser, 0, 1, 100)
        setSpeed(ser, 1, 1, 100)
        time.sleep(.5)
        setSpeed(ser, 0, 0, 0)
        setSpeed(ser, 1, 0, 0)
    if var == 'f':
        setSpeed(ser, 0, 0, 100)
        setSpeed(ser, 1, 1, 100)
        time.sleep(.5)
        setSpeed(ser, 0, 0, 0)
        setSpeed(ser, 1, 0, 0)
    if var == 'r':
        setSpeed(ser, 0, 1, 100)
        setSpeed(ser, 1, 0, 100)
        time.sleep(.5)
        setSpeed(ser, 0, 0, 0)
        setSpeed(ser, 1, 0, 0)
ser.close()
-UU-:----F1 remote.py      Bot L27      (Python)-----
```

- The `var = getch()` statement calls the function that returns the character, without having to type the *Enter* key. Now, it is important to note that the program changes the terminal settings. So, when you run your program, you can no longer stop the program by typing *Ctrl + C*. You'll have to type *q* to restore your terminal settings.

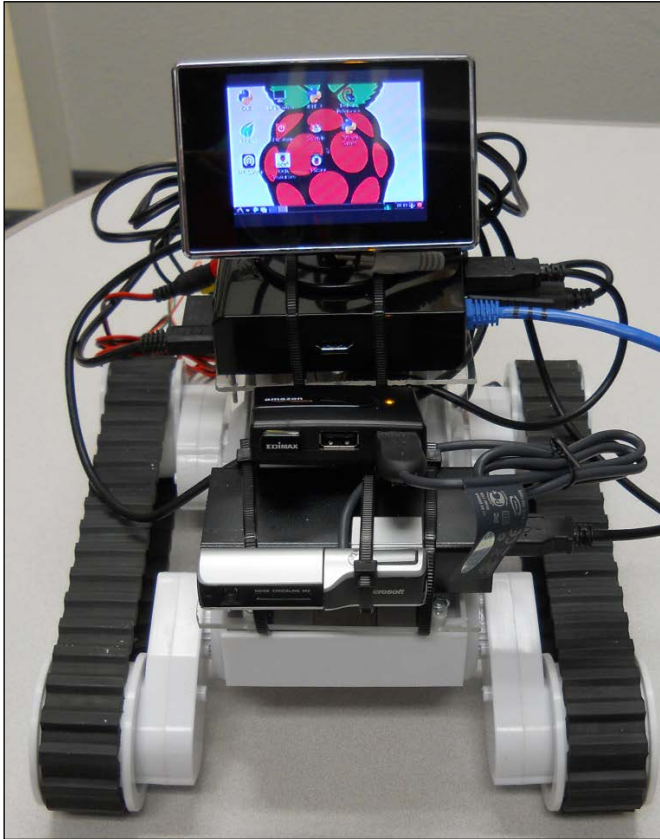
Many users are comfortable using gaming keypads. There are several that come with a wireless connection. You could try connecting one of these to your robot if you want it to seem more like a real video game. In my system, I used the wireless keypad by HausBell and mapped the arrow keys at the top of the keyboard to tell my robot to go forward, backward, left, and right. I figured out which key strokes these translated to by simply running my program and looking at the `print` statement in the program. You can also add additional functionality to the program to stop the robot if the program senses if it has been a long time since a key stroke has come in, in case you've lost connection to the keyboard.

Working remotely with your Raspberry Pi through a wireless LAN

The last section showed you how to control your robot projects through a wireless USB keyboard. But, what if you want more flexibility? You may not only want to control your robot but also monitor it, see what it is picturing, and so on. The easiest way to do this is using a wireless LAN connection.

To configure a wireless LAN connection, start by inserting the wireless LAN card into Raspberry Pi. Here is where it gets a bit tricky as you'll need a wireless LAN USB connection, a USB connection to your motor controller, and you'll also want to connect your USB webcam or perhaps, a distance sensor. You'll need to use a USB hub in order to connect all of these. So, connect the hub to Raspberry Pi and then connect the devices to the hub. You may want to choose a powered USB hub as it will give better power supply to the USB wireless LAN device.

The following is an image of the platform with the hub and the devices connected:

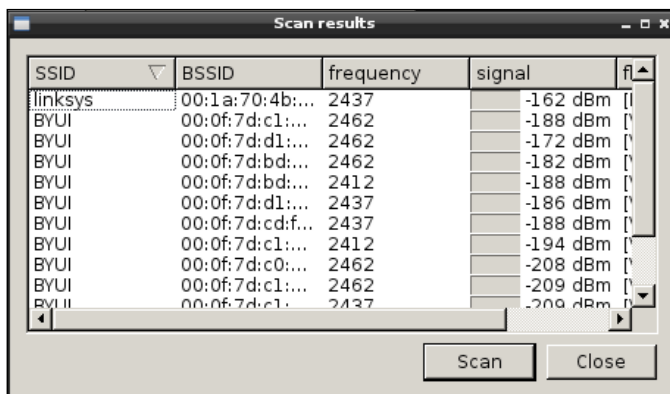


Make sure you use a powered USB hub as you'll be talking to a USB camera and a wireless LAN device that can take a bit of power. Additionally, you'll need to provide a network so that your robot can connect. You can use an available network if there is one in the area you are working in and you have access to any passwords. I like to create a dedicated wireless LAN by connecting a wireless router to my laptop, as I can take this configuration anywhere. The following is an image of this configuration:

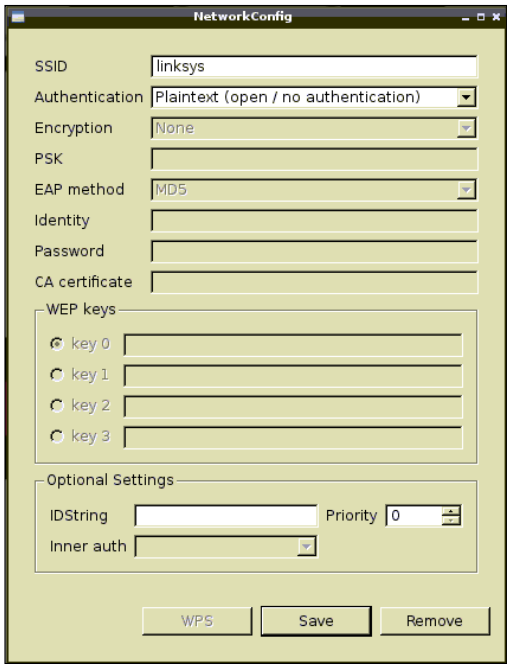


We are not going to walk through the configuration. It will be heavily dependent on your router, but we can just set up a simple, unsecured network since we don't use it all the time. Once you have connected all the devices, turn on the power to the robot. You'll need your screen and wireless keyboard to set up the wireless LAN network. Once the system has been booted, you can select the **WiFi Configuration** icon of Raspberry Pi. Now, you should see the **WiFi Config** tool.

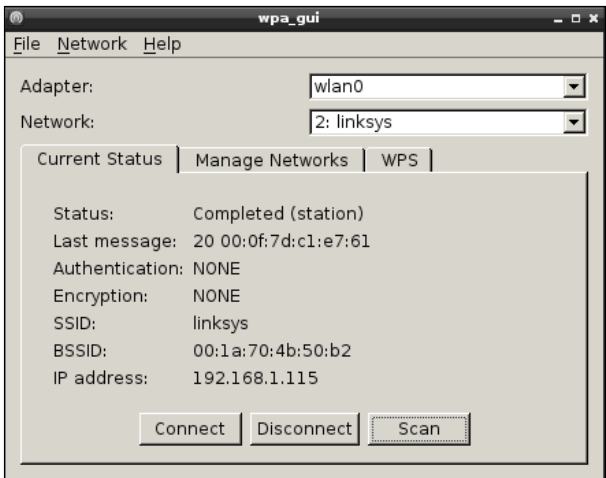
At this moment you are close to establishing a connection to the network. Your **wlan0** adapter should appear in the **Adapter:** option. Click on the **Scan** button as shown in the following screenshot of the pane:



As you can see in the preceding screenshot, the Linksys network is available, as are a number of secure networks. Now, if you select **linksys**, it will ask you to enter the specific network configuration parameters. I use the defaults for my open network, as follows:

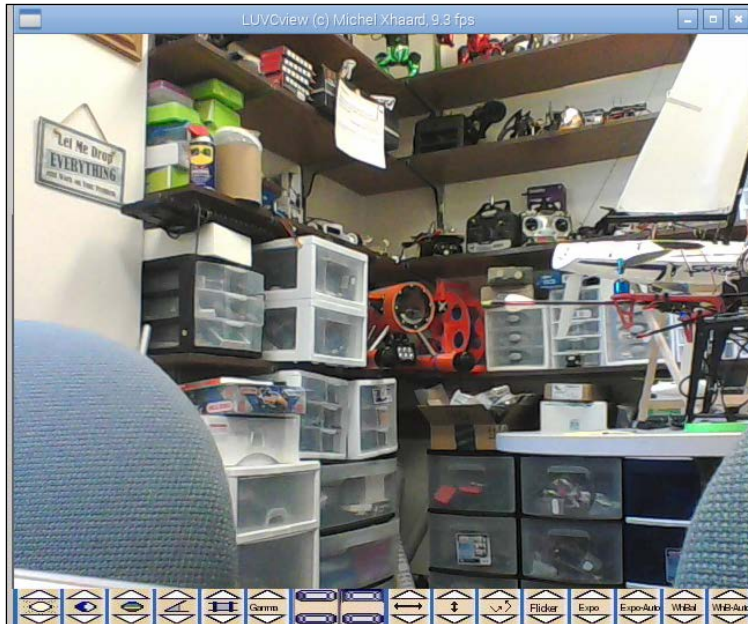


If you go back and click on **Connect**, it will connect to the network and give you an IP address as follows:



You can also configure the system using terminal commands. There are several tutorials that show you how to configure the system; try pingbin.com/2012/12/setup-wifi-raspberry-pi/ or <http://learn.adafruit.com/adafruits-raspberry-pi-lesson-3-network-setup/setting-up-wifi-with-occidentalis>.

Now, you can use this tool from your laptop by using PuTTY or the VNC server. The following is an image showing a **VNC Viewer** image of the webcam on Raspberry Pi communicating through the wireless LAN connection:



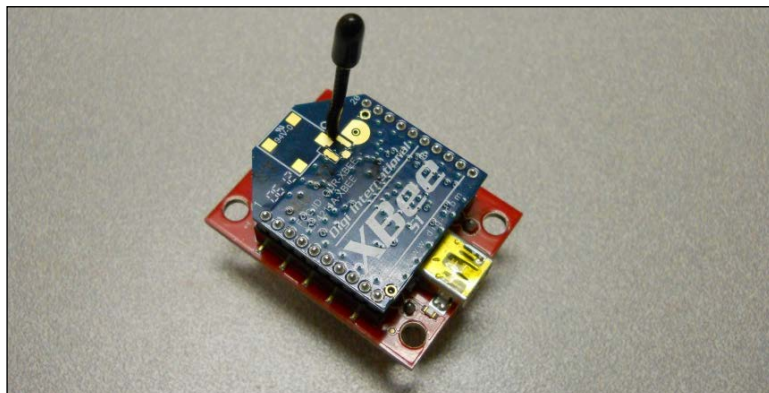
You can also do a similar sort of configuration by using your smartphone in hotspot mode. In this case, the cell phone provides the wireless LAN connection. Connect both the laptop and Raspberry Pi to the hotspot by using the information given to you on your cell phone. Now, you can communicate through the SSH and VNC servers wirelessly.

Working remotely with your Raspberry Pi through ZigBee

Now, you can remote to your device and control it through a wireless USB device as well as a wireless LAN connection. Now, let's look at a technology that can extend the wireless connection much further. The technology is ZigBee and it is made for longer-range wireless communications.

The ZigBee standard is built upon the IEEE 802.15.4 standard, a standard that was created to allow a set of devices to communicate with each other in order to enable a low data rate coordination of multiple devices. The ZigBee part of the standard ensures interoperability between the vendors of these low-rate devices. The IEEE 802.15.4 part of the standard specifies the physical interface and the ZigBee part of the standard defines the network and applications interface. To find out more about ZigBee, visit www.zigbee.org. Since we are only interested in the physical interface working together, you can buy IEEE 802.15.4 devices. However, ZigBee devices are a bit more prevalent, are supersets of IEEE 802.15.4, and are also quite inexpensive.

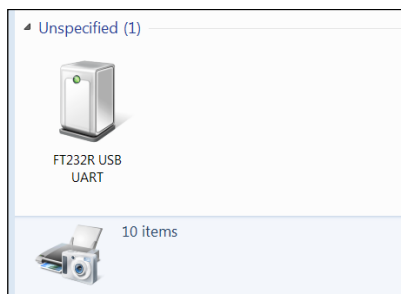
The other standard that you might hear as you try to purchase or use devices like these is XBee. This is a specific company's implementation (Digi) of several different wireless standards with standard hardware modules that can connect in many different ways to different embedded systems. They make some devices that support ZigBee; the following is an image of this type of device, which supports ZigBee, attached to a shield that provides a USB port:



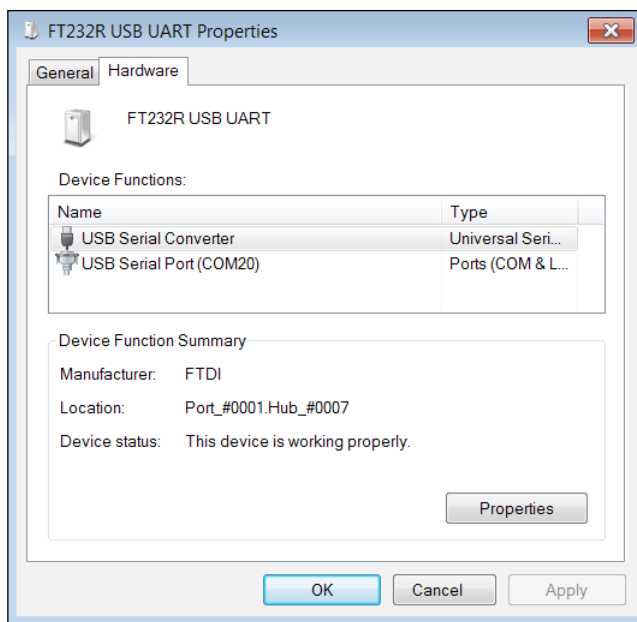
As noted at the beginning of this chapter, you will be learning how to use this specific device. The advantage of using this device is that it is configured to make it very easy to create and manage a simple link between two XBee Series 1 devices. Make sure you have an XBee device that supports ZigBee Series 1. You'll also need to purchase a shield that provides a USB port connection for the device.

Now, let's get started with configuring your two devices to make them talk. I'll give an example here by using Windows and a PC. A Linux user can do something similar by using a Linux terminal program. An excellent tutorial is available at <http://web.univ-pau.fr/~cpham/WSN/XBee.html>.

If you are using Windows, plug one of the devices into your personal computer. Your computer should find the latest drivers for the device. You should see your device when you click on the **Devices and Printers** option from the **Start** menu as follows:



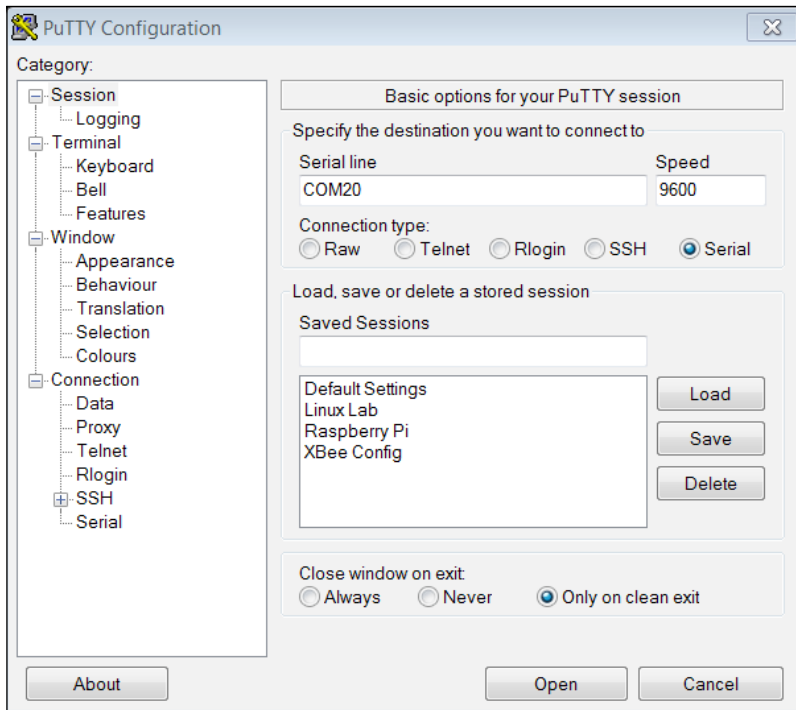
The device is now available to communicate with through the IEEE 802.15.4 wireless interface. We could set up a full ZigBee compliant network, but we're just going to communicate from one device to another directly, so we'll just use the device as a serial port connection. Double-click on the device icon, and then select the **Hardware** tab; you should see the following screenshot:



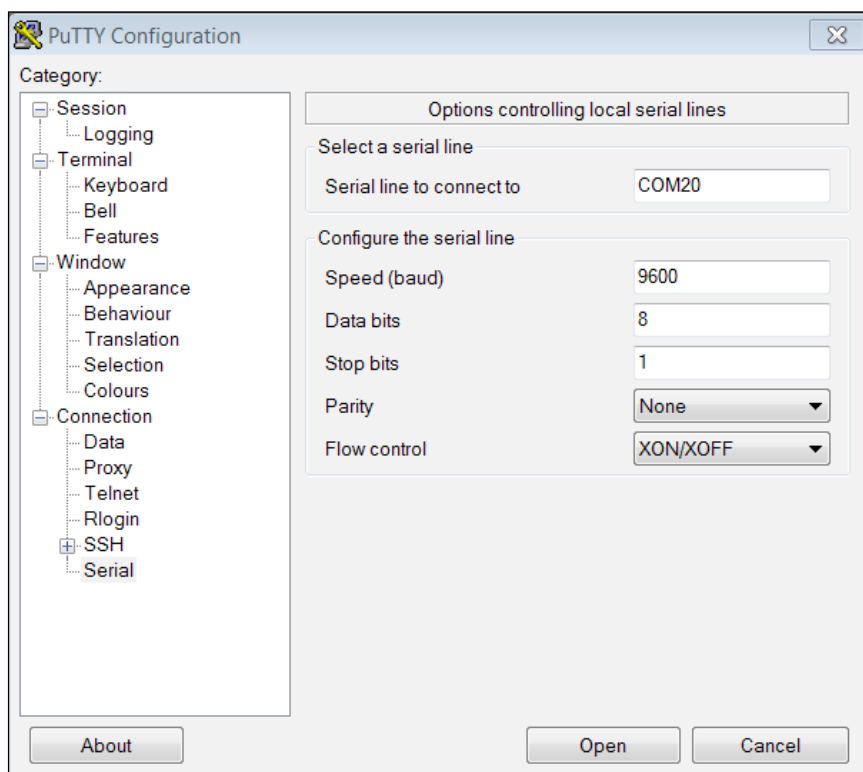
Note that the device is connected to the COM20 serial port. We'll use this to communicate with the device and configure it. You can use any terminal emulator program; I like to use PuTTY, which is already on my computer.

Perform the following steps to configure the device:

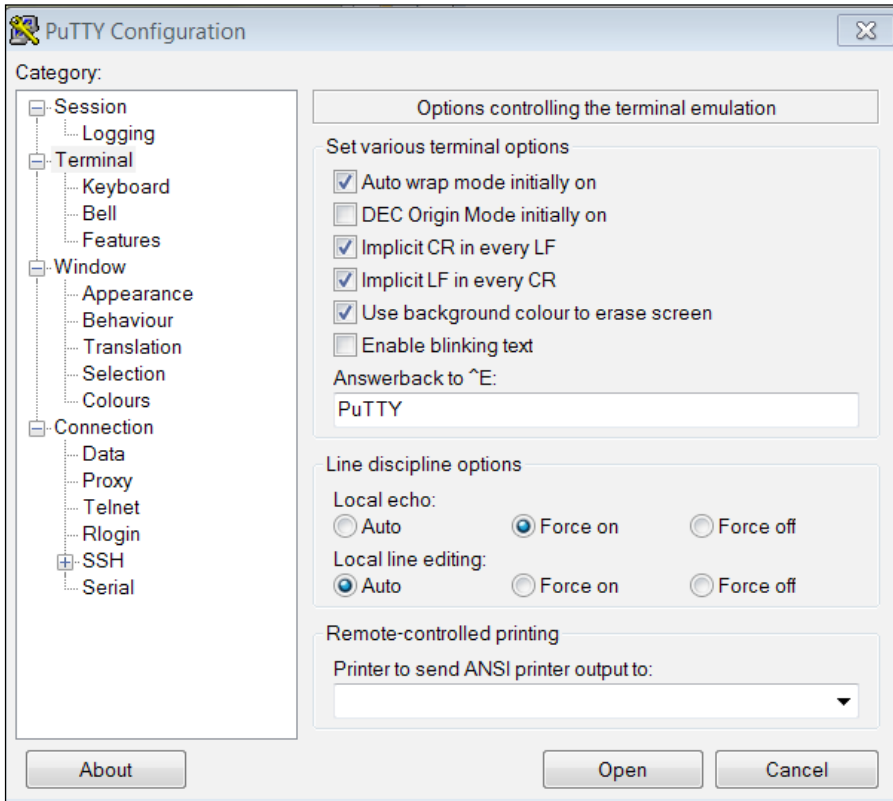
1. Open PuTTY and select the **Serial** option and (in this case) the **COM20** port. The following screenshot shows how to fill in the PuTTY window to configure the device:



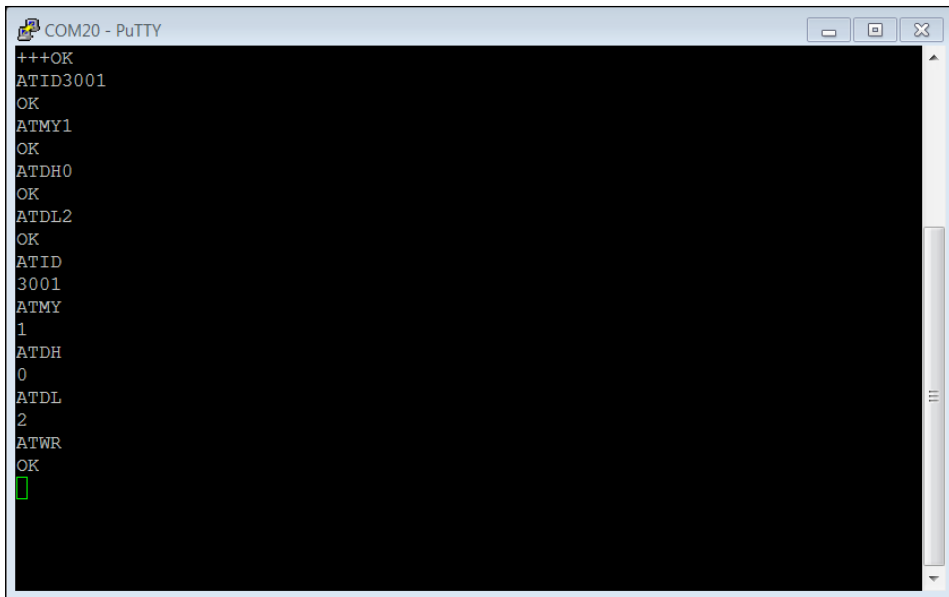
2. Configure the following parameters in the terminal window (the **Serial** option in the **Category:** selection set): Baudrate, 9600; the **Data bits** option, 8; **Parity**, **None**, and the **Stop bits** option, 1 as follows:



3. Make sure you also select **Force on** for the **Local echo** option and check the **Implicit CR in every LF** and **Implicit LF in every CR** options available under the **Terminal** tab of the **Category:** selection set as follows:

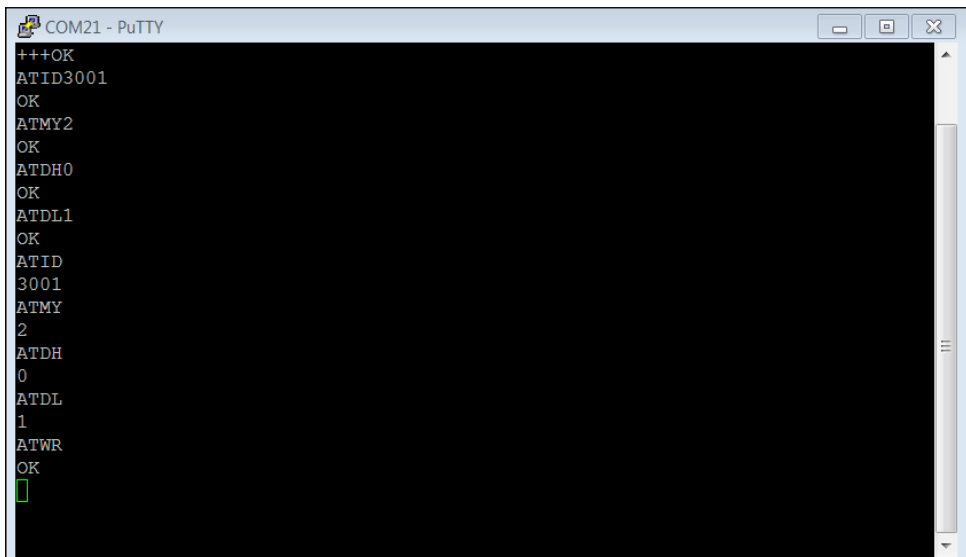


4. Connect to the device by clicking on **Open**.
5. Enter the commands to the device through the terminal window, as shown in the following screenshot:



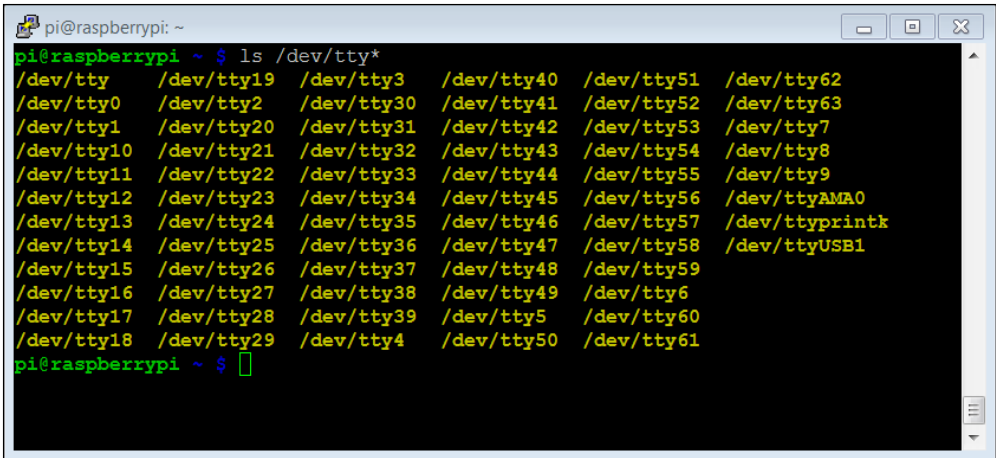
```
COM20 - PuTTY
+++OK
ATID3001
OK
ATMY1
OK
ATDH0
OK
ATDL2
OK
ATID
3001
ATMY
1
ATDH
0
ATDL
2
ATWR
OK
█
```

6. The OK response comes from the device as you enter each command. Now, plug the other device into the PC. Note that it might choose a different COM port; click on the **Devices and Printers** option, double-click on the device's icon, and select the **Hardware** tab to find the COM port. Follow the same steps to configure the second device, except there are just a few changes. The following is a screenshot of the terminal window for these commands:



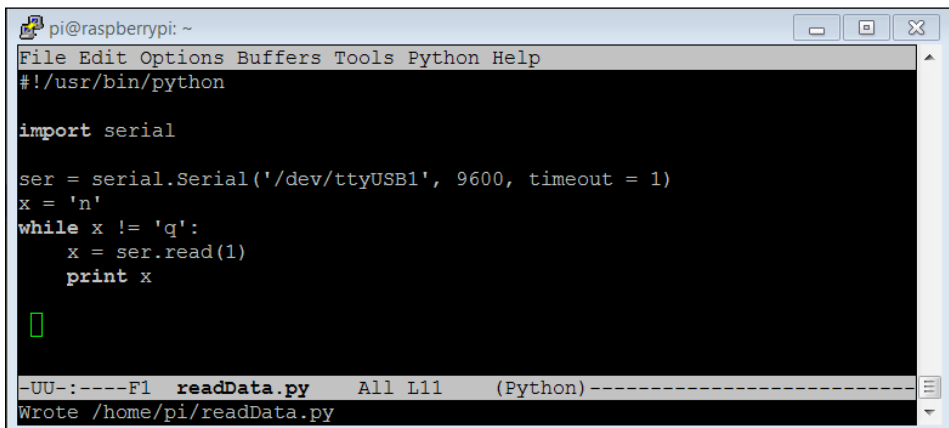
```
COM21 - PuTTY
+++OK
ATID3001
OK
ATMY2
OK
ATDH0
OK
ATDL1
OK
ATID
3001
ATMY
2
ATDH
0
ATDL
1
ATWR
OK
█
```


The devices are now ready to talk to one another. Plug one of the devices into the Raspberry Pi USB port. Using a terminal window, show the devices that are connected by typing `ls /dev/tty*`. It will look something like what is shown in the following screenshot:



```
pi@raspberrypi: ~  
pi@raspberrypi ~ $ ls /dev/tty*  
/dev/tty      /dev/tty19  /dev/tty3   /dev/tty40  /dev/tty51  /dev/tty62  
/dev/tty0     /dev/tty2   /dev/tty30  /dev/tty41  /dev/tty52  /dev/tty63  
/dev/tty1     /dev/tty20  /dev/tty31  /dev/tty42  /dev/tty53  /dev/tty7  
/dev/tty10    /dev/tty21  /dev/tty32  /dev/tty43  /dev/tty54  /dev/tty8  
/dev/tty11    /dev/tty22  /dev/tty33  /dev/tty44  /dev/tty55  /dev/tty9  
/dev/tty12    /dev/tty23  /dev/tty34  /dev/tty45  /dev/tty56  /dev/ttyAMA0  
/dev/tty13    /dev/tty24  /dev/tty35  /dev/tty46  /dev/tty57  /dev/ttyprintk  
/dev/tty14    /dev/tty25  /dev/tty36  /dev/tty47  /dev/tty58  /dev/ttyUSB1  
/dev/tty15    /dev/tty26  /dev/tty37  /dev/tty48  /dev/tty59  
/dev/tty16    /dev/tty27  /dev/tty38  /dev/tty49  /dev/tty6  
/dev/tty17    /dev/tty28  /dev/tty39  /dev/tty5   /dev/tty60  
/dev/tty18    /dev/tty29  /dev/tty4   /dev/tty50  /dev/tty61  
pi@raspberrypi ~ $
```

Note that the device appears at `/dev/ttyUSB1`. Now, you'll need to create a Python program that will read the preceding input. The following screenshot depicts the listing for such a program:

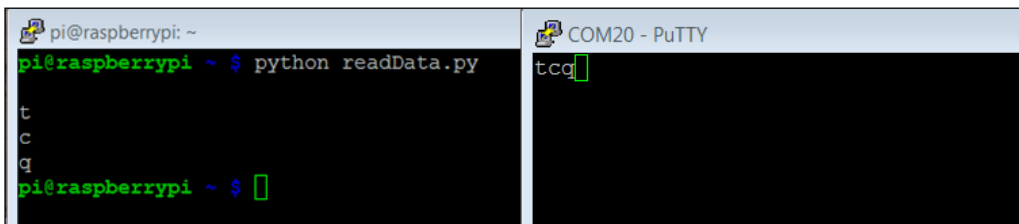


```
pi@raspberrypi: ~  
File Edit Options Buffers Tools Python Help  
#!/usr/bin/python  
  
import serial  
  
ser = serial.Serial('/dev/ttyUSB1', 9600, timeout = 1)  
x = 'n'  
while x != 'q':  
    x = ser.read(1)  
    print x  
  
[ ]  
--UU-:----F1  readData.py  All L11  (Python)-----  
Wrote /home/pi/readData.py
```

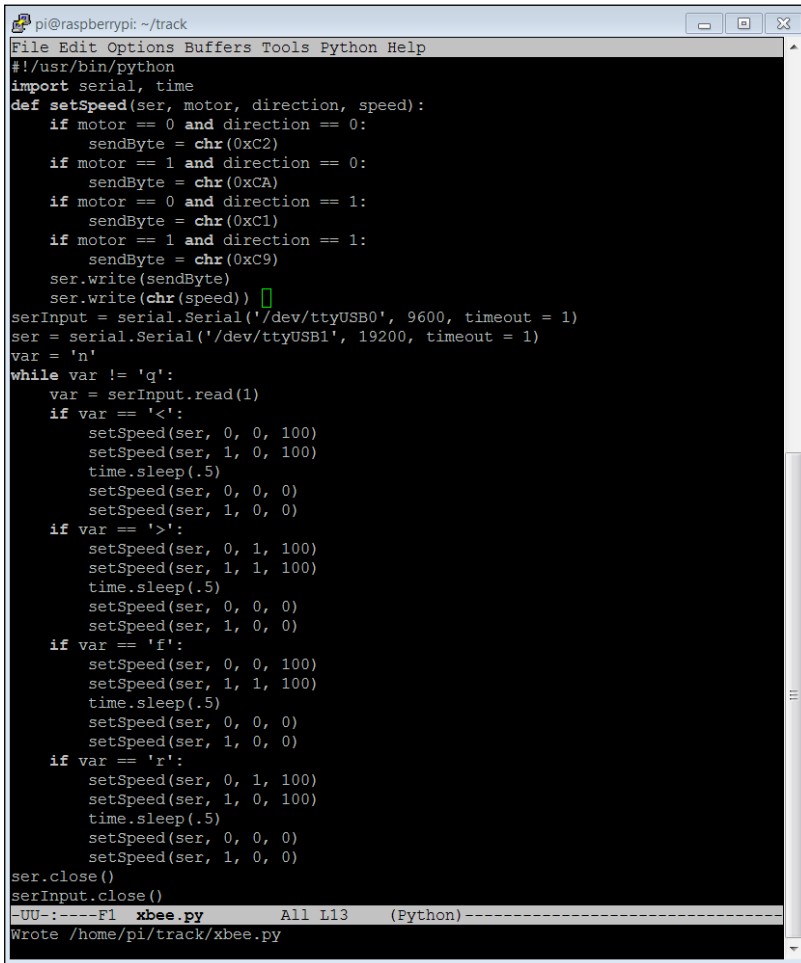
The following points explain the functionality of the code:

- The `#!/usr/bin/python` statement allows your program to run without invoking Python on the command line
- The `import serial` statement imports the Serial port library
- The `ser = serial.Serial('/dev/ttyUSB1', 9600, timeout = 1)` statement opens a serial port pointing to the `/dev/ttyUSB1` port with a baud rate of 9600 and a timeout of 1
- The `x = '\n'` statement defines a character variable and initializes it to `'\n'`, so we go through the loop at least once
- You will enter the while loop, while `x != 'q' :`, until the user enters the character `q`
- The `x = ser.read(1)` statement reads 1 byte from the serial port
- The `print x` statement prints out the value

Now, if you run `readData.py` in a terminal window and you have the PuTTY program on your personal computer connected to the other XBee module, you should see the characters that you type on the personal computer terminal windows, come out on the terminal windows running on Raspberry Pi. The following are the two screenshots given side by side:



Connecting this functionality to your robot is very easy. Start with the `remote.py` program that you created earlier in the chapter. Copy this into a new program by typing `cp remote.py xbee.py`. Now, let's remove some of the code, parts that you don't need, and add a bit that will accept the character input from the XBee module. The following screenshot shows a listing of the code:



```
pi@raspberrypi: ~/track
File Edit Options Buffers Tools Python Help
#!/usr/bin/python
import serial, time
def setSpeed(ser, motor, direction, speed):
    if motor == 0 and direction == 0:
        sendByte = chr(0xC2)
    if motor == 1 and direction == 0:
        sendByte = chr(0xCA)
    if motor == 0 and direction == 1:
        sendByte = chr(0xC1)
    if motor == 1 and direction == 1:
        sendByte = chr(0xC9)
    ser.write(sendByte)
    ser.write(chr(speed))
serInput = serial.Serial('/dev/ttyUSB0', 9600, timeout = 1)
ser = serial.Serial('/dev/ttyUSB1', 19200, timeout = 1)
var = 'n'
while var != 'q':
    var = serInput.read(1)
    if var == '<':
        setSpeed(ser, 0, 0, 100)
        setSpeed(ser, 1, 0, 100)
        time.sleep(.5)
        setSpeed(ser, 0, 0, 0)
        setSpeed(ser, 1, 0, 0)
    if var == '>':
        setSpeed(ser, 0, 1, 100)
        setSpeed(ser, 1, 1, 100)
        time.sleep(.5)
        setSpeed(ser, 0, 0, 0)
        setSpeed(ser, 1, 0, 0)
    if var == 'f':
        setSpeed(ser, 0, 0, 100)
        setSpeed(ser, 1, 1, 100)
        time.sleep(.5)
        setSpeed(ser, 0, 0, 0)
        setSpeed(ser, 1, 0, 0)
    if var == 'r':
        setSpeed(ser, 0, 1, 100)
        setSpeed(ser, 1, 0, 100)
        time.sleep(.5)
        setSpeed(ser, 0, 0, 0)
        setSpeed(ser, 1, 0, 0)
ser.close()
serInput.close()
-UU-:----F1  xbee.py          All L13      (Python)-----
Wrote /home/pi/track/xbee.py
```

There are only two meaningful changes as follows:

- `serInput = serial.Serial('/dev/ttyUSB0', 9600, timeout = 1):` This statement sets up a serial port, getting an input from the XBee device. It is important to note that the USB0 and USB1 settings might be different in your specific configuration based on whether the XBee serial device or the motor controller serial device configures first.
- `var = serInput.read(1):` With this statement, instead of getting the input from the user via the keyboard, you will be reading the characters from the XBee device.

That's it! Now, your robot should respond to commands sent from your terminal window on your personal computer. You could also create an application on your personal computer that could turn mouse movements or other inputs into proper commands for your robot.

Summary

Congratulations! Now, you can take your robot out into the big wide world. You can even use the LCD and the keyboard to make changes to your program, although the smaller screen size makes this a bit difficult. However, now that your robot is truly mobile, you may want to give it a sense of position and direction. The next chapter will show you how to add GPS to your robotic project.

9

Using a GPS Receiver to Locate Your Robot

Assuming that you have completed the tasks from the previous chapters, you should now have mobile robots that can move around, accept commands, see, and even avoid obstacles. This chapter will help you locate your robot if it moves, which can be useful for a robot that is fully autonomous. Your robot is mobile, but let's not let it get lost. You're going to add a GPS receiver, so that you can always—well, almost always—know where your robot is.

As you let your device loose, you may want it to not only know where it is, but also have a way of finding out if it has made it to the desired location. One of the coolest things to connect to your robot is a GPS location device. In this chapter, I'll show you how to connect a GPS receiver to your project and then use it to move in the correct direction.

In this chapter, we will cover the following topics:

- Connecting Raspberry Pi to a USB GPS device so that it can locate itself in relation to the world, at least wherever you can receive a GPS signal
- Connecting Raspberry Pi to a GPS device that can connect to the GPIO of Raspberry Pi
- Accessing the GPS programmatically and using its position information to move the robot to a specific location

To complete the tasks in this chapter, you'll need a USB GPS device, one that connects through an interface supported by the GPIO of Raspberry Pi. The following image is an example of a device that uses the USB interface:



The model number of the device in the preceding image is ND-100S from GlobalSat. It is small, inexpensive, and supports Windows, Mac OS X, and Linux, so our system should be able to interface with it. It is available on Amazon and other online electronics stores, so you should be able to get it almost anywhere. However, it does not have the sensitivity of some other GPS devices. So, if you will be using your robot in buildings or other locations that might stifle GPS signals, you should look for devices that are more sensitive to the signals from GPS satellites.

If you wish to choose a device with an interface that can be connected to the GPIO of Raspberry Pi, one that is quite easy to use is the VPN1513 GPS Receiver w/ Antenna, marketed by Parallax and available on their online store. It uses a simple RX/TX interface to talk with Raspberry Pi. The following is an image of the device:

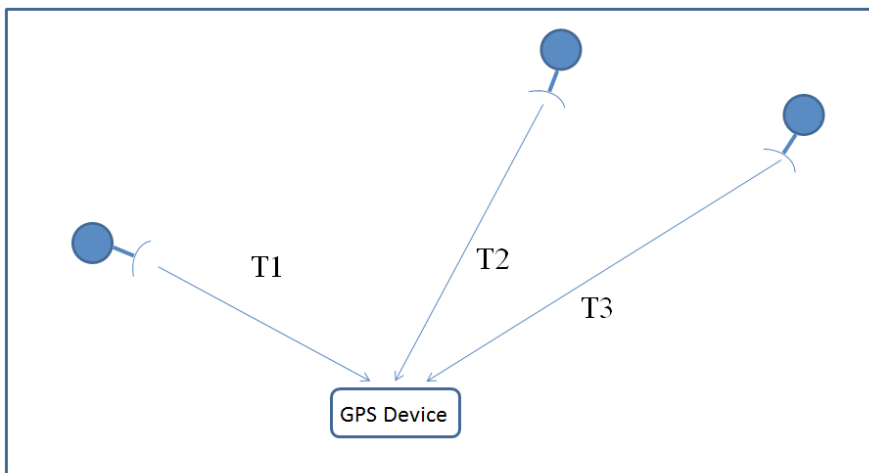


Now that you have your device, let's look at how to connect it to Raspberry Pi so that you can start collecting data.

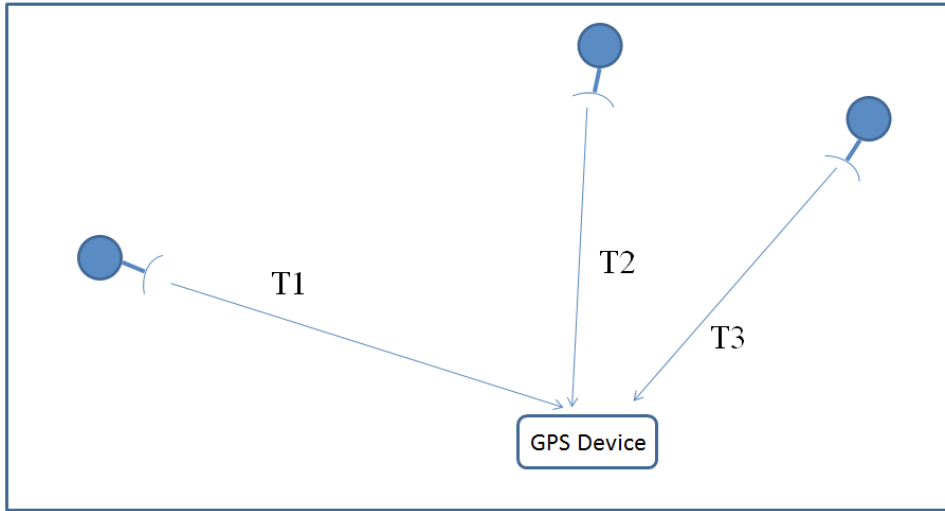
Connecting Raspberry Pi to a USB GPS device

Before you get started, let me first give you a brief tutorial on GPS. **GPS**, which stands for **Global Positioning System**, is a system of satellites that transmit signals. GPS devices use these signals to calculate a position of an object. There are a total of 24 satellites transmitting signals all around the earth at any given moment, but your device can only *see* the signal from a much smaller set of satellites.

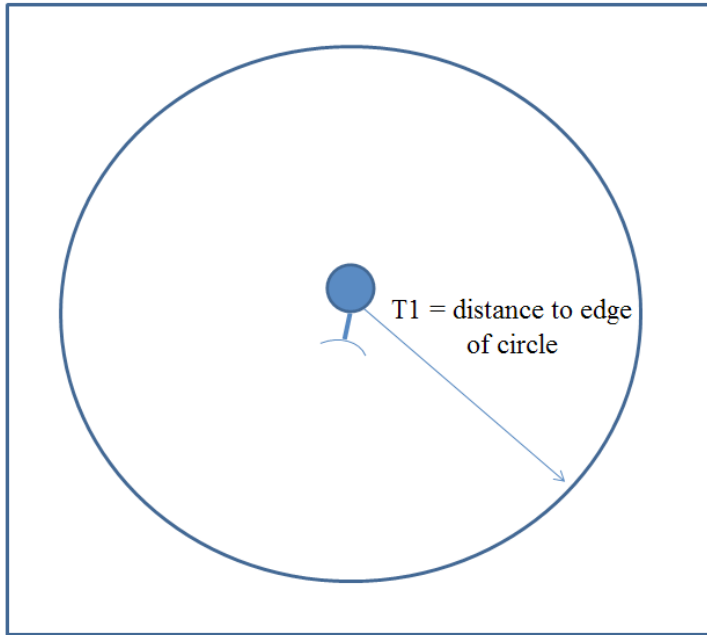
Each of these satellites transmits a very accurate time signal that your device can receive and interpret. It receives the time signal from each of these satellites, and then based on the delay—the time it takes the signal to reach the device—it calculates the receiver's position, based on a procedure called triangulation. The next two diagrams illustrate how a device uses the difference between the delay data from three satellites to calculate its position. The following is the first diagram, depicting the device at its initial position:



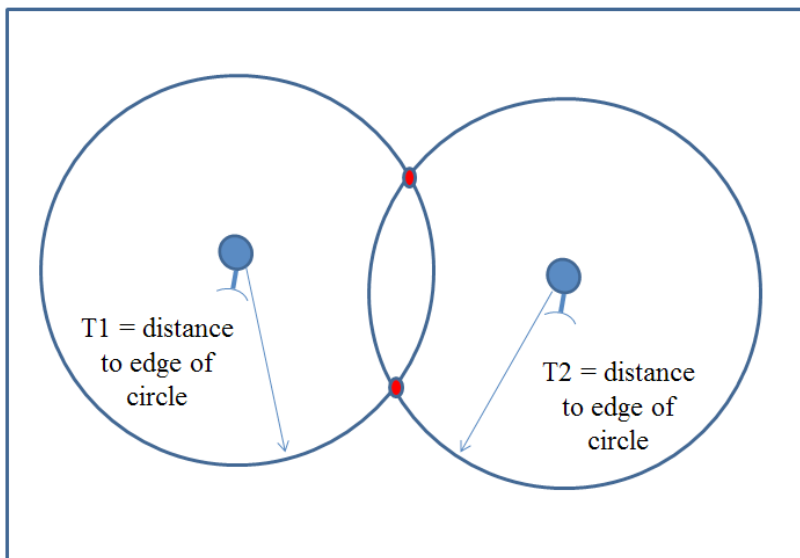
The GPS device is able to detect the three signals and the time delays associated with receiving them. In the following diagram, the device is at a different location, and the time delays associated with the three signals have changed from those in the preceding diagram:



The time delays of the signals **T1**, **T2**, and **T3** can provide the GPS with an absolute position using triangulation. Since the positions of the satellites are known, the amount of time that the signal takes to reach the GPS device is also a measure of the distance between that satellite and the GPS device. To simplify this concept, let's see an example in two dimensions. If the GPS device knows its distance from one satellite, based on the amount of time delay, you could draw a circle around the satellite at that distance and know that your GPS device is on the boundary of that sphere, as shown in the following diagram:



If you have two satellites' signals and know the distance between them, you can draw two circles, as shown in the following diagram:



However, you know that since you can only be at points on the boundary of the circle, you must be at one of the two points that are on the boundary of both the circles. Adding an additional satellite would eliminate one of these two points, providing your exact location. You need more satellites if you are going to do this in all three dimensions.

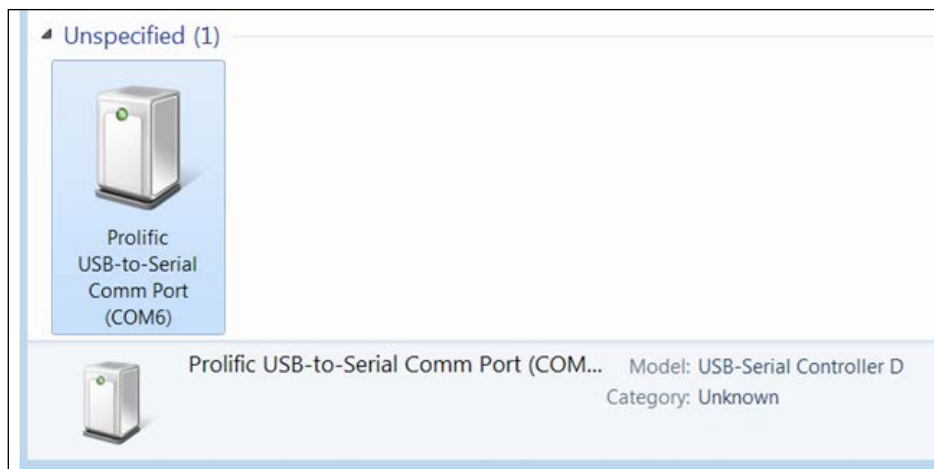
Now it's time to connect the device. While this is optional, I suggest that you connect a dongle to your PC. This will let you know whether or not the unit works, and help you understand the device a little better. Then you'll connect it to Raspberry Pi.

In order to install the GPS system on your PC, perform the following steps:

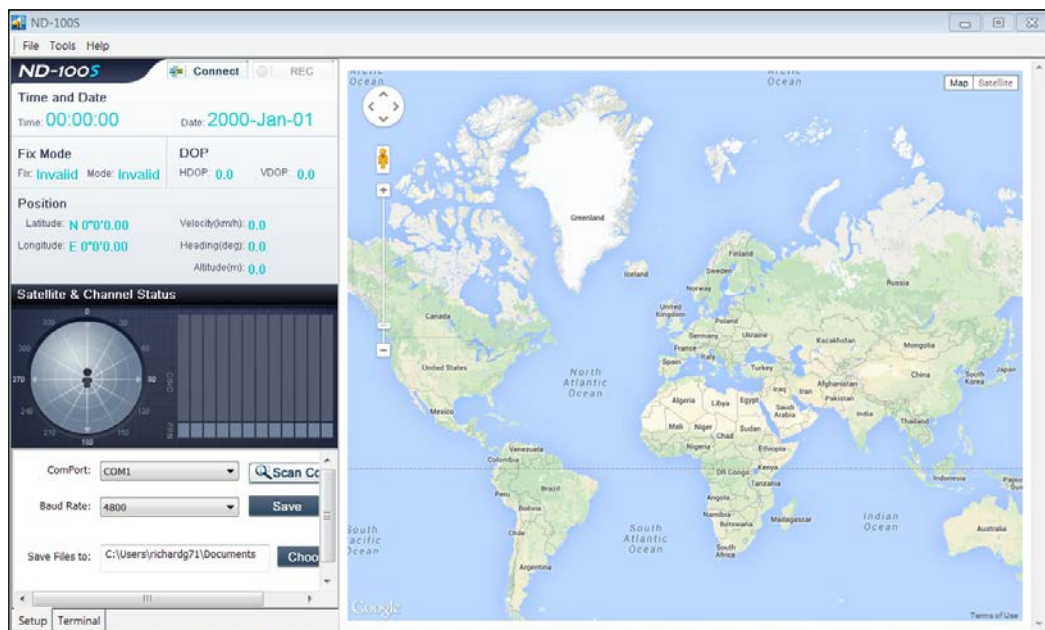
1. Insert the CD and run the setup program. You should see a window as shown in the following screenshot:



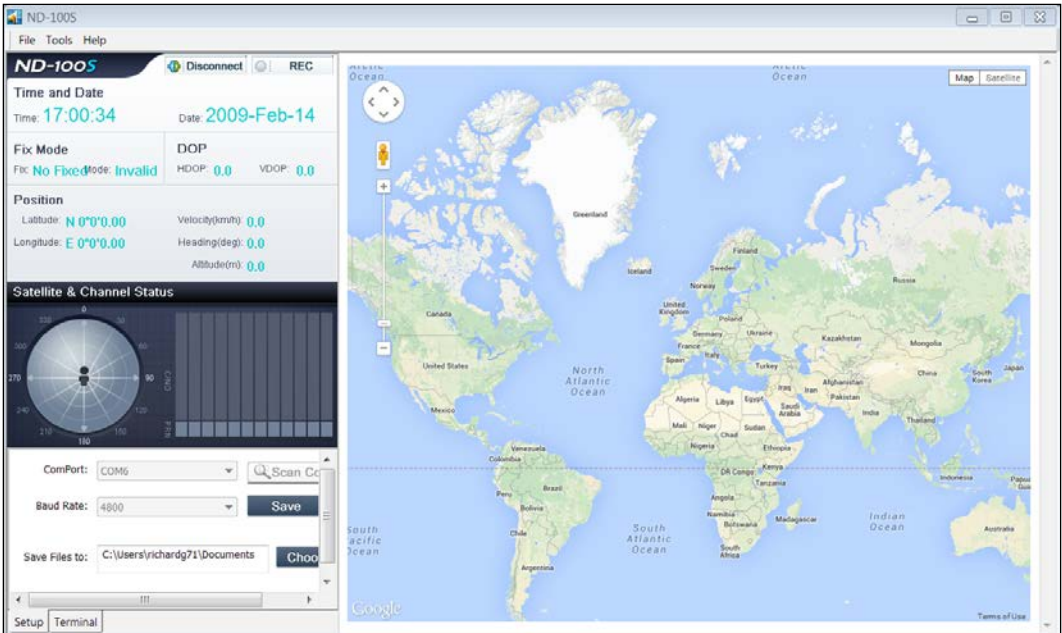
2. Click on both the **Install Driver** and **ND-100S Application** buttons and follow the default instruction procedures. If you have installed both the drivers and the application, you should be able to plug the GPS device into the USB port on your PC. A blue light at the end of the device should indicate that the device has been plugged in. The system will recognize the device, install the appropriate drivers, and give you access to it (this may take a few minutes). To ensure that the device has been installed, check your **Devices and Printers** start menu selection (if you're running Windows 7). You should see the following screenshot:



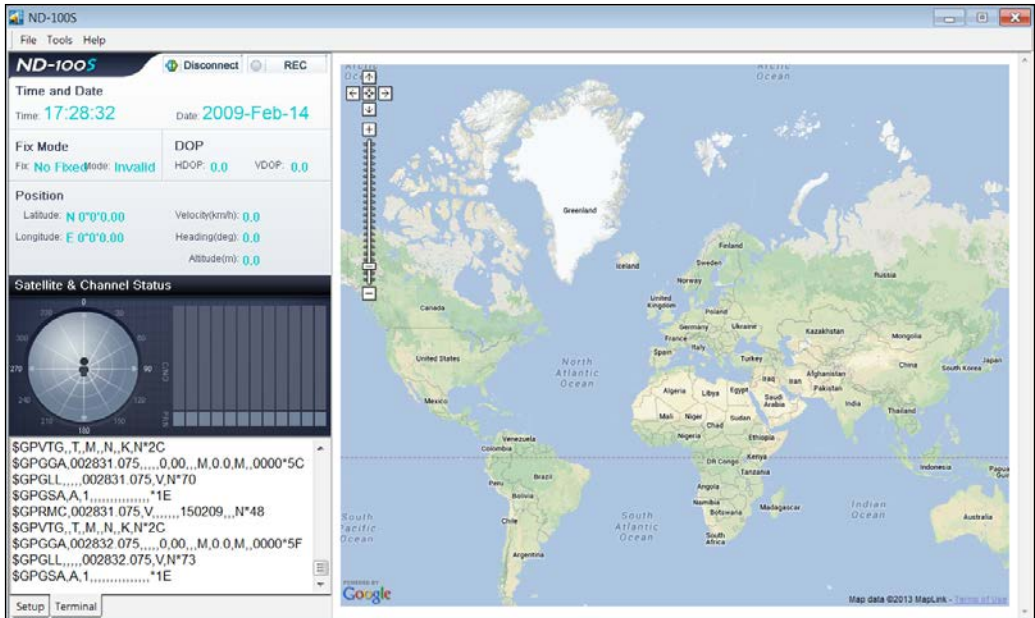
- Once the device is installed, you can also run the application that is available on the CD-ROM. At startup, it should look as shown in the following screenshot:



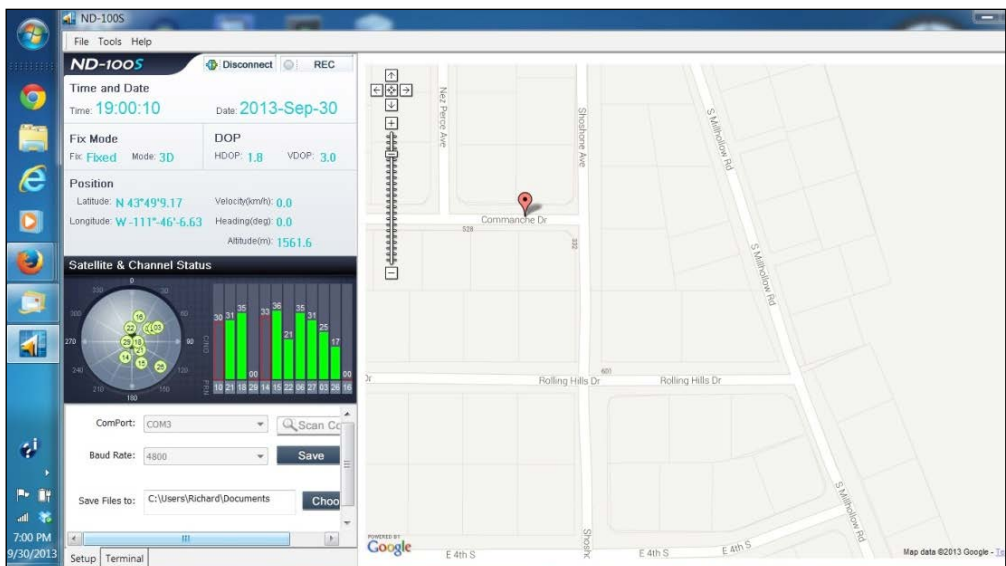
- Now click on the **Connect** button on the top-left of the screen. It should now look as shown in the following screenshot:



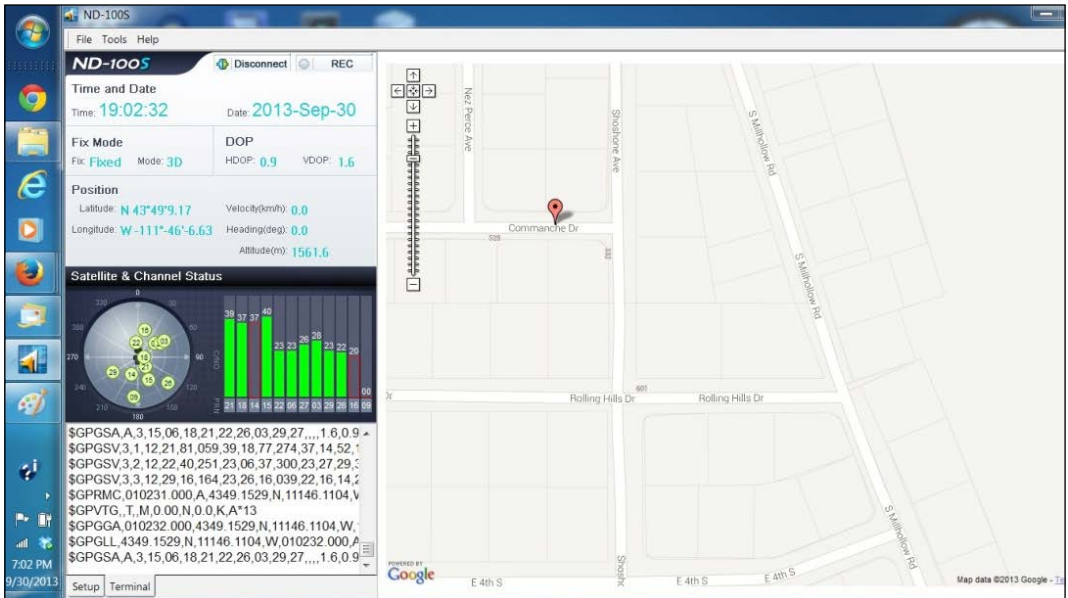
- Unfortunately, if you are in a building or a place where receiving information from the GPS satellites is difficult, the device may struggle to find its position. If you want to find out whether or not the system is working, even though it may struggle to find the signals, select the **Terminal** tab selection in the lower-left corner of the screen. You should see what is shown in the following screenshot:



6. Note that the lower-left window indicates that the device is trying to find its location. Initially, the unit in my office was unable to locate the satellites, which is not surprising when you're in a building designed to restrict the in and out transmission of signals. Following the procedure described in the preceding steps on my laptop shows this screenshot as a result:



7. You'll notice that the blue LED at the end of the GPS device is flashing. Now you have your position. When I select the **Terminal** tab, it shows the raw data returned by the GPS device:



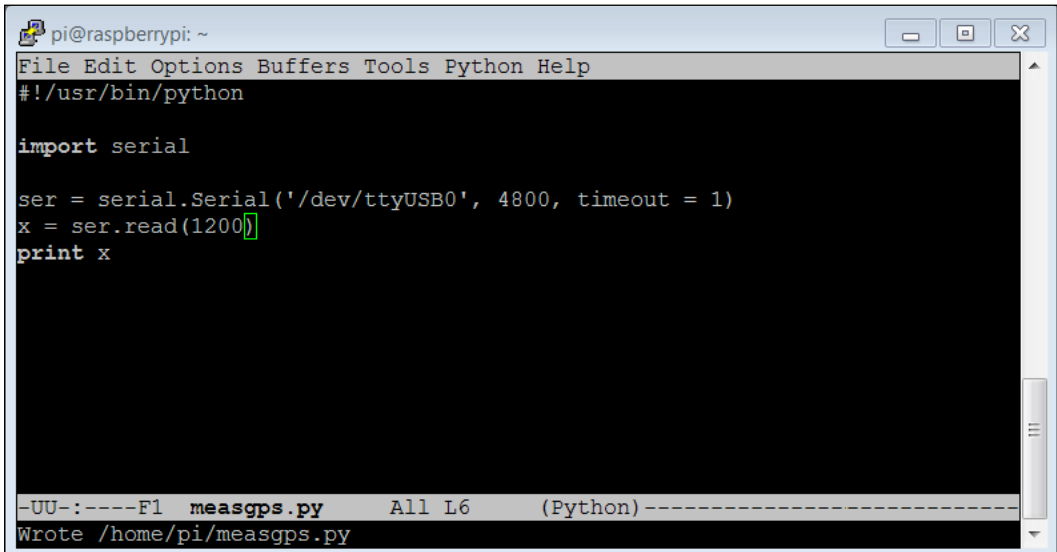
You'll use the raw data from the preceding screenshot in the next section, to plan your path to other positions. So, in an environment where GPS data is available, the unit is able to sync with it and show your position. The next step will be to hook it to your Raspberry Pi robot.

First, connect the GPS unit by plugging it into one of the free USB ports on the USB hub. Once it is plugged in and the unit is rebooted, type `lsusb` and you should see the output, as shown in the following screenshot:

```
pi@raspberrypi: ~  
pi@raspberrypi ~$ lsusb  
Bus 001 Device 002: ID 0424:9512 Standard Microsystems Corp.  
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub  
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.  
Bus 001 Device 018: ID 067b:2303 Prolific Technology, Inc. PL2303 Serial Port  
pi@raspberrypi ~$
```


The device is shown as Prolific Technology, Inc. PL2303 Serial Port. Your device is now connected to your Raspberry Pi.

Now create a simple Python program that will read the value from the GPS device. If you are using Emacs as an editor, type `emacs measgps.py`. A new file will be created called `measgps.py`. Then type the code as shown in the following screenshot:

A screenshot of the Emacs text editor window. The title bar shows 'pi@raspberrypi: ~'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Python', and 'Help'. The main text area contains the following Python code:

```
#!/usr/bin/python

import serial

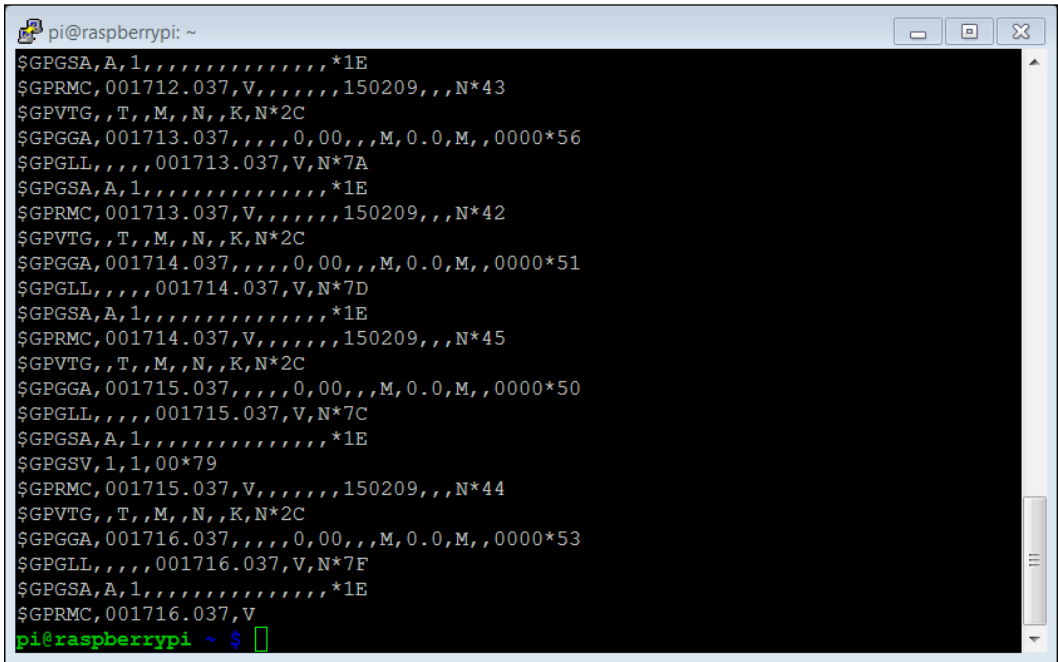
ser = serial.Serial('/dev/ttyUSB0', 4800, timeout = 1)
x = ser.read(1200)
print x
```

The status bar at the bottom displays '-UU-:----F1 measgps.py All L6 (Python)-----' and 'Wrote /home/pi/measgps.py'.

Let's go through the code to see what is happening:

- `#!/usr/bin/python`: As discussed earlier, this line simply makes this file available for you to execute from the command line.
- `import serial`: This imports the `serial` library. This will allow you to interface the USB GPS sensor with the GPS system.
- `ser = serial.Serial('/dev/ttyUSB0', 4800, timeout = 1)`: This command sets up the serial port to use the `/dev/ttyUSB0` device, which is your GPS sensor using a baud rate of 4800 and a timeout value of one second.
- `x = ser.read(1200)`: This command then reads a set of values from the USB port. In this case, you read 1200 bytes; this includes a fairly complete set of your GPS data.
- `print x`: This command then prints out the value obtained from the preceding command.

Once you have created this file, you can run the program and talk to the device. Do this by typing `python measgps.py` and the program will run. You should see the output as shown in the following screenshot:



```
pi@raspberrypi: ~  
$GPGSA,A,1,,,,,,,,,,,,,*1E  
$GPRMC,001712.037,V,,,,,,,,,150209,,,N*43  
$GPVTG,,T,,M,,N,,K,N*2C  
$GPGGA,001713.037,,,,,0,00,,,M,0.0,M,,0000*56  
$GPGLL,,,,,001713.037,V,N*7A  
$GPGSA,A,1,,,,,,,,,,,,,*1E  
$GPRMC,001713.037,V,,,,,,,,,150209,,,N*42  
$GPVTG,,T,,M,,N,,K,N*2C  
$GPGGA,001714.037,,,,,0,00,,,M,0.0,M,,0000*51  
$GPGLL,,,,,001714.037,V,N*7D  
$GPGSA,A,1,,,,,,,,,,,,,*1E  
$GPRMC,001714.037,V,,,,,,,,,150209,,,N*45  
$GPVTG,,T,,M,,N,,K,N*2C  
$GPGGA,001715.037,,,,,0,00,,,M,0.0,M,,0000*50  
$GPGLL,,,,,001715.037,V,N*7C  
$GPGSA,A,1,,,,,,,,,,,,,*1E  
$GPGSV,1,1,00*79  
$GPRMC,001715.037,V,,,,,,,,,150209,,,N*44  
$GPVTG,,T,,M,,N,,K,N*2C  
$GPGGA,001716.037,,,,,0,00,,,M,0.0,M,,0000*53  
$GPGLL,,,,,001716.037,V,N*7F  
$GPGSA,A,1,,,,,,,,,,,,,*1E  
$GPRMC,001716.037,V  
pi@raspberrypi ~ $
```

The device returns raw readings to you, which is a good sign. Unfortunately, there isn't much good data here, as the robot is again indoors. How do you know this? Look at one of the lines that starts with `$GPRMC`; this line should tell you your current latitude and longitude values. The GPRS reports the following code:

`$GPRMC,001714.037,V,,,,,,,,,150209,,,N*45`

The preceding line of data should take the form shown in the following table, with each field separated by a comma:

0	1	2	3	4	5	6	7	8	9	10	11	12
\$GPRMC	220516	A	5133.82	N	00042.24	W	173.8	231.8	130694	004.2	W	*7

The following table offers an explanation of each of the fields shown in the preceding table:

Field	Value	Explanation
1	220516	Timestamp
2	A	Validity: A (OK), V (invalid)
3	5133.82	Current latitude
4	N	North or south
5	00042.24	Current longitude
6	W	East or west
7	173.8	Speed in knots at which you are moving
8	3	Course: The angular direction in which you are moving
9	130694	Date stamp
10	0004.2	Magnetic variation: The variation between magnetic and true north
11	W	East or west
12	*70	Checksum

In this case, the field second value in the string is either reports v or that the unit cannot find enough satellites to get a position. Take the unit outdoors and you may get the result, as shown in the following screenshot from your `measgps.py` program:

```

pi@raspberrypi: ~
x,A*4F
$GPGSA,A,3,15,21,22,26,18,,,,,,,,,3.7,3.0,2.2*3F
$GPRMC,194824.000,A,4349.1418,N,11146.1046,W,0.00,,111213,,A*67
$GPVTG,,T,,M,0.00,N,0.0,K,A*13
$GPGGA,194825.000,4349.1418,N,11146.1046,W,1,05,3.0,1560.8,M,-16.9,M,,0000*54
$GPGLL,4349.1418,N,11146.1046,W,194825.000,A,A*4E
$GPGSA,A,3,15,21,22,26,18,,,,,,,,,3.7,3.0,2.2*3F
$GPRMC,194825.000,A,4349.1418,N,11146.1046,W,0.00,,111213,,A*66
$GPVTG,,T,,M,0.00,N,0.0,K,A*13
$GPGGA,194826.000,4349.1418,N,11146.1046,W,1,05,3.0,1560.8,M,-16.9,M,,0000*57
$GPGLL,4349.1418,N,11146.1046,W,194826.000,A,A*4D
$GPGSA,A,3,15,21,22,26,18,,,,,,,,,3.7,3.0,2.2*3F
$GPRMC,194826.000,A,4349.1418,N,11146.1046,W,0.00,,111213,,A*65
$GPVTG,,T,,M,0.00,N,0.0,K,A*13
$GPGGA,194827.000,4349.1418,N,11146.1046,W,1,05,3.0,1560.8,M,-16.9,M,,0000*56
$GPGLL,4349.1418,N,11146.1046,W,194827.000,A,A*4C
$GPGSA,A,3,15,21,22,26,18,,,,,,,,,3.7,3.0,2.2*3F
$GPGSV,3,1,12,21,81,018,35,18,71,255,31,15,50,083,35,22,33,245,30*7E
$GPGSV,3,2,12,06,32,307,23,26,23,045,32,27,23,314,21,29,22,161,*70
$GPGSV,3,3,12,16,18,283,20,03,13,319,,24,,123,,09,,019,*72
$GPRMC,194827.000,A,4349.1418,N,11146.1046,W,0.00,,111213,,A*64
$GPVTG,,T,,M,0.00,N,0.0,K,A*13
$GPGGA,194828.
pi@raspberrypi ~ $

```

Note that the \$GPRMC line now reads as follows:

```
$GPRMC,194827.000,A,4349.1418,N,11146.1046,W,0.00,,111213,,,A*64
```

Now the values will be as shown in the following table:

Field	Value	Explanation
1	020740.000	Timestamp
2	A	Validity: A (OK), V (invalid)
3	4349.1426	Current latitude
4	N	North or south
5	11146.1064	Current longitude
6	W	East or west
7	1.82	Speed in knots at which you are moving
8	214.11	Course: The angular direction in which you are moving
9	021013	Date stamp
10		Magnetic variation: The variation between magnetic and true north
11		East or west
12	*7B	Checksum

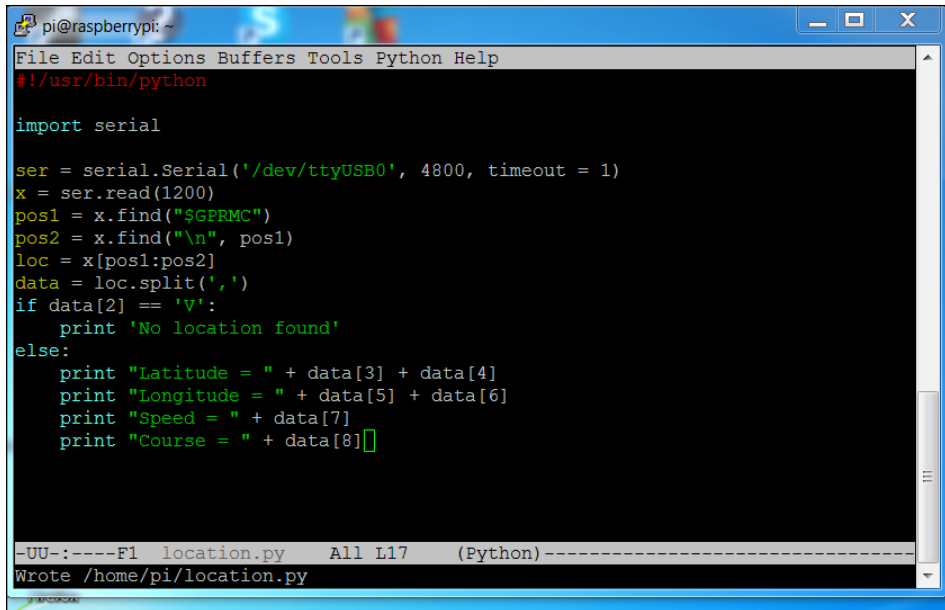
Now you have some indication of where you are; however, the GPS data is in a raw form that may not mean much. In the next section, you will figure out how to do something with these readings.

Accessing the USB GPS programmatically

Now that you can access the GPS device, let's work on accessing the data programmatically. Your project should now have the GPS device connected and have access to query the data via the serial port. In this section, you will create a program to use this data to discover where you are, and then you can determine what to do with that information.

If you've completed the previous section, you should be able to receive the raw data from the GPS unit. Now you want to be able to do something with this data; for example, find your current location and altitude, and then decide whether your target location is to the west, east, north, or south.

First, get the information from the raw data. As noted previously, the position and speed is in the \$GPRMC output of the GPS device. You will first write a program to simply parse out a couple of pieces of information from that data. So open a new file (you can name it `location.py`) and edit it, as shown in the following screenshot:



```

pi@raspberrypi: ~
File Edit Options Buffers Tools Python Help
#!/usr/bin/python

import serial

ser = serial.Serial('/dev/ttyUSB0', 4800, timeout = 1)
x = ser.read(1200)
pos1 = x.find("$GPRMC")
pos2 = x.find("\n", pos1)
loc = x[pos1:pos2]
data = loc.split(',')
if data[2] == 'V':
    print 'No location found'
else:
    print "Latitude = " + data[3] + data[4]
    print "Longitude = " + data[5] + data[6]
    print "Speed = " + data[7]
    print "Course = " + data[8]

-UU-:----F1  location.py  All L17  (Python)-----
Wrote /home/pi/location.py

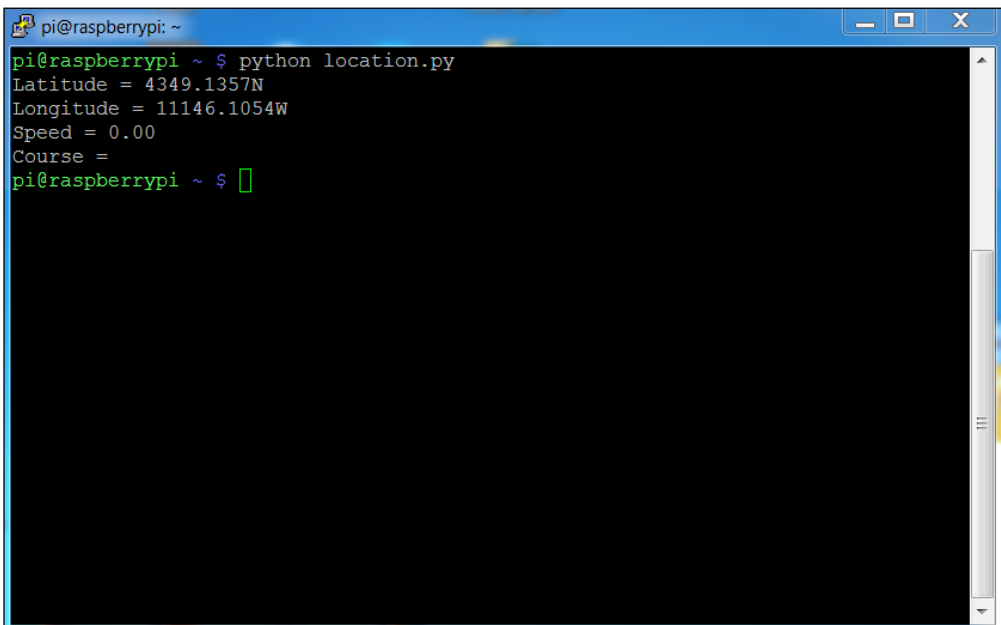
```

The code lines are explained as follows:

- `#!/usr/bin/Python`: As always, this line simply makes this file available for you to execute from the command line.
- `import serial`: You again import the `serial` library. This will allow you to interface the USB GPS sensor with the GPS system.
- `if __name__=="__main__":`: The main part of your program is then defined using this line.
- `ser = serial.Serial('/dev/ttyUSB0', 4800, timeout = 1)`: This command sets up the serial port to use the `/dev/ttyUSB0` device, which is your GPS sensor using a baud rate of 4800 and a timeout value of one second.
- `x = ser.read(500)`: This command then reads a set of values from the USB port. In this case, you read 500 values, which includes a fairly complete set of your GPS data.

- `pos1 = x.find("$GPRMC")`: This will find the first occurrence of `$GPRMC` and set the value `pos1` to that position. In this case, you want to isolate the `$GPRMC` response line.
- `pos2 = x.find("\n", pos1)`: This will find the end of this string of text.
- `loc = x[pos1:pos2]`: The `loc` variable will now hold the path, including all the information you are interested in.
- `data = loc.split(',')`: This will break your comma-separated line into an array of values.
- `if data[2] == 'V'::` You now check to see whether or not the data is valid. If not, the next line simply prints out that you did not find a valid location.
- `else::` If the data is valid, the next few lines print out the various pieces of data.

The following screenshot is an example showing the result that appeared when my device was able to find its location:



```
pi@raspberrypi: ~  
pi@raspberrypi ~ $ python location.py  
Latitude = 4349.1357N  
Longitude = 11146.1054W  
Speed = 0.00  
Course =  
pi@raspberrypi ~ $
```

Once you have the data, you can do some interesting things with it. For example, you might want to figure out the distance from and direction to another waypoint. There is a piece of code at <http://code.activestate.com/recipes/577594-GPS-distance-and-bearing-between-two-GPS-points/> that you can use to find the distances from and bearings to other waypoints, based on your current location. You can easily add this code to your `location.py` program to update your robot on the distances and bearings to other waypoints.

Now your robot knows where it is and the direction it needs to get to other locations! There is another way to configure your GPS device, that may make it a bit easier to access the data from other programs; it is using a functionality set held in the `gpsd` library. To install this capability, type `sudo apt-get install gpsd gpsd-clients` and this will install the `gpsd` software. For a tutorial on this software, go to http://wiki.ros.org/gpsd_client/Tutorials/Getting%20Started%20with%20gpsd_client. This software works by starting a background program (called a daemon) that communicates with your GPS device. You can then just query that program to get the GPS data. To start the process, type `sudo gpsd /dev/ttyUSB0 -F /var/run/gpsd.sock`. You can run the program by typing `cgps`. A sample result is shown in the following screenshot:

```

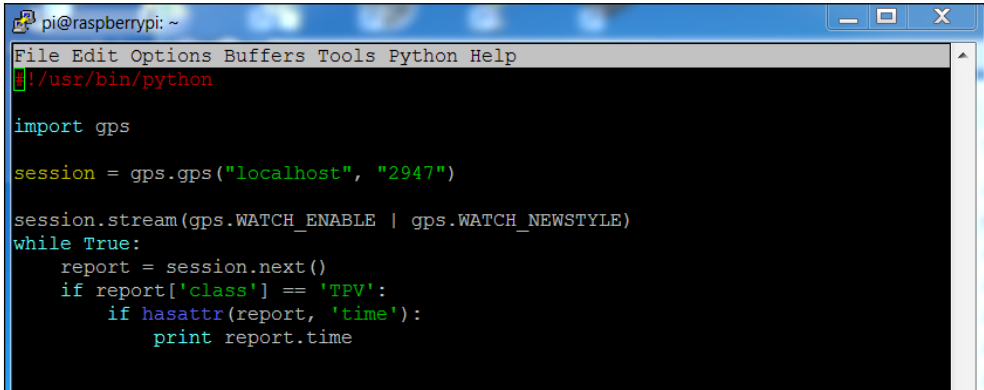
Time:      2013-12-15T17:35:59.000Z
Latitude:  43.818948 N
Longitude:  111.768426 W
Altitude:   1551.3 m
Speed:      0.0 kph
Heading:    0.0 deg (true)
Climb:      0.0 m/min
Status:     3D FIX (33 secs)
Longitude Err: +/- 11 m
Latitude Err: +/- 17 m
Altitude Err: +/- 44 m
Course Err:  n/a
Speed Err:   +/- 129 kph
Time offset: 1.097
Grid Square: DN43ct

PRN:  Elev:  Azim:  SNR:  Used:
 2    13     081    23     Y
25    24     205    20     Y
18    21     214    31     Y
 5    43     054    33     Y
29    82     144    26     Y
26    25     105    36     Y
21    43     277    16     Y

{"el":24,"az":205,"ss":20,"used":true},{ "PRN":18,"el":21,"az":214,"ss":31,"used":
true},{ "PRN":5,"el":43,"az":54,"ss":33,"used":true},{ "PRN":29,"el":82,"az":144,"
ss":26,"used":true},{ "PRN":26,"el":25,"az":105,"ss":36,"used":true},{ "PRN":21,"e
{"class":"TPV","tag":"MID2","device":"/dev/ttyUSB0","mode":3,"time":"2013-12-15T
17:35:59.000Z","ept":0.005,"lat":43.818948191,"lon":-111.768426061,"alt":1551.33
3,"epx":11.793,"epy":17.990,"epv":44.013,"track":0.0000,"speed":0.000,"climb":0.
000,"eps":35.98}

```

The preceding screenshot displays both the formatted and some of the raw data that is being received from the GPS sensor. If you get a timeout error when attempting to run this program, type `sudo killall gpstd` to kill all running instances of the daemon and then type `sudo gpstd /dev/ttyUSB0 -F /var/run/gpsd.sock` again. You can also access this information from a program. To do this, edit a new file called `gpstry1.py`. The code will look as shown in the following screenshot:



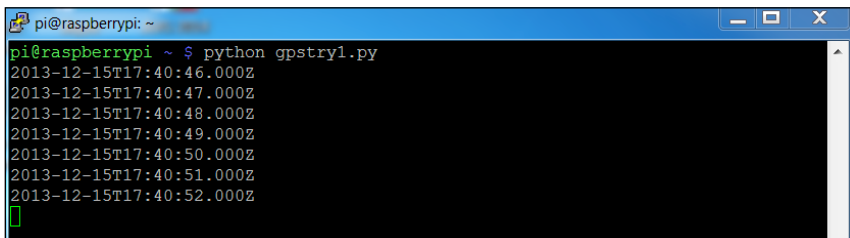
```
pi@raspberrypi: ~  
File Edit Options Buffers Tools Python Help  
#!/usr/bin/python  
  
import gps  
  
session = gps.gps("localhost", "2947")  
  
session.stream(gps.WATCH_ENABLE | gps.WATCH_NEWSTYLE)  
while True:  
    report = session.next()  
    if report['class'] == 'TPV':  
        if hasattr(report, 'time'):  
            print report.time
```

The following are the details of your code:

- `#!/usr/bin/Python`: As always, this line simply makes this file available for you to execute from the command line.
- `import gps`: In this case, you import the `gps` library. This will allow you to access the `gpsd` functionality.
- `session = gps.gps("localhost", "2947")`: This opens a communication path between the `gpsd` functionality and your program. It also opens port 2947, which is assigned to the `gpsd` functionality, on the localhost.
- `session.stream(GPS.WATCH_ENABLE | GPS.WATCH_NEWSTYLE)`: This tells the system to look for new GPS data as it becomes available.
- `while True::` This simply loops and processes information until you ask the system to stop (it can be stopped by pressing `Ctrl + C`).
- `report = session.next()`: When a report is ready, it is saved in the `report` variable.
- `if report['class'] == 'TPV'::` This line checks to see if the report will give you the type of data that you need.
- `if hasattr(report, 'time')::` This line makes sure that the report holds time data.

- `print report.time`: This prints the time data. I use this in my example because the time data is always returned, even if the GPS is not able to see enough satellites to return position data. To see other possible attributes, visit www.catb.org/gpsd/gpsd_json.html for details.

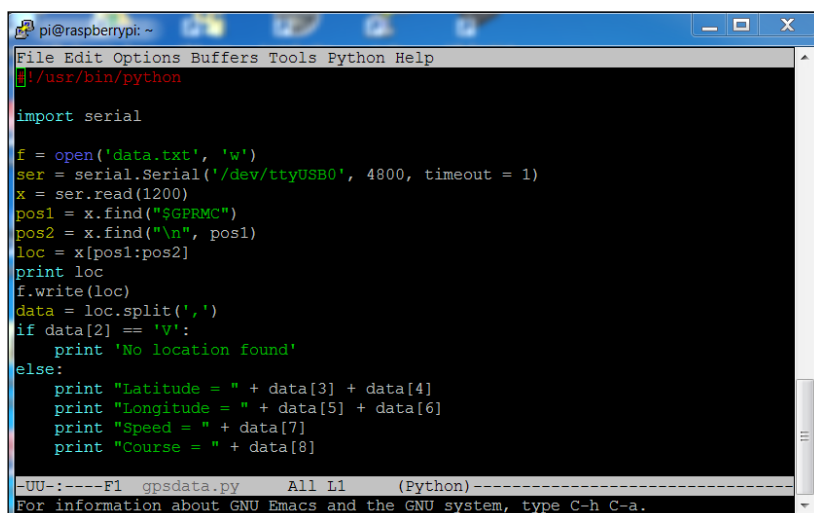
Once you have created the program, you can run it by typing `python gpstry1.py`. The following screenshot shows how the output should look after running the program:



```
pi@raspberrypi: ~
pi@raspberrypi ~ $ python gpstry1.py
2013-12-15T17:40:46.000Z
2013-12-15T17:40:47.000Z
2013-12-15T17:40:48.000Z
2013-12-15T17:40:49.000Z
2013-12-15T17:40:50.000Z
2013-12-15T17:40:51.000Z
2013-12-15T17:40:52.000Z
```

One cool way to display positional information is using a graphical display including a map of your current position. There are several map applications that can interface with your GPS to indicate your location on a map.

One map application that works well is GpsPrune. To get this application, type `sudo apt-get install gpsprune`. To run this command, you'll need to be in a graphical environment, so you'll need to run it either with a display and keyboard attached, or by using `vncserver`. You'll also need to store your data, so that the program can import it. To do this, let's amend the `location.py` program to save the data to a file. First, copy the program by typing `cp location.py gpsdata.py`. Now edit the program to make it look, as shown in the following screenshot:



```
pi@raspberrypi: ~
File Edit Options Buffers Tools Python Help
! /usr/bin/python

import serial

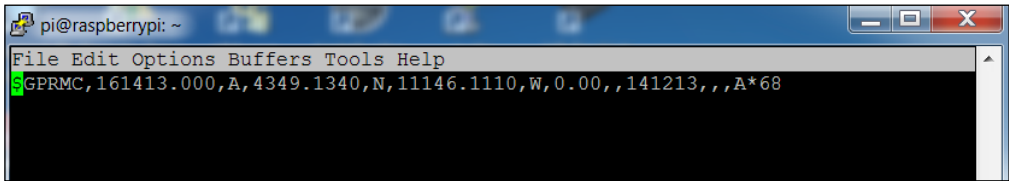
f = open('data.txt', 'w')
ser = serial.Serial('/dev/ttyUSB0', 4800, timeout = 1)
x = ser.read(1200)
pos1 = x.find("$GPRMC")
pos2 = x.find("\n", pos1)
loc = x[pos1:pos2]
print loc
f.write(loc)
data = loc.split(',')
if data[2] == 'V':
    print 'No location found'
else:
    print "Latitude = " + data[3] + data[4]
    print "Longitude = " + data[5] + data[6]
    print "Speed = " + data[7]
    print "Course = " + data[8]

-UU:-----F1  gpsdata.py      All L1      (Python)-----
For information about GNU Emacs and the GNU system, type C-h C-a.
```


The following are the two changes that you need to make:

- `f = open('data.txt', 'w')`: This line opens the file `data.txt` for writing the data in it
- `f.write(loc)`: This will write the line `loc` in the file which will hold the entire data set

Now run the program by typing `python gspdata.py`. After the program is run, you should see a new data file called `data.txt`. You can view the contents of the file by typing `emacs data.txt`. You should see the result, as shown in the following screenshot:



Once you have the data, you can look at it in various ways. Looking at your location on a map will be covered in the *Looking at the GPS data* section of this chapter.

Connecting Raspberry Pi to an RX/TX (UART) GPS device

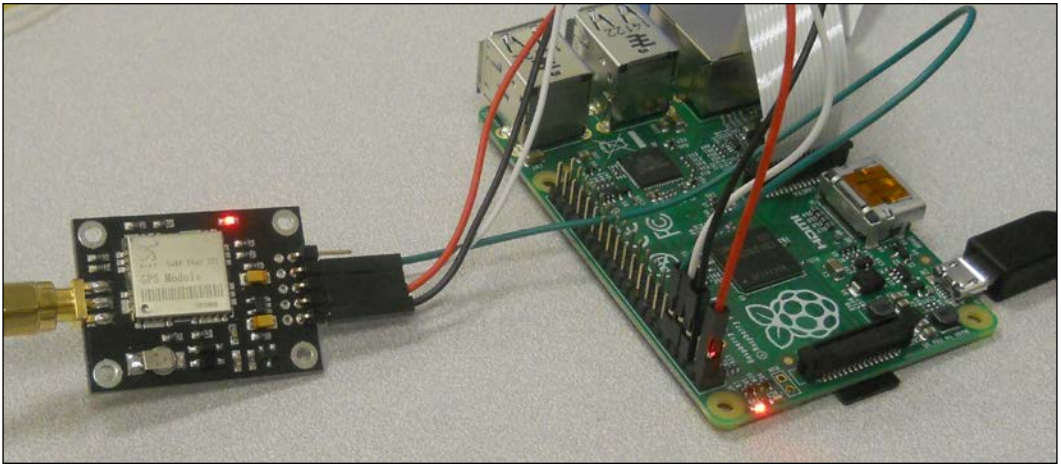
If you have chosen the GPS device that uses a standard RX/TX (UART) interface, you'll need to connect to the pins on the board. The following is an image of these pins:



You'll connect your Raspberry Pi using the male-to-female solderless jumper cables. The following are the connections on the BeagleBone Black:

BeagleBone Black pin	GPS Cable pin
P2	Vcc
P6	GND
P8	TX
P10	RX

The following is an image of the cables connected between the device and the GPS unit:

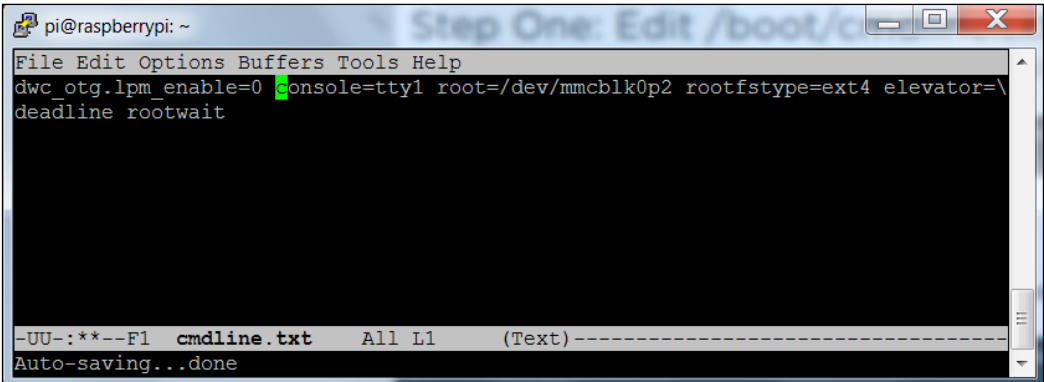


Now that the two devices are connected, you can access the device through Raspberry Pi.

Communicating with the RX/TX GPS programmatically

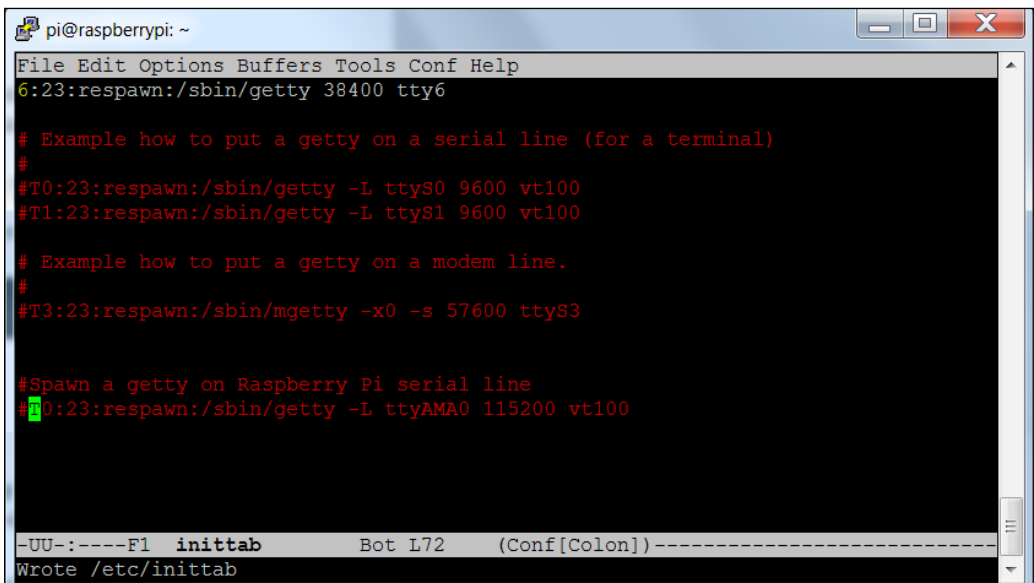
The GPS device will be talking over the RX/TX interface, so you'll need to edit two files to connect the device through the RX/TX connections on the GPIO pins for Raspberry Pi. Perform the following steps to connect the GPS for communicating:

1. First, type `sudo nano /boot/cmdline.txt`. Now edit the file to look as shown in the following screenshot:



```
pi@raspberrypi: ~  
File Edit Options Buffers Tools Help  
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=\  
deadline rootwait  
-UU-:***--F1 cmdline.txt All L1 (Text)-----  
Auto-saving...done
```

2. The second file you'll edit is `sudo nano /etc/inittab`. Edit this file by commenting out the last line, as shown in the following screenshot:



```
pi@raspberrypi: ~  
File Edit Options Buffers Tools Conf Help  
6:23:respawn:/sbin/getty 38400 tty6  
  
# Example how to put a getty on a serial line (for a terminal)  
#  
#T0:23:respawn:/sbin/getty -L ttyS0 9600 vt100  
#T1:23:respawn:/sbin/getty -L ttyS1 9600 vt100  
  
# Example how to put a getty on a modem line.  
#  
#T3:23:respawn:/sbin/mgetty -x0 -s 57600 ttyS3  
  
#Spawn a getty on Raspberry Pi serial line  
#0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100  
-UU-:----F1 inittab Bot L72 (Conf[Colon])-----  
Wrote /etc/inittab
```

3. Now restart your Raspberry Pi. You'll now create a program to communicate with the GPS unit. To do this, if you are using Emacs as an editor, type `emacs measgps.py`. A new file will be created called `measgps.py`. Then type the code as shown in the following screenshot:

```

pi@raspberrypi: ~
File Edit Options Buffers Tools Python Help
import serial

ser = serial.Serial(port = "/dev/ttyAMA0", baudrate = 9600)

x = ser.read(1200)
print x
ser.close()

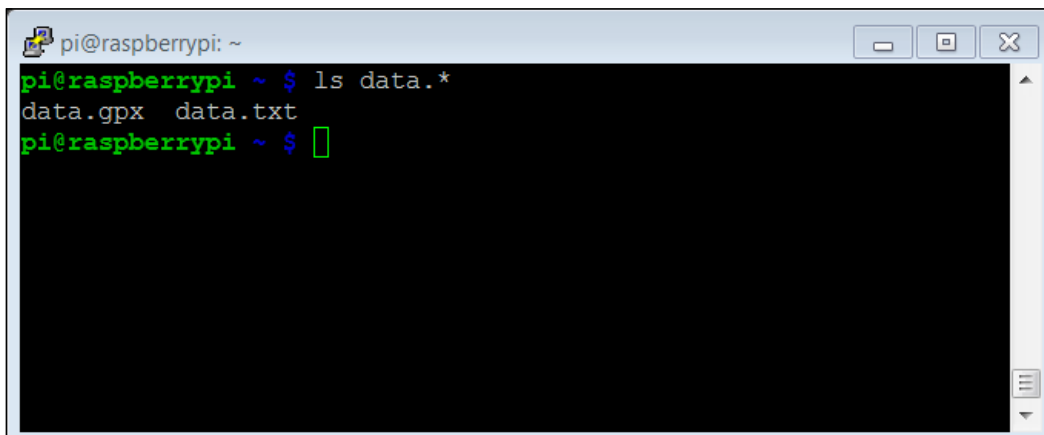
-UU-:----F1 meas_gps.py All L1 (Python)-----
For information about GNU Emacs and the GNU system, type C-h C-a.

```

4. Let's go through the code to see what is happening:
 - `#!/usr/bin/python`: As before, the first line simply makes this file available for you to execute from the command line.
 - `import serial`: You also import the serial library. This will allow you to interface with the RX/TX port.
 - `ser = serial.Serial(port = "/dev/ttyO1", baudrate=9600)`: This command sets up the serial port to use the `/dev/ttyO1` device, which is our GPS sensor, using a baud rate of 9600.
 - `x = ser.read(1200)`: This command then reads in a set of values from the RX/TX port. In this case, you read 1200 values, which will include a full set of GPS data.
 - `print x`: This command then prints out the value.
 - `ser.close()`: This closes the serial port.

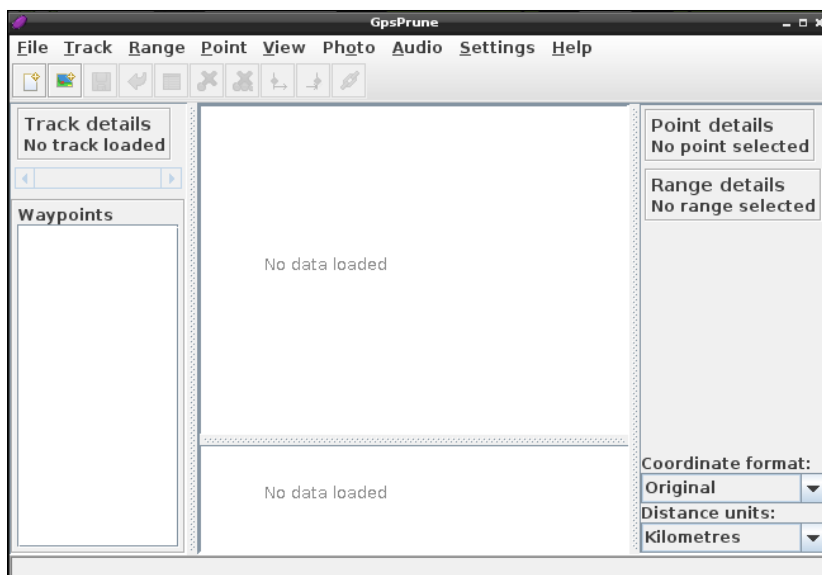
Taking a look at the GPS data

You can now look at the data in the GpsPrune application. First, you need to convert your **National Marine Electronics Association** (NMEA)-formatted data to a data format that GpsPrune can understand. To do this, type `gpsbabel -i NMEA -f data.txt -o GPX -F data.gpx`. Type a `ls data.*` command and you should see the files, as shown in the following screenshot:

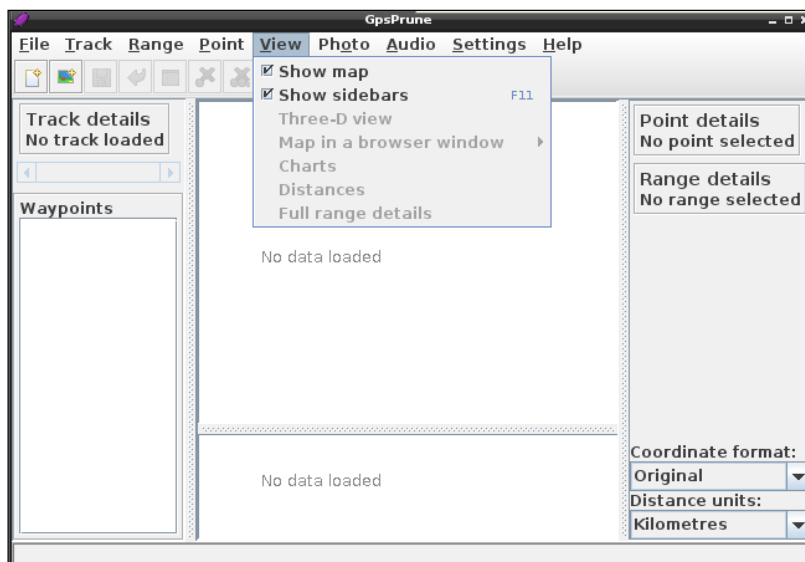


```
pi@raspberrypi: ~  
pi@raspberrypi ~$ ls data.*  
data.gpx data.txt  
pi@raspberrypi ~$
```

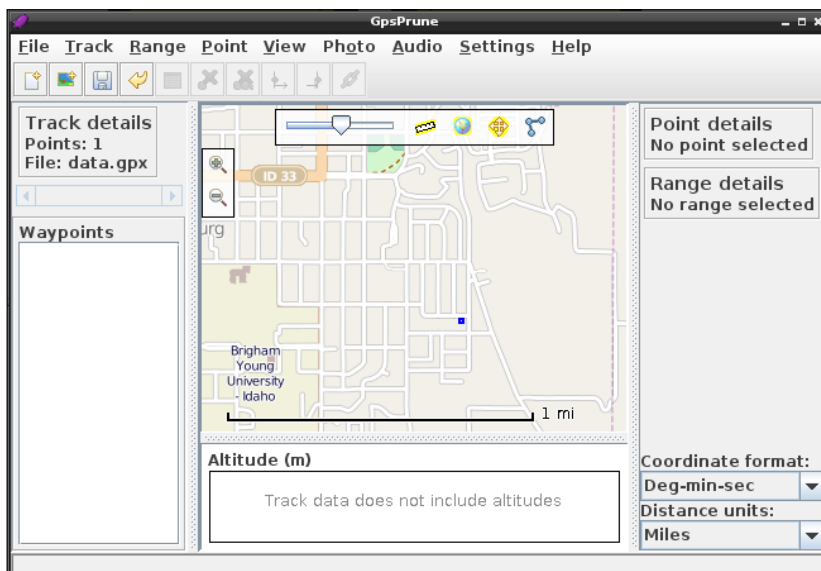
Now run the program by typing `gpsprune` in a terminal window. The application will open, as shown in the following screenshot:



Turn on the map function by navigating to **View | Show map**, as shown in the following screenshot:

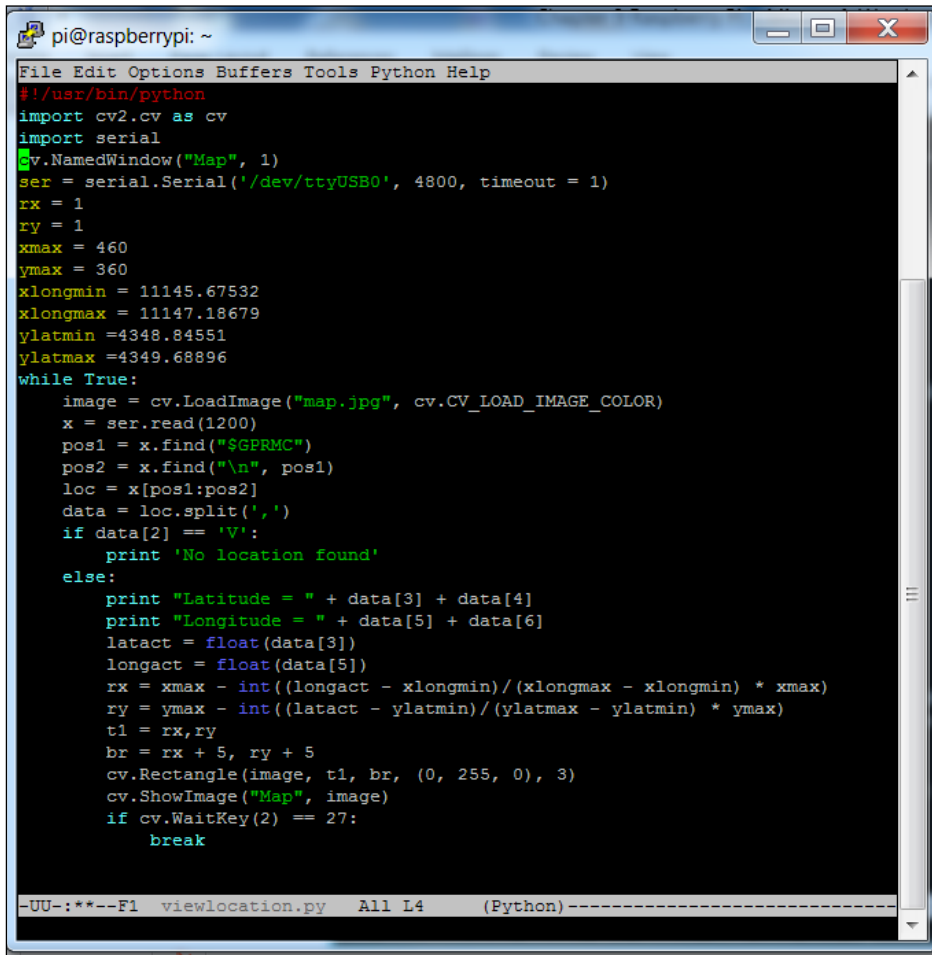


Now open your file by navigating to **File | Open** and then selecting your filename with the .gpx file extension. The window should open, as shown in the following screenshot:



You may need to zoom out to see your location, but it should be there on the map. You could also capture multiple locations and then display them as routes. But you might want something a bit simpler, such as a program that allows you to enter a waypoint and display a map that shows both the waypoint and your current location. You can do this with the skills you learned in *Chapter 4, Adding Vision to Raspberry Pi*.

Let's start with the code in `location.py`. Make a copy by typing `cp location.py viewlocation.py`. Now make the changes, as shown in the following screenshot:



```

pi@raspberrypi: ~
File Edit Options Buffers Tools Python Help
#!/usr/bin/python
import cv2.cv as cv
import serial
cv.NamedWindow("Map", 1)
ser = serial.Serial('/dev/ttyUSB0', 4800, timeout = 1)
rx = 1
ry = 1
xmax = 460
ymax = 360
xlongmin = 11145.67532
xlongmax = 11147.18679
ylatmin = 4348.84551
ylatmax = 4349.68896
while True:
    image = cv.LoadImage("map.jpg", cv.CV_LOAD_IMAGE_COLOR)
    x = ser.read(1200)
    pos1 = x.find("$GPRMC")
    pos2 = x.find("\n", pos1)
    loc = x[pos1:pos2]
    data = loc.split(',')
    if data[2] == 'V':
        print 'No location found'
    else:
        print "Latitude = " + data[3] + data[4]
        print "Longitude = " + data[5] + data[6]
        latact = float(data[3])
        longact = float(data[5])
        rx = xmax - int((longact - xlongmin)/(xlongmax - xlongmin) * xmax)
        ry = ymax - int((latact - ylatmin)/(ylatmax - ylatmin) * ymax)
        t1 = rx, ry
        br = rx + 5, ry + 5
        cv.Rectangle(image, t1, br, (0, 255, 0), 3)
        cv.ShowImage("Map", image)
        if cv.WaitKey(2) == 27:
            break
-UU-:***-F1 viewlocation.py All L4 (Python)-----

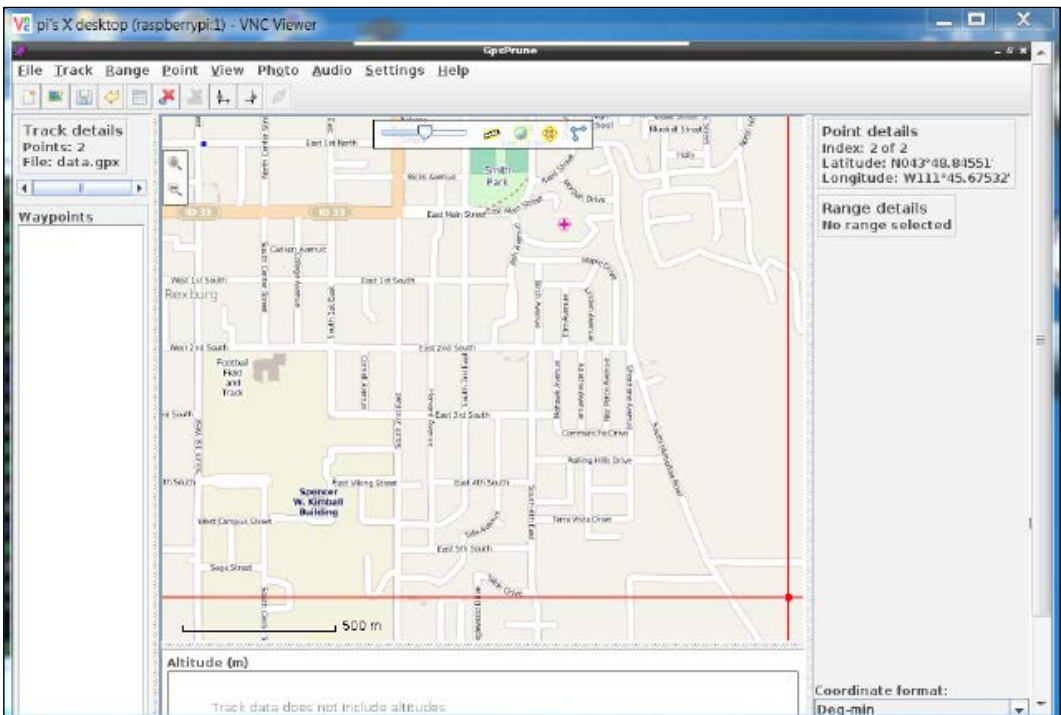
```


The following are the details of the changes:

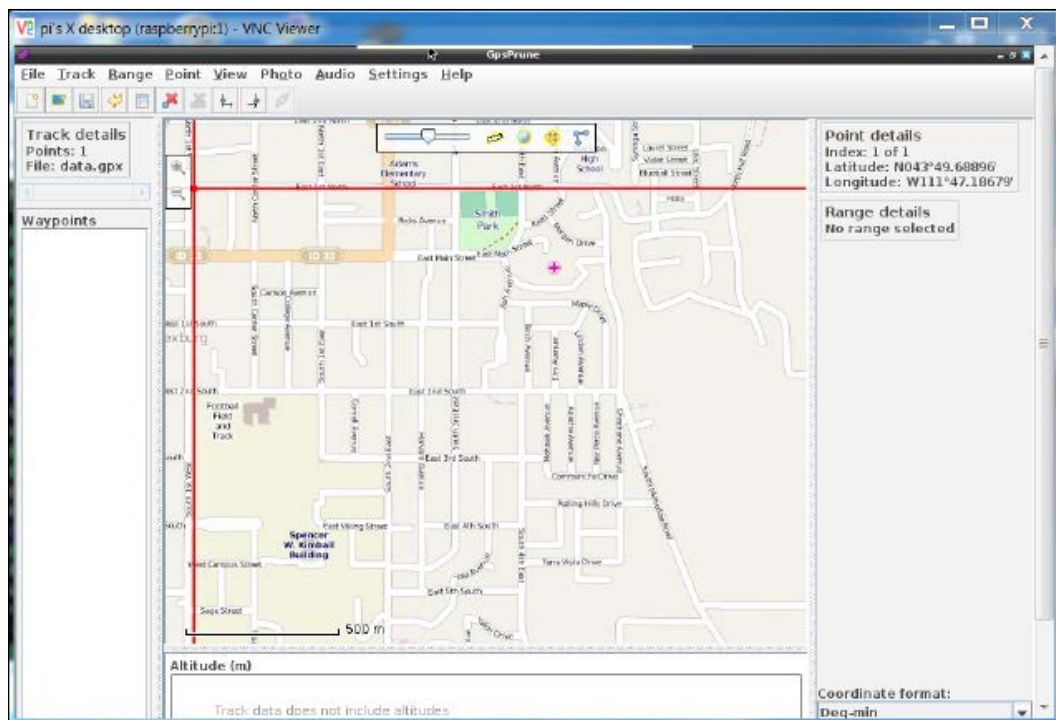
- `import cv2 as cv`: You'll be using the OpenCV library to draw the images, so you need to import it using this command.
- `rx = 1`: This variable will hold the *x* pixel value of your location.
- `ry = 1`: This variable will hold the *y* pixel value of your location.
- `xmax = 460`: This is the maximum *x* pixel value in the map you created.
- `ymax = 360`: This is the maximum *y* pixel value in the map you created.
- `xlongmin = 11145.67532`: This is the minimum longitude value in the map you created.
- `xlongmax = 11147.18679`: This is the maximum longitude value in the map you created.
- `ylatmin = 4348.84551`: This is the minimum latitude value in the map you created.
- `ylatmax = 4349.68896`: This is the maximum latitude value in the map you created.
- `while True::` This loops through the program, drawing the map and the position each time you get a new position from the GPS system.
- `image = cv.LoadImage("map.jpg", cv.CV_LOAD_IMAGE_COLOR)`: This creates a new data structure called `image`, and initializes it with the map image.
- `latact = float(data[3])`: This command turns the string that is the latitude into a float value.
- `longact = float(data[5])`: This command turns the string that is the longitude into a float value.
- `rx = xmax - int((longact - xlongmin)/(xlongmax - xlongmin) * xmax)`: This calculates the *x* value in pixels, by taking the ratio of the range from the actual longitude value to the minimum longitude value, divided by the total longitude range.
- `ry = ymax - int((latact - ylatmin)/(ylatmax - ylatmin) * ymax)`: This calculates the *y* value in pixels, by taking the ratio of the range from the actual latitude value to the minimum latitude value, divided by the total latitude range.

- `t1 = rx, ry`: This creates a point value that holds the `x` and `y` values of your position.
- `br = rx + 5, ry + 5`: You'll want to draw a rectangle so you can see the created point; this draws a rectangle value that is 5x5 in size.
- `cv.Rectangle(image, t1, br, (0, 255, 0), 3)`: This adds the drawn rectangle to the map image.
- `cv.ShowImage("Map", image)`: This shows the map with the rectangle in a window.
- `if cv.WaitKey(2) == 27`: The `WaitKey` object displays the image, and also checks to see whether or not the `Esc` key has been pressed. If it has been pressed, it will execute the next statement.
- `break`: This closes the loop and the program.

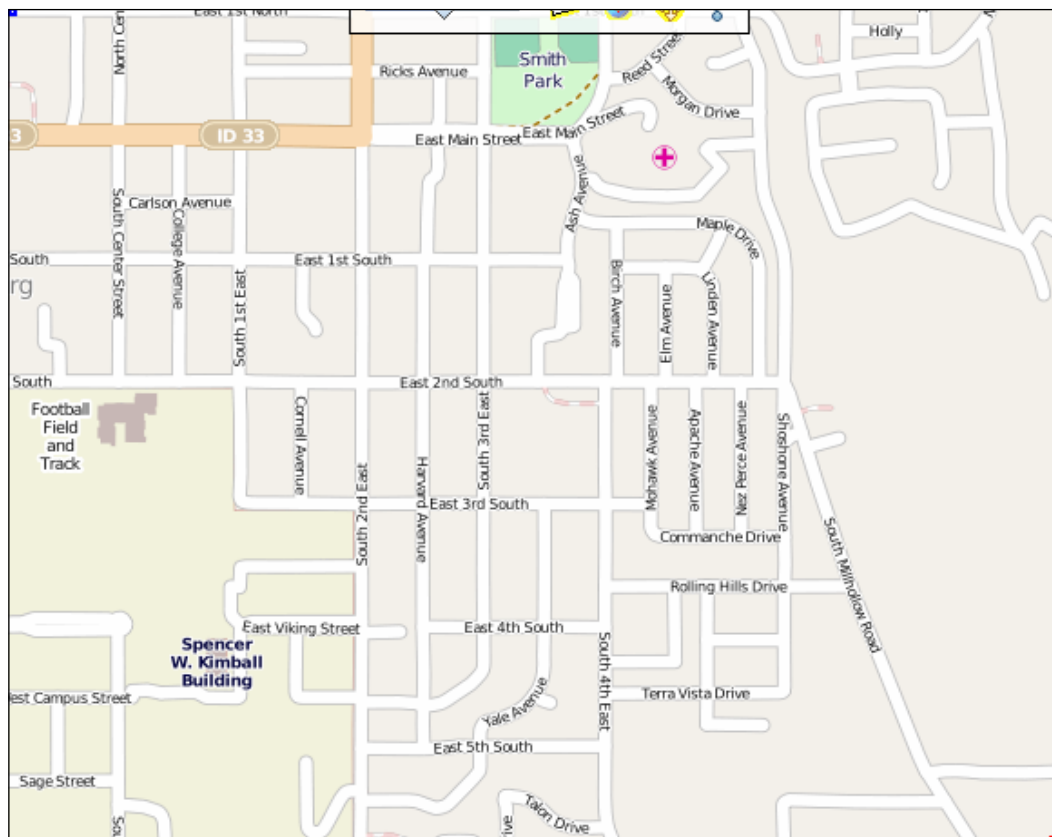
You'll also need to build a map to be displayed. You'll need to know the coordinates of the corners of your map. I used GpsPrune to build a map. I first removed the point that was plotted by opening the file. Then, I selected a point in the bottom-right corner. The following is a screenshot of the application:



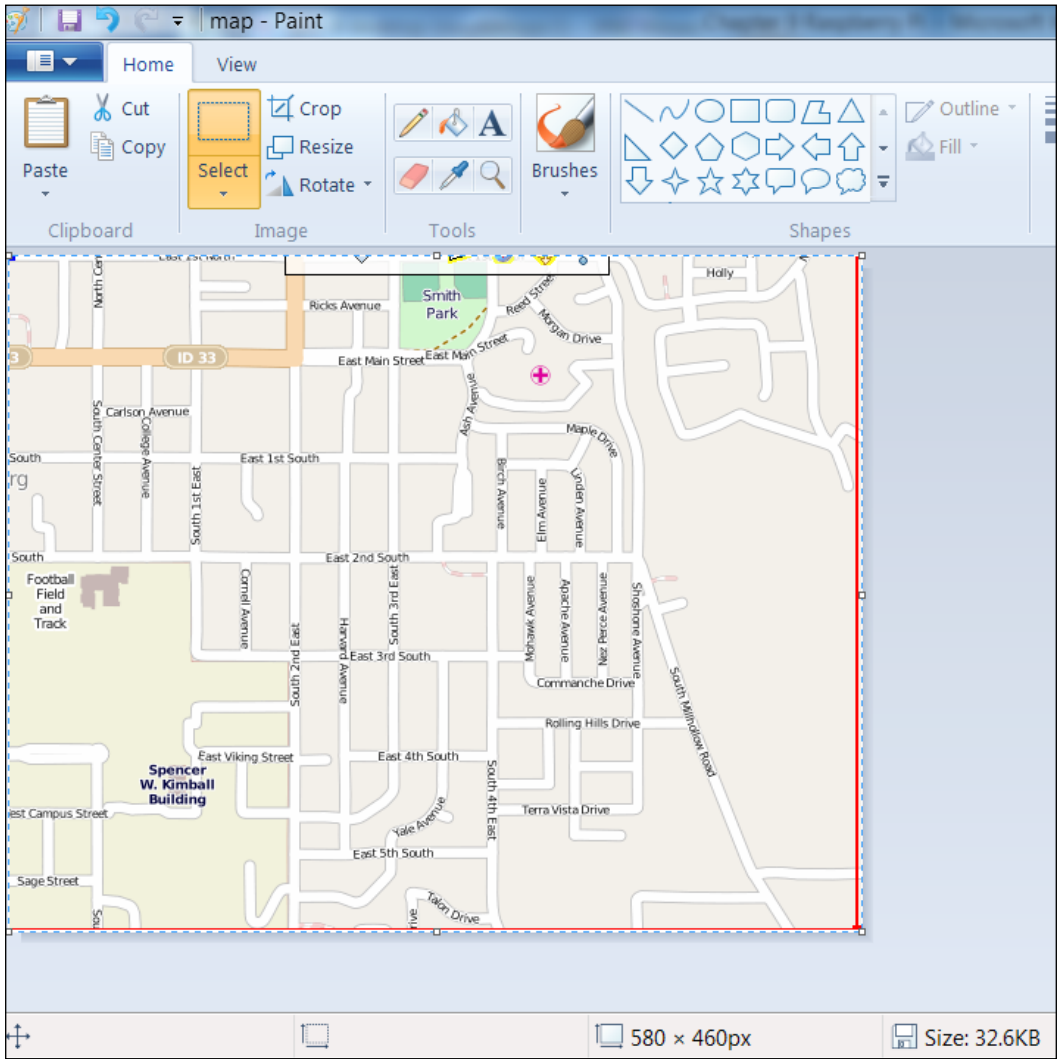
Make sure the **Coordinate format** dropdown is set to **Deg-min**. Note the **Latitude** and **Longitude** values in the upper-right corner, which in this case are **43°48.84551'** and **111°45.67532'** respectively. Now add another point in the upper-left corner, as shown in the following screenshot:



In this case, the **Latitude** value is **43°49.68896'** and the **Longitude** value is **111°47.18679'**. These are the corners of your map. Now take a screenshot of the map and use a program to crop the map at these two corners, as shown in the following screenshot:



For this example, I was using the **VNC Viewer** application on my PC to run GpsPrune on Raspberry Pi. This made it easy to take a screenshot by pressing *Ctrl + PrtScr* and then import the image to the Paint application. I then cropped the image and saved it as `map.png`. The following screenshot shows the view of the image as seen in the Paint application:



Summary

Congratulations! Your robot can now move around without getting lost. This capability is more useful for robots that can go far from home. You can use the information to identify routes to different waypoints and track where your robot has been.

In the next chapter, we'll cover how to bring all the various functionalities you have been working on together in a single, integrated system.

10

System Dynamics

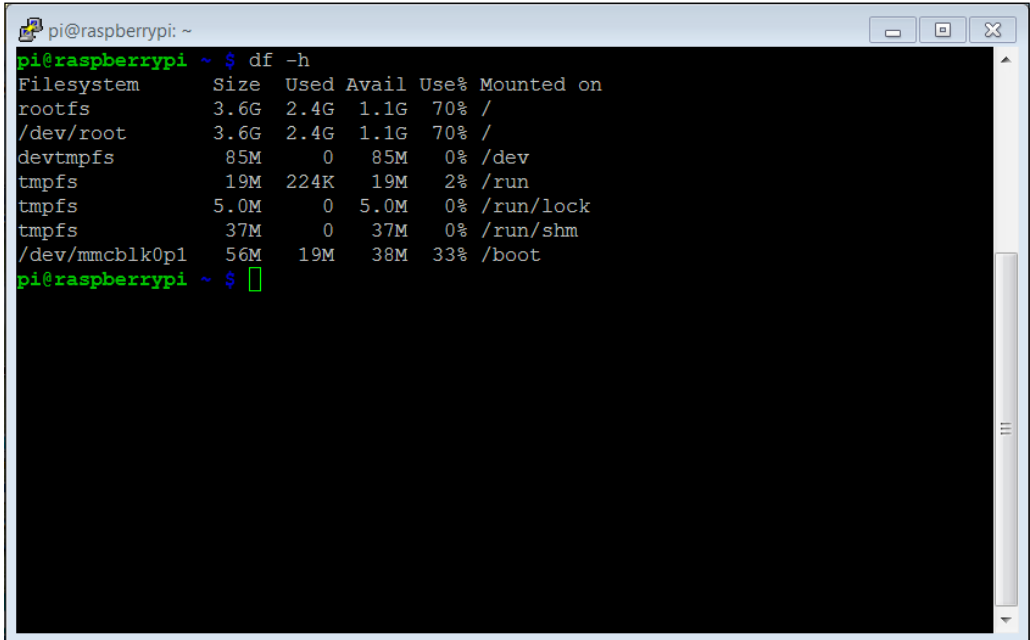
In the previous chapters, you've spent time learning a lot about individual functionalities that you can add to your robotic projects. In this chapter, you'll learn how to integrate these different parts into a single system.

You've spent a large amount of time on individual functionalities and your robotic projects now have many different capabilities that you can add to them. This chapter will bring all these parts together into a framework that allows the different parts to work together. You don't want the robot to just walk, talk, or see. You want it to perform all of these actions in a coordinated package. In this chapter, you'll learn how to programmatically connect all these individual capabilities and make your projects seem intelligent.

In this chapter, we will:

- Create a general control structure so that different capabilities can work together through system calls
- Introduce the **Robot Operating System (ROS)** as a supported framework for robotic capabilities

You're finally done with purchasing hardware! In this chapter, you'll add functionality through software. You'll need ample storage space for an array of the new software. First, let's check how much space you have in your memory card. You can also use the `df -h` command to see this information. You should see something like the following screenshot when you type the `df -h` command:



```
pi@raspberrypi: ~  
pi@raspberrypi ~ $ df -h  
Filesystem      Size  Used Avail Use% Mounted on  
rootfs          3.6G  2.4G  1.1G   70% /  
/dev/root       3.6G  2.4G  1.1G   70% /  
devtmpfs        85M    0   85M    0% /dev  
tmpfs           19M  224K   19M    2% /run  
tmpfs           5.0M    0   5.0M    0% /run/lock  
tmpfs           37M    0   37M    0% /run/shm  
/dev/mmcblk0p1  56M   19M   38M   33% /boot  
pi@raspberrypi ~ $
```

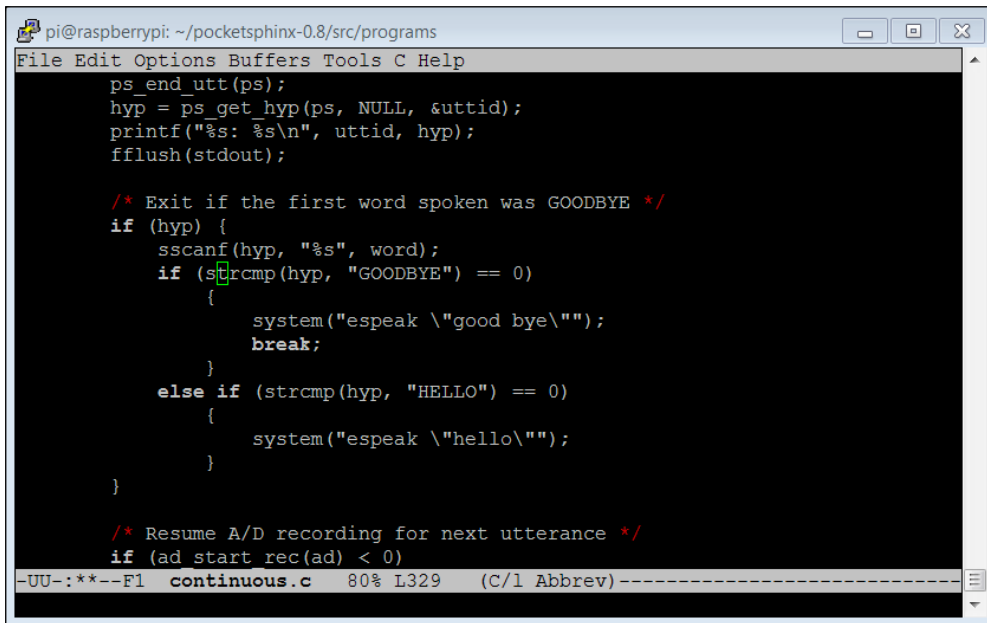
In this case, I have 1.1 GB, which should be enough to add the capability I need. You'll want at least 1 GB to make sure you don't run out of space when dealing with images.

Creating a general control structure

Now that you have a mobile robot, you will want to coordinate all of its different abilities. Let's start with the simplest approach. We will use a single control program that can call other programs and enable all the capabilities.

You've already done this once in *Chapter 3, Providing Speech Input and Output*.

Here, you edit the `continuous.c` code to allow it to call other programs to execute functionality you'll want your project to execute, for example movement, or speech response. Here is a screenshot of the code that we used from the `/home/pi/pocketsphinx-0.8/programs/src/` directory:



```

pi@raspberrypi: ~/pocketsphinx-0.8/src/programs
File Edit Options Buffers Tools C Help
ps_end_utt(ps);
hyp = ps_get_hyp(ps, NULL, &uttid);
printf("%s: %s\n", uttid, hyp);
fflush(stdout);

/* Exit if the first word spoken was GOODBYE */
if (hyp) {
    sscanf(hyp, "%s", word);
    if (strcmp(hyp, "GOODBYE") == 0)
    {
        system("espeak \"good bye\"");
        break;
    }
    else if (strcmp(hyp, "HELLO") == 0)
    {
        system("espeak \"hello\"");
    }
}

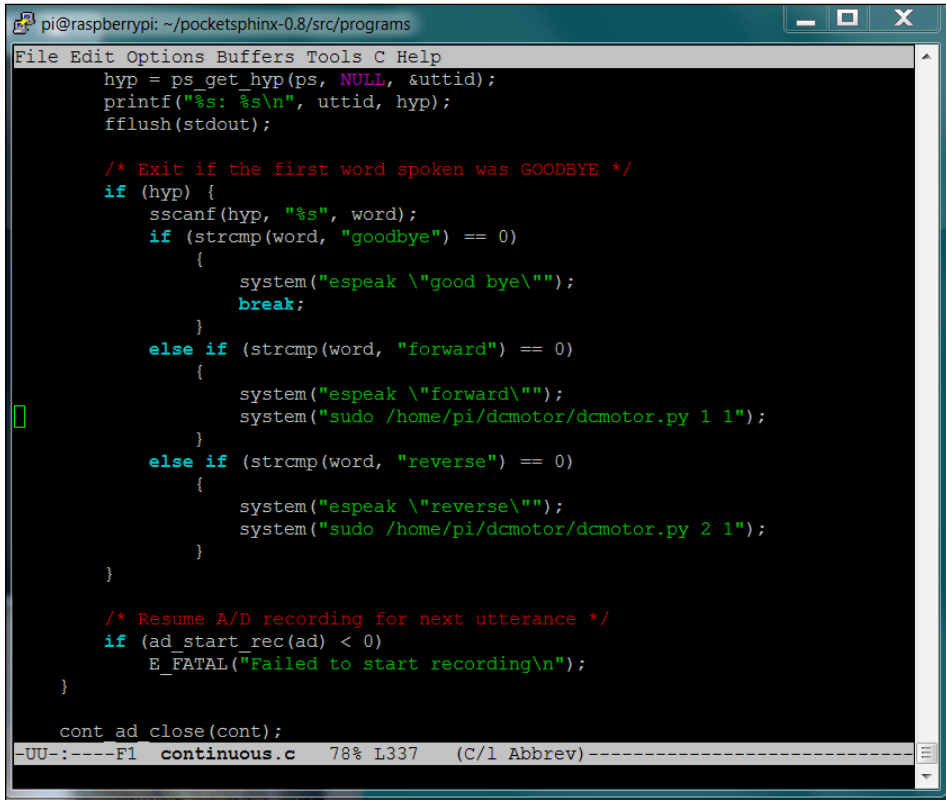
/* Resume A/D recording for next utterance */
if (ad_start_rec(ad) < 0)

```

UU-:***-F1 continuous.c 80% L329 (C/l Abbrev)

The functionality that is important to us is the `system("espeak \"good bye\")`; line of code. When you use the `system` function call, the program actually calls a different program, in this case the `espeak` program. The functionality passes the "good bye" parameter to the program so that the words good and bye come out of the speaker.

The following screenshot is another example from *Chapter 5, Creating Mobile Robots on Wheels*, where you wanted to command your robot to move:



```

pi@raspberrypi: ~/pocketsphinx-0.8/src/programs
File Edit Options Buffers Tools C Help
hyp = ps_get_hyp(ps, NULL, &uttid);
printf("%s: %s\n", uttid, hyp);
fflush(stdout);

/* Exit if the first word spoken was GOODBYE */
if (hyp) {
    sscanf(hyp, "%s", word);
    if (strcmp(word, "goodbye") == 0)
    {
        system("espeak \"good bye\"");
        break;
    }
    else if (strcmp(word, "forward") == 0)
    {
        system("espeak \"forward\"");
        system("sudo /home/pi/dcmotor/dcmotor.py 1 1");
    }
    else if (strcmp(word, "reverse") == 0)
    {
        system("espeak \"reverse\"");
        system("sudo /home/pi/dcmotor/dcmotor.py 2 1");
    }
}

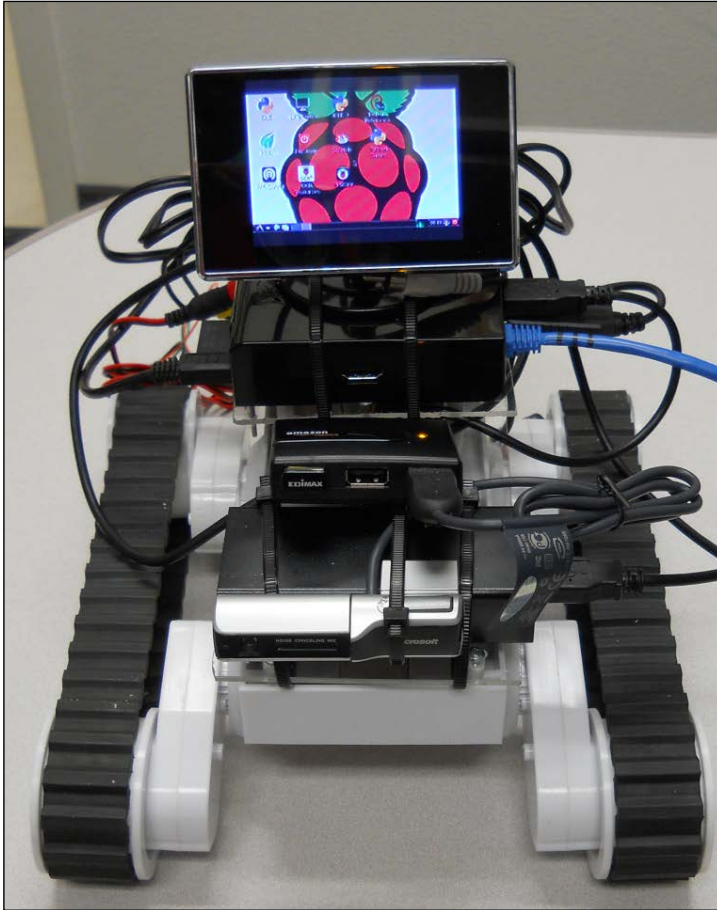
/* Resume A/D recording for next utterance */
if (ad_start_rec(ad) < 0)
    E_FATAL("Failed to start recording\n");
}

cont_ad close(cont);
-UU-:----F1 continuous.c 78% L337 (C/l Abbrev)-----

```

In this case, if you say *forward* to your robot, it will execute two programs. The first program you call is the *espeak* program with the "moving robot" parameter. These words should then come out of the speaker on the robot. The second program is the *dcmotor.py 1 1* program. The, *espeak* program will interpret this program through the system arguments so that the robot moves forward for one second.

I will now include an example in Python; it is my preferred language. I will use my tracked robot, which is shown in the following image:



This robot has a camera and is also able to communicate through a speaker. You can also control it via a wireless keyboard. You will now add the functionality to follow a colored ball. The functionality should also turn as the ball goes right or left and tell you when it is turning.

You also need to make sure that all of your devices are available to your programs. For this, you need to make sure that your USB camera as well as the two DC motor controllers are connected. To connect the camera, follow the steps mentioned in *Chapter 4, Adding Vision to Raspberry Pi*, in the *Connecting the USB camera to Raspberry Pi and viewing the images* section. It works best to connect the USB camera first, before connecting any other USB devices.

When the camera is up and running, you'll want to connect and check the DC motor controllers, as described in *Chapter 5, Creating Mobile Robots on Wheels*. You may want to run the `dcmotor.py` program just to make sure you are connected and both the motors work.

In this project, you will involve three different programs. First, you will create a program that will find out whether the ball is on the right-hand side or the left-hand side. This will be your main control program. You will also create a program that moves your robot approximately 45-degrees to the right and another program that moves it 45-degrees to the left. You will keep these programs very simple and you may just want to put them all in the same source file. However, as the complexity of each of these programs grows, it will make more sense for them to be separate. So, this is a good starting point for your robotic code. Also, if you want to use the code in another project or want to share it, this sort of separation helps.

You will create three programs for this project. In order to keep this organized, I created a new directory in my home directory by typing `mkdir robot` in my home directory. I will now put all my files in this directory.

The next step is to create two files that can move your robot, one file to move it to the left and the other to move it to the right. For this, you will have to create two copies of the `dcmotor.py` code, as you did in *Chapter 5, Creating Mobile Robots on Wheels*, in your `robot` directory. If you have created that file in your home directory, type `cp dcmotor.py ./robot/move_left.py` `cp dcmotor.py ./robot/move_right.py`. Now, you'll edit the files, changing two numbers in the program. Edit the code shown in the following screenshot in the `move_left.py` file:

```

pi@raspberrypi: ~/robot
File Edit Options Buffers Tools Python Help
#!/usr/bin/python
import serial
import time

def setSpeed(ser, motor, direction, speed):
    if motor == 0 and direction == 0:
        sendByte = chr(0xC2)
    if motor == 1 and direction == 0:
        sendByte = chr(0xCA)
    if motor == 0 and direction == 1:
        sendByte = chr(0xC1)
    if motor == 1 and direction == 1:
        sendByte = chr(0xC9)
    ser.write(sendByte)
    ser.write(chr(speed))

ser = serial.Serial('/dev/ttyUSB0', 19200, timeout = 1)
setSpeed(ser, 0, 0, 100)
setSpeed(ser, 1, 0, 100)
time.sleep(.5)
setSpeed(ser, 0, 0, 0)
setSpeed(ser, 1, 0, 0)
ser.close()

-UU-:----F1 move_left.py All L1 (Python)-----
For information about GNU Emacs and the GNU system, type C-h C-a.

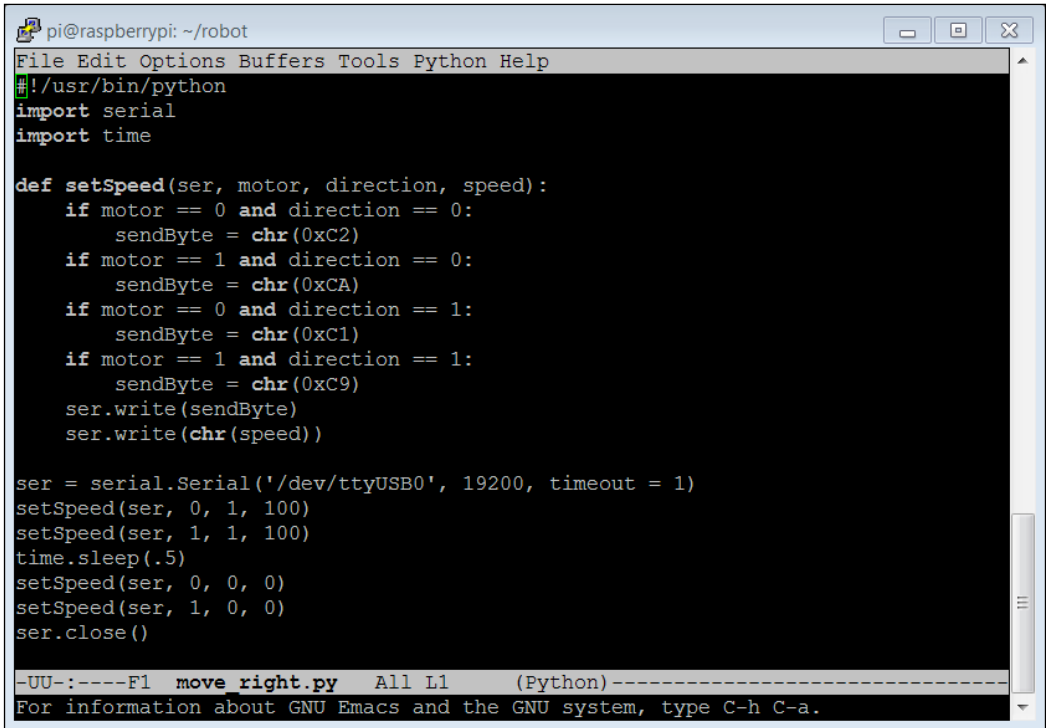
```

The following are the details for this code:

- `#!/usr/bin/python`: This statement sets the program so that it can be run directly from the program line.
- `import serial`: This statement imports the `serial` library so that you can talk to the motor controller.
- `import time`: This statement imports the `time` library so that you can use the `time.sleep()` function to add a fixed delay.
- `def setSpeed(ser, motor, direction, speed):`: This statement has the `setSpeed` function you will call in your program. This function sets the speed and direction for a given motor.
- `if motor == 0 and direction == 0:` This statement sets motor 1 in the forward direction.

- `sendByte = chr(0xC2):` This statement is the actual byte command to the motor controller.
- `if motor == 1 and direction == 0::` This statement sets motor 2 to go in a backward direction.
- `sendByte = chr(0xCA):` This statement is the actual byte command to the motor controller.
- `if motor == 0 and direction == 1::` This statement sets motor 1 to go in a backward direction.
- `sendByte = chr(0xC1):` This statement is the actual byte command for the motor controller.
- `if motor == 1 and direction == 1::` This statement sets motor 2 to go in a forward direction.
- `sendByte = chr(0xC9):` This statement is the actual byte command for the motor controller.
- `ser.write(sendByte):` This statement sends the byte command out of the serial port.
- `ser.write(chr(speed)):` This statement sends the speed byte out of the serial port.
- `ser = serial.Serial('/dev/ttyUSB0', 19200, timeout = 1):` This statement initializes and opens the serial port.
- `setSpeed(ser, 0, 0, 100):` This statement sends the forward command to motor 1 at a speed of 100 units.
- `setSpeed(ser, 1, 0, 100):` This statement sends the reverse command to motor 2 at a speed of 100 units.
- `time.sleep(.5):` This statement causes the motor to wait for 0.5 seconds.
- `setSpeed(ser, 0, 0, 0):` This statement sets the speed of motor 1 to 0 units.
- `setSpeed(ser, 1, 0, 0):` This statement sets the speed of motor 2 to 0 units.
- `ser.close():` This statement closes the serial port.

Similarly, you will also need to edit `moveright.py`, as shown in the following screenshot:



```
pi@raspberrypi: ~/robot
File Edit Options Buffers Tools Python Help
#!/usr/bin/python
import serial
import time

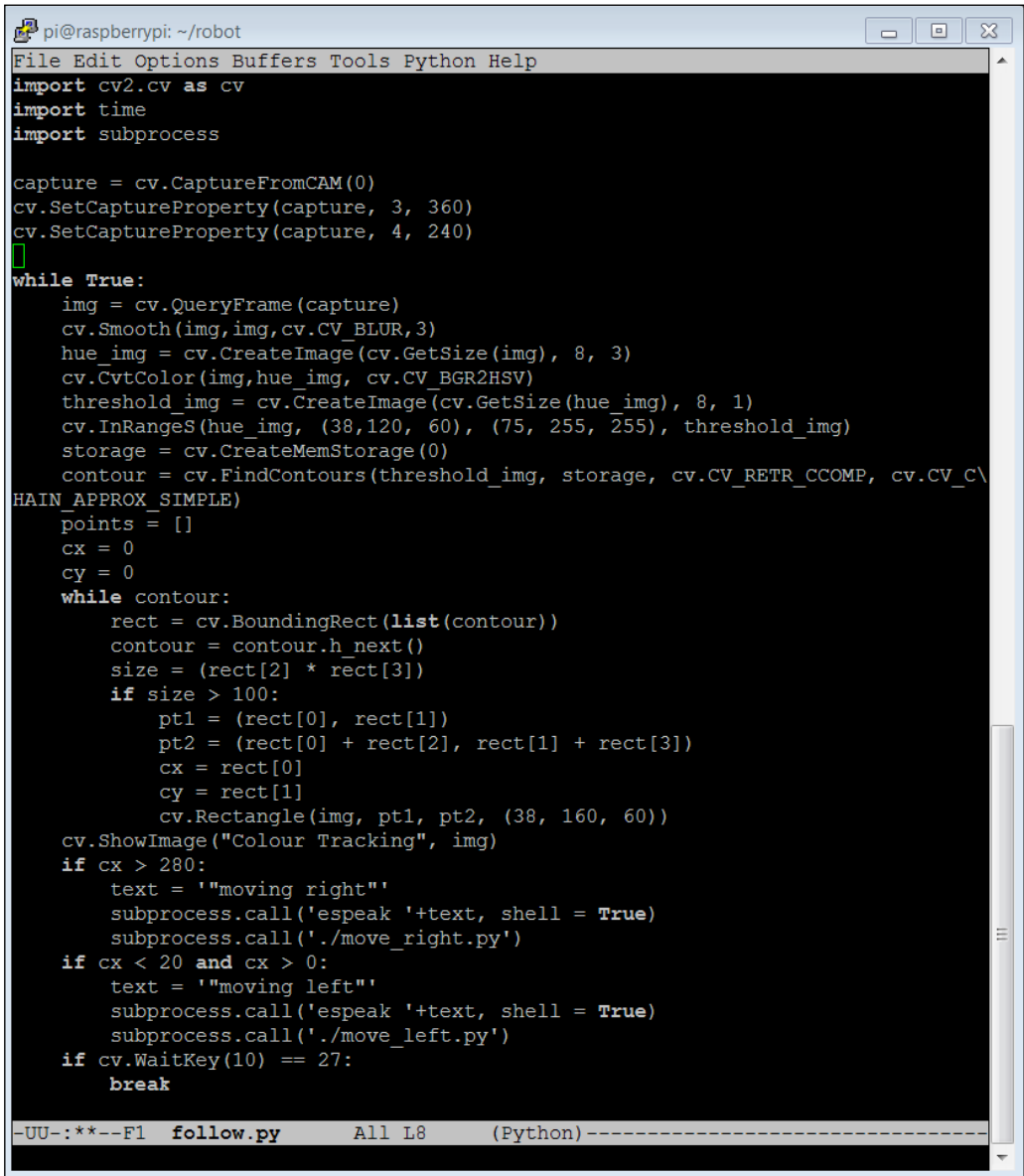
def setSpeed(ser, motor, direction, speed):
    if motor == 0 and direction == 0:
        sendByte = chr(0xC2)
    if motor == 1 and direction == 0:
        sendByte = chr(0xCA)
    if motor == 0 and direction == 1:
        sendByte = chr(0xC1)
    if motor == 1 and direction == 1:
        sendByte = chr(0xC9)
    ser.write(sendByte)
    ser.write(chr(speed))

ser = serial.Serial('/dev/ttyUSB0', 19200, timeout = 1)
setSpeed(ser, 0, 1, 100)
setSpeed(ser, 1, 1, 100)
time.sleep(.5)
setSpeed(ser, 0, 0, 0)
setSpeed(ser, 1, 0, 0)
ser.close()

-UU-:----F1 move_right.py All L1 (Python)-----
For information about GNU Emacs and the GNU system, type C-h C-a.
```

This time, the `setSpeed` numbers are changed to run the motors in the opposite direction, turning the robot to the right.

The final step is to create the main control program. You will start with the `camera.py` program you edited in *Chapter 4, Adding Vision to Raspberry Pi*. Execute this by typing `cp /home/pi/example/python/camera.py ./follow.py` while you are in your robot directory. Open this file with your editor. If you are using emacs, type `emacs follow.py` and then edit the code, as shown in the following screenshot:



```

pi@raspberrypi: ~/robot
File Edit Options Buffers Tools Python Help
import cv2.cv as cv
import time
import subprocess

capture = cv.CaptureFromCAM(0)
cv.SetCaptureProperty(capture, 3, 360)
cv.SetCaptureProperty(capture, 4, 240)
while True:
    img = cv.QueryFrame(capture)
    cv.Smooth(img, img, cv.CV_BLUR, 3)
    hue_img = cv.CreateImage(cv.GetSize(img), 8, 3)
    cv.CvtColor(img, hue_img, cv.CV_BGR2HSV)
    threshold_img = cv.CreateImage(cv.GetSize(hue_img), 8, 1)
    cv.InRangeS(hue_img, (38, 120, 60), (75, 255, 255), threshold_img)
    storage = cv.CreateMemStorage(0)
    contour = cv.FindContours(threshold_img, storage, cv.CV_RETR_CCOMP, cv.CV_C\
CHAIN_APPROX_SIMPLE)
    points = []
    cx = 0
    cy = 0
    while contour:
        rect = cv.BoundingRect(list(contour))
        contour = contour.h_next()
        size = (rect[2] * rect[3])
        if size > 100:
            pt1 = (rect[0], rect[1])
            pt2 = (rect[0] + rect[2], rect[1] + rect[3])
            cx = rect[0]
            cy = rect[1]
            cv.Rectangle(img, pt1, pt2, (38, 160, 60))
    cv.ShowImage("Colour Tracking", img)
    if cx > 280:
        text = "moving right"
        subprocess.call('espeak '+text, shell = True)
        subprocess.call('./move_right.py')
    if cx < 20 and cx > 0:
        text = "moving left"
        subprocess.call('espeak '+text, shell = True)
        subprocess.call('./move_left.py')
    if cv.WaitKey(10) == 27:
        break
-UU-:***-F1 follow.py All L8 (Python)-----

```

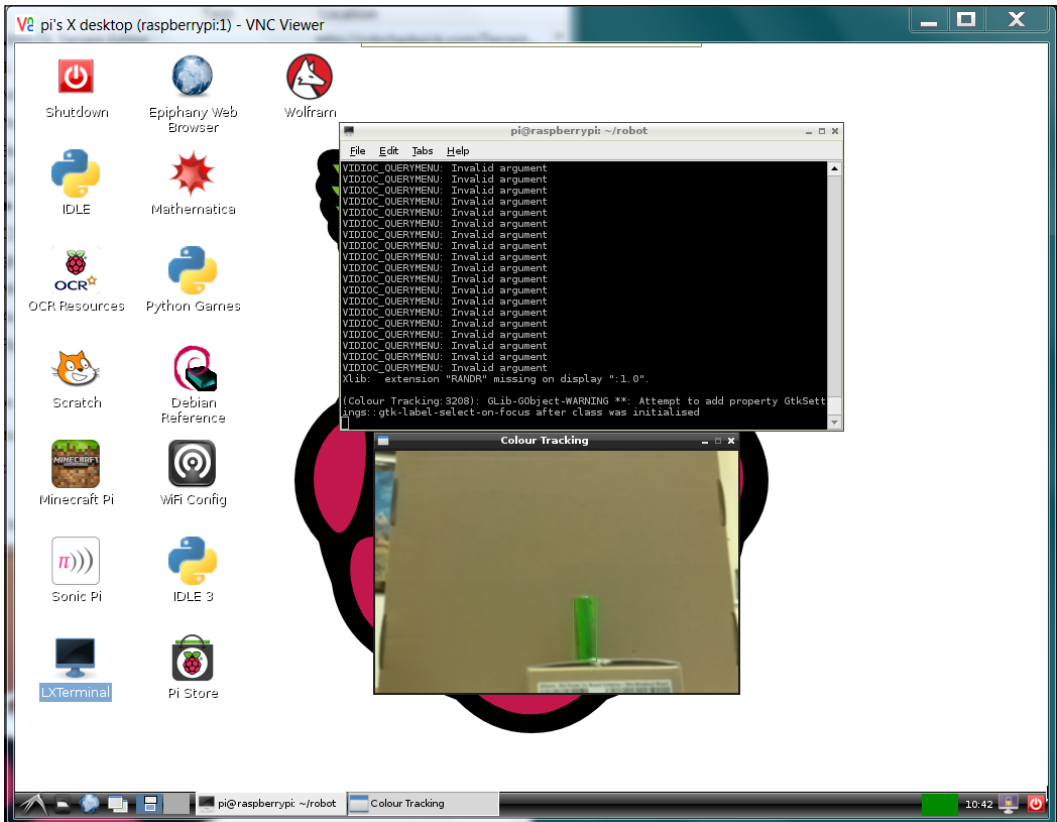
Let's look at the following code statements along with their functions:

- `import cv2.cv as cv`: This statement imports the `cv` library.
- `import time`: This statement imports the `time` library.
- `import subprocess`: This statement imports the `subprocess` library. This will allow you to call other programs within your Python program.
- `capture = cv.CaptureFromCAM(0)`: This statement sets up the program to capture your images from the webcam.
- `cv.SetCaptureProperty(capture, 3, 360)`: This statement sets the `x` resolution of the image to 360.
- `cv.SetCaptureProperty(capture, 4, 240)`: This statement sets the `y` resolution of the image to 240.
- `while True::` This statement executes the loop over and over until the `Esc` key is pressed.
- `img = cv.QueryFrame(capture)`: This statement brings in the image.
- `cv.Smooth(img, img, cv.CV_BLUR, 3)`: This statement smoothes the image.
- `hue_img = cv.CreateImage(cv.GetSize(img), 8, 3)`: This statement creates a new image that will hold the hue-based image.
- `cv.CvtColor(img, hue_img, cv.CV_BGR2HSV)`: This statement moves a copy of the image using the hue values in `hue_img`.
- `threshold_img = cv.CreateImage(cv.GetSize(hue_img), 8, 1)`: This statement creates a new image that will hold all the blobs of colors.
- `cv.InRangeS(hue_img, (38, 120, 60), (75, 255, 255), threshold_img)`: This statement now fills in `hue_img` from `threshold_img`.
- `storage = cv.CreateMemStorage(0)`: This line creates some memory for you to manipulate the images.
- `contour = cv.FindContours(threshold_img, storage, cv.CV_RETR_CCOMP, cv.CV_CHAIN_APPROX_SIMPLE)`: This statement finds all the areas on your image that are within the threshold. There may be more than one, so you want to capture them all.
- `points = []`: This statement creates an array to hold all the different possible points of color.
- `cx = 0`: This statement gives a variable to hold the `x` location of color.
- `cy = 0`: This statement gives a variable to hold the `y` location of color.

- `while contour::` This statement now adds a `while` loop that will let you step through all the possible contours. By the way, it is important to note that if there is another larger green blob in the background, you would find that `x` and `y` location. Just to keep this simple, we'll assume that your green ball is unique.
- `rect = cv.BoundingRect(list(contour)):` This statement creates a bounding rectangle for each area of `color`. The rectangle is defined by its corners around the blob of `color`.
- `contour = contour.h_next():` This statement will prepare you for the next `contour` statement (if one exists).
- `size = (rect[2] * rect[3]):` This statement calculates the diagonal length of the rectangle you are evaluating.
- `if size > 100::` This checks to see whether the area is big enough for our purpose.
- `pt1 = (rect[0], rect[1]):` This statement defines a `pt` variable at the `x` and `y` coordinates on the left-hand side of the blob's rectangular location.
- `pt2 = (rect[0] + rect[2], rect[1] + rect[3]):` This statement defines a `pt` variable at the `x` and `y` coordinates on the right-hand side of the blob's rectangular location.
- `cx = rect[0]:` This statement sets the value of `cx` to the `x` location of `color`.
- `cy = rect[1]:` This statement sets the value of `cy` to the `y` location of the `color`.
- `cv.Rectangle(img, pt1, pt2, (38, 160, 60)):` Here, you add a rectangle to your original image, identifying where you think it is located.
- `cv.ShowImage("Colour Tracking", img):` This statement displays the image on the screen.
- `if cx > 280::` This statement checks to see whether the object is too far to the right.
- `text = "moving right":` This statement gets you ready to call the user.
- `subprocess.call('espeak '+text, shell = True):` This statement calls `espeak` with a message.
- `subprocess.call('./move_right.py'):` This statement calls the Python program, which will move the unit to the right-hand side.
- `if cx < 20 and cx > 0::` This statement checks to see whether the object is too far to the left. Make sure you exclude 0, which would be the case initially if there is no object.

- `text = "moving left":` This statement gets you ready to call the user.
- `subprocess.call('espeak '+text, shell = True):` This statement calls `espeak` with a message.
- `subprocess.call('./move_left.py'):` This statement calls the Python program, which will move the unit to the left.
- `if cv.WaitKey(10) == 27:` This statement kills the program if you press the *Esc* key.
- `break:` This statement stops the program.

Now, you can run the program by typing `python ./follow.py`. You can also type `chmod +x follow.py` and then run the program by typing `./follow.py`. The window should be displayed as shown in the following screenshot:



The green rectangle indicates that the program is following the color green. As the green color is moved towards the edge on the left-hand side, the robot should rotate slightly to the left. As the green color is moved towards the edge on the right-hand side, the robot should rotate slightly towards the right.

With OpenCV, it is also possible to perform motion detection. There are a couple of good tutorials on how to do this with OpenCV. One simple example is at <http://www.steinm.com/blog/motion-detection-webcam-python-opencv-differential-images/>. Another example, a bit more complex but more elegant, can be found at <http://stackoverflow.com/questions/3374828/how-do-i-track-motion-using-opencv-in-python>.

While using motion detection, if you put your wind-up walker toy in front of the camera, you would see the output on the webcam (using the code from the second tutorial) as follows:



You can then use the x and y location data to move the robot by following the motion.

Using the structure of the Robot Operating System to enable complex functionalities

As you can see, communicating between different aspects of our project can be challenging. In this section, I will introduce you to a special operating system that is designed specifically for use with robotic projects, the **Robot Operating System (ROS)**. This operating system works on top of Linux and provides some interesting functionality.

The operating system is available at www.ros.org. However, the most useful link to the Wiki for the ROS is wiki.ros.org. If you visit this link, you will find a complete set of documentation and downloads. There are also a number of resources that could be useful if you'd like to learn more about the ROS in depth. One of the better resources is the book *Learning ROS for Robotics Programming*, Martinez and Fernandez, Packt Publishing.

This section will not cover the ROS in detail but will introduce you to some of the basics. Start by going to the www.ros.org website. If you select **Install**, you'll note that there is a wide range of Linux operating systems and hardware support. You'll also note that there are several different versions or releases of the ROS. Some of the later releases are Hydro, Fuerte, and Groovy. One of the challenges of using an operating system like this is to decide which release to use. The most recent release will have the largest number of features. It may also have a significant number of issues that may cause problems.

I often prefer using a past release that has been used by a larger number of people; this way, I run into fewer problems. I also don't like to build large packages like this myself. It can take a great deal of time, and you'll often run into cryptic error messages that can take days to resolve. So, for this tutorial, I will use an older version that I have been successful with in the past. It is called Groovy, and while it is not the latest version, it is very stable and easy to use.



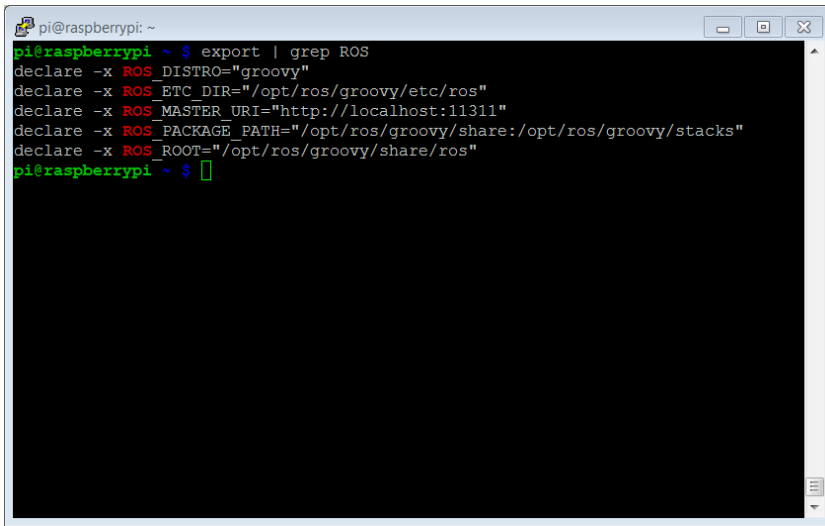
The ROS brings with it quite a bit of code. If you are going to work with the ROS, I would recommend that you have at least an 8 GB card installed on Raspberry Pi.

I normally follow the installation instructions at <http://wiki.ros.org/groovy/Installation/Raspbian>. The following are the instructions in a step-by-step form:

1. Add the repository to your apt sources. This is the place where your apt-get command appears when it is trying to find packages to install. Here is the command:

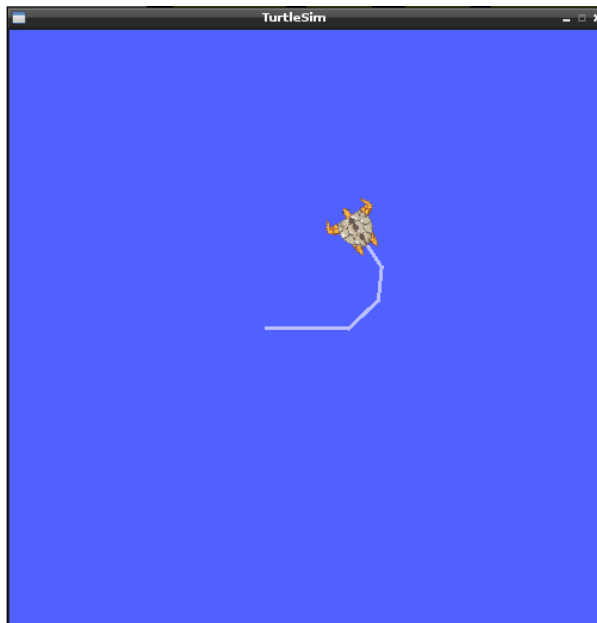
```
sudo sh -c 'echo "deb http://64.91.227.57/repos/rospbian wheezy main" > /etc/apt/sources.list.d/rospbian.list'
```
2. Next, add the following apt key:

```
wget http://64.91.227.57/repos/rospbian.key -O - | sudo apt-key add -
```
3. Now, reload the apt sources so that your Raspberry Pi will know where the files are, by typing `sudo apt-get update`.
4. Now, install the ROS packages. Installing `ros_comm` will install all the significant packages you'll need. You need to type `sudo apt-get install ros-groovy-ros-comm`. The package is quite large and will take some time.
5. Before you can use the ROS, you need to install and initialize `rosdep` to let you track dependencies and run some core features. Type `sudo rosdep init` and then `rosdep update`.
6. You also need to set up the ROS environment variables so that they are automatically added to your session every time you launch a terminal window. To perform this, type `echo "source /opt/ros/groovy/setup.bash" >> ~/.bashrc` and then `source ~/.bashrc`.
7. The, add one more tool, `python-rosinstall`. This tool can help in installing the ROS packages. To add this tool, type `sudo apt-get install python-rosinstall`.
8. To make sure the ROS is set up correctly, type `export | grep ROS`. You should see the following screenshot:

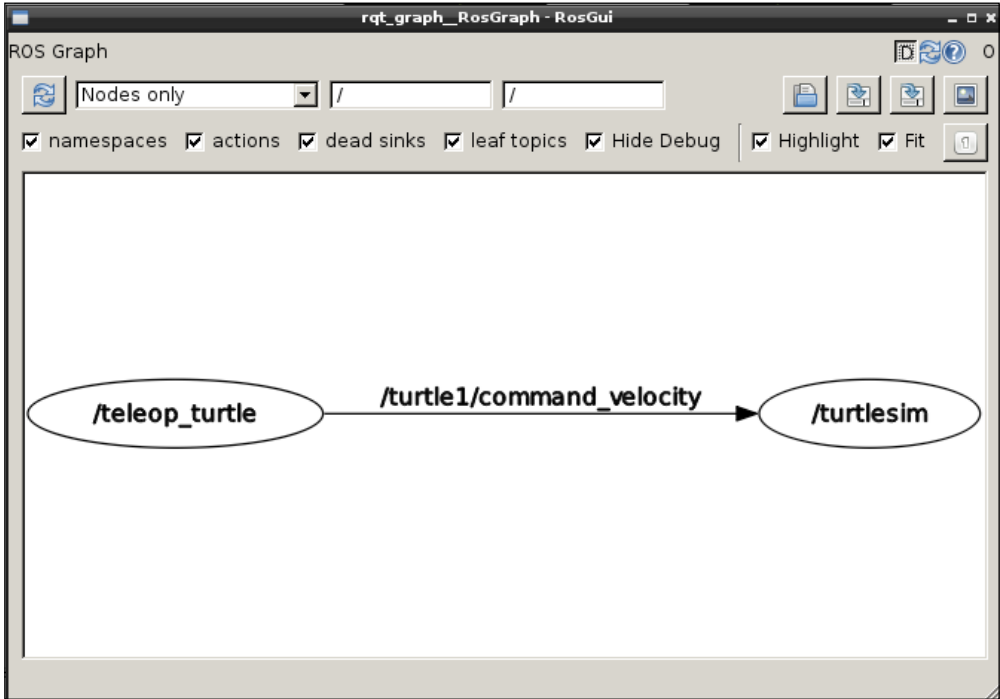


```
pi@raspberrypi: ~  
pi@raspberrypi ~$ export | grep ROS  
declare -x ROS_DISTRO="groovy"  
declare -x ROS_ETC_DIR="/opt/ros/groovy/etc/ros"  
declare -x ROS_MASTER_URI="http://localhost:11311"  
declare -x ROS_PACKAGE_PATH="/opt/ros/groovy/share:/opt/ros/groovy/stacks"  
declare -x ROS_ROOT="/opt/ros/groovy/share/ros"  
pi@raspberrypi ~$
```

9. Once installed, you should go through the tutorials at wiki.ros.org/ROS/Tutorials. They will introduce you to the features of the ROS and how to use it in your robotic projects. You will learn how it can provide a systematic way of configuring and communicating between multiple features that run in different programs. It even comes with some programs that implement some interesting vision and motor control capabilities. The following is the **TurtleSim** tutorial running on Raspberry Pi:



One of the really powerful features of the ROS is its ability to show you how your information is flowing between applications. As you follow the tutorial, you will end up with two running applications, `teleop_turtle` and `turtlesim`. In this example, you can use an application called `rqt_graph` to record the flow of information between the two applications, as shown in the output in the following screenshot:



It will be very difficult to illustrate all of the functionalities of the ROS here. It may take a bit of time, but you can learn to use the ROS to give your robot more functionality.

Summary

Now you can coordinate complex functionalities for your robot. Your robot can walk, talk, see, hear, and even sense its environment, all at the same time. In the next chapter, you'll learn how to construct robots that can fly, sail, and even go under water.

Fortunately, the ROS is free and open source. It has a very complex set of functionalities. However, if you spend some time learning it, you could start using some of the most comprehensive functionalities being developed in robotics research today.

11

By Land, Sea, and Air

You've built robots that can navigate on land. Now, let's look at some possibilities to utilize the tools you have used so far to build some robots that dazzle the imagination. By now, I hope you are comfortable accessing the USB control channels and talking to servo controllers and other devices that can communicate over USB. Instead of leading you through each step, in this chapter, I will point you in the right direction and then allow you to explore a bit. I'll try to give you some examples using some of the projects that are popular on the Internet.

You don't want to limit your robotic possibilities to just walking or rolling. You'll want your robot to fly, sail, or swim. In this chapter, you'll see how you can use the capabilities you have already mastered in projects that defy gravity, explore the open sea, or navigate below the sea.

In this chapter, we will:

- Use Raspberry Pi in robots that can sail
- Use Raspberry Pi in robots that can fly
- Use Raspberry Pi in robots that can go underwater

We need to add hardware to our robotics in order to complete these projects. Since the hardware is different for each of these projects, I'll introduce them in each individual section.

Using Raspberry Pi to sail

Now that you've created platforms that can move on land, let's turn to a completely different type of mobile platform, one that can sail. In this section, you'll discover how to use Raspberry Pi to control your sailboat.

Getting started

Fortunately, making a robot sail on water is as simple as making it walk on land. First, however, you need a sailing platform. The following is an image of an **Radio Controlled (RC)** sailing platform that can be modified to accept control from Raspberry Pi:



In fact, many RC-controlled boats can be modified to use Raspberry Pi. All you need is space to put the processor, the battery, and any additional control circuitry. In this case, the sailing platform has two controls, a rudder, which is controlled by a servo, and a second servo, which controls the position of the sail, as shown in the following image:



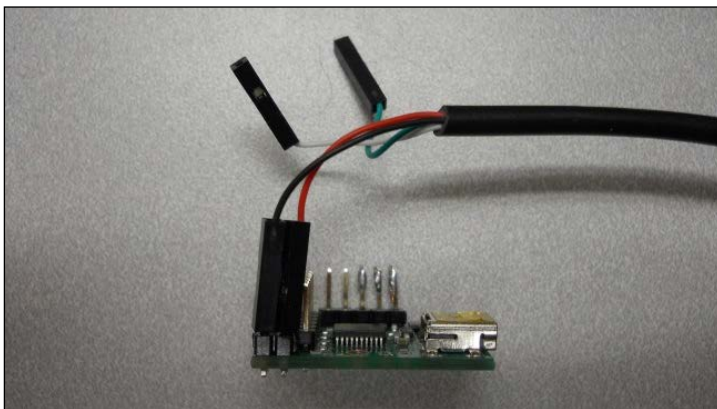
To automate control of the sailboat, you'll need Raspberry Pi, a battery, and a servo controller. The servo controller I would advise for this project is the one that you used in *Chapter 6, Controlling the Movement of a Robot with Legs*. It is a six-servo controller made by Pololu. It is available at www.pololu.com, and it looks as like the following image:



The advantage is that this servo controller is very small and fits in a limited space. The only challenge is getting a power connection for the device. Fortunately, there is a cable that you can purchase. This cable makes these power connections available from a standard cable. The cable you want is a USB-to-TTL serial/RS232 adapter cable. Make sure that the TTL end of the cable has individual female connectors. You can get this cable at www.amazon.com and also at www.adafruit.com. The following is an image of the cable:



The red and black wires are for connection to the power. These can be connected to the servo controller, as shown in the following image:



Once you have assembled your sailboat, you will first need to hook up the servo controller to the servos on the boat. You should try to control the servos before installing all the electronics inside the boat, as shown in the following image:

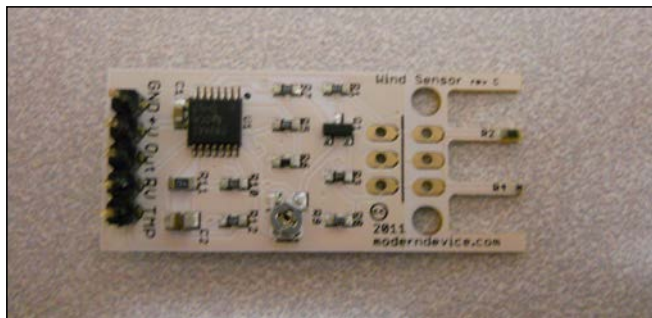


Just as in *Chapter 6, Controlling the Movement of a Robot with Legs*, you can use the Maestro Servo Controller software to control the servo controller from your PC. When you are ready to hook it up to Raspberry Pi, you can start with the same Python program you used in *Chapter 6, Controlling the Movement of a Robot with Legs*. You will probably want to control the system without a wired connection. So, you can use the principles that you learned in *Chapter 8, Going Truly Mobile – The Remote Control of Your Robot*.

It may be a bit challenging if you are using the standard 2.4 GHz keyboard or a smaller 2.4 GHz controller. One possible solution is a wireless LAN, where you can set up your own ad hoc wireless network using a router connected to a laptop. Also, as noted in *Chapter 8, Going Truly Mobile – The Remote Control of Your Robot*, many cell phones have the ability to set up a wireless hot spot. This hot spot can create a wireless network so that you can communicate remotely with your sailboat.

Another possible solution is to use ZigBee wireless devices to connect your sailboat to a computer. We already covered the details in *Chapter 8, Going Truly Mobile – The Remote Control of Your Robot*. You'll need two of these devices and a USB shield for each. You can get these at a number of places, including www.adafruit.com. If you can connect your computer and Raspberry Pi through this wireless network, the advantage is that it carries communications to and from your sailboat. It can also have a range of up to a mile using the right devices.

Now, you can sail your boat, controlling it through an external keyboard or a ZigBee wireless network from your computer. If you want to fully automate your system, you could add your GPS and then have your sailboat sail to each of the programmed positions. One additional item you may want to add to make the system fully automated is a wind sensor. The following is an image of a wind sensor that is be fairly inexpensive if you buy it from www.moderndevices.com:



You can mount it to the mast if you'd like. I used a small piece of heavy-duty tape and mounted it to the top of the mast, as shown in the following image:



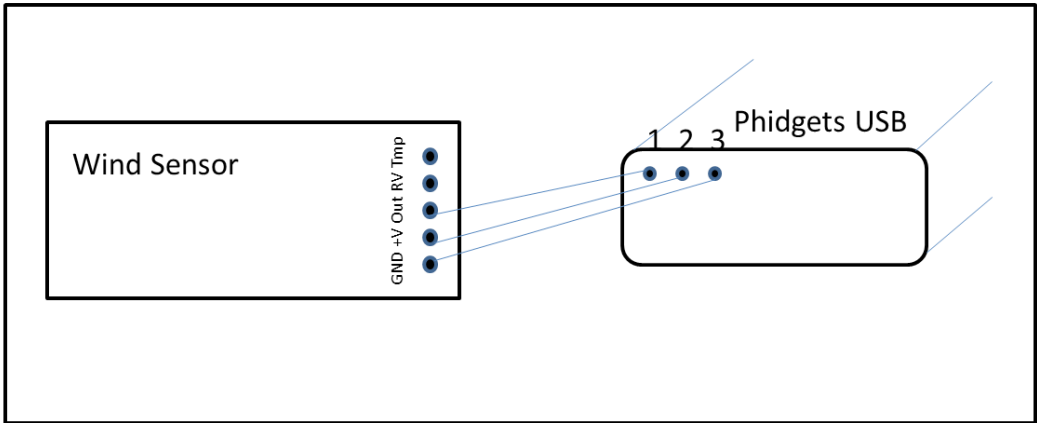
To add this to your system, you'll also need a way to take the analog input from the sensor and send it to Raspberry Pi. There are two ways to do this. One way is to add a USB device that samples the analog signal and reports the measurements over the USB bus. The device I like to use is the PhidgetInterfaceKit 2/2/2 from www.phidgets.com. The following is an image of this device:



The following is an image of the wind sensor connected to the converter:



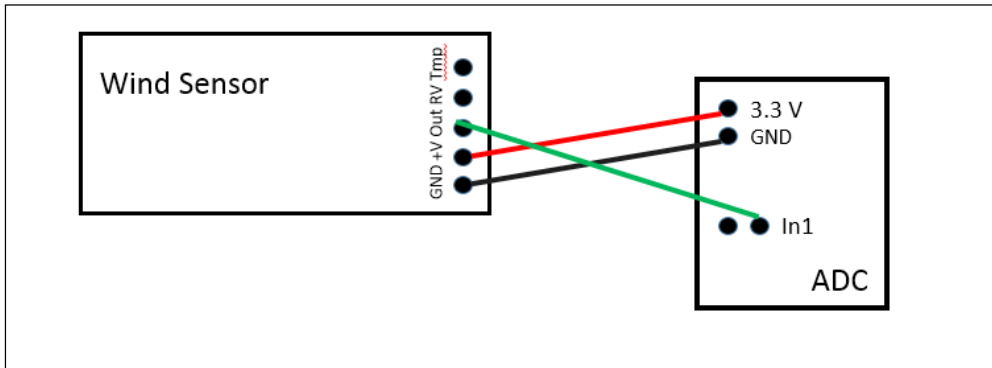
The following wiring diagram shows how the wind sensor interfaces with the Phidgets USB device:



The Phidgets website will lead you through the download process. I chose Python as my language and downloaded the appropriate libraries and sample code. When I run this sample code, I get the following output while blowing on the sensor:

```
ubuntu@ubuntu-armhf: ~/Python_2.1.8.20130926/Python
InterfaceKit 268671: Sensor 0: 495
InterfaceKit 268671: Sensor 0: 505
InterfaceKit 268671: Sensor 0: 515
InterfaceKit 268671: Sensor 0: 526
InterfaceKit 268671: Sensor 0: 536
InterfaceKit 268671: Sensor 0: 545
InterfaceKit 268671: Sensor 0: 555
InterfaceKit 268671: Sensor 0: 547
InterfaceKit 268671: Sensor 0: 537
InterfaceKit 268671: Sensor 0: 526
InterfaceKit 268671: Sensor 0: 516
InterfaceKit 268671: Sensor 0: 507
InterfaceKit 268671: Sensor 0: 497
InterfaceKit 268671: Sensor 0: 487
InterfaceKit 268671: Sensor 0: 478
InterfaceKit 268671: Sensor 0: 468
InterfaceKit 268671: Sensor 0: 458
InterfaceKit 268671: Sensor 0: 448
InterfaceKit 268671: Sensor 0: 438
InterfaceKit 268671: Sensor 0: 428
InterfaceKit 268671: Sensor 0: 418
InterfaceKit 268671: Sensor 0: 408
InterfaceKit 268671: Sensor 0: 399
```

Another way to get the analog signal into your Raspberry Pi is to use the ADC board, covered in *Chapter 7, Avoiding Obstacles Using Sensors*. Here, you can use this board to process the infrared sensor data. Here is the wiring diagram for this solution:



Now, you can access the wind speed from the USB connection in the same way that you received data from the other USB devices that you have already used. Now that you have a way to measure your location and the wind, you can use your Raspberry Pi to sail your boat. I am not a sailor myself, but you'll want to use the wind sensor to get a sense of the direction of the wind and then use classic sailing techniques such as tacking (moving back and forth to use the wind effectively) to sail.

While constructing this, you'll need to be careful with waterproofing, especially while sailing in heavy wind. Think about attaching a hatch that covers the electronics securely. I added small screws and tabs and also some waterproof sealant to hold the hatch.

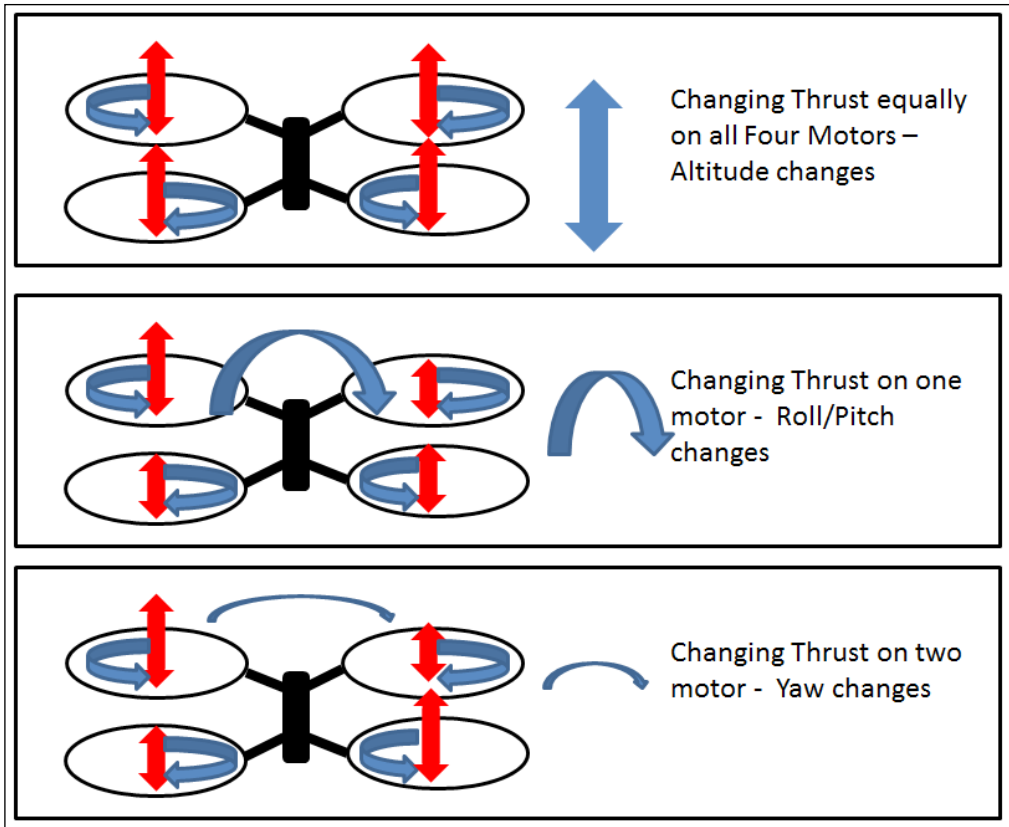
Using Raspberry Pi to fly robots

You've now built robots that can move around on a wheeled structure, robots that have legs, and robots that can sail. You can also build robots that can fly by relying on Raspberry Pi to control their flight. There are several possible ways to incorporate Raspberry Pi into a flying robotic project, but the most straightforward way is to add it to a quadcopter project.

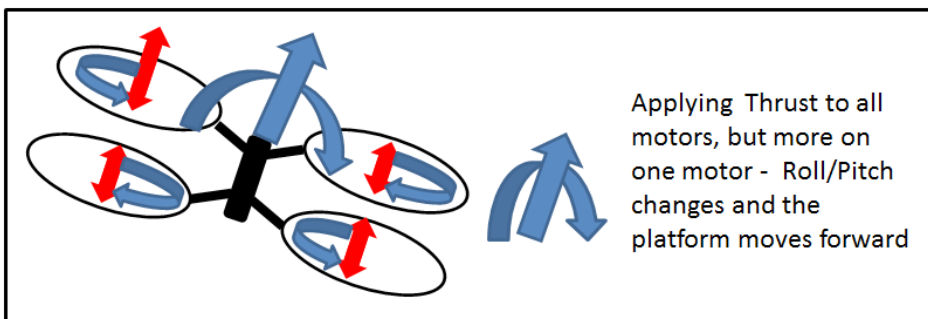
Quadcopters are a unique subset of flying platforms that have become very popular in the last few years. They are flying platforms that utilize the same vertical lift concept as helicopters. However, they employ not one, but four motor/propeller combinations to provide an enhanced level of stability. If you'd like to know more about how quadcopters work, see <http://quadcopter101.blogspot.com/2013/10/chapter-1-introduction-to-quadcopters.html>. The following is an image of such a platform:



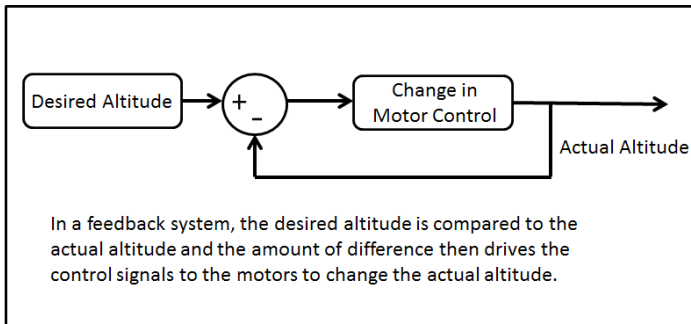
The quadcopter has two sets of counter-rotating propellers. This simply means that two of the propellers rotate one way, while the other two rotate the other way, to provide thrust in the same direction. This provides a platform that is inherently stable. Controlling the thrust on all four motors allows you to change the pitch, roll, and yaw of the device. The following figure may be helpful to understand this concept:



As you can see, controlling the relative speed of the four motors allows you to control the various ways in which the device can change position. To move forward, or in any direction really, we can combine a change in the roll/pitch with a change in the thrust. We do this so that instead of going up, the device will move forward, as shown in the following figure:



In a perfect world, knowing the components you used to build your quadcopter, you might know exactly how much control signal to apply to get a certain change in the roll/pitch/yaw or altitude of your quadcopter. You cannot rely on a fixed set of signals, as there are simply too many aspects of your device that can vary. Instead, this platform uses a series of measurements for its position, pitch/roll/yaw, and altitude. Then, it adjusts the control signals to the motors to achieve the desired result. We call this feedback control. The following figure denotes a feedback system:



As you can see, if your quadcopter is too low, the difference between the desired altitude and the actual altitude will be positive. The motor control will increase the voltage supplied to the motors, increasing the altitude. If the quadcopter is too high, the difference between the desired altitude and the actual altitude will be negative. The motor control will decrease the voltage supplied to the motors, decreasing the altitude. If the desired altitude and the actual altitude are equal, then the difference between the two will be zero, and the motor control will be held at its current value. Thus, the system stabilizes even if the components aren't perfect or if a wind comes along and blows the quadcopter up or down.

One function of Raspberry Pi in this type of robotic project is to actually coordinate the measurement and control of the quadcopter's pitch, roll, yaw, and altitude. This can be done. However, it is a very complex task, and the details of its implementation are beyond the scope of this book. It is unclear whether Raspberry Pi has the horsepower to execute and keep up with this type of application.

However, Raspberry Pi can still be utilized in this type of robotic project by introducing another embedded processor to carry out the low-level control and using Raspberry Pi to manage high-level tasks, such as using the vision system of Raspberry Pi to identify a colored ball and then guiding the platform toward it. Alternatively, as in the sailboat example, you can use Raspberry Pi to coordinate GPS tracking and long-range communications through ZigBee or a wireless LAN. I'll cover an example of this in this section.

The first thing you'll need is a quadcopter. There are three approaches to this, which are as follows:

- Purchase an already assembled quadcopter
- Purchase a kit and construct it yourself
- Buy the parts separately and construct the quadcopter

In any case, one of the easiest ways to add Raspberry Pi to your quadcopter is to choose one that uses ArduPilot as its flight-control system. This system uses a flight version of Arduino to do the low-level feedback control we talked about earlier. The advantage of this system is that you can talk to the flight control system through USB.

There are a number of assembled quadcopters available that use this flight controller. One place to start is at www.ardupilot.com. This website will give you some information on the flight controller and the store has several preassembled quadcopters. If you are thinking of assembling your own quadcopter, a kit would be the right approach. Try www.unmannedtechshop.co.uk/multi-rotor.html or www.buildyourowndrone.co.uk/ArduCopter-Kits-s/33.htm, as these websites sell not only assembled quadcopters, but assembling kits as well.

If you'd like to assemble your own kit, there are several good tutorials on choosing all the right parts and assembling your quadcopter. Try one of the following links:

- blog.tkjelectronics.dk/2012/03/quadcopters-how-to-get-started
- blog.oscarliang.net/build-a-quadcopter-beginners-tutorial-1/
- <http://www.arducopter.co.uk/what-do-i-need.html>

All of these links have excellent instructions.

You may be tempted to purchase one of the very inexpensive quadcopters that are being offered on the market. For this project, you will need the following two key characteristics of the quadcopter:

- The quadcopter flight control will need a USB port so that you can connect Raspberry Pi to it
- It will need to be large enough and have enough thrust to carry the extra weight of Raspberry Pi, a battery, and perhaps a webcam or other sensing devices

No matter which path you choose, another excellent source of information is <http://code.google.com/p/ardupilot>. This gives you some information on how ArduPilot works and also talks about Mission Planner, an open source control software that will be used to control ArduPilot on your quadcopter. This software runs on PC and communicates to the quadcopter in one of two ways, either through a USB connection directly or through a radio connection. It is the USB connection that you will use to communicate between Raspberry Pi and ArduPilot.

The first step to work in this space is to build your quadcopter and get it to work with an RC radio. When you allow Raspberry Pi to control it later, you may still want to have the RC radio handy, if things don't go quite as planned.

If the quadcopter is flying well, based on your ability to control it using the RC radio, you should begin to use ArduPilot in the autopilot mode. To do this, download the software from www.ardupilot.com/downloads. You can then run the software. You should see something like the following screenshot:



You can then connect ArduPilot to the software and click on the **CONNECT** button in the upper-right corner. You should then see something like the following screenshot:

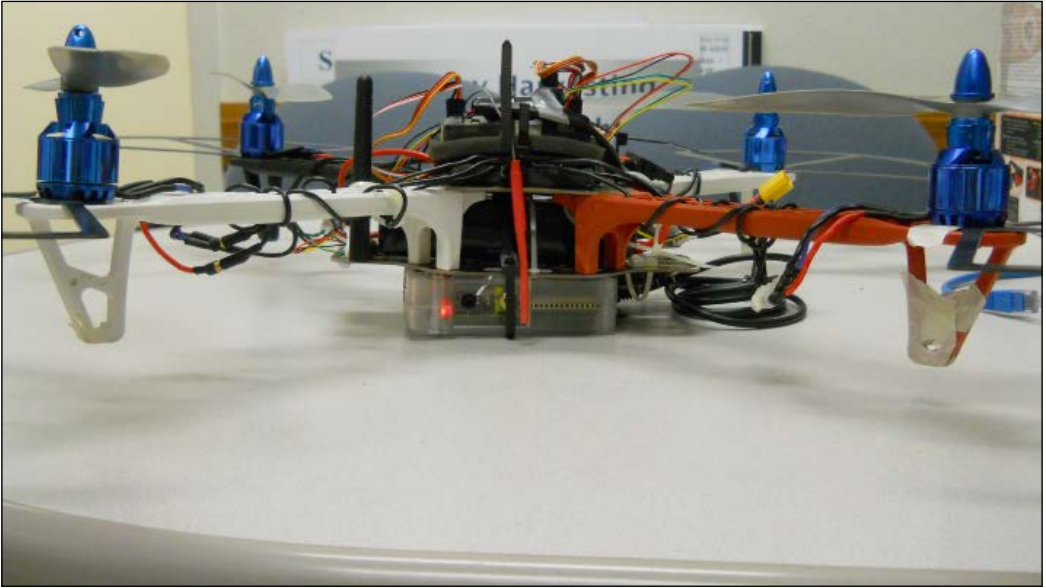


We will not walk through how to use the software to plan an automated flight path. There is plenty of documentation for that on the www.ardupilot.com website. Note that in this configuration, you have not connected the GPS on ArduPilot.

You want to hook up Raspberry Pi to ArduPilot on your quadcopter so that it can control the flight of your quadcopter just as Mission Planner does, but at a much lower and more specific level. You will use the USB interface just as Mission Planner does.

To connect the two devices, you'll need to modify the Arduino code, create some Raspberry Pi code, and then simply connect the USB interface of Raspberry Pi to ArduPilot. You can issue the yaw, pitch, and roll commands to the Arduino to guide your quadcopter wherever you want it to go. The Arduino will take care of keeping the quadcopter stable. An excellent tutorial on how to accomplish this can be found at <http://oweng.myweb.port.ac.uk/build-your-own-quadcopter-autopilot/>.

The following is an image of my configuration. I put Raspberry Pi in a plastic case and mounted the battery and Raspberry Pi below the quadcopter's main chassis:



Now that you can fly your quadcopter using Raspberry Pi, you can use the same GPS and ZigBee or wireless LAN capabilities mentioned in the previous section to make your quadcopter semiautonomous.

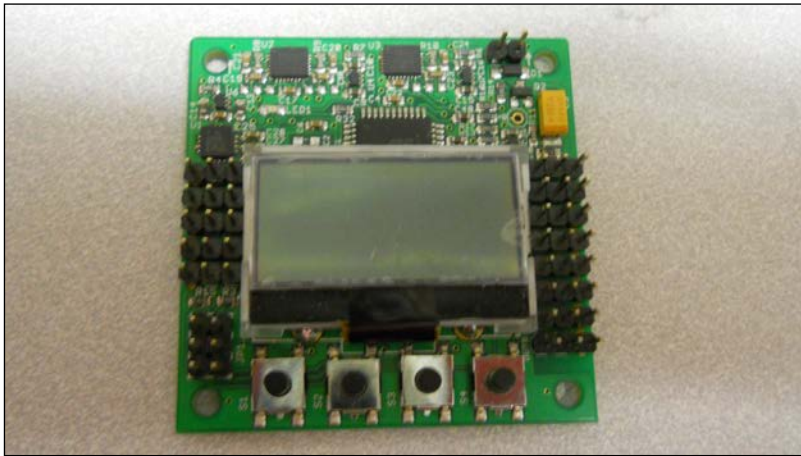
Your quadcopter can act completely autonomously as well. Adding a 3G modem to the project allows you to track your quadcopter, no matter where it might go, as long as it can receive a cell signal. The following is an image of such a modem:



This can be purchased on Amazon or from any cellular service provider. Once you have purchased your modem, simply use Google and look for instructions on how to configure it in Linux. A sample project that puts it all together can be found at <http://www.skydrone.aero>. This project is based on BeagleBone Black, a small Linux processor with capabilities similar to Raspberry Pi.

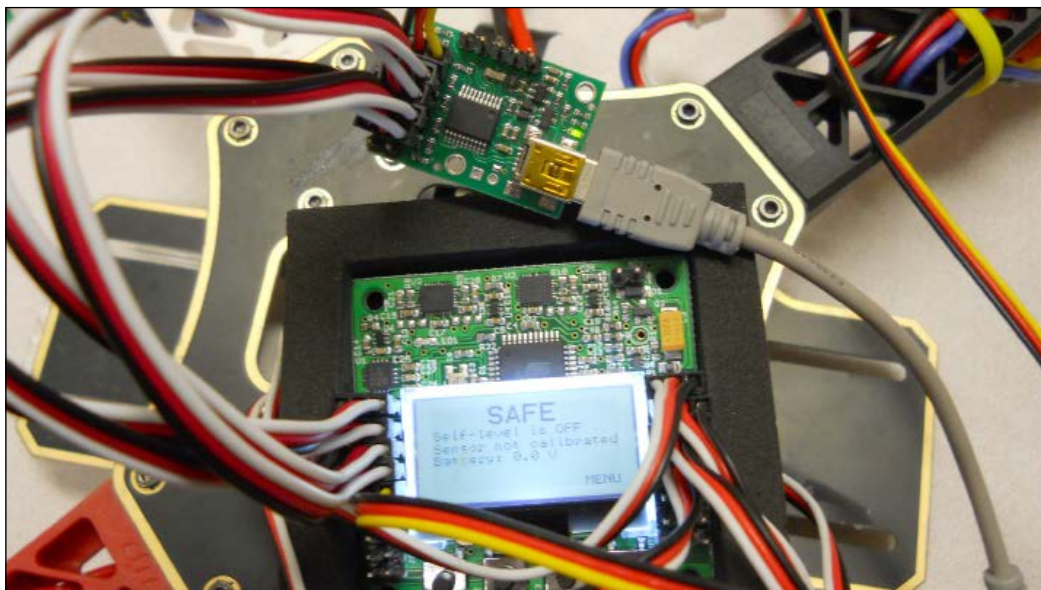
Another possibility for an aerial project is a plane based on ArduPilot and controlled by Raspberry Pi. Look at <http://plane.ardupilot.com/> for information on controlling a fixed-wing aircraft with ArduPilot. It would be fairly straightforward to add Raspberry Pi to this configuration.

If you are a bit more confident in your aeronautic capabilities, you can also build a quadcopter using Raspberry Pi and a simpler, less-expensive flight-control board. The one I like to use is the Hobby King KK2.0 flight-control board, shown in the following image:



This flight-control board takes its flight inputs through a set of input signals and then sends out control signals to the four motor controllers. The good thing about this controller is that it has built-in flight sensors, so it can handle the feedback control for the motors. The inputs will come in as electric commands to turn, bank, go forward, or increase the altitude. Normally, these would come from the RC radio receiver. For this project, you can insert Raspberry Pi and the Maestro Servo Controller we covered in *Chapter 6, Controlling the Movement of a Robot with Legs*.

A close-up of the connections between the servo controller and the flight controller board is shown in the following image. Make sure you do not connect power to the servo controller; it does not need a power supply. Just send the appropriate signals through the servo lines to the receiver input.



You can follow the standard instructions to calibrate and control your quadcopter, with Raspberry Pi creating the commands using the servo controller. It is perhaps best to first use the Pololu Maestro Control Center software to do the calibration. You can then write a program based on the control program you wrote in *Chapter 8, Going Truly Mobile – The Remote Control of Your Robot*. Using this program, you can use the keyboard to control the quadcopter to make it go up, down, right or left, bank, or turn. An example of this type of system, albeit using an even less expensive controller, is shown at <http://www.instructables.com/id/Autonomous-Cardboard-Raspberry-Pi-Controlled-Quad/>. Another example, this time with a tricopter, can be found at <http://bitoniau.blogspot.com/2013/05/using-raspberry-pi-wifi-as-transmission.html>.

Using Raspberry Pi to make the robot swim underwater

You've explored the possibilities of walking robots, flying robots, and sailing robots. The final frontier is robots that can actually maneuver under the water. It makes sense to use the same techniques that you've mastered to explore the undersea world. In this section, I'll explain how to use the capabilities that you have already developed in a **Remote Operated Vehicle (ROV)** robot. There are, of course, some interesting challenges that come with this type of project, so get ready to get wet!

As with the other projects in this chapter, there is the possibility of either buying an assembled robot or assembling one yourself. If you'd like to buy an assembled ROV, try <http://openrov.com>. This project, funded by Kickstarter, provides a complete package, albeit with electronics based on the BeagleBone Black. If you are looking to build your own robot, there are several websites that document possible instructions for you to follow, such as <http://dzlsevilgeniuslair.blogspot.dk/search/label/ROV>. Additionally, <http://www.mbari.org/education/rov/> and <http://www.engadget.com/2007/09/04/build-your-own-underwater-rov-for-250/show> platforms to which you can add your Raspberry Pi.

Whether you have purchased a platform or designed your own, the first step is to engage Raspberry Pi to control the motors. Fortunately, you should have a good idea of how to do this. *Chapter 5, Creating Mobile Robots on Wheels*, covers how to use a set of DC motor controllers to control DC motors. In this case, you will need to control three or four motors based on which kind of platform you build. Interestingly, the problem of control is quite similar to the quadcopter control problem. If you use four motors, the problem is almost exactly the same, except instead of focusing on moving the ROV up and down, you are focusing on moving the ROV forward.

There is one significant difference, the ROV is inherently more stable. In the quadcopter, your platform needed to hover in the air. This is a challenging control problem because the resistance of air is very small and the platform responds very quickly to changes. As the system is so dynamic, a microprocessor is needed to respond to the real-time measurements and individually control the four motors to achieve a stable flight.

This is not the case underwater, where our platform does not want to move suddenly. In fact, it takes a good bit of power to make the platform move through water. You, as the operator, can control the motors with enough precision to get the ROV moving in the direction you want.

Another difference is that wireless communication is not available to you underwater. So, you'll be tethering your device and running controls from the surface to the ROV through wires. You'll need to send control signals and video so you can control the ROV in real time.

You already have all the tools at your disposal for this project. As noted in *Chapter 5, Creating Mobile Robots on Wheels*, you know how to hook up DC motor controllers. You'll need one for each motor on your platform. *Chapter 4, Adding Vision to Raspberry Pi*, shows you how to set up a webcam so that you can see what is around you. All of this can be controlled from a laptop at the surface. This laptop should be connected through a LAN cable and should run vncserver.

Creating the basic ROV platform should open the possibility of exploring the undersea world. An ROV platform has some significant advantages. It is very difficult to lose (you have a cable attached). As the device tends to move quite slowly, the potential for catastrophic collisions is significantly less than many other projects. The biggest problem, however, is keeping everything dry!

Summary

Now, you have access to a wide array of different robotics projects that can take you over land, on the sea, or in the air. Be prepared for some challenges and always plan on a bit of rework. Well, we've reached the end of the book, which, hopefully, is only the beginning of your robotic adventures. We've covered how to construct robots that can see, talk, listen, and move in a wonderful variety of ways. Feel free to experiment; there are so many additional capabilities that can be added to make your robot even more amazing.

Adding infrastructure such as the Robot Operating System opens even more opportunities for complex robotic systems. Perhaps one day, you'll even build a robot that passes the Turing test; that would be a robot that may be mistaken for a human being!

Index

A

A+ board 3, 4

actions

initiating 80-83

ADC-DAC pi

URL 143

Advanced IP Scanner 32

Advanced Linux Sound Architecture
(ALSA) libraries 65

alsamixer 66

arducopter

URL 270

ArduPilot

URL 273

B

B+ board 3

board

accessing, remotely 18

powering 5

C

camera board

connecting 89-92

connecting, to view images 89-92

URL 90

Carnegie Mellon University (CMU)

URL 73

C/C++ programming language 55-58

CMU web tool

URL 78

color

tutorial, URL 100

commands

interpreting 80-83

control structure

general control structure, creating 241-252

C tutorial

URL 58

D

daemon 221

data

URL 221

degrees of freedom (DOFs)

URL 124

display

hooking up 6-8

E

Eclipse

URL 59

Emacs

about 41

commands 43

Espeak

used for allowing robotic voice response,
by projects 71, 72

G

Global Positioning System (GPS)

about 207

data 229-237

GPIO ADC

URL 161

used, for connecting IR sensor 155-162

H

hardware

- about 62
- hooking up 63-70

I

if statement 46, 47

infrared sensor

- Raspberry Pi connecting to, USB used 146

Integrated Development Environment (IDE)

- URL 59

Internet access

- establishing, on Raspberry Pi A+ 18, 19
- establishing, on Raspberry Pi B+ 18

IR sensor

- connecting, GPIO ADC used 155

K

keyboard

- hooking up 6-8

L

legged robot

- hardware, gathering 124-127

Lightweight X11 Desktop Environment (LXDE) 16

Linux commands

- on Raspberry Pi 36-41

Linux program

- creating, to control mobile platform 135-137

Linux terminal program

- URL 195

M

MaxBotix

- URL 173

mobile platform

- controlling, by creating Linux program 135-137
- controlling programmatically, Raspberry Pi used 111-114
- hardware, gathering 176-183

- making truly mobile, by issuing voice commands 138-140

- Raspberry Pi connecting to, servo controller used 127

mobile robots

- creating, on wheels 105
- hardware, gathering 105-108

motion detection

- tutorials, URL 104

motor speed

- controlling, PWM used 114-117

mouse

- hooking up 6-8

N

Nmap

- URL 33

O

OpenCV

- downloading 93-98
- installing 93-98
- URL 104, 252

operating system

- installing 8-17

P

Phidget

- URL 149, 152

picamera 92

platform

- controlling, by adding program arguments 117-119
- making truly mobile, by issuing voice commands 119-121
- speed controlling, Raspberry Pi GPIO used 108-111

PocketSphinx

- used, for accepting voice commands 73, 74

Polulu software

- URL 129, 132

program

- creating in Linux, to control mobile platform 135-137

programming tools, Raspberry Pi

- functions, working with 49, 50
- if statement 46, 47
- object-oriented code 52-54
- Python, libraries 51, 52
- Python, modules 51, 52
- while statement 48, 49

Pulse Width Modulation (PWM)

- used, for controlling motors speed 114-117

PuTTY

- URL 22

Python

- modules, URL 149
- tutorials, URL 43

Python 2

- URL 44

Python programs

- creating 43-46
- running 43-46

Q

quadcopter

- controlling, URL 274
- URL 266, 269, 272

R

Radio Controlled (RC) 258

Raspberry Pi

- A+ board 3, 4
- about 1
- accessing, from host PC 20-33
- B+ board 3
- camera board connecting, to view images 89-92
- connecting to infrared sensor, USB used 146, 147
- connecting to mobile platform, servo controller used 127
- connecting, to RX/TX (UART) GPS device 224, 225
- connecting, to USB GPS device 207-218
- connecting, to USB sonar sensor 162, 163
- connecting to, wireless USB keyboard used 183
- files, creating 41-43

- files, editing 41-43

- files, saving 41, 42

- functions, working with 49

- if statement 46, 47

- Linux commands 36-41

- programming constructs 46

- sensor connecting, USB interface used 147-154

- URL 9

- used, for controlling mobile platform programmatically 111-114

- used, for making robot swim underwater 275, 276

- used, for sailing 258-265

- used, to fly robots 265-274

- while statement 48, 49

- working remotely, through wireless LAN 189-193

- working remotely, through ZigBee 194-203

Raspberry Pi A+

- Internet access, establishing 18, 19

Raspberry Pi B+

- Internet access, establishing 18

Raspberry Pi GPIO

- used, for controlling speed on platform 108-111

Raspbian

- about 36

- URL 254

RealVNC 24

Remote Operated Vehicle (ROV) robot

- about 275

- URL 275

Robot Operating System (ROS) structure

- used, for enabling complex functionalities 253-256

robots

- flying, Raspberry Pi used 265-274

- legged robot 124

- making to swim underwater,

 - Raspberry Pi used 275, 276

- sailing, Raspberry Pi used 258-265

RX/TX GPS 225-228

RX/TX (UART) GPS device

- Raspberry Pi, connecting to 224, 225

S

Secure Shell Hyperterminal (SSH)
 connection 22

servo controller

 hardware, connecting 127-129

 software, configuring 129-134

 URL 125, 127

 used, for connecting Raspberry Pi

 to mobile platform 127

 used, for moving single sensor 169-173

single sensor

 moving, sensor used 169-173

sonar sensors

 URL 173

Sphinx

 URL 73

square-wave pulse width modulation
 (SW PWM) signals 126

system

 configuring, URL 193

T

terminal emulator software

 URL 164

triangulation 207

tricopter

 URL 274

two-wheeled platforms

 URL 106

U

USB camera

 connecting to, Raspberry Pi 86-89

 connecting, to view images 86-89

USB GPS

 accessing, programmatically 218-224

USB GPS device

 Raspberry Pi, connecting 207-218

USB interface

 used, for connecting sensor 147-155

 using 206

USB sonar sensor

 hardware, connecting 164-168

 Raspberry Pi, connecting to 162, 163

V

vision library

 used, for detecting colored objects 98-104

voice commands

 accepting, PocketSphinx used 73-80

 issued, for making platform truly

 mobile 119-121

W

while statement 48, 49

wireless device

 URL 18

wireless keyboards 2.4-GHz

 URL 181

wireless LAN

 URL 181

 used, for Raspberry Pi remote

 connection 189-193

wireless USB keyboard

 used, for controlling project 183-189

 used, for Raspberry Pi connection 183

Z

ZigBee

 about 176

 used, for Raspberry Pi remote

 connection 194-203