ISSUE 30 - FEB 2015

Get printed copies at themagpi.com

The MagPi

A Magazine for Raspberry Pi Users

Electronic Ping Pong
PWM Motor Control
C++ Inheritance
Introducing C#
Raspberry Pi 2
Maze Builder
Weaved IoT
Air Hockey

Enriching Marine Electronics

ISSN 2051-9990

http://www.themagpi.com

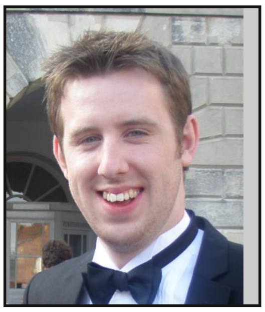
The MagPi 30

Welcome to Issue 30 of The MagPi and another bumper issue full of interesting articles and projects. Before we describe the content of this Issue, let us tell you about our new Kickstarter project to print Volume 3 of The MagPi (Issues 20 to 29) plus a superb new binder. Many of you have asked us for printed copies of The MagPi. Therefore, please check it out at:
https://www.kickstarter.com/projects/themagpi/the-magpi-magazine-for-raspberry-pi-volume-3

Did you miss our surprise? Over the Christmas period we released a special edition of The MagPi... a massive 132 pages of articles from previous issues that we think are great for folks who are new to the Raspberry Pi. You can download it for free from http://www.themagpi.com.

In this Issue, Olivier LeDiouris demonstrates how the Raspberry Pi can be used to enhance navigation data, sampling sensors over a long voyage. João Matos explains how to produce an electronic tennis game with the GPIO pins, LEDs and numeric LED displays. Philip Munts discusses the basics of controlling a small hobby DC motor with his expansion board. Eric Ptak introduces a new P2P solution that allows a secure connection to Raspberry Pi projects that are not directly available on a public network connection. Finally, Ian McAlpine rounds off the hardware section by presenting the new Raspberry Pi 2.

Programming is part of any Raspberry Pi project. In this Issue, Mubarak Abdu-Aguye introduces the C# programming language with an I²C project. William Bell demonstrates how to use C++ inheritance and interface classes. William also discusses another Scratch arcade game, with a two player air hockey simulation. Finally, Martin Meier rounds off this Issue with a Python maze generator.



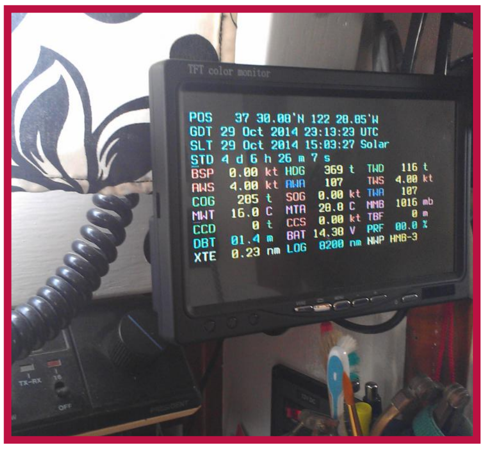
Ash Stone

Chief Editor of The MagPi

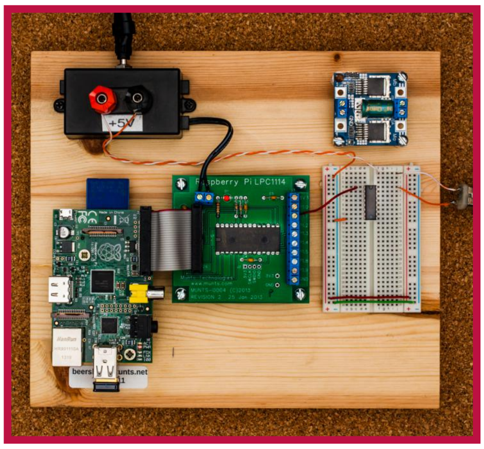
The MagPi Team

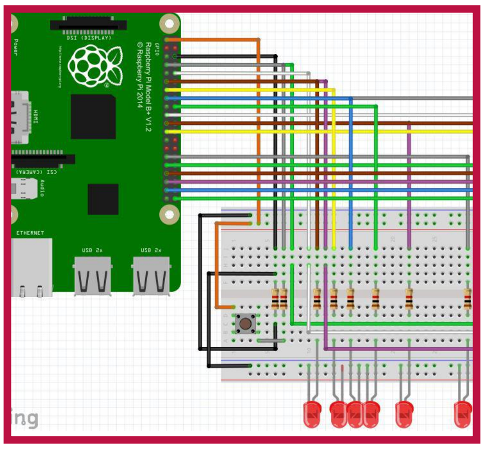
Ash Stone - Chief Editor / Administration
Ian McAlpine - Layout / Testing / Proof Reading
W.H. Bell - Issue Editor / Administration / Layout
Bryan Butler - Page Design / Graphics
Matt Judge - Website
Nick Hitch - Administration
Colin Deady - Layout / Proof Reading
Aaron Shaw - Administration
Dougie Lawson - Testing
Paul Capenter - Testing
Mark Pearson - Layout
John Stap - Layout
Nick Liversidge - Layout / Proof Reading
Martin Wolstencroft - Proof Reading
David Bannon - Layout / Proof Reading
Shelton Caruthers - Proof Reading
Rita Smith - Proof Reading
Claire Price - Proof Reading

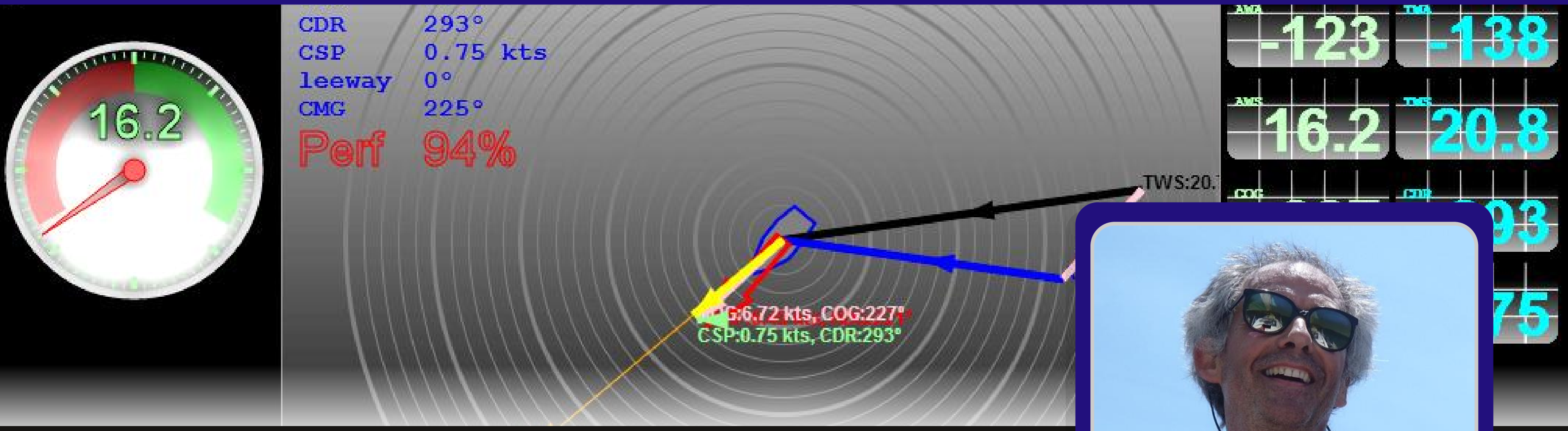
Contents



```
POS   37 30.08'N 122 28.05'W
GDT 29 Oct 2014 23:13:23 UTC
SLT 29 Oct 2014 15:03:27 Solar
STD 4 d 6 h 26 m 7 s
BSP  0.00 kt HDG   369 t  TWD   116 t
AWS  4.00 kt AWA   107    TWS  4.00 kt
COG   285 t   SOG  0.00 kt TWA   107
MWT  16.0 C   MTA  28.8 C  MMB  1016 mb
CCD     0 t   CCS  0.00 kt TBF     0 m
DBT  01.4 m   BAT 14.38 V  PRF  00.0 %
XTE  0.23 nm  LOG  8200 nm NWP HNB-3
```







Enriching the NMEA stream using Java

Olivier LeDiouris

Guest Writer

SKILL LEVEL : INTERMEDIATE

Background

Various commercial electronic instruments are available for boats to read, compute, and display different data used for navigation. My idea was to provide easy access to the data and allow display of the data on several kinds of devices, not just the expensive commercial displays sold by the nautical equipment providers. I also wanted to be able to add more data sources and display the merged information.

Raspberry Pi setup inside the chart table

In the picture above, the Raspberry Pi is fitted with a Slice of Pi break-out board to host the battery monitoring device. Also visible is the small breadboard with a BMP180 barometric pressure and temperature sensor board.

The Raspberry Pi is the only device connected to the boat's NMEA (National Marine Electronics Association) Interface. It is the first to be turned on, and the last to be switched off. I've been running it continuously for weeks, without any glitch.

In this role the Raspberry Pi fulfils two primary functions. Firstly it enriches the NMEA Stream read from the NMEA Station, by calculating new data values, and adding data read from the connected sensors. Secondly it re-broadcasts the enhanced data stream on its own network, so it can be read and used simultaneously by a variety of devices.

Optionally the Raspberry Pi can also read its own data stream and format a character console display, this is discussed later.

The NMEA format

The electronic instruments available on boats are used to measure data like the boat speed, position and heading, the wind speed and direction, all the kind of things required for navigation. Some data are read from transducers (boat speed through water, wind speed, compass heading, etc.), and some are computed (true wind speed and direction, true heading, etc.). Those data are usually available in the NMEA format. The definition of the NMEA strings is one of the oldest electronic standards. The NMEA sentences strings are human readable, that is it is a text format although it can be a bit cryptic. Here is an example of what an NMEA stream looks like:

```
$IIMWV,112,R,00.8,N,A*19
$IIMWV,109,T,00.7,N,A*1A
$IIMTA,31.5,C*02
$IIRMB,A,0.23,R,,HMB-3,,,,,001.20,184,,V,A*1F
$IIXDR,P,1.0156,B,0*71
$WIMDA,29.991,I,1.016,B,31.5,C,,,,,,,,,,,,,,*
3B
$IIRMC,062658,A,1111.464,S,14235.335,W,05.6,2
26,231110,10,E,A*0A
...
```

Extracting the data

The information we want can be extracted from the NMEA data stream using a suitable parser, which is a piece of software that turns text into a programming language "structure". As an example, let us take an RMC (Recommended Minimum) NMEA sentence, the last complete sentence shown above:

A JSON (JavaScript Object Notation) parser could turn that string into a JSON object like this:

```
{
  type: 'RMC',
  active: true,
  cog: 226,
  sog: 5.6,
  declination: 10.0,
  date: 1290493618000,
  pos: {
    latitude: -11.191066,
    longitude: -142.5889166
  }
}
```

The JSON object can then be used in your program. In the object above:

cog (Course Over Ground) 226º
sog (Speed Over Ground) 5.6 knots
dec (Declination) 10º E
date (Date milliseconds since 00:00:00 Jan 1st 1970) 1290493618000 is 23-Nov-2010 06:26:58 UTC.
pos (Position decimal degrees from deg/min)
(Latitude) 11º 11.464' South
(Longitude) 142º 35.335' West

Exclusive serial port access

Navigation stations usually deliver the NMEA data through a serial interface, which is compatible with many devices (like laptop computers). The unfortunate thing about serial ports is that they require exclusive access. That means that when one program, on a laptop for example, is reading the data sent by the NMEA station, the data stream is not available for any other program until the serial connection is released by the program that was using it.

This can be painful if you want to use the data simultaneously in different applications on different platforms. For example displaying a chart plotter (like OpenCPN) while presenting readings in a browser-based graphical interface and reading the GPS data from Airmail, to use a propagation chart to aim for the most appropriate SSB (Single Side Band radio land-station.

Introducing data from other sensors

In some cases, you could be interested in monitoring data that are not managed by an NMEA station (like battery voltage). You could also be interested in monitoring data managed by NMEA, but not by your NMEA station (like air temperature, atmospheric pressure).

Injecting those data in the NMEA Stream you read from your station would make them available for all programs reading the NMEA stream and allow you to log them to monitor, display, or replay them.

You might be interested in some computed data that are not returned by your NMEA station (like true wind speed and direction). These results could be computed from the existing data, and then injected into the broadcast stream.

Power consumption

A laptop can sound like an obvious solution for your on-board computing needs, but the power consumption of a laptop is not always negligible. The NMEA stations usually draw a very small amount of current, but a laptop power adaptor can draw between 1 and 3 Amps, which is a lot when you have limited energy available, like on a sail boat for example. Using a Raspberry Pi as the only device running at all times, this can be reduced to 0.13 and 0.19 Amps with short increases when another device is switched on to use and display the data.

The Java language

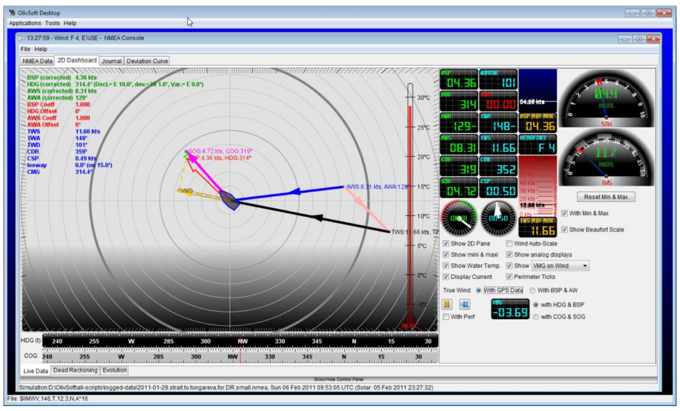
The computer you would use to read the NMEA stream will probably sit close to the chart table, but you might very well be interested in visualizing the data it computes on deck, from existing displays, or from some wireless devices, like tablets or smart-phones.

Java provides a number of libraries that make it an appropriate choice for this application. There is a huge Java community and a lot of open source material. Java is very scalable, so once it runs, it runs the same on all systems (Windows, Mac, Linux, etc.). A jar file (Java ARchive) that runs somewhere will run the same anywhere Java runs, no re-compilation is required. Many good IDEs (Integrated Development Environment) are available for free, and although they are too demanding to run on the Raspberry Pi it is possible to run a very good remote debugging environment.

All the software I wrote to run on the Raspberry Pi is written in Java and is Open Source. Remember Java and JavaScript are totally different languages, and are used for different tasks.

The Navigation Console

The Navigation Console is a program I wrote (in Java) and I've been using on board for several years. When run on a laptop it can provide, among other features, a graphical user interface to the navigation data (as shown below).



I have recently enhanced it to run in headless mode (i.e. without a graphical user interface), to run on the Raspberry Pi. When headless, the role of the Navigation Console is to read and compute the data, optionally save to a log, and then re-broadcast them on one or more channels. For such a re-broadcasting to happen, the Raspberry Pi creates its own ad-hoc wireless network. Other devices will join this network, and will then have access to the re-broadcasted data.

Multiplexing the data

Multiplexing is a technique that takes data from several sources and merges them into a single channel. Here we read data from the NMEA station (already NMEA formatted, obviously), and the program puts them into a cache (technically, it is a Java dynamic structure called a HashMap, living in a singleton). The other software components, such as the different servers (TCP, UDP, etc) read that cache as the single point of truth. Computed data are also read from the cache. As a matter of fact, all computed data (like True Wind Direction and Speed, Current Speed and Direction, Performance, etc.) are re-calculated every time a new value is inserted into the cache by the NMEA reader. The cache also uses a Publish/Subscribe architecture that implements a listener pattern.

We will use those aspects to inject extra data in the cache. For example, you can read the battery voltage from some equipment, turn this value into an NMEA Sentence, and inject it into the cache. There is not a specific NMEA sentence for battery data, so I defined my own BAT (battery) sentence. Any component that has subscribed to the manageEvent event in its listener will be notified of the injection of the NMEA sentence.

The same process can be followed for any other data source. I used a BMP180 PCB (from Adafruit) to get the air temperature and the atmospheric pressure. (Note that unlike the battery voltage, those data do have an NMEA equivalent, but are not available on the NMEA station I have on board). They are read from the sensors, turned into the appropriate NMEA string, and injected in the cache. See in the picture below the data prefixed with BAT (custom NMEA chain for battery), MTA (Air Temperature), MMB (Barometric Pressure). The Character Console featured below is reading the data from the cache that they are injected into.

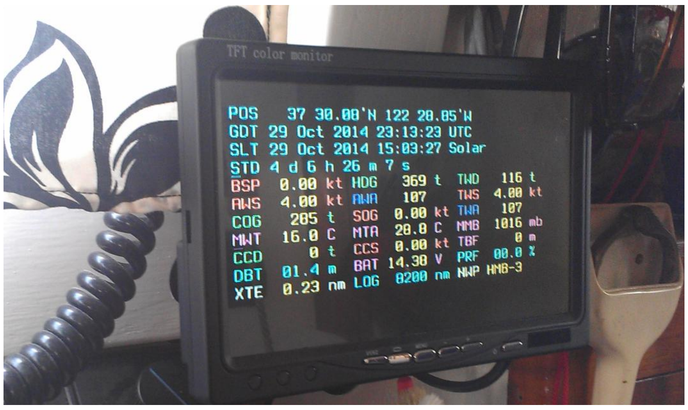
Re-broadcasting

Many navigation programs provide the possibility to read the NMEA streams from channels other than the serial connection. Those other protocols are usually TCP (Transfer Control Protocol) or UDP (User Defined Protocol). Also available for the same purpose: HTTP (Hyper Text Transfer Protocol), RMI (Remote Method Invocation), WebSocket.

The Raspberry Pi re-broadcast the merged data stream on one or more other channels. All the channels can be consumed simultaneously by several clients, so the data broadcast by the Raspberry Pi is simultaneously available to all interested devices.

The Navigation Console provides TCP, UDP, HTTP, and RMI servers. Those servers are very tiny, and do not overload the Raspberry Pi. The HTML5 WebSocket protocol is also available, through node.js and a user-exit.

The Character Console

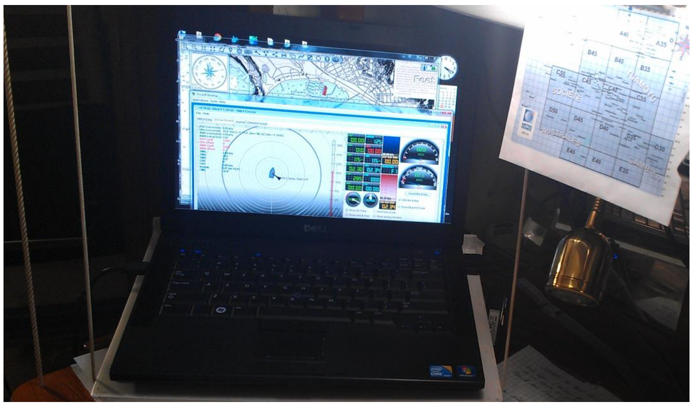


```
POS   37 30.08'N 122 28.85'W
GDT 29 Oct 2014 23:13:23 UTC
SLT 29 Oct 2014 15:03:27 Solar
STD 4 d 6 h 26 m 7 s
BSP  0.00 kt HDG   369 t  TWD   116 t
AWS  4.00 kt AWA   107    TWS  4.00 kt
COG   285 t  SOG  0.00 kt TWA   107
MWT  16.0 C  MTA  20.8 C  MMB  1016 mb
CCD     0 t  CCS  0.00 kt TBF     0 m
DBT  01.4 m  BAT 14.30 V  PRF  00.0 %
XTE  0.23 nm LOG  0200 nm NWP HMB-3
```

The Character Console on a 7" screen

This provides access to the data computed by the Raspberry Pi and can be customized by the user. The goal here is to have an access to those data, without having to turn on a laptop. The only thing to switch on and off is the screen. The Character Console process is also running on the Raspberry Pi, but is separate to the Navigation Console process that is reading and re-broadcasting the data.

Other devices

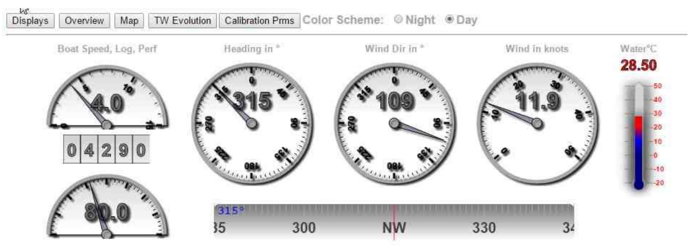


The laptop can use TCP to receive data from the Raspberry Pi and present it simultaneously in my Graphical Console, and the popular open source package OpenCPN (Open Chart Plotter Navigator).

TCP and UDP are both light protocols, designed for computer-to-computer communication. They are both based on a socket mechanism. Once a socket is established between two computers (a client, and a server), then the logic of the dialog will be implemented by the programs running on both client(s) and server, which they have to agree on (to understand each other). In our case, this is extremely simple, once a client is connected, the server is sending it all the valid NMEA strings it reads.

HTTP

HTTP has always been HTML's best friend. HTML is a markup language (widely used to design web pages), HTTP is a transport protocol that can convey HTML streams. HTTP is based on TCP, but is has been designed to be a request-response protocol. For the server to do something, a client has to ask first. As long as you have a browser on the device you want to use, then HTTP would be an obvious choice. To refresh the data, we would use AJAX, in order to avoid a refresh to be performed by the client. HTML5 provides elaborated graphic capabilities that will give us the possibility to come up with a nice graphical user interface. JavaScript is used to handle user interactions.



The HTML5 console displayed in a browser

The HTML5 console can be viewed on any device with a browser such as a laptop, tablet or smart-phone.

WebSocket

The WebSocket protocol has been introduced with the release of HTML5. It is also based on TCP. One of the drawbacks of HTTP is that it is a request-response (a.k.a. push-pull) protocol. You have to make a request to get a response. For example, if you want to get a flight status from some airline website, the first time you reach the page, it gives you the expected arrival time of the flight. Every time you want to see if this estimated time has changed, you must refresh your page. In other words, request it again.

The WebSocket protocol precisely addresses (among others) this kind of issue. Once the client (i.e. your browser) is connected to the server, data will be pushed (by the server, to the client) as needed, without requiring the client to refresh its page. This clearly divides the traffic by two. The browser you use must be WebSocket aware though. As of now (2015), some browsers (like Internet Explorer 9) still do not support it.

In the Navigation Console, the WebSocket interface is implemented as a user-exit. It requires a WebSocket server to be available and we can run this on the Raspberry Pi. Node.js is the one we use, with its WebSocket module.

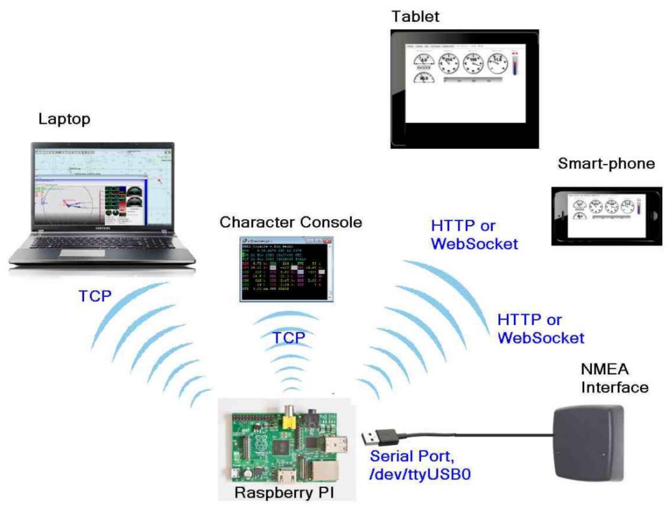
In short, this is what happens:

1) An HTTP/WebSocket server is started on the Raspberry Pi
2) A user-exit (listening to the cache) is pinging the WebSocket server everytime some data is inserted
3) A web page (WebSocket client) will reach the WebSocket server to display real-time data, pushed by the server, from as many clients as needed

The WebSocket console looks exactly like the HTML5 one featured above. But it takes about half the resources and the data are refreshed regularly by the server.

Summary of the architecture

This shows all the possible components with the Raspberry Pi at the heart. The Raspberry Pi rebroadcasts the data using TCP and both HTTP and WebSocket (for browser-based clients). Data can also be logged on the Raspberry Pi's SD card. The laptop uses TCP to consume the data, and can run simultaneously several programs using the NMEA data. Tablets and smart-phones can also be added into the wireless network, and by using HTTP or WebSocket they can also display the HTML5 Console.



Notice in the diagram, the central place taken by the Raspberry Pi. It is acting as the main hub of all the system. It is the only one reading the NMEA data out of the NMEA station. In addition, it manages its own sensors to read extra data (in this case: battery voltage, atmospheric pressure and air temperature). All the data gathered by the Raspberry Pi are re-broadcasted onto several channels, depending on the devices connected on the Raspberry Pi's own ad-hoc wireless network.

Those devices can join or leave the network at their leisure, without disturbing the others. The only piece remaining active will be the Raspberry Pi, maintaining all the infrastructure, and all this for between 0.13 and 0.19 Amps.

It is interesting though, to notice that this architecture is the exact same one that was used during the recent 34th America's Cup by those fantastic multi-million dollar boats. Well, now you can get to the data, you can analyse them and the next America's Cup is all yours.

Resources and Useful Links

This article is based on my project description which can be found at:
`http://www.lediouris.net/RaspberryPI/_Articles/readme.html`

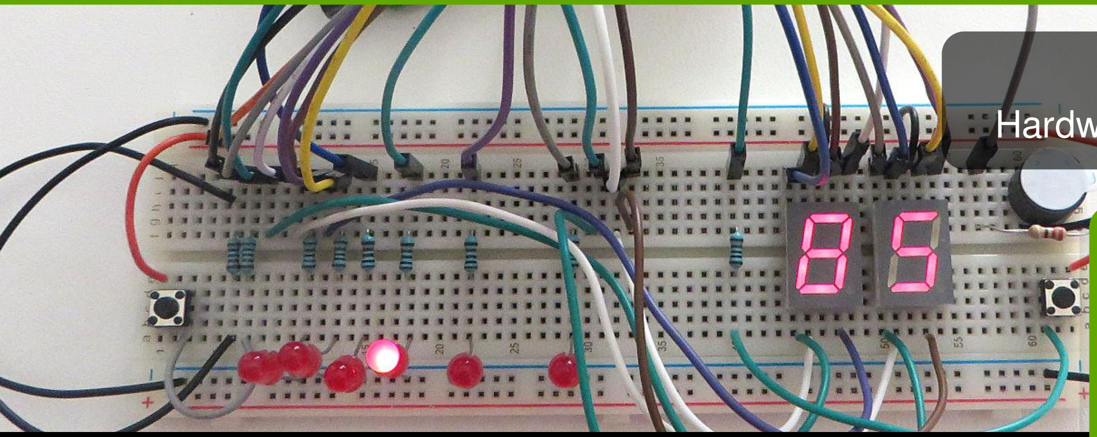
For simplicity most of this software is available as a pre-built Raspberry Pi archive :
`https://code.google.com/p/weatherwizard/wiki/NewDowloadPage`
Unpack the `.tar.gz` archive file tar with `xavf` then run `../all-scripts/olivsoft` for the main menu. Of course you will need a suitable Java Runtime environment, this is provided in recent releases of Raspbian, you may need to install it for other platforms. This page also has links for Windows and Linux install packages, and a User Manual for the Navigation Console.

To create a build environment for this projects (and some of my others) see the instructions at:
`https://code.google.com/p/oliv-soft-project-builder/`
You will need a Java Development Kit (JKD), minimum version 7, and a build system like Ant or Gradle.

The Java code for the Navigation Console is described at:
`https://code.google.com/p/fullnmeaconsole/`
Instructions are given to download the source using an SVN (Subversion) client.

OpenCPN - Open Source Chartplotter and GPS Navigation Software can be found here:
`http://opencpn.org/ocpn/`

Some of the Java code uses Pi4J a project to link Java with the Raspberry Pi I/O:
`http://pi4j.com/`



PING PONG

Hardware reaction testing game

Build a hardware based "tennis" game and test your reactions

João Matos

Guest Writer

SKILL LEVEL : BEGINNER

Introduction

I was approximately 10 years old when I received my first TV game console. It was a gift from a long time friend of my parents who lived in France. At that time there was nothing like this game in Portugal.

The main game was a simple tennis match, which was composed of two small vertical bars on each side of the screen and a square as a ball. It was the leading edge of technology - all in black and white! I loved it and played it for years.

I remembered those times and decided to build a hardware based Ping Pong look-alike project using my new Raspberry Pi B+. I also wanted to see the reaction of my kids (Tiago age 13 and Sara age 11) who are used to playing with modern games consoles.

I was pleased to see how excited they were while I was building it and also to play with it. I hope you will have some fun with it too.

Rules of the game

A point is won when the other player does not press the button in time to return the “ball” (represented by a lit LED) to the opposite side, or if the other player presses their button too soon. The player who wins the point starts the next serve.

A game is won after 9 points are scored. A match should consist of any odd number of games (usually five or seven).

For the first serve the starting player is randomly selected. The time between LED jumps is random to make it more difficult to predict. The buzzer will sound every time a point is won.

Before you start

This project requires the use of a Raspberry Pi A+/B+ as it uses some of the new GPIO pins. It is assumed you are already using a Raspbian distribution.

The software for this project was written using Python, an easy to learn programming language.

If you want to learn more about Python programming, check out http://www.python.org and also the book "Welcome to Python for You and Me" at http://pymbook.readthedocs.org/en/latest/.

When you are learning Python it is usually better to type in the code yourself and a full listing is provided, starting on page 13. This helps you to understand what is going on and how the program works. However you can also download the Python code, plus the breadboard design shown below, from http://fritzing.org/projects/ping-pong-with-raspberry-pi.

Parts needed

1x Raspberry Pi A+/B+
1x buzzer (I used a 8-15V 85dB Mag)
2x 7-segment red display 13mm, red, common cathode
2x 4-pin push button (3.5mm)
6x red LED 5mm
1x 220R resistor for the buzzer
9x 1k resistors for the LEDs, push-buttons and displays
1x breadboard (aka Protoboard)

You also need several coloured jumper cables (male-female and male-male)

Hardware

The easiest way to assemble the circuit is to follow the picture below with the breadboard design.

The Raspberry Pi has 2 different voltages in its GPIO (General Purpose Input Output) pins - 5V and 3.3V (or 3V3). All pins only work with 3V3, except the 5V and Ground (GND) pins.

NOTE: You should be careful to never connect a 5V pin to a 3V3 pin, nor should you connect a 3V3 or 5V pin to a GND pin.

Because of that, I like to follow some safe rules when building circuits for my Raspberry Pi:

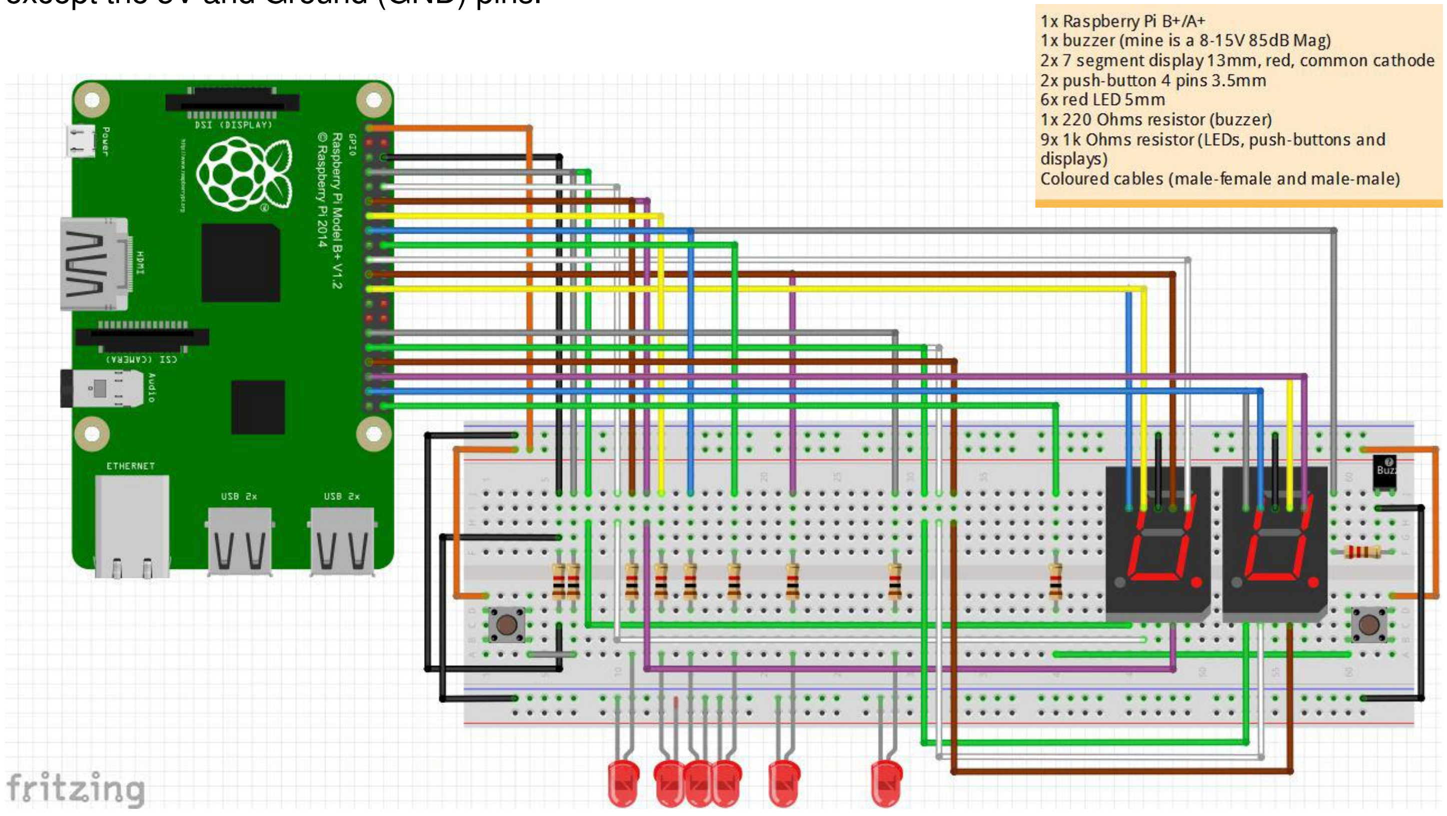
1) Use coloured cables to ease identification (red for 5V, orange for 3V3 and black for GND)

2) Assemble everything before connecting the Raspberry Pi

3) After completing all the assembly work, double check everything before powering the Raspberry Pi.

It may sound basic and boring, but it will save time searching for malfunctions and it may save your Raspberry Pi from getting damaged. Of course, the great thing about the Raspberry Pi is that if it should get damaged, it does not cost much to get another!

I like to have a coloured printout of the GPIO pinout near my Raspberry Pi to make it easy to assemble and double check everything. You can find one at http://www.raspberrypi-spy.co.uk/2014/07/raspberry-pi-b-gpio-header-details-and-pinout/.


1x Raspberry Pi B+/A+
1x buzzer (mine is a 8-15V 85dB Mag)
2x 7 segment display 13mm, red, common cathode
2x push-button 4 pins 3.5mm
6x red LED 5mm
1x 220 Ohms resistor (buzzer)
9x 1k Ohms resistor (LEDs, push-buttons and displays)
Coloured cables (male-female and male-male)

Circuit description

In the following description a high signal means 3V3 and a low signal means 0V (GND).

The 1k resistors are there to limit the electrical current and to protect the Raspberry Pi GPIO pins. The red LEDs are lit by activating a high signal on their anode/positive side (this is the longer leg). The LEDs are connected to GND. When either push-button is pressed it sends a high signal to the anode of the LED.

The buzzer is very simple. Just apply a high signal to its positive pin (it should be marked) and it will sound until the signal is dropped. If the buzzer does not sound or sounds very weak, try removing the 220R resistor.

The 7-segment displays I selected for this project have individual anode/positive pins but a common cathode/negative/GND pin. Think of them as having 7 different LEDs (plus one for the decimal point, which is not used in this project). If the displays are too dim replace the 1k resistor beside the left display with a 220R resistor.

[Ed: Experienced readers may choose to use a BCD - 7-segment decoder to reduce the wiring.]

Software

If you are typing in the Python code you can do so from the command line with,

```
nano ~/ping-pongv1.py
```

or if you are using the GUI then you can use the Python IDLE editor.

The program is listed on the next page. After various functions are defined, the main part of the program starts with the `try` block. Here the pin numbering system is defined and any warnings about the previous state of the pins is disabled. Then it defines which pins are used for input and output and sets all output pins with a low signal (0V).

The second block sets the variable `cur_pos` to keep track of the current position of the “ball” and randomly defines the starting player. In doing so it also defines the game direction (right or left), which is used later to test for the first/last position. You can probably guess what the `show_scores()` function does!

In the next block, the `while` loop is the inner core of the program. This loop repeats itself until one of the players reaches the maximum score of 9 points. First it turns the “ball” LED on and waits for a period of time to allow the players to press their button. Then it turns the LED off and checks if the “ball” is in the final position on either side.

If the "ball" is not in the final position, it checks if either player pressed their button early. If either button is pressed that player loses and a function called `end_service()` is called to sound the buzzer and update the displays. If no button is pressed then it increases the “ball” position.

The next block starts with the `else` clause, from the first `if` statement after the `while` loop. This indicates that the "ball" is in the first or last position. It checks if the player pressed the button on time or not. If they did it changes the "ball" direction, but if they did not then the score is increased for the other player and the position and direction are updated for the new game. As before, the `end_service()` function is called to sound the buzzer and update the displays.

The `out_sequence()` function is called when a player reaches the maximum score. It is an LED special effect to show the end of the game.

The last command is `GPIO.cleanup()` which is used to restore all the pins to their pre-game state, as good practice.

The main part is enclosed in a `try-except-finally` block to make sure the `GPIO.cleanup()` function is called, even if there is an error or the user interrupts the program.

The start of the listing is where libraries of functions are imported, global constants (all caps) and variables are created and functions are defined. To understand the `DISPLAY_1`, `DISPLAY_2` and `DIGITS_MASKS` constants please refer to the 7-segment display documentation and to https://en.wikipedia.org/wiki/Seven-segment_display.

Due to space restrictions I cannot explain the entire source code, but hopefully you will take this as a challenge to learn and understand what the code is doing.

Play the game

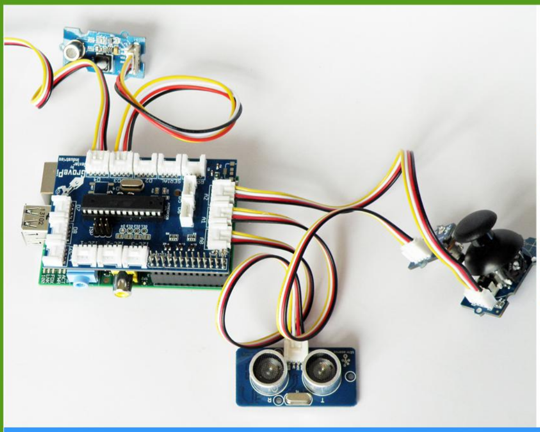
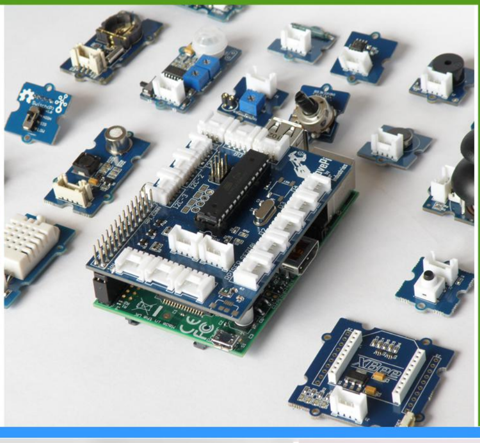
Once the hardware is built and checked and the code has been entered, to play the game enter the following on the command line:

```
sudo python ~/ping-pongv1.py
```

Python source code

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""Ping-pong.
This is a ping-pong look-alike but the maximum
score is 9 instead of 11 due to the use of a
7-segment display.

For the first service, the starting player is
randomly selected. The time between LED jumps
is random to make it more difficult to
predict. The buzzer will sound every time a
point is won."""

import random
import time
import RPi.GPIO as GPIO

# Constants use BOARD pin numbering system
BUTTON_1 = 7
BUTTON_2 = 40
IN = [BUTTON_1, BUTTON_2]  # input pins

LEDS = [11, 13, 15, 18, 22, 29]
FIRST = 0  # first position
LAST = len(LEDS) - 1  # last position

BUZZER = 16

# pins are ordered by letter according to
# the 7-segment display documentation
DISPLAY_1 = [21, 19, 12, 10, 8, 23, 24]
DISPLAY_2 = [36, 35, 33, 32, 31, 37, 38]

OUT = LEDS + [BUZZER] + DISPLAY_1 + DISPLAY_2
# output pins

# masks are ordered by letter according
# to the 7 segment display documentation

DIGITS_MASKS = {0: [1, 1, 1, 1, 1, 1, 0],
 1: [0, 1, 1, 0, 0, 0, 0],
 2: [1, 1, 0, 1, 1, 0, 1],
 3: [1, 1, 1, 1, 0, 0, 1],
 4: [0, 1, 1, 0, 0, 1, 1],
 5: [1, 0, 1, 1, 0, 1, 1],
 6: [1, 0, 1, 1, 1, 1, 1],
 7: [1, 1, 1, 0, 0, 0, 0],
 8: [1, 1, 1, 1, 1, 1, 1],
 9: [1, 1, 1, 0, 0, 1, 1]}

# LED direction
RIGHT = 1
LEFT = -1

MAX_SCORE = 9

# global score vars
player_1 = 0
player_2 = 0

def set_in_pins(pins):
  """Setup input pins."""
  for pin in pins:
    GPIO.setup(pin, GPIO.IN, GPIO.PUD_DOWN) #
activates pull down resistor

def set_out_pins(pins):
  """Setup output pins."""
  for pin in pins:
    GPIO.setup(pin, GPIO.OUT)

def pin_low(pin):
  """Put low 0V signal on pin."""
  GPIO.output(pin, GPIO.LOW)

def all_low(pins_lst):
  """Put low 0V on all pins in list."""
  for pin in pins_lst:
    pin_low(pin)

def show_digit(digit=0, display=DISPLAY_1):
  """Show digit in specifed display."""
  # enumerate returns pos and value from mask
  for pos, value in enumerate(DIGITS_MASKS
[digit]):
    GPIO.output(display[pos], value)

def show_scores():
  """Shows players scores."""
  show_digit(player_1, DISPLAY_1) # show
player 1 score on 1st display
  show_digit(player_2, DISPLAY_2) # show
player 2 score on 2nd display

def pin_high(pin):
  """Activates high 3V3 signal on pin."""
  GPIO.output(pin, GPIO.HIGH)

def led_on(pos):
  """Switch LED on and pauses."""
  pin_high(LEDS[pos])
  if pos in [FIRST, LAST]: # first or last LED
    pause = 0.3 # time for button response
  else:
    pause = random.choice([0.5, 1, 1.5, 2]) #
random delay
  time.sleep(pause)

def led_off(pos):
  """Switch LED off."""
  pin_low(LEDS[pos])

def is_on(pin):
  """True if pin has a high signal."""
  return GPIO.input(pin) == GPIO.HIGH

def buzzer(pin, pause=1):
  """Activates buzzer."""
  pin_high(pin)
  time.sleep(pause)
  pin_low(pin)

def end_service():
  """Buzzes and updates scores."""
  buzzer(BUZZER)
  show_scores()
  time.sleep(2)

def out_sequence(pause=1):
  """Shows LED sequence from inside out."""
  nr_leds = len(LEDS)
  for pos in range(nr_leds / 2, 0, -1): #
reverse order
    # activate pair
    pin_high(LEDS[pos - 1])
    pin_high(LEDS[-pos + nr_leds])

    time.sleep(pause)

    # deactivate pair
    pin_low(LEDS[pos - 1])
    pin_low(LEDS[-pos + nr_leds])
 time.sleep(pause)

try:
  GPIO.setmode(GPIO.BOARD) # BOARD numbering
  GPIO.setwarnings(False)

  set_in_pins(IN) # setup input pins
  set_out_pins(OUT) # setup output pins

  all_low(OUT) # clear all output pins

  cur_pos = random.choice([FIRST, LAST]) #
define first LED to be lit

  if cur_pos == FIRST:
    direction = RIGHT
  else:
    direction = LEFT
```

```
  show_scores()

  while player_1 < MAX_SCORE and player_2 <
MAX_SCORE:
    # light LED (ball) and pause for a time to
    # allow the players to press the button
    led_on(cur_pos)

    led_off(cur_pos)

    if ((direction == RIGHT and cur_pos <
LAST) or # not in last position
        (direction == LEFT and cur_pos >
FIRST)): # not in first position

      if is_on(BUTTON_1): # if button 1 was
pressed player 1 loses
        player_2 += 1
        cur_pos = LAST
        direction = LEFT
        end_service()
      elif is_on(BUTTON_2): # if button 2 was
pressed player 2 loses
        player_1 += 1
        cur_pos = FIRST
        direction = RIGHT
        end_service()
      else:
        cur_pos += direction # update pos.

    else: # it is the first or last position

      if ((cur_pos == LAST and
is_on(BUTTON_2)) or
          (cur_pos == FIRST and
is_on(BUTTON_1))):

        direction *= -1 # change direction
        cur_pos += direction

      else: # didn't push button on time

        if cur_pos == FIRST: # player 1 loses
          player_2 += 1
          cur_pos = LAST
          direction = LEFT
        else: # player 2 loses
          player_1 += 1
          cur_pos = FIRST
          direction = RIGHT
        end_service()

  out_sequence()

except KeyboardInterrupt as error:
  pass

finally: # always executed
  GPIO.cleanup()  # restore to pre-game state
```

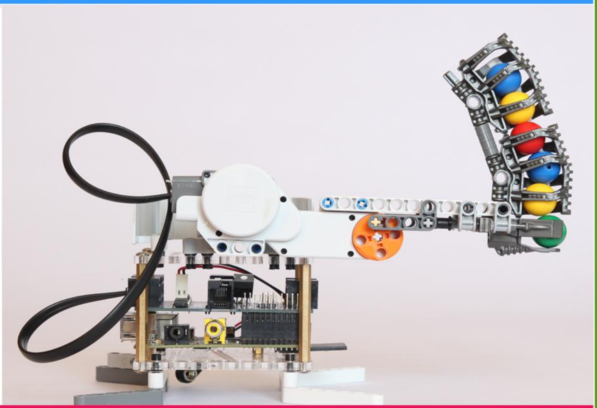
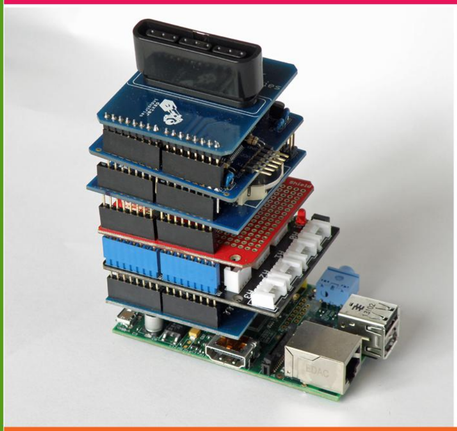
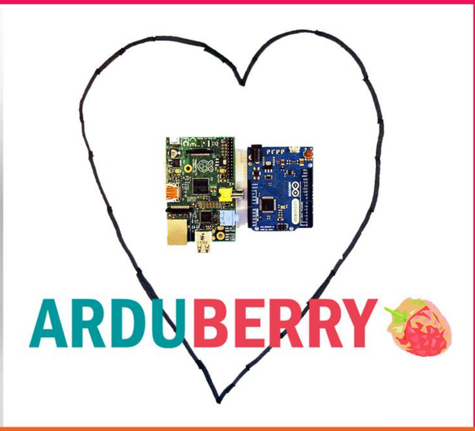



GrovePi

Connect Hundreds of Sensors to your Raspberry Pi

BrickPi

Turn your Raspberry Pi into a LEGO© Robot

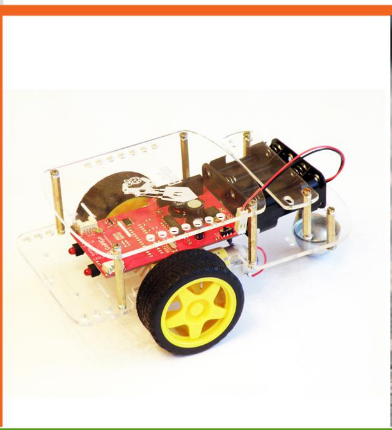
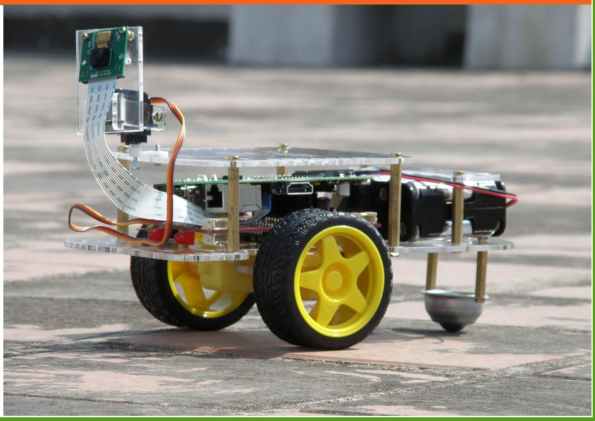




Arduberry

Unite the Raspberry Pi and Arduino

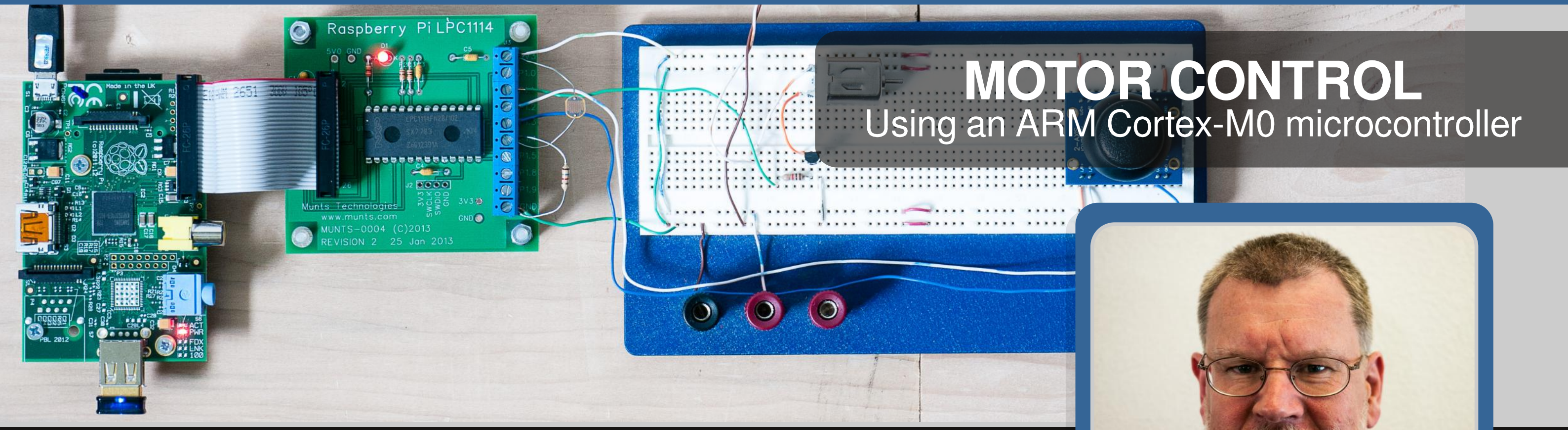
GoPiGo

Turn Your Raspberry Pi into a Robot

dexterindustries.com

MagPi Readers! Use the code "MagPi14" for a 10% discount in our store.



Pulse width modulation motor control

Philip Munts

Guest Writer

SKILL LEVEL : ADVANCED

Introduction

Electric motors are pervasive in our industrial civilization. Motors can open and close our doors, make our elevators go up and down, chill our refrigerators, and make our toys go. They are among the more interesting devices one can control from a computer like the Raspberry Pi, whether for a robot, a vehicle, or even a piece of art.

The Raspberry Pi LPC1114 I/O Processor Expansion Board was first introduced in Issue 14 of the MagPi. It has since been discussed in Issues 17 and 22 of the MagPi. The LPC1114 expansion board has three terminals that can be configured as PWM (Pulse Width Modulation) outputs, suitable for controlling DC (Direct Current) motors. PWM is a technique for controlling the amount of energy delivered to a load (motor or other device) by switching its power supply on and off very rapidly.

The percentage of time that the power supply is switched on is called the duty cycle. If the power supply is switched on most of the time (high duty cycle), maximum energy will be delivered to the load. If the switch is off most of the time (low duty cycle), minimum energy will be delivered to the load. Figure 1 shows oscilloscope traces of PWM signals that correspond to low (yellow) and high (blue) duty cycles.

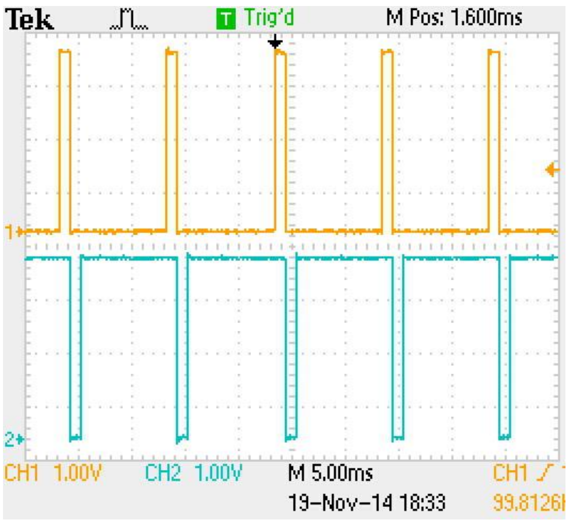


Figure 1 - Pulse width modulation signals

The LPC1114 timer subsystem can be configured to drive up to three PWM outputs, without software control beyond setup and duty cycle adjustment. This article describes how to control both DC motors and remote control servo motors with the LPC1114 PWM outputs.

DC motors

DC (Direct Current) motors always have power supply voltage and speed ratings. At the specified power supply voltage, a motor will spin at the rated speed (assuming no load, a mechanical load will slow down the motor). If the power supply voltage is less than the rated voltage, a DC motor will spin at a slower speed. Therefore, the speed of a DC motor can be controlled by adjusting the voltage supplied to the motor. Every motor has a low voltage limit, below which it will not spin reliably. Rotation speeds that are below the speed corresponding to the low voltage limit can be accessed by using a gear train.

Due to mechanical (inertia) and electrical effects (inductance), a DC motor's speed depends on its average power supply voltage. Therefore, it is possible to use a PWM signal to switch a motor's power supply on and off to control its speed. A low PWM duty cycle will result in a low average voltage and a slow speed. A high PWM duty cycle will result in a high average voltage and a high speed.

The LPC1114 PWM outputs are incapable of delivering enough energy to control anything but the tiniest DC motors. Therefore, to use small or medium sized motors it is necessary to connect some sort of power switch or driver between the LPC1114 PWM output and the DC motor. Many different DC motor driver chips and modules are available, capable of driving anything from small hobby motors to large industrial motors.

First test your motor with power connected at maximum speed (maximum voltage for the motor) and measure the current that the motor draws using a digital volt meter. In addition to the current associated with the maximum rotation speed, measure the current the motor draws when it is prevented from moving. This is the stall current. The current ratings of the motor determine the type of drive circuit that should be used with the motor.

The ULN2003 is a simple and inexpensive driver that can be used with small DC hobby motors. The ULN2003 is an integrated circuit with 7 low side switches, implying that each switch will sink current to ground when you turn it on. With a low side switch, the low side of the load is connected to the switch and the high side of the load is connected to a positive supply voltage such as +5V or +12V.

The ULN2003 is rated for voltages of up to 50V and a current of 500mA for any one output. To drive more than one load, the total current for all of the outputs should also be limited to 500 mA. For larger motor currents, more complex driver solutions are required. The ULN2003 and its slightly larger companion device the ULN2803 (with 8 switches) can be used to drive all sorts of real world loads, including motors, LED's, relays, solenoids, etc..

There is also another companion device, the UDN2981, that is nearly pin compatible with the ULN2803 but contains 8 high side switches. With a high side switch, the high side of the load is connected to the switch and the low side of the load is connected to ground. The UDN2981 is not as common and is more expensive than either the ULN2003 or the ULN2803.

A circuit diagram suitable for driving a small DC hobby motor using the ULN2003 is given in Figure 2, whereas Figure 3 shows the complete test system used for this article. Note that this arrangement can only spin the motor in one direction. To spin a motor in either direction, it is necessary to use a more complicated driver circuit called an H-Bridge. An H-Bridge circuit has 4 switches, two-low side and two-high side, which allows it to deliver either polarity voltage to the motor.

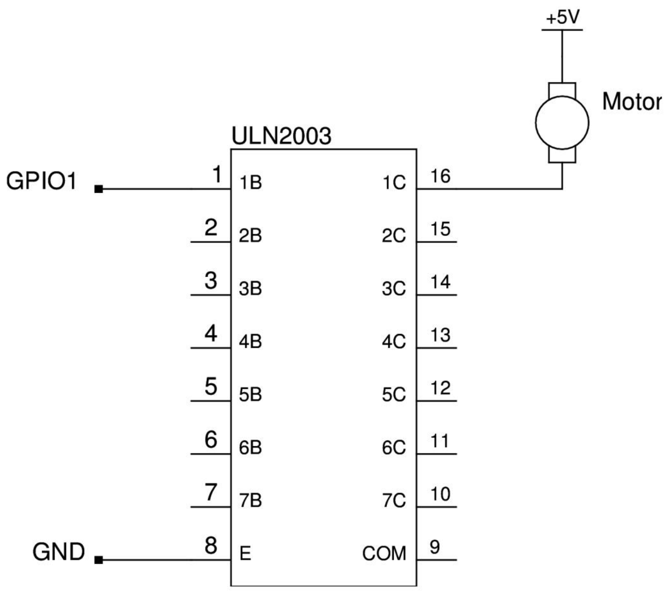


Figure 2 - ULN2003 and small DC motor

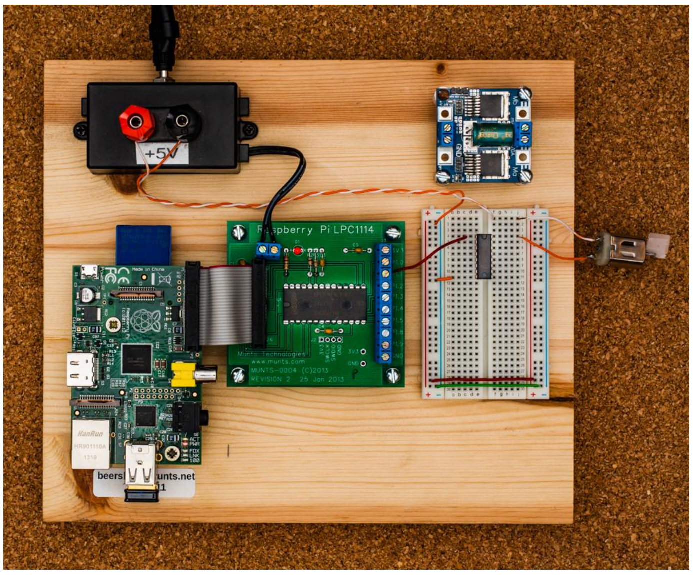


Figure 3 - Small DC motor test stand

Programming interface

Complete documentation of the LPC1114 I/O expansion board and the associated application programming interface (API) is given in the user guide:
http://munts.com/rpi-lpc1114/doc/UserGuide.pdf

The C programming API includes several functions that can be used to control a DC motor with a LPC1114 PWM output:

```
spiagent_open(SERVERNAME,&error);
spiagent_pwm_set_frequency(100,&error);
spiagent_pwm_configure(LPC1114_PWM4,&error);
spiagent_pwm_put(pin,50.0F,&error);
```

where the frequency parameter ranges from 50 to 50000 Hz, the duty cycle parameter ranges from 0.0 to 100.0 (percent), with 0.0 selecting minimum power to the motor and 100.0 selecting maximum power. Small hobby motors tend to function better with a low PWM frequency. Larger industrial motors may work with higher PWM frequencies.

Servo motors

Servo motors have been used for many years in radio controlled model airplanes, cars, and boats. Each unit has three wired connected to it, a voltage supply (4.8V to 6V for small servos, 12V or more for large servos), control signal, and ground. The servo unit contains an electric motor, gears, and a control circuit. Two example small servo motors are shown in Figure 4.

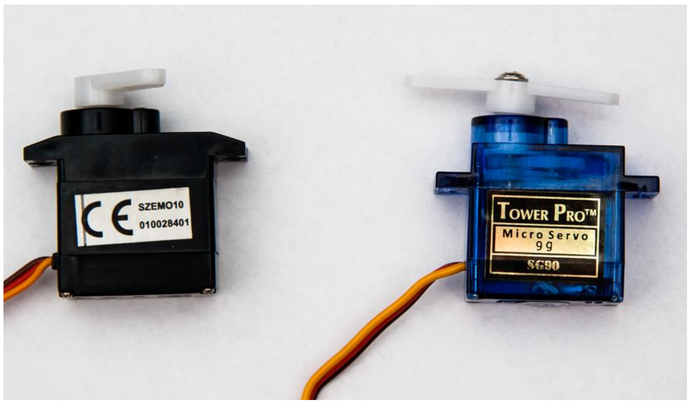


Figure 4 - Small servo motors

The servo motor control signal is a form of pulse duration modulation. The servo motor expects a continuous train of control pulses that are 1.0 to 2.0 milliseconds long repeated 50 times a second. The duration of the control pulse sets the position of the servo motor.

A control pulse width of 1.0 milliseconds sets the servo to one end of its rotation, 1.5 milliseconds sets the servo to the midpoint/zero/neutral position and 2.0 milliseconds sets the servo to other end of its rotation. If we constrain the PWM frequency to 50 Hz and the pulse width from 1.0 to 2.0 milliseconds (5 to 10% duty cycle), we can use the LPC1114 PWM outputs to control servo motors as well.

Small servo motors typically have a 90 degree rotation, but some are geared for 180 degree or 360 degree rotation. There are also linear servos available, with up to 300 mm stroke and 80 kg thrust, and continuous rotation servos that rotate at a speed and direction proportional to the duration of the control pulse.

Since servo motors expect a 3V to 5V control signal, they are very easy to drive directly from the LPC1114 I/O Processor Expansion Board. Just connect +5V, ground, and one of GPIO1, GPIO2, or GPIO3 to the servo. The following C code fragment illustrates how to control a servo motor with an LPC1114 PWM output:

```
spiagent_open(SERVERNAME,&error);
spiagent_pwm_set_frequency(50,&error);
spiagent_pwm_configure(LPC1114_PWM4,&error);
spiagent_servo_set_position(pin,0.0F,&error);
```

where position parameter ranges from -1.0 to +1.0, with 0.0 selecting the midpoint or neutral position. The servo library functions limit the PWM frequency to 50 Hz and the pulse width to 1.0 to 2.0 milliseconds, as required by typical RC servos. Further details are given in the LPC1114 I/O expansion board user guide: http://munts.com/rpi-lpc1114/doc/UserGuide.pdf

Grounding

For anything larger than a small hobby motor, make sure that the power and ground connections for the driver chip or module are directly connected to the system power supply. Do not use the power and ground terminals on the expansion board. Otherwise, electric current for the motor will flow through the expansion board and Raspberry Pi. Neither of these were designed to carry much electric current.

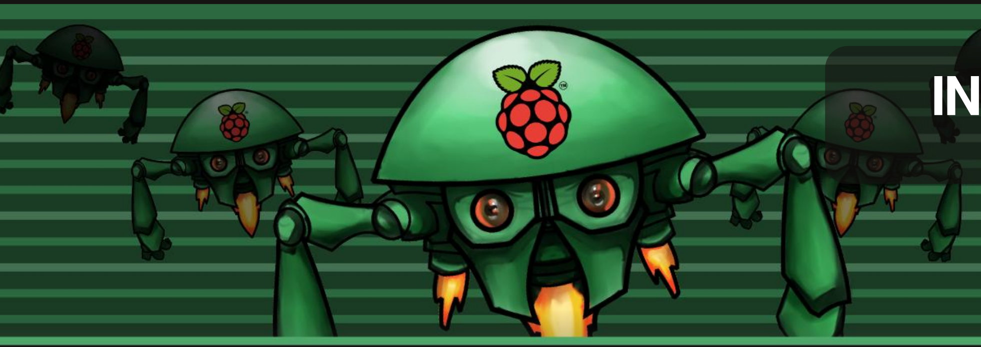
Attention !

1) It is important to MEASURE the motor current when running at full speed and when the motor is STALLED (which is forcibly stopped while trying to turn at full speed).

2) The circuit shown in this article will work for ONE motor up to a maximum current of 500 mA, where this includes surges.

Links

LPC1114 I/O Processor Expansion Board documentation and purchase information:
http://munts.com/rpi-lpc1114
Wikipedia on Pulse Width Modulation: http://en.wikipedia.org/wiki/Pulse-width_modulation
ULN2003 data sheet: http://www.ti.com/lit/ds/symlink/uln2003a.pdf
Servo City, a source for rotary and linear servos: https://www.servocity.com



Eric PTAK

Guest Writer

Raspberry Pi access over the Internet, say goodbye to port forwarding

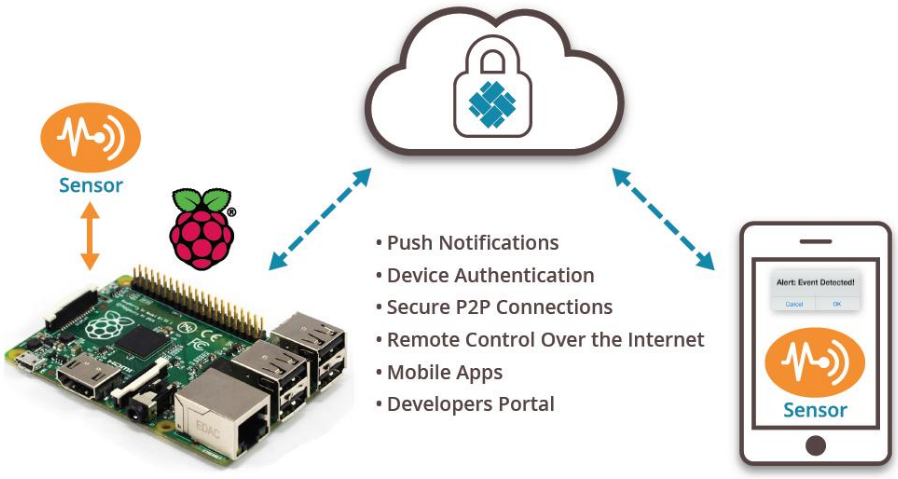
SKILL LEVEL : BEGINNER

Ever since I released WebIOPi in 2012, in Issue 9 of The MagPi, I have received many messages from users asking about accessing the Raspberry Pi over the Internet.

The problem is that typically a Raspberry Pi is connected to the Internet with a home router that shares a single public IP with devices. The router performs network address translation and often includes a firewall. This protects your Raspberry Pi from the Internet, but prevents you from accessing it from a friend's house.

It is possible to get around this problem by configuring port forwarding in your router. However, this depends on configuring the router correctly. Every router is different and it is impossible to provide a set of unique instructions that will suit everyone. Once you have correctly configured port forwarding, you will realise that your public IP can also change. Therefore, you will need a dynamic DNS solution to be able to connect. Each step required is possible, but it can take a lot of time that could be better spent on another project.

I recently discovered the Internet of Things (IoT) Kit from Weaved Inc., a start up in the Silicon Valley. Weaved Inc. provides a free, secure, easy-to-use and efficient way to access your Raspberry Pi from anywhere on the Internet. Just install the Weaved IoT Kit to access your



Raspberry Pi via the Weaved web portal or mobile application.

1. Ensure that the service (WebIOPi, Apache, SSH...) you want to access is running on your Raspberry Pi.

2. Create an account for free on https://developer.weaved.com/

3. Download the Weaved IoT Kit installer on your Pi using a terminal window or SSH connection:

```
wget https://github.com/weaved/installer/raw/
master/binaries/weaved-nixinstaller_1.2.x.bin
```

Where "1.2.x" is the latest version of the installer software on the Weaved website.

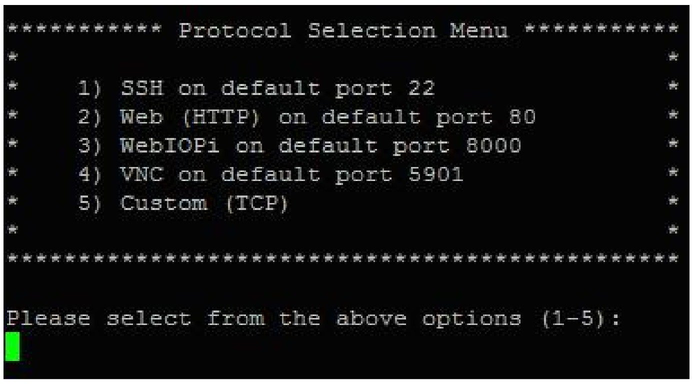
4. Make the installer executable:

```
chmod +x weaved-nixinstaller_1.2.x.bin
```

5. Run the installer:

```
sudo ./weaved-nixinstaller_1.2.x.bin
```

6. Log into the installer using the account you created at step 2.



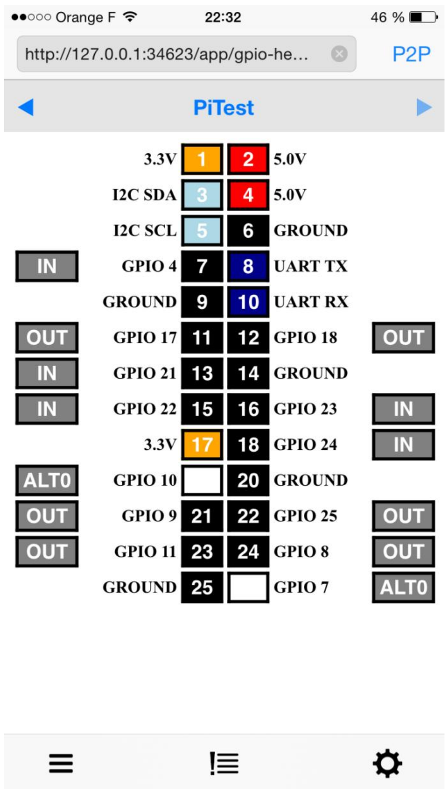
```
*********** Protocol Selection Menu ***********
*                                             *
*     1) SSH on default port 22               *
*     2) Web (HTTP) on default port 80        *
*     3) WebIOPi on default port 8000         *
*     4) VNC on default port 5901             *
*     5) Custom (TCP)                         *
*                                             *
***********************************************

Please select from the above options (1-5):
```

7. Choose the service you want to access, it can be either Apache, SSH, WebIOPi or a custom service.

8. Enter an alias to identify your device and service; the installer will register your device, which can take a while.

9. Connect to your device via https://developer.weaved.com/ or download the Weaved free iOS application. That's it; you're done; no complex configuration.

The mobile application is a convenient way to access the WebIOPi GUI on your Raspberry Pi from a mobile device in a single tap.



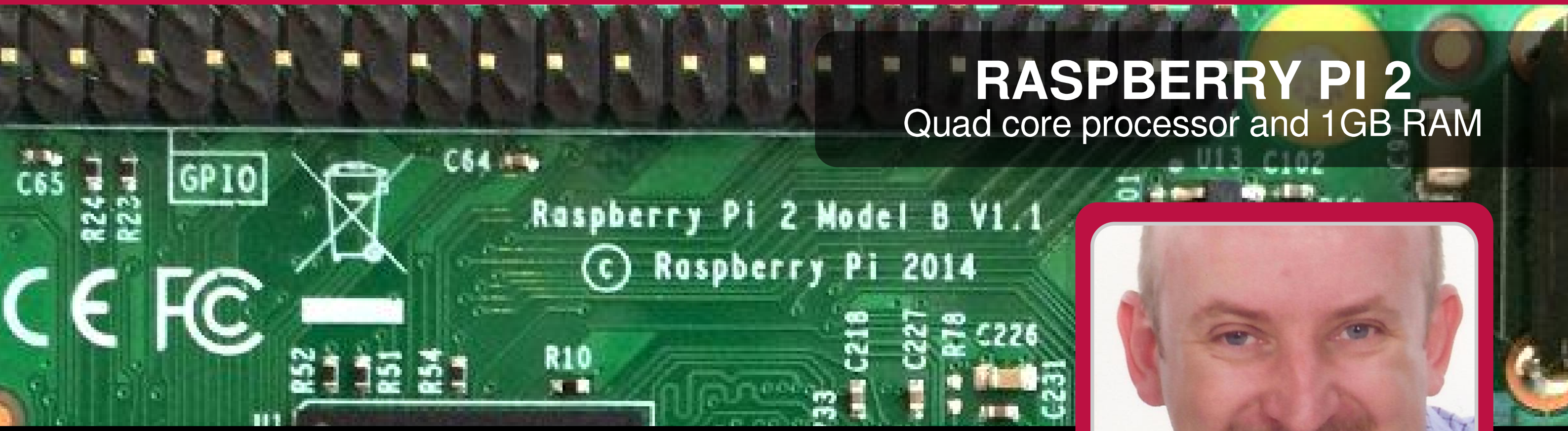
	3.3V	1	2	5.0V	
	I2C SDA	3	4	5.0V	
	I2C SCL	5	6	GROUND	
IN	GPIO 4	7	8	UART TX	
	GROUND	9	10	UART RX	
OUT	GPIO 17	11	12	GPIO 18	OUT
IN	GPIO 21	13	14	GROUND	
IN	GPIO 22	15	16	GPIO 23	IN
	3.3V	17	18	GPIO 24	IN
ALT0	GPIO 10		20	GROUND	
OUT	GPIO 9	21	22	GPIO 25	OUT
OUT	GPIO 11	23	24	GPIO 8	OUT
	GROUND	25		GPIO 7	ALT0

With the Weaved IoT Kit on your Raspberry Pi, you won't have to take care about network configuration or your public IP anymore. If your public IP changes, or your router restarts, then the Weaved service will automatically detect the change, and update the connection to your Raspberry Pi.

You can even move your Raspberry Pi to another location, on another network, and you will still be able to access it with no change.

Weaved IoT Kit also allows you to send Push Notifications from the Raspberry Pi to your mobile phone. Therefore, you could be notified upon an event happening on your Raspberry Pi.

http://code.google.com/p/webiopi/
http://developer.weaved.com



Ian McAlpine

MagPi Writer

Introducing the "game changing" Raspberry Pi 2

New hardware

When the Raspberry Pi Foundation announced the new Raspberry Pi 2, it came as a surprise to many folks. But really it should not have been a surprise. The Raspberry Pi is now 3 years old and in the life cycle of a technology product it was long overdue an update. Indeed, when the Model B+ was launched in August 2014 many were disappointed that increases in processor performance or RAM size did not happen then.

Replacing the BCM2835 (and its 700MHz single core ARM11 CPU) with the BCM2836 (and its 900MHz quad-core ARM Cortex-A7 CPU), plus doubling the RAM to 1GB, is a MAJOR performance update. These have been widely documented so I will not go into the details here, but you no longer have to make excuses for the Raspberry Pi’s (lack of) performance, just because it only costs US$35.

Backwards compatibility

The Raspberry Pi Foundation has always tried very hard to retain as much hardware and software backwards compatibility as possible between revisions. The update between Revision 1 and Revision 2 of the Model B saw some GPIO differences which required software updates. The update between the Model B and Model B+ was much more significant and from a software perspective it was 100% backwards compatible. But the many hardware changes to the layout of the components meant that some accessories no longer fitted and almost every case was rendered useless. However, the update between the Model B+ and the Raspberry Pi 2 is 100% backwards compatible, both with hardware and software. With hindsight it is now clear that the hardware updates introduced with the Model B+ paved the way for the Raspberry Pi 2.

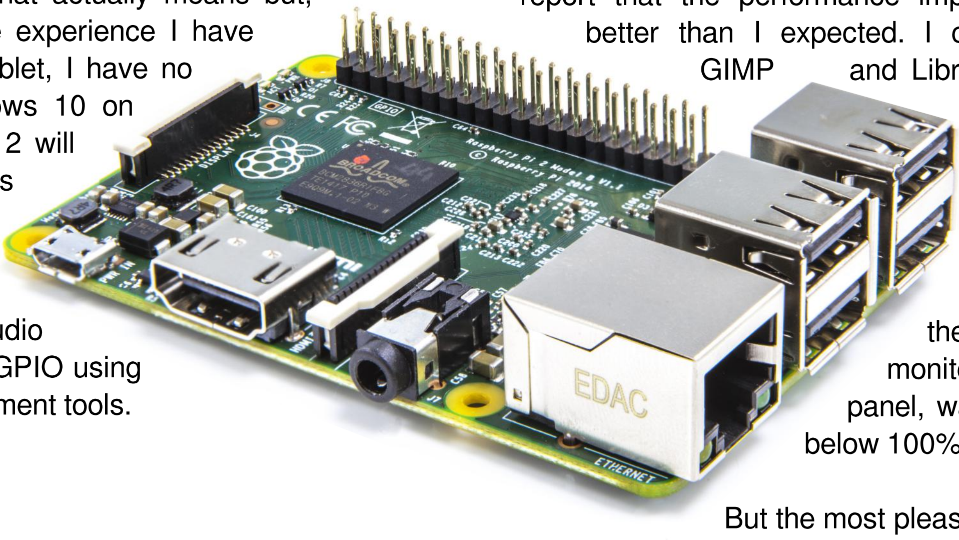
Ubuntu and Windows 10

There were some genuine surprises during the Raspberry Pi 2 launch announcement. First, with its ARM v7 processor, more operating systems become available such as Ubuntu. But later in 2015 there will also be Windows 10 for the Raspberry Pi!. This will be made available to the maker community for FREE through Microsoft’s Windows Developer Program for IoT (Internet of Things). You can sign up for this at http://dev.windows.com/en-us/featured/raspberrypi2support.

The availability of Windows 10 on the Raspberry Pi 2 is a game changer, both for the Raspberry Pi Foundation and for Microsoft. Rightly or wrongly, any computer running Linux is deemed too difficult to use and certainly not for the masses. But if that computer runs Windows then suddenly everything is different. Suddenly you have access to something familiar, something that does not require new training... and if that computer only costs US$35 then it becomes a complete no-brainer. Remember the now defunct US$100 one laptop per child initiative? With its access to Windows and its developer eco-system, the Raspberry Pi 2 could now make that a true reality.

There has been some speculation about just how capable Windows 10 will be on the Raspberry Pi 2. There are several Windows 10 IoT derivatives including "Mobile" for phones, low-end tablets, etc. and "Athens" for resource constrained devices. I have a US$99 tablet with an Intel Atom quad core CPU and 1GB RAM. It runs the FULL version of Windows 8.1 very well.

Steve Teixeira from Microsoft has commented that they are running "real" Windows 10 on the Raspberry Pi 2, but as it is optimised for the Raspberry Pi 2 it does not include the full Windows experience. Time will reveal what that actually means but, given the positive experience I have with my cheap tablet, I have no doubt that Windows 10 on the Raspberry Pi 2 will be equally as good. It will certainly be interesting to explore Visual Studio and program the GPIO using Windows development tools.



How much?

The second big surprise for me is the price. Hands up who would have bought a Raspberry Pi 2 if it had cost US$45? The Raspberry Pi Foundation is predicting sales in 2015 of 3 million Raspberry Pis. IMHO that is a pessimistic figure (especially as the initial batch of 150,000 sold out in 24 hours) and could have been easily achieved even if the price was US$45. That would have provided US$30m (US$10 x 3m) extra funds for the Raspberry Pi Foundation to use for good causes.

I was disappointed that the choice of the Raspberry Pi 2 name breaks the homage to the BBC Microcomputer. With the Compute Module, plus the US$20 Model A+ and US$35 Model B+, the Foundation could have easily introduced a US$45 Raspberry Pi Master. Each of these has a very clear use case - Compute Module for industrial and embedded applications, Model A+ for portable and disconnected applications, Model B+ for everything which does not require a GUI and the Raspberry Pi Master for education and home computing.

But the Foundation did not do either of these things. Instead they priced the Raspberry Pi 2 at US$35 thus instantly cannibalising sales of the Model B+. (If you want a cheap Model B+ now is the time to buy!) It is a brave decision and both the Raspberry Pi Foundation and Broadcom must be applauded for offering such an amazing update while still keeping the price at an incredible US$35.

Real world performance

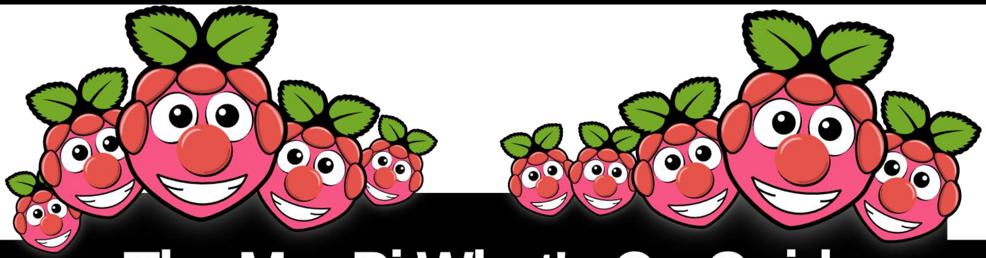
I use my Raspberry Pi almost every day when producing The MagPi. Today I had the opportunity to spend some time with the Raspberry Pi 2 and I can report that the performance improvement is even better than I expected. I can have Scribus, GIMP and LibreOffice all running at the same time, while generating PDFs, and the performance remains great. No more watching the CPU usage monitor on the desktop panel, waiting for it to drop below 100%!

But the most pleasing improvement is the usability of Epiphany. Using a web browser on the Raspberry Pi is something you used to do only if there was no other option... but not any more. With the Raspberry Pi 2, browsing the web using Epiphany is now a pleasure!

Conclusion

The Raspberry Pi 2 is a genuinely usable computer for everyday use. No more excuses. History will record that the Arduino and the Raspberry Pi were instrumental in creating a new generation of innovators, but it will also record that the Raspberry Pi 2 finally made computers a commodity item - accessible to everyone, everywhere.

Huge thanks to **Cana Kit** (http://www.canakit.com) for getting a Raspberry Pi 2 to me in Pacific Canada in an impossibly short period of time.



The MagPi What's On Guide

Want to keep up to date with all things Raspberry Pi in your area?
Then this section of The MagPi is for you! We aim to list Raspberry Jam events in your area, providing you with a Raspberry Pi calendar for the month ahead.

Are you in charge of running a Raspberry Pi event? Want to publicise it?
Email us at: editor@themagpi.com

Mechelen Raspberry Jam

Wanneer: Donderdag 12 februari 2015
Waar: Kanunnik De Deckerstraat 20A, Mechelen, België

Na het enthousiasme van de vorige editie kon een tweede Raspberry Jam niet uitblijven. De activiteiten vereisen géén Raspberry Pi ervaring. http://raspberryjam.be/tweede-raspberry-jam-op-2015-02-12

Northern Ireland Raspberry Jam 9

When: Saturday 14th February 2015, 1.00pm to 5.00pm
Where: Farset Labs, Linfield Industrial Estate, Belfast, BT12 5GH, UK

The Raspberry Jam sessions are both for complete beginners and those looking for more complicated challanges. https://www.eventbrite.co.uk/e/14676410539

Rhône Valley Raspberry Jam

Quand: Dimanche 22 février 2015 de 10:00 à 17:00 (Heure : France)
Où: Foyer Laique, 13 Avenue Jean Jaurès, 84350 Courthézon, France

Journée de découverte et d'échange autour du Raspberry Pi et d'autres technologies numérique. http://www.eventbrite.fr/e/15438067676

HacDC's Raspberry Jam

When: Saturday February 28 2015, 1.00pm to 5.00pm (EST)
Where: 1525 Newton St NW, Washington, USA

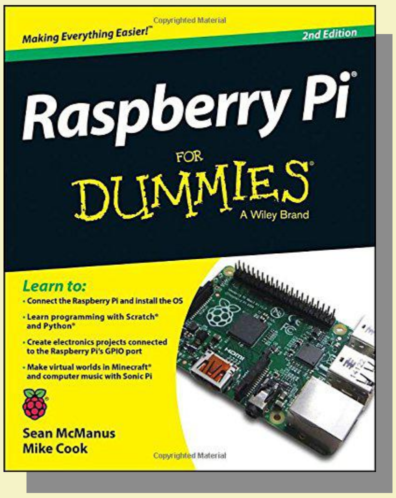
Come to learn and share: several Raspberry Pi computers will be networked together and available for those attending to explore. http://www.meetup.com/hac-dc/events/219990940

Bristol Digimakers

When: Saturday 28th February 2015, 10.00am to 5.00pm
Where: @Bristol, Anchor Road, Bristol, BS1 5DB, UK

A series of technology events aimed at children (7+), teachers and parents. A great opportunity to learn about electronics and computing. http://www.eventbrite.com/e/15201494078

Raspberry Pi For Dummies (2nd Edition)
Sean McManus and Mike Cook
Wiley


Making Everything Easier!
2nd Edition
Raspberry Pi
FOR
DUMMIES
A Wiley Brand
Learn to:
• Connect the Raspberry Pi and install the OS
• Learn programming with Scratch and Python
• Create electronics projects connected to the Raspberry Pi's GPIO port
• Make virtual worlds in Minecraft and computer music with Sonic Pi
Sean McManus
Mike Cook

We first reviewed *Raspberry Pi For Dummies* back in Issue 13 of The MagPi. Eighteen months later and the book has been updated with a second edition. A lot has happened in the Raspberry Pi world during this time and this edition describes many of those changes.

Raspberry Pi For Dummies (2nd Edition) comprises of six parts. Part 1 describes getting started with the Raspberry Pi including an introduction, downloading an operating system and a description of how to connect everything together. It is similar content that you will find in almost every Raspberry Pi book.

Part 2 is where things get much more interesting with an explanation of getting started with Linux. There is a chapter that explains the Desktop Environment followed by a chapter on using the Linux command line. This is particularly useful with a description of many of the common shell commands plus how to keep your system up-to-date. There is even a good description of the Linux directory structure and what each contains.

With Part 3 the book explores common uses of the Raspberry Pi for both work and play. There is a detailed chapter about LibreOffice - a word processor, spreadsheet and presentation software suite that is compatible with Microsoft Office - followed by a chapter on editing photos using GIMP, a very powerful image manipulation program. Part 3 concludes with a chapter on audio and video. This includes setting your Raspberry Pi up as a media centre, probably one of the most popular uses for the Raspberry Pi.

Part 4 describes programming the Raspberry Pi and unsurprisingly it is the largest part in the book. First there is an introduction to Scratch followed by programming an arcade game using Scratch. Next up is a good introduction to Python followed by creating a game using Python and Pygame. Part 4 concludes with chapters on programming Minecraft and Sonic Pi.

One of the big appeals of the Raspberry Pi is the ease that it can be interfaced to physical devices. Many books gloss over this aspect but not *Raspberry Pi For Dummies*. Part 5 is the second largest part in the book and it starts with how to understand circuits and soldering before spending the next three chapters describing the GPIO plus a wide variety of hardware projects.

Dummies books always conclude with *The Part of Tens* and this book is no exception. There is a chapter that highlights ten great programs for the Raspberry Pi plus a chapter that describes ten projects for you to try.

At 436 pages the second edition is 24 pages longer than the original but the changes are much more than the modest page increase suggests. For example, a large chapter on building a website has been removed and replaced with two new chapters - *Programming Minecraft with Python* and *Making Music with Sonic Pi*. Additionally there is an additional Appendix that describes how to use RISC OS plus there is an online bonus chapter about Mathematica. Of course every chapter has been updated appropriately with recent information, such as the use of NOOBS and the latest Raspberry Pi hardware.

Raspberry Pi For Dummies (2nd Edition) is a thorough and excellent book for those who want to maximise their use of the Raspberry Pi.

The MagPi readers are able to receive a 25% discount. To claim, purchase from www.dummies.com and enter the promo code **VBJ92** when prompted.
Please note: this discount is only valid from the 1st February to the 31st July 2015 on the book listed here.

C#

```
static ushort ReadUnsignedWord(int msbAddress, int lsbAddress)
{
    bus.WriteByte(deviceID, (byte)msbAddress); //read the MSB
    byte[] bytes = bus.ReadBytes(deviceID, 1);
    ushort raw = (ushort)((ushort)bytes[0] << 8);
    bus.WriteByte(deviceID, (byte)lsbAddress); //re
    bytes = bus.ReadBytes(deviceID, 1);
    raw |= (ushort)bytes[0];
    return raw;
}
```

Getting started with Mono and I^{2}C devices

Mubarak Abdu-Aguye
Guest Writer

SKILL LEVEL : ADVANCED

Introduction

The first version of the C# programming language was released in January 2002, alongside the Microsoft dotNet framework. This framework which was intended to provide a secure runtime environment in a similar manner as Java and its associated virtual machine. The C# programming language is similar to Java, where the syntax of both languages were inspired by C and C++.

Although the initial releases of the dotNet framework ran exclusively on Microsoft Windows operating systems, the dotNet framework specification was released in 2000, which allowed anyone to implement it on any other operating system. Fast forward a few years later, and Novell came up with the Mono project, which was aimed at providing a dotNet-compatible runtime environment for Linux (initially) and MacOS.

Today, Mono is maintained by Xamarin Ltd and supported on Android, iOS and Xamarin. The current version of the Mono framework available in the Debian Wheezy Linux repositories is 3.2.8, which supports up to dotNet framework version 4.5.

Installing Mono

To be able to run C# (and essentially, any application written in a dotNet language such as VB.NET, F#.NET, etc) on the Raspberry Pi, the Mono runtime environment has to be installed. For the most basic purposes, it is possible to just install the base mono-runtime package. However, for the majority of development activities, it is necessary to install the complete package. Starting from the latest fully updated Raspbian Linux distribution, type:

```
sudo apt-get install -y mono-complete
```

This will install a bundle of development tools and several dotNet libraries. The installation will take several minutes to finish. Once it has finished, the version of Mono installed can be verified by typing

```
mono --version
```

This should return something similar to:

```
Mono JIT compiler version 3.2.8 (Debian
3.2.8+dfsg-4+rpi1)
Copyright (C) 2002-2014 Novell, Inc, Xamarin
Inc and Contributors. www.mono-project.com
TLS:          __thread
SIGSEGV:      normal
```

```
Notifications: epoll
Architecture:  armel,vfp+hard
Disabled:      none
Misc:          softdebug
LLVM:          supported, not enabled.
GC:            sgen
```

Enabling I²C communication

The Linux module that handles I²C communication on the Raspberry is disabled by default in Raspbian Linux. To enable the I²C module, type:

```
sudo raspi-config
```

Then select the Advanced Options and then I²C. Select `Yes`, `Ok`, `Yes`, `Ok`, `Finish` and `No`. This will remove the `i2c-bcm2708` kernel module from the blacklist and load it.

To provide some diagnostic information on the I²C bus, type:

```
sudo apt-get install -y i2c-tools
```

Then enable the `i2c-dev` kernel module, by:

```
sudo nano /etc/modules
```

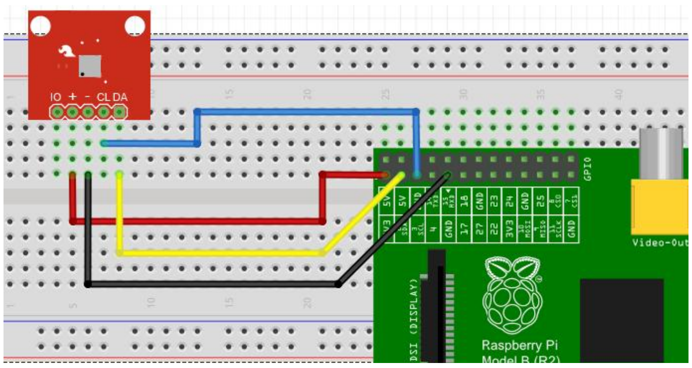
and add then adding

```
i2c-dev
```

to the end of the file. Save the file, shutdown the Raspberry Pi and disconnect it from its power source.

Connecting the BMP180

The BMP180 barometric pressure, temperature and altitude sensor can be purchased from Sparkfun at https://www.sparkfun.com/products/11824 The sensor was connected to the Raspberry Pi using a breadboard and four male-to-male jumper wires, as shown at the top of this page. Pay attention to the power connections, to avoid damage to the sensor and the Raspberry Pi.



Then turn the Raspberry Pi on again and type:

```
sudo i2cdetect –y 1
```

This will run the `i2cdetect` program, which will show us all devices on the I²C bus. The –y part is to override a yes/no prompt by providing a default 'yes' answer, and 1 specifies the I²C bus to be scanned. The I²C bus number depends on the model of the Raspberry Pi, where more information can be found at: http://elinux.org/RPi_Low-level_peripherals

Typing the `i2detect` command should produce

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- 77
```

which reports that a device with address `0x77` has been detected. This device is the BMP180 sensor, where the address is hardcoded at the Bosch factory.

Using I²C with C#

The I²C bus can be accessed from C# by installing Rpi.I2C.Net.dll written by Max Shmelev. To get this package type:

```
git clone https://github.com/mshmelev/RPi.I2C.Net
.git
```

The package contains classes and a supporting shared library called libnativei2c.so. To use

Rpi.I2C.Net.dll on Raspbian, the supporting library should be compiled by typing:

```
cd RPi.I2C.Net/Lib/LibNativeI2C/src
make
```

Now set the LD_LIBRARY_PATH, to include the directory that contains the library.

```
export LD_LIBRARY_PATH=$HOME/RPi.I2C.Net/Lib/LibNat
iveI2C
```

Now the Rpi.I2C.Net.dll library can be built:

```
cd ~/Rpi.I2C.Net/RPi.I2C.Net
xbuild RPi.I2C.Net.csproj
```

where the first command assumes the package was cloned into the users home directory. When the package builds it will produce a warning, but will compile correctly.

When mono runs it looks for .dll libraries in the present working directory and within the `MONO_PATH` directory list. Once the package has been built the `Rpi.I2C.Net.dll` file should be in the `bin/Debug` directory of `RPi.I2C.Net`.

The `libnativei2c.so` and `RPi.I2C.Net.dll` files should be made available to mono, either by editing `~/.bashrc` and appending:

```
i2c_net="$HOME/RPi.I2C.Net/RPi.I2C.Net/bin/Debug"
if [[ -z $MONO_PATH ]]; then
  export MONO_PATH="$i2c_net"
elif [[ $MONO_PATH != *"$i2c_net"* ]]; then
  export MONO_PATH=$MONO_PATH:"$i2c_net"
fi
unset i2c_net
natlib=$HOME/RPi.I2C.Net/Lib/LibNativeI2C
if [[ -z $LD_LIBRARY_PATH ]]; then
  export LD_LIBRARY_PATH="$natlib"
elif [[ $LD_LIBRARY_PATH != *"$natlib"* ]]; then
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:"$natlib"
fi
unset natlib
```

or by copying `libnativei2c.so` and `RPi.I2C.Net.dll` into the present working directory where the mono program is being run.

Test program

To check the I^{2}C library and the circuit, download the test program and run it:

```
wget http://www.themagpi.com/resources/CSharp/Pi_I2
CTest.exe
mono Pi_I2CTest.exe
```

The program will take sixty readings, print "Done" and then close. The output from the program should be something like this:

```
Temperature: 27.9 Pressure: 938.918264411984
Temperature: 27.9 Pressure: 938.933390108141
Temperature: 27.9 Pressure: 939.001455765812
Temperature: 27.9 Pressure: 938.948515806314
```

This program will only work correctly if the BMP180 sensor is connected to the first I^{2}C bus.

Hello World

Programs can be developed on the Raspberry Pi or on Microsoft Windows using the Visual Studio. Some information on development with Visual Studio is given a at: https://erictummers.wordpress.com/2012/01/25/target-mono-from-visual-studio/

To build on the Raspberry Pi, use a text editor such as `nano` or `emacs` and create a file called `HelloWorld.cs` that contains:

```
using System;
public class HelloWorld {
  public static void Main() {
    Console.WriteLine("Hello World!");
  }
}
```

Then compile and run this example, by typing:

```
mcs HelloWorld.cs
mono HelloWorld.exe
```

Building the test program

Download and unpack the program files by typing:

```
wget http://www.themagpi.com/resources/CSharp/Pi_I2C
Test.zip
unzip Pi_I2CTest.zip
```

Similar to the I²C communication library, this package was developed using Microsoft Visual Studio. The C# program files can be found in the `Pi_I2CTest/Pi_I2CTest/Pi_I2CTest` directory. The example program can be built and run by typing:

```
cd Pi_I2CTest/Pi_I2CTest/Pi_I2CTest
xbuild Pi_I2CTest.csproj
mono bin/Debug/Pi_I2CTest.exe
```

The I²C readout program

The Pi_I2CTest program source code can be found in `Program.cs`. This program contains commands to connect to the I²C bus, take data and decode the values.

After the I²C kernel modules have been setup as described earlier in this article, the I²C buses will be visible as Linux devices `/dev/i2c-0` and `/dev/i2c-1`. A connection to the second I²C bus can be made by using

```
I2CBus.Open("/dev/i2c-1");
```

The method returns an I2CBus reference to the chosen bus. This I2CBus class is part of the `Rpi.I2C.Net` namespace. Therefore a using directive is needed. Thus, the code in the `Main()` method is:

```
using System;
//other using directives
using RPi.I2C.Net;

//namespace and class declaration
static I2CBus bus;

static void Main(string[] args) {
  bus = I2CBus.Open("/dev/i2c-1");
```

Using the I2CBus reference, it is possible to read from and write to any devices on the bus. In the code snippet below, the reference is declared with class scope such that it is accessible to other methods in the program.

The BMP180 sensor contains a number of registers, each containing configuration or measurement data. Configuration of the sensor can be performed by writing the appropriate bit combinations to the appropriate registers. Similarly, temperature and pressure measurements can be made by writing certain bits to certain registers, waiting for specified periods of time and then reading the contents of registers designated to contain the results of the measurements.

To explore basic read-write functionality, the program reads the chip ID register of the sensor. This is specified on the datasheet for the BMP180 as being at 0xD0. A correctly functioning BMP180 sensor should contain the value 0x55 in that register. Therefore, to ensure that there is in fact a BMP180 sensor connected to the Raspberry Pi the program:

1) Reads the value in register 0xD0
2) Compares the value read to 0x55

To read a value from any BMP180 register, the address of the desired register should be written to the BMP180 and then a single byte is read back from the sensor. The single byte thus read is the value in the register specified in the preceding write operation. Thus, to complete step 1) above, the program:

i) Writes the value 0xD0 to the BMP180 (which selects that register for the next operation)
ii) Reads a single byte back from the BMP180

To perform (i), the value is written with the `WriteByte()` method of the I2CBus class. This method takes two arguments: the first is the device address (integer) and the second is the byte to be written (byte). The device address for the BMP180 is 0x77 and the byte to be written is 0xD0:

```
bus.WriteByte(0x77, 0xD0);
```

To perform (ii), the `ReadBytes()` method of the I2CBus class is used. This method takes two arguments: the device address (integer) and the number of bytes to read (integer). Unlike the `WriteByte()` method, this method returns a byte array (`byte[]`) containing the bytes read from the device in the order that they were read. In this case, only one byte is being read:

```
byte[] results = bus.ReadBytes(0x77, 1);
```

Assuming that all goes well, the results array should contain a single value that should be 0x55. This can be confirmed by using an if statement, comparing `results[0]` with 0x55. If `results[0]` is indeed 0x55, then a BMP180 sensor is present and functional. If it is not, then perhaps there might be a wiring issue or the sensor is not working correctly.

The complete code for the detection routine is contained in the `Detect()` method in the `Program.cs` file.

Reading the temperature

The BMP180 has certain calibration constants, which are required to calculate the temperature and pressure read from the sensor. These constants are designated as AC1 to AC6, B1, B2, and MB to MD. They are all sixteen-bit values, stored in register pairs starting from address 0xAA to 0xBE. Therefore, AC1's MSB and LSB are stored in 0xAA and 0xAB respectively and so on.

For simplicity, only AC1 shall be considered. To read it, a single byte should be read from addresses 0xAA and 0xAB. Next, the first byte should be shifted by 8 positions to the left, such that it becomes the MSB. Then the resultant value should be OR'ed with the second byte. This corresponds to:

```
bus.WriteByte(0x77, 0xAA); // select the MSB of AC1

// read one byte from 0xAA
byte[] result = bus.ReadBytes(0x77, 1);
```

```
// select the LSB of AC1
bus.WriteByte(0x77, 0xAB);

// read one byte from 0xAB
byte[] result2 = bus.ReadBytes(0x77, 1);

// shift the MSB, store it in a 16bit variable
short finalValue = (short)(result[0] << 8);

// OR the MSB with the LSB
finalValue |= (short)result2[0];
// Variable finalValue now contains the calibration
// constant AC1
```

A similar process is required for the other calibration constants. Therefore, the functionality was written as a method called `ReadSignedWord()`. Note that constants AC4 to AC6 are unsigned, therefore another function is needed to read them. For this reason, another method called `ReadUnsignedWord()` was implemented, within which the variable holding the final result is declared as ushort (unsigned short) as opposed to short (which is a signed type). Finally, a method called `InitParams()` was implemented, which reads and stores all of the calibration constants by using the `ReadSignedWord()` and `ReadUnsignedWord()` methods. The reader is strongly encouraged to read the implementation of these three methods to gain a clearer understanding.

To read the temperature from the sensor, the value 0x2E is written to register address 0xF4 to begin the measurement process. After a delay of 4.5 milliseconds, the result can be read from registers 0xF6 (MSB) and 0xF7 (LSB). The value read from the sensor is unsigned and is converted to a temperature value using the formula given on the data sheet, available from Sparkfun:

$X1 = (UT-AC6) \times AC5/2^{15}$
$X2 = MC \times 2^{11}/(X1+MD)$
$B5 = X1 + X2$
$T = (B5 + 8)/2^{4}$

where UT is the raw value read from the sensor. To write multiple bytes at a time, the I2CBus instance provides a `WriteBytes()` method. This method differs from the `WriteByte()` method in that the second parameter is not a single byte but a byte array (`byte[]`), containing the bytes to be written. This corresponds to:

```
// 0xF4 (address), 0x2E (data)
byte[] bytes = new byte[] { 0xF4, 0x2E };
bus.WriteBytes(0x77, bytes); // address, then data
System.Threading.Thread.Sleep(7); // delay

// Read raw value from source file
long ut = ReadUnsignedWord(0xF6, 0xF7);

// Convert ut to a temperature value
long x1 = (long)((ut - AC6) * AC5/Math.Pow(2, 15));
long x2 = (long)(MC * Math.Pow(2, 11) / (x1 + MD));
long b5 = x1 + x2;
long temp = (long)((b5 + 8) / Math.Pow(2, 4));

temp = temp * 0.1; // convert to degC
```

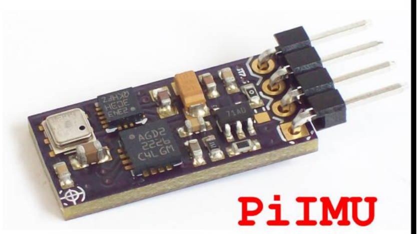
The code above is used in the `ReadTemperature()` method. Try reading the datasheet and comparing it with the code for reading the pressure.

Conclusion

The Rpi.I2C.Net library provides a well-designed, low-level interface to the I^{2}C bus. This makes it possible to write drivers for particular I^{2}C devices with maximal flexibility. A basic BMP180 driver is implemented in the `BMP180.cs` file.

Note also that error checking/exception handling code has been omitted both above and in the driver code. In practice, all calls to the I2CBus methods should be wrapped in try-catch blocks. Finally, the `Dispose()` method should be called on the I2CBus instance once it is no longer required. This explicitly ensures that other programs can access it after your program exits.

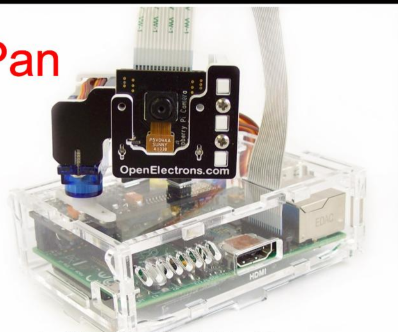
Happy hacking!



PiIMU

10 DOF Gyro Compass Accelerometer & Altimeter for RPi

Pi-Pan



a Pan-Tilt for RPi Camera

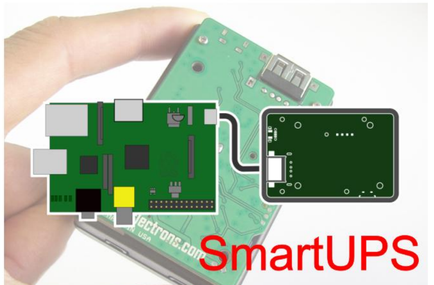
SmartUPS Hacker's Kit: Innovate power for your Pi!

PiConsole

Access RPi Console on your Android or iPhone!

PiServoController

6 Channel Servo Controller for Raspberry Pi



SmartUPS

Uninterruptible power supply for Raspberry Pi

MagPi Special 10% off coupon: MAGPIQE4WE
How to use coupon:
http://www.openelectrons.com/pages/37

OpenElectrons.com

SHORT CRUST plus

www.shortcrust.net

the perfect base for your Raspberry Pi

thanks to RasPi.TV.com for the great pic :)

simple, attractive look to suit any application
high quality, durable PC/ABS and PC materials
easy to use, tooless design with no rattling
expandable with 12mm snap in spacer

now available at

Pi Supply
www.pi-supply.com

Data Streaming & Visualization

See what happened.
Real-time data streaming from your Raspberry Pi with just one line of code. It's that easy.

Interactive data visualizations.
Visualize and mine your data in waveforms, stacked line graphs, and statistics. Data logs streamed in less than 1ms!

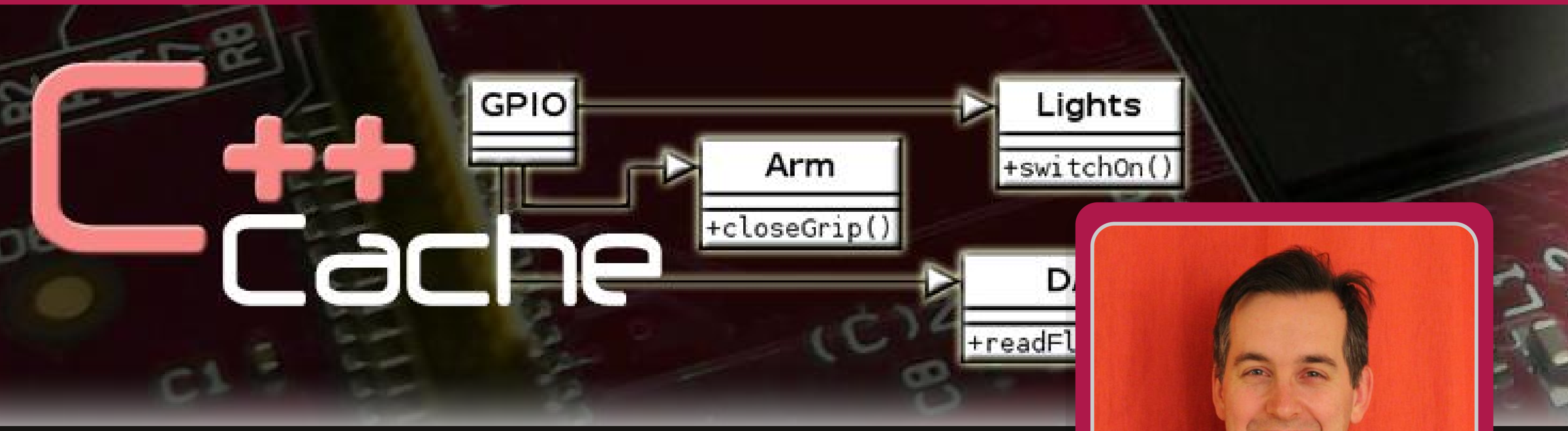
Rest easy, your data is secure.
Every data stream is secured by enterprise-grade SSL encryption from the millisecond you stream it. While many of our users share their data with friends and coworkers, others choose to work privately.

Special Offer for Mag Pi Readers

initialstate.com/magpi

---

initial state



8 - Inheritance and polymorphism

W. H. Bell

MagPi Writer

SKILL LEVEL : ADVANCED

Object-orientated languages such as Java, Python and C++ all allow inheritance structures. The idea of inheritance is that several classes may have a basic set of member functions or data members that are common between them. Rather than implement these members functions or data members several times, they can instead be implemented in a base class. The other classes then inherit from the base class, to include the member functions or data members.

Inheritance structures can include a large number of classes. As the number of classes involved becomes larger, the compilation time will increase and the effects of changing the base class design will be more dramatic. Therefore, inheritance should be used carefully, where the base class design should be as robust as possible. Unlike Java, C++ allows a derived class to inherit from several base classes. However, this feature should be used sparingly, to avoid software from becoming too complicated for other developers to understand.

The syntax and usage of C++ classes is discussed in Issues 23, 24 and 27 of The MagPi. This article builds on concepts introduced in these previous articles. This tutorial assumes that g++ and make have been installed. Using the latest Raspbian image, these can be installed by typing:

```
sudo apt-get install -y g++ make
```

Three layer inheritance

Open a text editor such as nano or emacs and create a file called Bag.h. Then add the C++ code on the top of the next page and save the file. This is the base class declaration and implementation. Since the functionality of the member functions is very simplistic, it has been implemented within the header file. The class contains one data member that holds the volume of the bag. There is a constructor that has a default value. This implies that the constructor can be used with a value or without a value. The two member functions return and set the data member value. The data member is declared as private. This implies that it will not be directly accessible in the derived class. To allow direct access to the data member in the derived class, the data member should be declared as protected or public.

```
#ifndef BAG_H
#define BAG_H
class Bag {
public:
  Bag(double volume=0):  // Constructor, with default value
  m_volume(volume) { // Set the value of the private data member m_volume
  }

  double volume(void){  // Return the volume of this bag object
    return m_volume;
  }

  void setVolume(double volume){  // Set the volume of this bag object
    m_volume = volume;
  }

private:
  double m_volume; // A data member to store the volume
};
#endif
```

Now that a class has been written, another class can inherit from it. Create another file called ColouredBag.h and add the C++ code given below.

```
#ifndef COLOUREDBAG_H
#define COLOUREDBAG_H
// Include the base class.
#include "Bag.h"

class ColouredBag: public Bag {  // ColouredBag inherits from Bag
public:
  ColouredBag(char c='0'): // Set the bag colour
  Bag(),  // Set the volume of the bag to zero
  m_bagColour(c){  // Set the bag colour
  }

  char colour(void) {  // Return the bag colour
    return m_bagColour;
  }

  void setColour(char c) { // Set the bag colour
    m_bagColour = c;
  }

private:
  char m_bagColour; // Variable to store a colour character
};
#endif
```

The ColouredBag class is a simple container class that inherits from the Bag class. This implies that it has one data member within the class declaration and one data member by inheritance. The ColouredBag constructor calls the Bag constructor and sets the value of the data member. There are two accessor functions in the ColouredBag class and two more that are inherited from the Bag class.

Inheritance structures can include several layers of classes. To demonstrate this, create another file called BeanBag.h that contains the C++ code given below:

```
#ifndef BEANBAG_H
#define BEANBAG_H
// Include the base class
#include "ColouredBag.h"

class BeanBag: public ColouredBag {   // BeanBag inherits from ColouredBag
public:
  BeanBag(char colour='0', int beans=0): // Constructor with default values
    ColouredBag(colour),  // Set the value of the colour, by calling the constructor
    m_beans(beans) {  // Set the number of beans
  }

  int beans(void){  // Return the number of beans
    return m_beans;
  }

  void setBeans(int beans){ // Set the number of beans
    m_beans = beans;
  }

private:
  int m_beans;  // The number of beans in the bag
};
#endif
```

The BeanBag class inherits from the ColouredBag class, which in turn inherits from the Bag class. This implies that the BeanBag class includes functions and data members that are present in both base classes. Now create another file called main.cpp and add the C++ code given below. Then compile the main.cpp file using g++ or a

```
#include <iostream>
#include "Bag.h"
#include "ColouredBag.h"
#include "BeanBag.h"
using namespace std;

int main() {
  Bag bag(30.0); // Create an Bag object
  ColouredBag colouredBag;  // Create a ColouredBag object
  colouredBag.setVolume(40.0);  // Set the volume of the coloured bag object
  colouredBag.setColour('r');  // Set the colour character of the coloured bag
  BeanBag beanBag('b');  // Create a BeanBag object
  beanBag.setVolume(50.0);  // Set the volume of the BeanBag object
  beanBag.setBeans(100);  // Set the number of beans in the BeanBag object
  cout << "Volume of bag = " << bag.volume() << endl << endl;  // Print the volume
  cout << "Volume of colouredBag = " << colouredBag.volume() << endl;  // Print the volume
  cout << "Colour of colouredBag = " << colouredBag.colour() << endl << endl; // Print the colour
  cout << "Volume of BeanBag = " << beanBag.volume() << endl; // Print the volume
  cout << "Colour of BeanBag = " << beanBag.colour() << endl; // Print the colour
  cout << "Beans in BeanBag = " << beanBag.beans() << endl; // Print the number of beans
  return 0;
}
```

Makefile as demonstrated in the previous C++ articles. Then run the program to print the values on the screen.

Polymorphism and interfaces

Polymorphism can be used to define interface classes. An interface class is an abstract base class that contains purely virtual functions and a virtual destructor. Purely virtual functions are not implemented within the declaration of the base class. Therefore, when a derived class inherits from the interface class it has to implement the purely virtual functions otherwise it will not compile. Having defined an interface class, an interface pointer can be used to access many different implementations of the interface. This implies that large sections of a program can rely on the presence of the member functions defined in the interface class, without needing to know which implementation of the interface is actually being used. To demonstrate this principle, create a file called IVectorTransform.h and add the C++ code given below:

```
#ifndef IVECTORTRANSFORM_H
#define IVECTORTRANSFORM_H

#include "TwoVector.h"
#include <string>

// A transform interface: a purely abstract base class.
class IVectorTransform {
 public:
  virtual ~IVectorTransform(){};  // The destructor should be implemented in the interface
  virtual TwoVector transform(const TwoVector &) = 0; // Purely virtual
  virtual std::string str() = 0; // Purely virtual
};

#endif
```

This interface class contains two purely virtual member functions, to transform a two-dimensional vector and return the transform as a string. The TwoVector class is given in Issue 27 of The MagPi, where both the TwoVector.h and TwoVector.cpp files are needed. Next, create a file called MatrixTransform.h that contains:

```
#ifndef MATRIXTRANSFORM_H
#define MATRIXTRANSFORM_H
#include "IVectorTransform.h"

// A matrix transformation, inheriting from the IVectorTransform interface
class MatrixTransform: public IVectorTransform {
 public:
  MatrixTransform(double norm=1.0): // With default normalisation
    m_normalisation(norm){  // setting the normalisation of the matrix
    m_matrix[0][0] = 0.; m_matrix[1][0] = 1.;  // The first two matrix elements
    m_matrix[0][1] = -1.; m_matrix[1][1] = 0.;  // the remaining two matrix elements
  }
  virtual ~MatrixTransform(){};  // A virtual destructor
  virtual TwoVector transform(const TwoVector &);  // A virtual transform function
  virtual std::string str();  // A virtual str function to return a string form of the transform

 private:
  double m_normalisation;  // The normalisation of this matrix transformation
  double m_matrix[2][2];  // The matrix that will be multiplied with the 2d vector
};
#endif
```

The MatrixTransform class includes the declaration of the transform and str member functions without the =0. This implies that these functions are virtual, but not purely virtual. Therefore, they are implemented for this class. The class also contains a constructor to set the matrix normalisation and initialise the values used for each of the matrix elements. Create another file called MatrixTransform.cpp and add:

```
#include "MatrixTransform.h"
#include <sstream>
#include <iomanip>

TwoVector MatrixTransform::transform(const TwoVector &twoVector){
  double x = twoVector.x();  // Get the x-component of the 2d vector
  double y = twoVector.y();  // Get the y-component of the 2d vector
  double new_x = m_normalisation * (x*m_matrix[0][0] + y*m_matrix[0][1]);  // matrix mult.
  double new_y = m_normalisation * (x*m_matrix[1][0] + y*m_matrix[1][1]);  // matrix mult.
  return TwoVector(new_x,new_y);
}

std::string MatrixTransform::str(){
  std::stringstream ss;  // Create a string stream instance
  ss << std::setw(4) << m_normalisation << " x ";  // Print the normalisation and matrix
  ss << "| " << std::setw(3) << m_matrix[0][0] << ", " << std::setw(3) << m_matrix[0][1]
     << " |" << std::endl;
  ss << "       | " << std::setw(3) << m_matrix[1][0] << ", " << std::setw(3) << m_matrix[1][1]
     << " |" << std::endl;
  return ss.str();  // Return the string value from the string stream instance
}
```

The transform function performs matrix multiplication between the two-dimensional vector and the 2x2 matrix. The results from this multiplication are then used to create a new TwoVector instance, which is returned by the function. The str function uses a string stream instance to return a string representation of the matrix and its normalisation. To complete the example program, create another file called main.cpp and append:

```
#include "TwoVector.h"
#include "MatrixTransform.h"
#include <iostream>
#include <cmath>

// A program to demonstrate how to use an interface class
int main() {
  TwoVector vec1(3.,4.);  // A 2d vector (x=3, y=4)
  IVectorTransform *t = new MatrixTransform(1); // Create a new transform on the heap
  TwoVector vec2 = t->transform(vec1);  // Transform the vector, to produce a new vector
  std::cout << t->str() << std::endl;  // Print the string form of the transform
  std::cout << "vec1=" << vec1 << ", vec2=" << vec2 << std::endl;  // Print the vectors
  delete t; // Delete the matrix transform pointer
  return 0;
}
```

Then, similar to the article in Issue 27, use a Makefile to compile the .cpp files together to make an executable. When the program is run it will print the values of the two-dimensional vector components before and after the matrix transformation. Notice that the pointer type used to refer to the MatrixTransform object is an IVectorTransform pointer. While the pointer type is the base class, the derived class member functions are called. If the base class member functions were not virtual, then the base class member functions would have been called instead.

Wyliodrin

ARDUINO

Have you tried Wyliodrin yet? It's free!

Use a browser from any device to program and monitor your Raspberry Pi

Program and monitor your Pi from anywhere in the Internet

Use our graphs to display your sensors' data

Just drag & drop blocks to create your applications, using Visual Programming

www.wyliodrin.com

python C++ node js Perl Java ARDUINO php

THE SCRATCH PATCH

BOUNCING SURFACES
Two player arcade game

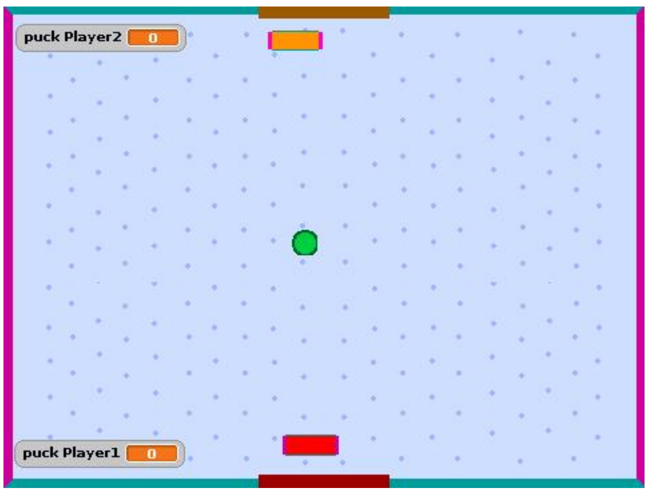
Air hockey arcade game

W. H. Bell

MagPi Writer

SKILL LEVEL : BEGINNER

Many games require sprites to bounce on a surface. While Scratch provides a block that causes a sprite to bounce on the edge of the screen, games often require interactions with other edges. In this month's article, a simple air hockey arcade game is introduced. There are two paddles and two sets of controls, one for each player. If the puck touches the sides of the table or one of the paddles, it will bounce off the surface. The first player to get to seven goals wins the game.

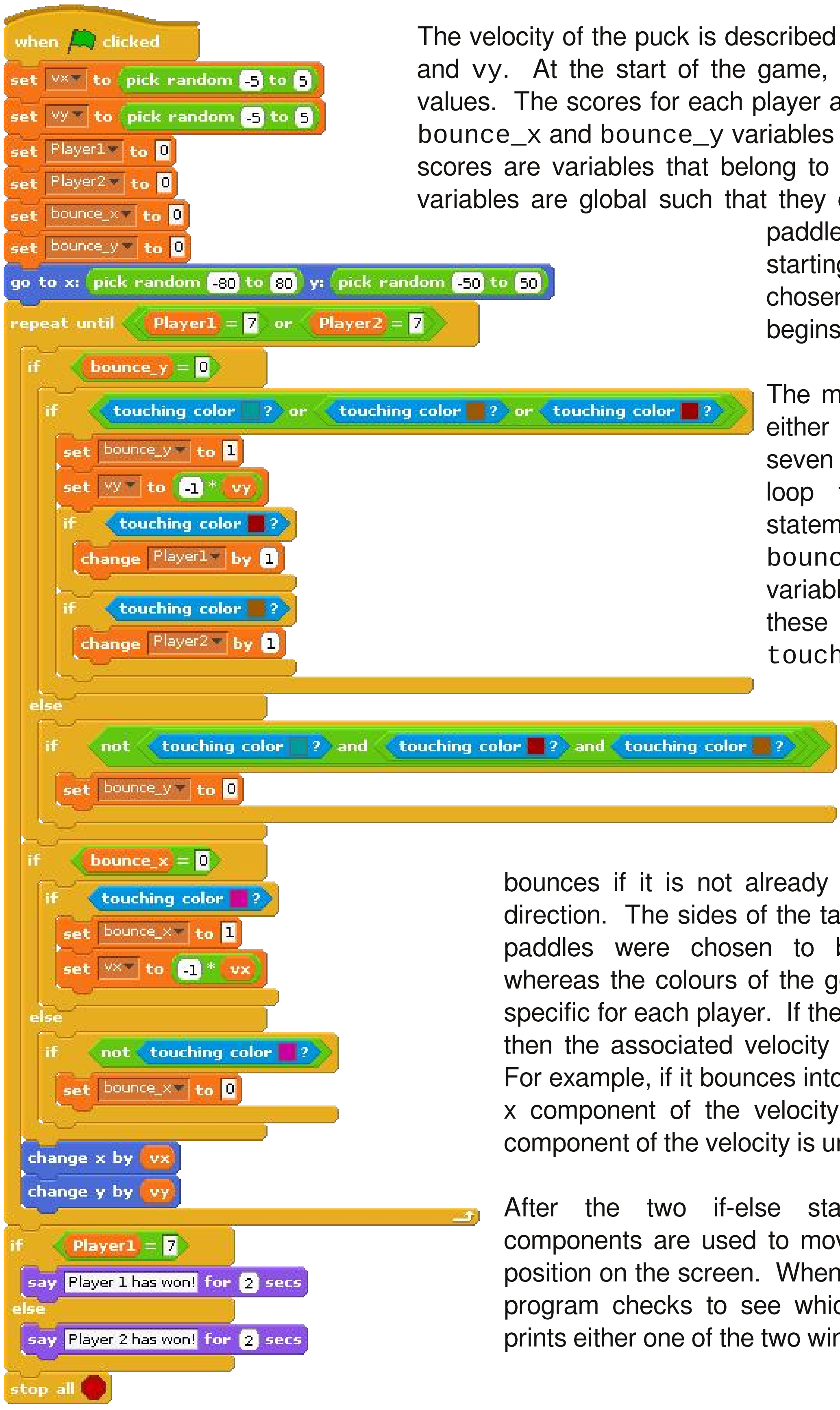


Bouncing

When a sprite bounces off a surface, the surface behaves as a mirror flipping the velocity. The surface can have a perfect bounce where no energy is lost, or a non-perfect bounce where energy is lost as a function of the velocity of the bouncing sprite at the point of impact with the surface. The game in this article includes a perfect bounce, since it is the simplest option the implement.

Bouncing in scratch can be implemented by checking if a sprite touches a specific colour. This implies that games can be constructed without using the position of the bouncing sprite on the screen. When a sprite moves more than one unit of distance at a time the `touching color` sensor can fire twice before the sprite leaves the coloured area. This implies that a simple if statement might cause the velocity to mirror twice and the sprite to fail to bounce correctly.

The puck



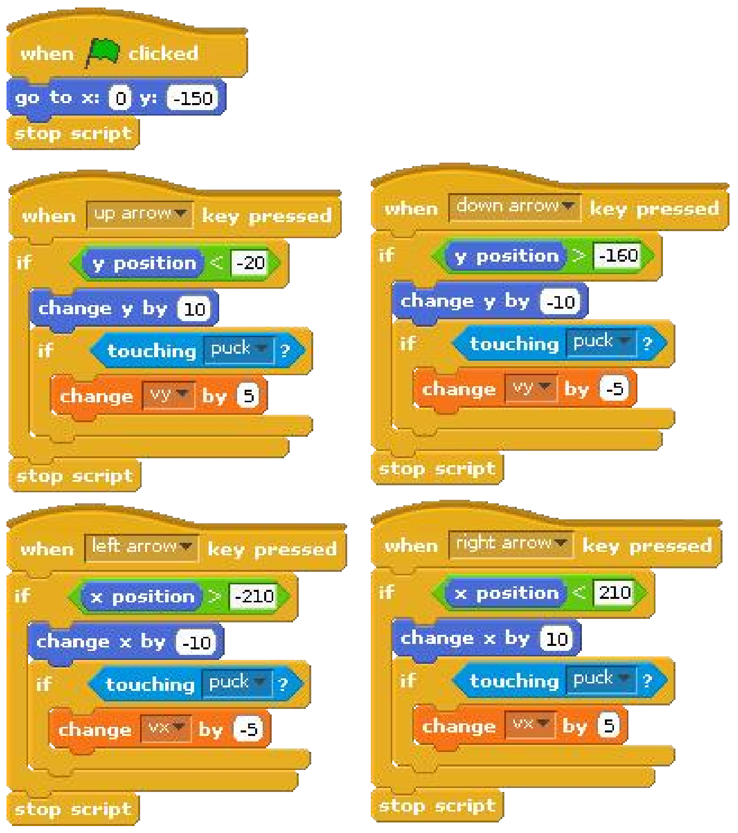
```
when [flag] clicked
set vx to pick random -5 to 5
set vy to pick random -5 to 5
set Player1 to 0
set Player2 to 0
set bounce_x to 0
set bounce_y to 0
go to x: pick random -80 to 80 y: pick random -50 to 50
repeat until <Player1 = 7> or <Player2 = 7>
    if <bounce_y = 0>
        if <touching color ■ ?> or <touching color ■ ?> or <touching color ■ ?>
            set bounce_y to 1
            set vy to -1 * vy
            if <touching color ■ ?>
                change Player1 by 1
            if <touching color ■ ?>
                change Player2 by 1
    else
        if not <touching color ■ ?> and <touching color ■ ?> and <touching color ■ ?>
            set bounce_y to 0
    if <bounce_x = 0>
        if <touching color ■ ?>
            set bounce_x to 1
            set vx to -1 * vx
    else
        if not <touching color ■ ?>
            set bounce_x to 0
    change x by vx
    change y by vy
if <Player1 = 7>
    say Player 1 has won! for 2 secs
else
    say Player 2 has won! for 2 secs
stop all
```

The velocity of the puck is described by two components, `vx` and `vy`. At the start of the game, they are set to random values. The scores for each player are reset to zero and the `bounce_x` and `bounce_y` variables are set to 0. The player scores are variables that belong to the puck and the other variables are global such that they can be modified by the paddle sprite programs. The starting position of the puck is chosen and the main loop begins.

The main loop continues until either one of the players has seven points. Inside the main loop there are two if-else statements that check the `bounce_x` and `bounce_y` variables. The purpose of these tests is to prevent the `touching color` from being true twice due to the sprite moving more than one pixel at a time.

The puck only bounces if it is not already bouncing in the x or y direction. The sides of the table and the sides of the paddles were chosen to be the same colours, whereas the colours of the goals were chosen to be specific for each player. If the puck touches a surface then the associated velocity component is reflected. For example, if it bounces into the right-hand side, the x component of the velocity is mirrored and the y component of the velocity is unaffected.

After the two if-else statements, the velocity components are used to move the puck to the next position on the screen. When the main loop exits, the program checks to see which player has won and prints either one of the two winning messages.



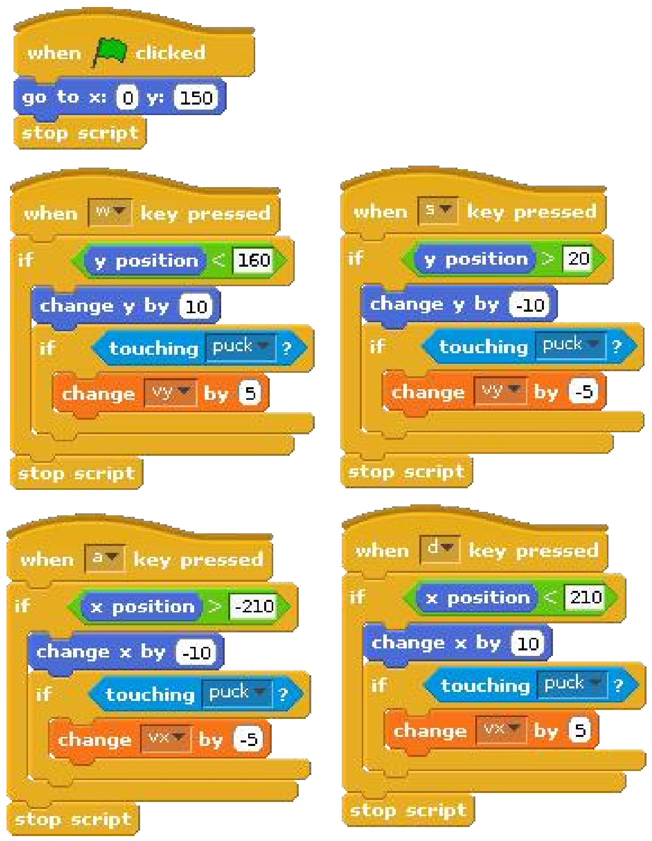
```
when [flag] clicked
go to x: 0 y: -150
stop script

when up arrow key pressed
if y position < -20
    change y by 10
    if touching puck ?
        change vy by 5
stop script

when down arrow key pressed
if y position > -160
    change y by -10
    if touching puck ?
        change vy by -5
stop script

when left arrow key pressed
if x position > -210
    change x by -10
    if touching puck ?
        change vx by -5
stop script

when right arrow key pressed
if x position < 210
    change x by 10
    if touching puck ?
        change vx by 5
stop script
```

The paddles

Two paddle sprites were created, one for each player. To help the players recognise which paddle is which, the bulk of the two paddles are coloured differently. The first player is assigned the cursor keys to move the paddle around and the second player is assigned w, s, a and d. When the green flag is clicked, the two paddles move to their default positions along the x-axis.

In a normal game of air hockey, each player is not allowed to reach into the other half of the table. Therefore, the paddles are not allowed to be moved to the other side of the table. The paddles are also not allowed to be moved off the side of the table. These restrictions are implemented by adding an if statement to each of the paddle control scripts.

When a moving object hits another object, the object it hits is given an impulse. For example, when a ball is hit with a bat the ball is sent into the air. If the bat is not moving, then the ball will just bounce off it. To simulate the effect of an impulse, the velocity component of the puck is incremented according to the direction the paddle is moving when it is hit by the puck.



```
when [flag] clicked
go to x: 0 y: 150
stop script

when w key pressed
if y position < 160
    change y by 10
    if touching puck ?
        change vy by 5
stop script

when s key pressed
if y position > 20
    change y by -10
    if touching puck ?
        change vy by -5
stop script

when a key pressed
if x position > -210
    change x by -10
    if touching puck ?
        change vx by -5
stop script

when d key pressed
if x position < 210
    change x by 10
    if touching puck ?
        change vx by 5
stop script
```

Possible extensions

Try changing the speed and effect of the impulse of the paddles. Try starting the game without an initial puck velocity, where the puck is in range of one of the two players.

Bouncing can be used for several different games. For example, a vehicle that drives from the left to the right of the screen could bounce over the surface of some terrain.

Expand your Pi

Stackable Raspberry Pi expansion boards for the A+ and B+

Serial Pi Plus

RS232 serial communication board. Control your Raspberry Pi over RS232 or connect to external serial accessories.

Breakout Pi Plus

The Breakout Pi Plus is a useful and versatile prototyping expansion board for the Raspberry Pi A+ and B+.

ADC Pi Plus

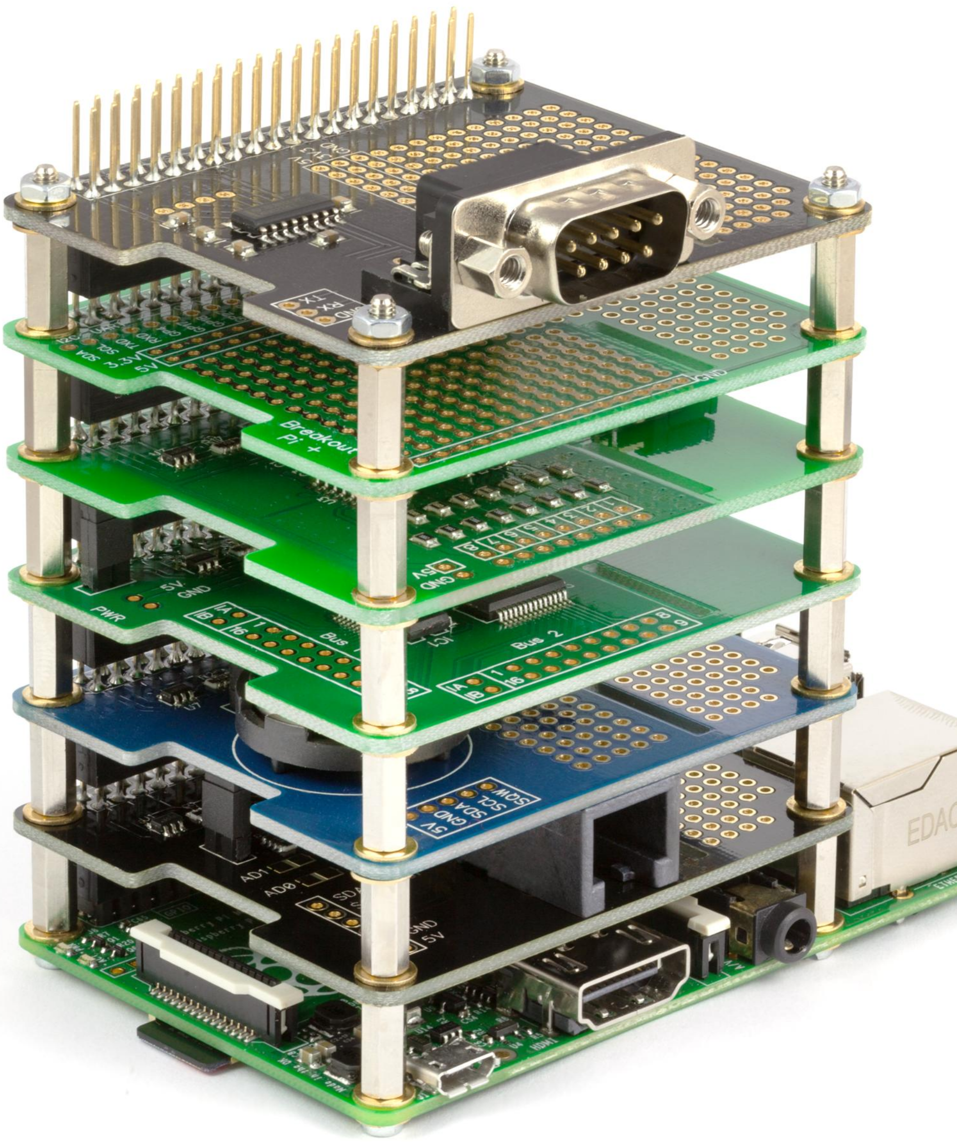
8 channel analogue to digital converter. I²C address selection allows you to add up to 32 analogue channels to your Raspberry Pi.

IO Pi Plus

32 digital 5V inputs or outputs. I²C address selection allows you to stack up to 4 IO Pi Plus boards on your Raspberry Pi.

RTC Pi Plus

Real-time clock with battery backup and 5V I²C level converter for adding external 5V I²C devices to your Raspberry Pi.



1 Wire Pi Plus

1-Wire® to I²C host interface with ESD protection diode and I²C address selection.

We also stock a wide range of expansion boards for the original Raspberry Pi models A and B.

AB electronics UK

www.abelectronics.co.uk

```
class Factorial:
    def __init__(self,n):
        self.n = n
        self.fact = 0
        self.count = 0
    def next(self):
        if self.count > self.
            raise StopIterat
        self.fact *= self.count
        if self.fact < 1:
            self.fact = 1
        self.count+=1
        return self.fact
```

MAZE BUILDER

written in Python

Simply Amazing !

SKILL LEVEL : BEGINNER

Martin Meier

Guest Writer

My education in programming truly began when I traded my hard-earned money for a copy of “The Creative Atari”. Sure, I knew my way around the BASIC programming language, but so did most kids back in the mid-eighties. Similar to most kids back then, I was far more interested in typing in the games than I was in actually learning something. Luckily for me, this new tome was not just a collection of few BASIC programs. Instead, each programmer had been thoughtful enough to include a write up with his contribution that not only explained the nuts and bolts of the code, but took us deep into the development process. These articles were written in such a way that it stuck with me for the next thirty years.

Maze Master

“Maze Master” was a fun little offering by Fred Brunyate, which would give the player a new maze each time it was run. Having a limited supply of quarters had already prodded me to attempt to program my own games. A program that generated a random maze with only one path from start to finish would give me lot to work with. Even better, was that Fred’s explanation gave me everything I needed to adapt this concept to an entertaining dungeon-crawler game. Years later, while learning C, I wrote a version of the maze builder in C. Now learning Python, I repeated the process and produced another maze builder. Clearly, Fred had provided me with a fun little exercise to encourage me while learning new programming languages. This article contains the Python version, such that that others may benefit.

The ideas Fred provided are pretty straight-forward. Start at any point within a two-dimensional array. Mark the block you are standing in, so you cannot move back into it. Now look at the four squares around you, and see which ones you can move into. Once you have made that list of potential moves, pick one at random. Repeat that process until there is nowhere left for you to move. Once you have boxed yourself into a dead end, simply choose a new position in the array that you have already been in and continue moving. Choosing a location, in which you have already been, insures that all the maze branches are connected together. Repeat this process until all the squares in the array have been tagged.

In Python

Python is a great choice for this exercise. If you look at the program listing, you can see that I have divided it up into three main parts: `build_maze`, `print_maze`, and `main`. Starting with `main`, we can see that the first thing that happens is that the program asks the user how big the maze should be. With that information, the programme then initialises the two-dimensional array of the requested size to all '0's.

The main loop in the programme first checks to see if there are any cells left in the array that have not been moved into. If there are still some available, it makes a list of possible moves (left, right, up, and down) based on what is around the cell it is in. The program then makes a random choice based on the list it has compiled. Now things begin to get a little confusing. The same array that keeps tracks of whether or not a cell has been moved into, also keeps track of how the walls on each cell are set up. This program will only concern itself with the lower wall and the right wall of each cell, with the upper wall being handled by the cell above and the left wall being defined by the cell to the left. A value of 4 means that both the lower and the right-hand wall of a cell are in place. A value of 1, means that both the lower wall and right-hand wall are absent. Values of 2 and 3 correspond to only the right-hand wall or lower wall respectively. Based on which random direction the program has decided move in, it will update both the cell it is currently in and the cell it moves into.

The last part of the program is `print_asci_maze`. All this does is print out the completed maze, based on the array it is fed. Back in my Atari days, I had solid block graphics that made excellent walls and intersections. As I am still feeling out Python's capabilities, I have decided to just use the characters +, -, and | for now.

Hopefully, you have enjoyed this exercise as much as I have. More importantly, I hope that any budding programmers will come away with an understanding of the application of a simple concept that can be applied to help you learn new languages. Keep on coding!

```
Maze Builder. Inspired By
    "The Creative Atari" 1983


What should the width be (2-10)?10
What should the depth be (2-10)?5
Your maze will be 10 by 5
Am building a 10 by 5 maze
+s+-+-+-+-+-+-+-+-+-+
|   |     |   |     |
+-+ +-+-+ +-+ + +-+ +
| |       |   | |   |
+ + +-+-+-+ + + + +-+
| |         | | | | |
+ + +-+-+-+ + + + + +
|   |   |   |   | | |
+-+ + +-+ + +-+-+ + +
|   |     |     |   |
+-+-+-+-+-+-+-+-+-+f+
```

```
Maze Builder. Inspired By
    "The Creative Atari" 1983


What should the width be (2-10)?10
What should the depth be (2-10)?5
Your maze will be 10 by 5
Am building a 10 by 5 maze
+s+-+-+-+-+-+-+-+-+-+
| |     |       |   |
+ +-+-+ +-+ +-+ + + +
|       |   | |   | |
+-+-+-+ + +-+ + +-+ +
|     |   |   | |   |
+-+-+ +-+-+ +-+ +-+-+
|     |     |       |
+ +-+-+ + +-+ +-+ + +
|       |     |   | |
+-+-+-+-+-+-+-+-+-+f+
```

where S is the start and F is the finish. Every time the program is run, the maze is different!

```
#!/usr/bin/env python
import random


def build_maze(maze):
  deep = len(maze)
  wide = len(maze[0])
  print "Am building a",wide,"by",deep,"maze"

  x = 0
  y = 0
  maze[y][x]= 4
  total = wide*deep
  blocks_tagged = 1

  while (blocks_tagged < total):
    options=[]
    if (x > 0):
      if (maze[y][x-1] == 0):
        options.append("left")
    if (x < wide-1):
      if (maze[y][x+1] == 0):
        options.append("right")
    if (y > 0):
      if (maze[y-1][x] == 0):
        options.append("up")
    if (y < deep-1):
      if (maze[y+1][x] == 0):
        options.append("down")

    if (options):
      choice_index = random.randrange(len(options))
      choice = options[choice_index]
      blocks_tagged +=1

      if (choice == 'down'):
        if (maze[y][x] == 3):
          maze[y][x] =1
        if (maze[y][x] == 4):
          maze[y][x] =2
        y = y + 1
        maze[y][x] = 4

      if (choice == 'up'):
        y = y -1
        maze[y][x]=2

      if (choice == 'right'):
        if (maze[y][x] == 4):
          maze[y][x] = 3
        if (maze[y][x] == 2):
          maze[y][x] = 1
        x = x+1
        maze[y][x] = 4

      if (choice == 'left'):
        x = x -1
        maze[y][x] = 3
    else:
      while(1):
        y = random.randrange(deep)
        x = random.randrange(wide)
        if (maze[y][x] > 0):
          break

def print_asci_maze(maze):
  deep = len(maze)
  wide = len(maze[0])
  right_wall = (0,2,4)
  bottom_wall = (0,3, 4)
  # print top border
  row = "+s+"
  for index in range (0,wide-1):
    row +="-+"
  print row

  for index1 in range (0,deep):
    # print upper half of row
    row = "|"
    for index in range (0,wide):
      if (maze[index1][index] in right_wall):
        row += " |"
      else:
        row += "  "
    print row
    # print lower half of row
    row = "+"
    for index in range (0,wide):
      if (index == wide-1) and (index1==deep-1):
        row +="f+"
      else:
        if (maze[index1][index] in bottom_wall):
          row +="-+"
        else:
          row += " +"
    print row


# The main function
def main():
  print  """\n\nMaze Builder. Inspired By
  "The Creative Atari" 1983\n\n"""

  width = 0
  depth = 0
  while (width < 2) or (width > 10):
    width = int(raw_input("What should the width be (2-10)?"))
  while (depth < 2) or (depth > 10):
    depth = int(raw_input("What should the depth be (2-10)?"))
  print  "Your maze will be",width,"by",depth

  # set up a list of lists referenced as maze[row][column]
  maze = [0 for index in range (0, depth)]
  for index in range (0, depth):
    maze[index] = [0]*width

  build_maze(maze)
  print_asci_maze(maze)

if __name__ == '__main__':
  main()
```

A Challenge !

The author has deliberately kept this script simple, using text characters to draw his maze implies that the code is very portable. However, MagPi readers are all hackers at heart, our valued editor would love to see what they can do with this code. Instead of poking things like "-+" into the variable, 'row', how about using some of the very many unicode characters ?

As a hint, try firing up your python environment and typing "print unichr(0x2588)", don't like that "character" ? Try 0x254b. Nearby numbers produce similar characters. There are a lot of alternatives scattered across the unicode set.

We may be able to convince the Editor to publish the best versions so please send your effort into articles@themagpi.com

PRINT EDITION AVAILABLE WORLDWIDE

The MagPi is available for FREE from http://www.themagpi.com, from The MagPi iOS and Android apps and also from the Pi Store. However, because so many readers have asked us to produce printed copies of the magazine, we are pleased to announce that printed copies are now regularly available for purchase at the following Raspberry Pi retailers...

Americas — adafruit INDUSTRIES

EMEA — MODMYPI, Authentic SWAG Educating since '12, The Pi Hut, Pi Supply

AsiaPac — AusPi Technologies Australia

Have Your Say...

The MagPi is produced by the Raspberry Pi community, for the Raspberry Pi community. Each month we aim to educate and entertain you with exciting projects for every skill level. We are always looking for new ideas, opinions and feedback, to help us continue to produce the kind of magazine you want to read.

Please send your feedback to editor@themagpi.com, or post to our Facebook page at http://www.facebook.com/MagPiMagazine, or send a tweet to @TheMagP1. Please send your article ideas to articles@themagpi.com. We look forward to reading your comments.