ISSUE 25 - JUL 2014

Get printed copies at themagpi.com

The MagPi

A Magazine for Raspberry Pi Users

Custom Autopilot
PiBot Robotics
WiFi Sniffing
Time Lapse
MouseAir
Fish Dish
BitScope
BASIC
JAVA

Enigma Cipher

ISSN 2051-9990

http://www.themagpi.com

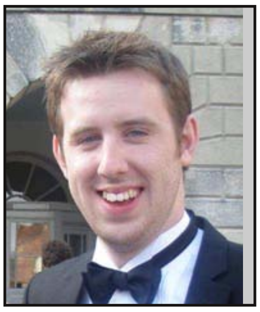
Welcome to issue 25,

This month's MagPi contains a wealth of different material. There is a review of Raspberry Pi powered autopilot technology, the first of a mini-series from the PiBot robotics team, data acquisition with BitScope and more automation projects. We are also very pleased to host a review of the FishDish electronics board, written by our youngest guest author so far.

With standard peripherals such as the Raspberry Pi camera, a lot of interesting projects can be implemented. For example, time lapse photography is introduced this month. The Raspberry Pi, installed with Linux, provides many network diagnostic tools. Following last month's WiFi sniffing article, some more networking tips and tricks are discussed.

The MagPi is devoted to providing programming articles without the need for additional hardware. This month, Java classes are introduced and we are please to provide the first in a series of articles on the BASIC programming language. Completing this month's programming content, a full Python version of the Pocket Enigma Cipher machine is given. The Pocket version uses wheels, similar to the original Enigma machine shown on the cover.

If you have a Raspberry Pi, it would be great to hear how you have used it. We are looking for feedback or contributions, from young authors to seasoned professionals.



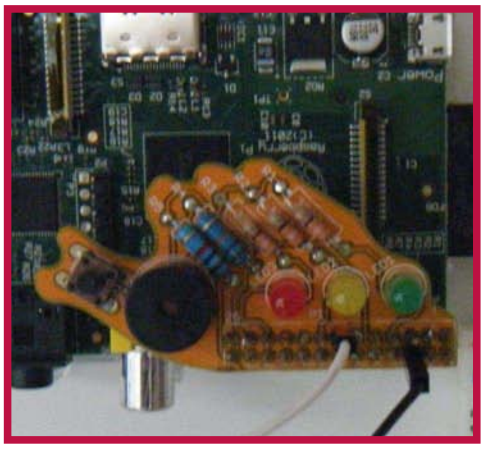
[illegible]

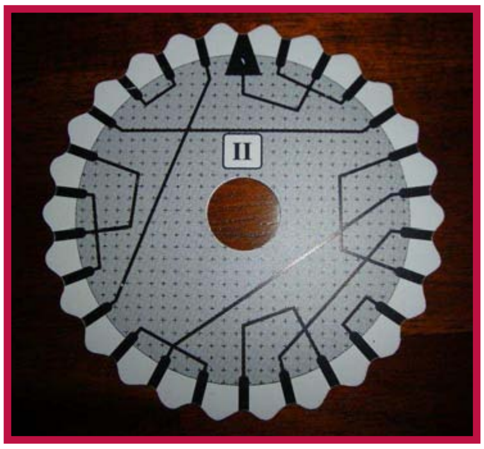
Chief Editor of The MagPi

The MagPi Team

Ash Stone - Chief Editor / Administration / Layout
Les Pounder - Issue Editor / Testing / Proof Reading
W.H. Bell - Layout / Administration
Bryan Butler - Page Design / Graphics
Ian McAlpine - Layout
Matt Judge - Website / Administration
Aaron Shaw - Administration / Proof Reading
Nick Hitch - Administration / Layout
Colin Deady - Layout / Proof Reading
Dougie Lawson - Testing
Tim Cox - Proof Reading
Paul Carpenter - Testing
Nick Liversidge - Layout / Proof Reading
Claire Price - Proof Reading
Chris 'tzj' Stagg - Testing
Rita Smith - Proof Reading

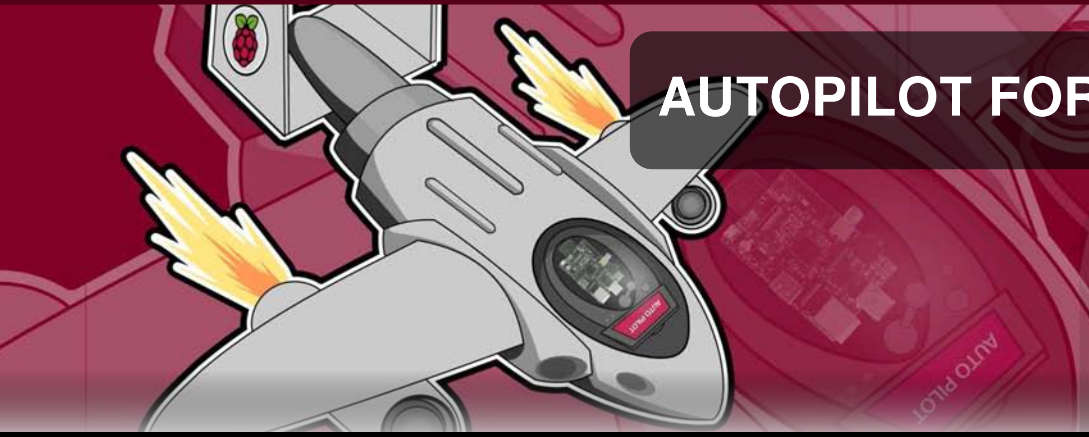
Contents





Cover photo courtesy of Bletchley Park, ©shaunarmstrong/mubsta.com

http://www.themagpi.com



The Navio autopilot shield

Igor Vereninov

Guest Writer

SKILL LEVEL : ADVANCED

There is no need to explain all of the features of the Raspberry Pi and how it is making embedded development easier. When working with the Raspberry Pi platform, we did however find that autopilot applications are some what limited and thus we established the Navio project to overcome this.

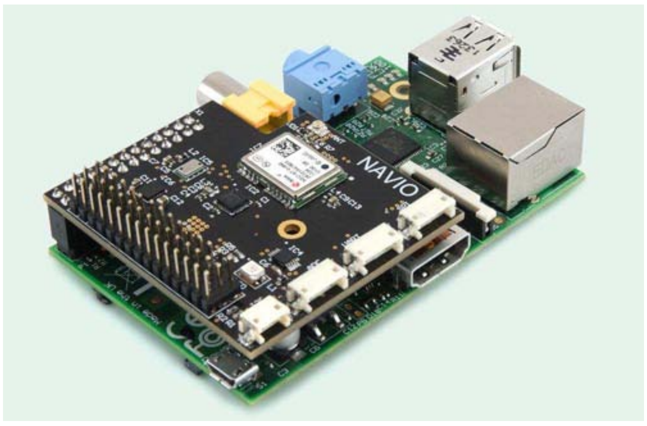
Prior to using the Raspberry Pi, a lot of time was invested in building a commercial UAV autopilot based on the STM32 microcontroller, the same microcontroller that is used in most autopilots. When development of the core software was completed our focus became integration of new payloads, i.e. a new means of communication and new sensors. We found this process very complicated, having to write drivers for each new piece of electronics, even though we knew that they already existed. Development and debugging tools were bulky and kept us tied to the workplace. It was from these issues that the idea of a Linux autopilot first crossed our minds.

At first Navio started as a hobby project at weekends. With progression of the project and increasing group confidence we decided to quit our jobs to work on the Linux autopilot full time.

We started investigating the options available to create an open and powerful Linux autopilot platform, deducing the Raspberry Pi as the obvious choice. We received great support from the community, finding a lot of readily available code and tutorials combining all of the power and flexibility of Linux. When we first sent data over a 3G network we just couldn't believe how easy it was, compared to our previous setup. It was just like on a desktop PC. Simply install the drivers and you are ready to go!

Autopilot is much more than just a processor running an OS. It needs a way to determine its state, its place on Earth, and be able to control actuators. It takes many sensors and servos to fly a plane or multicopter. We needed a 9DOF inertial measurement unit for orientation, GPS for position, barometric pressure sensor for altitude and multiple control outputs. We thus returned to the drawing board, well technically not to a drawing board, but to CAD software, and designed the Navio. In the process an ADC was added for voltage and current monitoring, an RGB LED to show statuses and connectors were made compatible with existing hardware.

Navio sits on top of the Raspberry Pi GPIO header and is fixed with a standoff and bolt. It met our specification of being very compact, making it possible to fit in tight spaces inside small drones, whilst packing everything needed for a full featured autopilot. For our own plane we have chosen to remove some of the sockets on Raspberry Pi to save weight and reduce size.

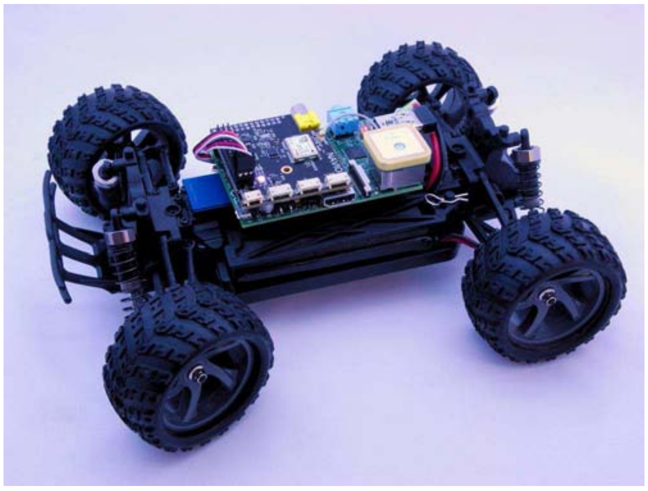


When we first presented the idea to the community, the feedback was great. A Raspberry Pi autopilot actually sounds sweet. It was quickly drawn to our attention that the autopilot has to be strict real-time, and it is not something you expect from Linux straight out of the box. We started experimenting. From the team's previous experience with low latency audio processing on Linux, we knew that it can work real-time if tuned the right way.

With a real-time kernel and proper task priority settings we have outperformed what we had on STM32. We are able to run inertial measurement unit code at 3000 times per second, including reading data from sensors, computing roll, pitch and yaw angles and outputting them through Wi-Fi to console. Even when running alongside computational heavy processes like compiling, the performance was stable. This assured us that there is no need to overcomplicate the setup with a secondary processor for sensor reading.

In our opinion, the Raspberry Pi with Navio is not a commercial autopilot product, but something that lets you evaluate your ideas and test new algorithms. The key factor was the ability to run multiple applications with autopilot in parallel. The other feature of Raspberry Pi is the ease of development; just open your favourite code editor, compile the file and run the application.

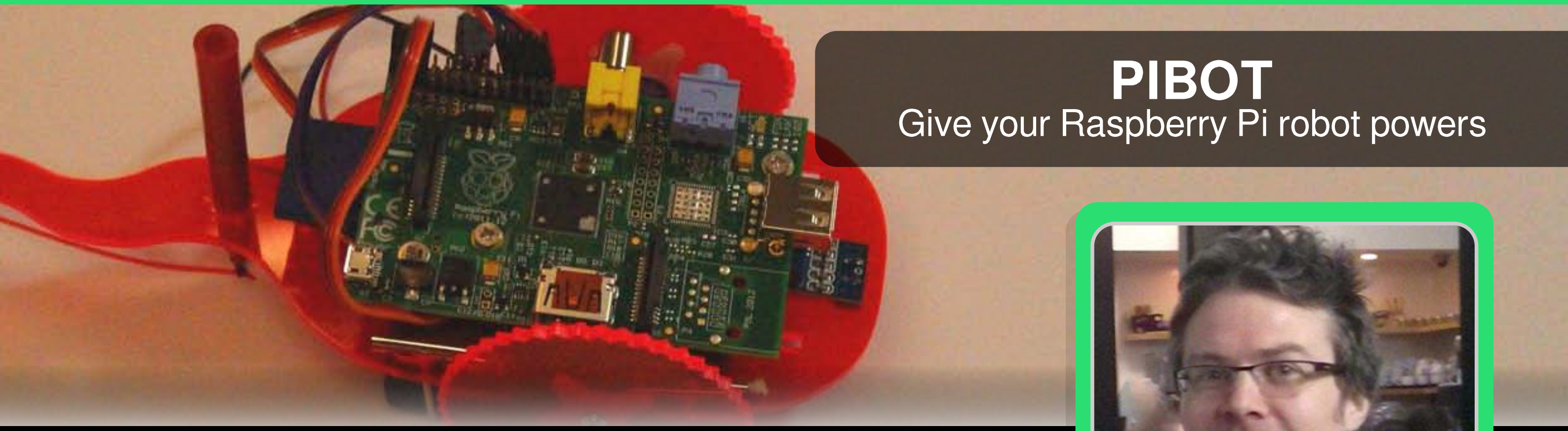
With hardware and OS complete we moved to the autopilot code itself. At first we trialled code to check if the sensors and peripherals were working as expected. When we got Navio to control servos it was time to try it on something real. We took an RC car off the shelf, installed Navio on top and added webcam and Wi-Fi. After several hours of coding we were able to control it from a laptop keyboard and to get live video streaming on screen! Immediately we took a tour around the house, feeling like secret service ROV operators. We revealed some corners that have not been vacuumed for years, and even found the place where second socks go.



We then identified that we needed software to actually fly a plane, but we did not want to reinvent the wheel. From our previous experience it was clear that there is no quick way to write an autopilot. It takes a lot of testing and working on the details to actually make a reliable system. We all know how many open source autopilots are out there, it was just a question of picking the right one.

We were following the ArduPilot project from the very beginning, and when we found out that the team is working on a Linux port, we decided that we should make it run on our platform. Mostly this requires small changes to the hardware specific parts, something we are actively working on. The test platform is already assembled, and we are sure that we are going to take flight soon!

Navio: http://www.emlid.com
ArduPilot: http://ardupilot.com



PIBOT
Give your Raspberry Pi robot powers

Learn the fundamentals of robotics - Part 1

SKILL LEVEL : BEGINNER

Harry Gee

Guest Writer

Introduction

The robot revolution is coming. Robots are no longer just machines from science fiction. From self-driving cars, to flying drones, robots are on the march. Over the next few years robots are going to be seen all over the place and will be increasingly used in agriculture, manufacture, medicine and education, as well as in our own homes. The amazing thing is that now almost anyone can become a roboticist and if you have a Raspberry Pi you are already half way there.

In the first part of this two part article, I will give you an introduction to the exciting new world of robotics and will detail all the things you need to consider when embarking on building your own robot. As an example we will cover the building of the simplest robot possible for the Raspberry Pi. Next time, in part 2 of the article, I will cover more advanced robots and show how you can build and then program these robots to do some really interesting things.

Before we get into the what and how-to of any robotics, the first question may be why would you want to build a robot with a Raspberry Pi in the first place? Also, what kind of things will a Raspberry Pi robot be able to do?

I first got interested in Raspberry Pi robots when I realised how good they can be for learning about technology and also the teaching of technology to others. The great thing about robots is that they are physical and immersive. Instead of providing output to pixels on a screen, a robot is in your personal space and its movements, lights and sounds go way beyond the limits of a screen. With the Raspberry Pi and some low cost hardware, not only can you make a robot move, you can make a robot that can speak, dance and a whole lot more besides! Why wouldn't you want to turn a Raspberry Pi into a robot?

What makes the Raspberry Pi so good for robotics (as well as so many other projects) is its special GPIO port (General Purpose Input Output). This allows the Raspberry Pi to connect to all kinds of electronics and hardware. The fundamental requirements for the most interesting robots are the ability to both sense and interact with their environment, and it is the GPIO port of the Raspberry Pi that makes this possible.

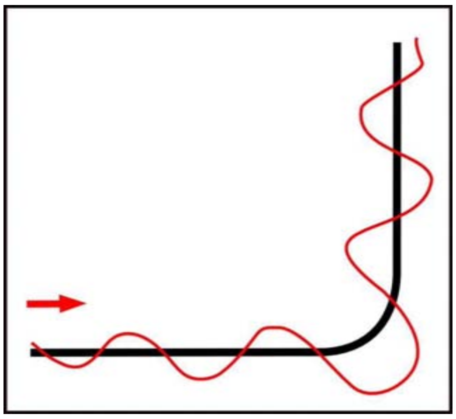
Before going into further details for building a simple robot, let's first consider some robot fundamentals.

What makes a robot robotic?

Wikipedia defines a robot as "*a machine which can be electronically programmed to carry out a variety of physical tasks or actions*". In addition to this requirement to perform physical actions, a proper robot should also have an autonomous ability. Autonomy allows a robot to act independently in the world and this makes them different from things like radio controlled vehicles that are not able to function by themselves. The Raspberry Pi's processing ability (both CPU and GPU), along with its GPIO port, gives it lots of potential for developing autonomous behaviour. As a simple example let's take a look at the classic line following robot.

A line following robot has the ability to move forward, change direction and detect a line on the ground. This is a classic example of autonomous behaviour. No matter where the line leads, the robot is programmed to follow the line without the need for any external control.

For our simple robot we are going to add an infrared (IR) sensor that detects a change in reflectivity. The reflectivity of the ground changes when the robot moves off the line. When this happens it changes direction to get back on the line.



While autonomous behavior can be very useful in a robot it can also be a lot of fun to control it directly. Some of the most interesting applications happen when a robot combines the two. Imagine a flying robot that can fly around at your command but can also be programmed never to crash into walls and other obstacles.

Anatomy of a Raspberry Pi robot

Let us go through all the things you need to build your own robot using a Raspberry Pi. I will detail a minimal possible Raspberry Pi robot step by step. These basic steps are:

- Remote access to the Raspberry Pi
- Powering a Raspberry Pi without cables
- Building a robot chassis
- Making the robot move
- Adding sensors

The first hurdle to turning a Raspberry Pi into a robot is to untether it from all wires and cables.

Remote access

Most people will be familiar with interfacing a Raspberry Pi to a monitor/TV, keyboard and mouse, but this is just not going to work for a robot. The best thing for connecting remotely is a wireless dongle connected to the Raspberry Pi USB port. I won't go into all the details for doing this here. Instead, visit http://pihw.wordpress.com/guides/guide-to-remote-connections/.

Another interesting thing to note is that you can either connect wirelessly across an existing network or you could turn your Raspberry Pi into a wireless hotspot and connect to it via its own network. See Issue 11 of The MagPi, page 10 (http://www.themagpi.com/issue/issue11/article/turn-your-raspberry-pi-into-a-wireless-access-point) for details.

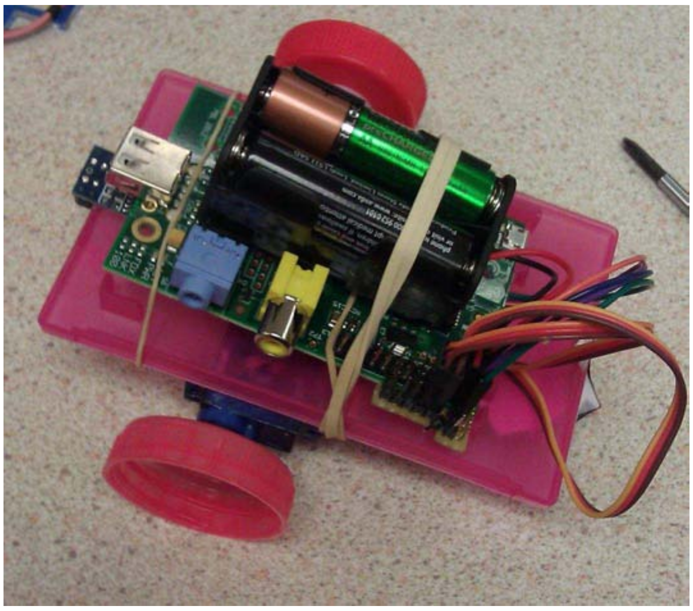
Rather than using the Raspbian Desktop on the robot, a terminal interface is more suitable. When communicating across the wireless network, a remote SSH terminal session is the best way to send commands. Details are in the previously mentioned guide to remote connections.

Portable robot power

The next essential for the robot is to get the Raspberry Pi running on portable electrical power. The simplist way to do this is to use a backup USB mobile phone charger. Personally I opted for a basic 4 x AA battery pack with a 5V voltage regulator. If you are using a Model B then a 8 x AA battery pack may be preferable to get longer running time.

Building a robot chassis

While it is possible to use some kind of existing chassis to build your robot, it can also be fun to make your own. Even some stiff cardboard can be used for this. The photo overleaf shows a basic design.



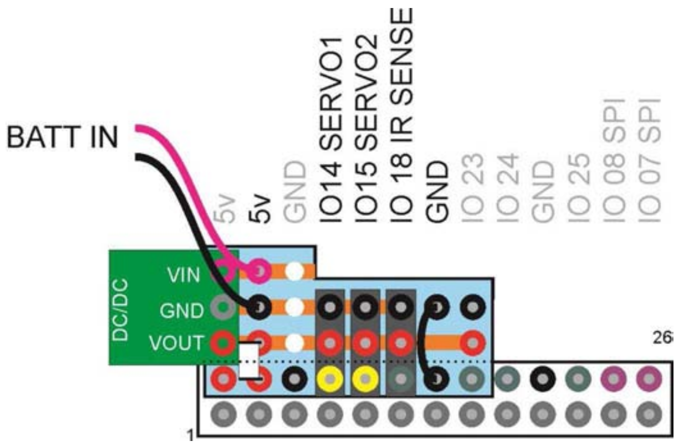
Making the robot move

Now that you have successfully untethered your Raspberry Pi and can connect to it remotely, we can give it some robot powers - motion! The simplest and probably cheapest way to give your robot motion is by connecting two **continuous rotation** servos.

Normally servos are designed to turn just 90 or 180 degrees though modified versions exist that provide continuous rotation to be able to drive wheels. Adafruit show how to modify a low cost servo for continuous rotation (http://learn.adafruit.com/modifying-servos-for-continuous-rotation). If you want to be really inventive you can also make your own wheels. In the photo above, two large bottle tops are being used for wheels.

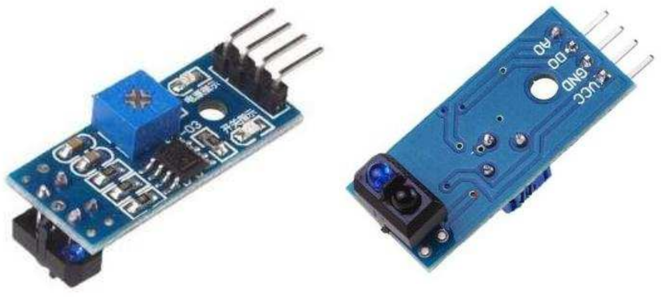
With two wheels to drive the robot a third wheel is usually required for support. It is positioned in the middle towards the back of the chassis. Ball bearing castors or furniture wheels can work well but to keep things really simple a smooth sliding support can be made using a paperclip.

Now that the robot has wheels the next important step is to connect the servos to the GPIO port. As a hack I made up a simple circuit. This connects the data pins of the servos and the IR sensor to the GPIO and also connects the positive and negative power pins of the servo to a 5V voltage regulator. The circuit is shown on the right.



Making your robot sensational

To keep things simple we will only add an IR sensor to our robot. The way this works is that it has one LED that is an IR emitter and one that is an IR receiver. The receiver measures the light reflected from the ground and the amount received will depend of the reflectivity of the surface. A line that has a different reflectivity to the ground around it can be measured by the sensor so it knows whether it is on or off the line.



A suitable sensor can be obtained from http://www.banggood.com (search for LM393 IR).

Coding autonomous behaviour

Now that we have the hardware of the robot sorted, it is time for the code. The Python script shown on the next page is all that is required to add both remote control and autonomous line-following behaviour to our robot. A detailed explanation of the code is available at http://www.pibot.org/tiddlybot/code.

Beyond the basics

Next month we will look at some more capable robots with features like speech, voice recognition and environment mapping. We will also discuss adding more hardware via an Arduino interface.

In this first part I hope you've become more familiar with the basics of building a robot and hope you agree that building robots with the Raspberry Pi is affordable and fun.

To help make it easier for Raspberry Pi users to get into building robots I have now developed the "TiddlyBot". This brings the ideas described in this article to an easier "build-your-own" robot kit. For "TiddlyBot" I have also added some exciting software as well as things like line drawing capabilities. To find out more and to support the "TiddlyBot" project, please see our **Kickstarter** (http://kck.st/1pOgU1J) until 26 July 2014.

With the Raspberry Pi anyone can now be a roboticist and I hope this helps people to both learn and have fun!

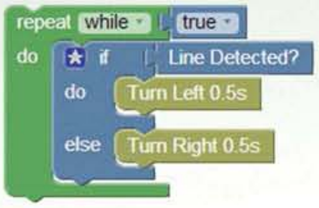
```
import RPi.GPIO as GPIO
from RPIO import PWM
import time
import curses
import sys

left_motor = 14
right_motor = 15
line_sensor = 18

GPIO.setmode(GPIO.BCM)
GPIO.setup(line_sensor, GPIO.IN)

servo = PWM.Servo()

def display_input(stdscr, kboutput):
  stdscr.clear()
  stdscr.addstr(kboutput)
  stdscr.refresh()
  stdscr.move(0, 0)

def turn_movement_servos(left_speed,
right_speed):
  servo.set_servo(left_motor, 1520 + (-1 *
left_speed))
  servo.set_servo(right_motor, 1520 + (1 *
right_speed))

def check_on_line():
  if GPIO.input(line_sensor) == GPIO.LOW:
    on_line = True
  else:
    on_line = False
  return on_line

def kb_input(stdscr):
  k = 0
  in_line_follow = False
  try:
    stdscr.nodelay(1)
    while True:
      c = stdscr.getch()

      ## Remote control button actions
      if (c != -1) and (in_line_follow is
False):
        if (c == 261) and (c != k): # right
          k = c
          turn_movement_servos(0, 80)
          display_input(stdscr, "Right")
        elif (c == 260) and (c != k): # left
          k = c
          turn_movement_servos(80, 0)
          display_input(stdscr, "Left")
        elif (c == 259) and (c != k): # up
          k = c
          turn_movement_servos(100, 100)
          display_input(stdscr, "Forwards")
        elif (c == 258) and (c != k): # down
          k = c
          turn_movement_servos(-100, -100)
          display_input(stdscr, "Backwards")
        elif (c == 10) and (c != k): # enter
          k = c
          turn_movement_servos(0, 0)
          display_input(stdscr, "Stopped")
        elif (c == 113) and (c != k): # quit
          display_input(stdscr, "QUITING!")
          time.sleep(0.5)
          RPIO.cleanup()
          sys.exit()
        elif (c == 32) and (c != k): # space
          k = c
          in_line_follow = True
          line_following()
          display_input(stdscr,"Follow Line")
        else:
        if k != c:
          d = "ASCII "+ str(c) +" not valid!"
          display_input(stdscr, d)

      ## Line following button actions
      elif c == 32:
        in_line_follow = False
        stop()  # stop wheels turning
        display_input(stdscr, "Stopped")
      elif c == 113:
        display_input(stdscr, "QUITING!")
        time.sleep(0.5)
        RPIO.cleanup()
        sys.exit()
      else:
        if in_line_follow:
          line_following()
          display_input(stdscr,"Follow Line")

  except curses.error:
    curses.wrapper(kb_input)

def line_following():
  if check_on_line():
    turn_movement_servos(100, 40)
  else:
    turn_movement_servos(40, 100)
  time.sleep(0.1)

## Start program
curses.wrapper(kb_input)
```

Introducing..

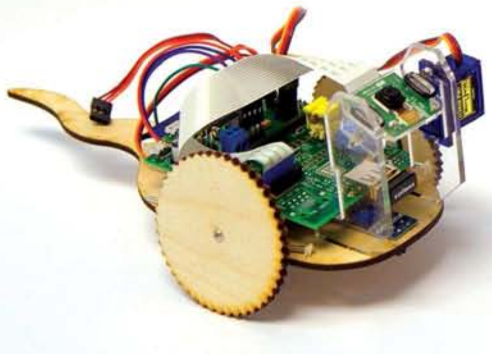
tiddlybot

The Awesome new Robot for the Raspberry Pi

Featuring a Blockly Interface to control Tiddles with any smartphone or tablet



```
repeat while ▾ true ▾
do  if  Line Detected?
    do    Turn Left 0.5s
    else  Turn Right 0.5s
```



TiddlyBot can: Draw, Follow Lines & Play Games + more!

Discover TiddlyBot Only on:

KICKSTARTER

Get the kit for as little as **£26** or **$44**!

Search 'TiddlyBot' on www.kickstarter.com

http://kck.st/1pOgU1J

See what the **buzz** is about and **support** the **project**!

VISIT US ON

WORLD'S MOST VERSATILE CIRCUIT BOARD HOLDERS

Model 209
VACUUM
BASE PV JR.

Model 201
PV JR.

- Work-holding tools for electronics projects
- Circuit board holders make soldering easy & fun
- Versatile hobby vises for any project

Model 207
VISE BUDDY JR.

PANAVISE®
Innovative Holding Solutions

www.panavise.com

PIMORONI

PANAVISE IS AVAILABLE AT
http://shop.pimoroni.com

Andrew Suttle

Guest Writer

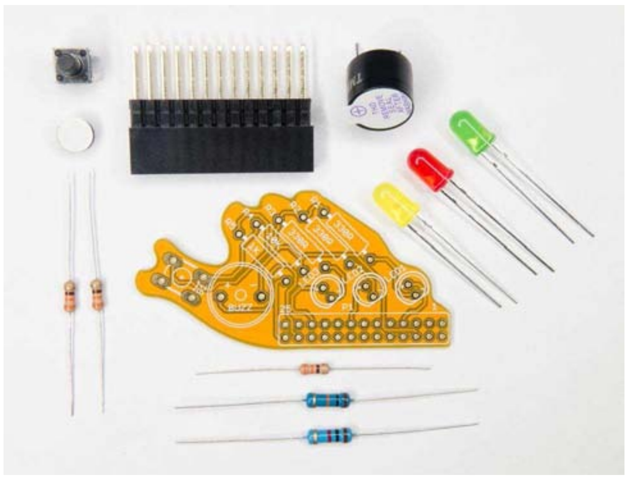
A Review of the Fish Dish

SKILL LEVEL : BEGINNER

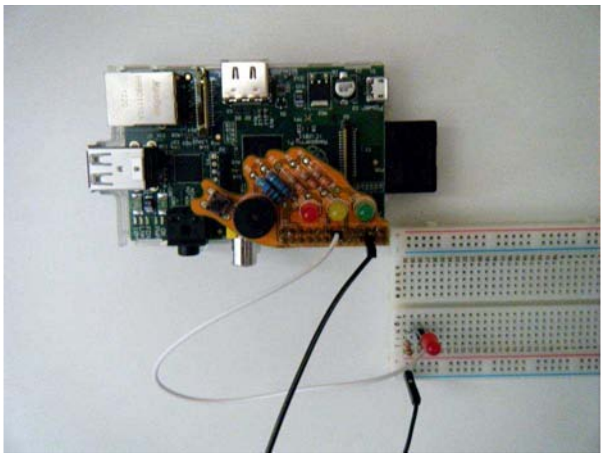
The Fish Dish (http://www.pi-supply.com/product/fish-dish-raspberry-pi-led-buzzer-board) is a little circuit board, which is simple to build by yourself and ideal for children learning about the GPIO. Children please ask an adult to help with the soldering – I accidentally melted part of the connector, but it still works fine!

The circuit board has three LEDs, one buzzer and a switch. There is a Python program available from the internet, but for children, Scratch is better. I only had normal Scratch on my Raspberry Pi, but it will not use the GPIO pins. I downloaded ScratchGPIO, as introduced in issue 9 and 10 of the MagPi and documented at http://cymplecy.wordpress.com/scratchgpio/scratch-raspberrypi-gpio/ . This put a new Scratch on my desktop.

I could not copy the pin numbers from the Python program as they are different in Scratch GPIO and the allon command only turned on two LEDs and not the buzzer. I tried all the pins and worked out that if you turn them on then the allon command works. I made a simple program that turned all on and then all off when the button was pushed. I also ran a scratch program and the Python program at the same time, which was interesting as you can see what happens when two programs think they are controlling the same thing.



Fish bits to make a Fish Dish



Fish Dish with Pi and side order of LED

The **right arrow** code [1] adjusts the brightness of the red LED I connected to pin 13. It gets brighter then dimmer and then turns off. It will work for any LED and even for the buzzer, which can be very annoying if you do it for a long time. Just change the **power13** to **power7** for Green, **power15** for Yellow and **power21** for Red. The buzzer is on pin 24.

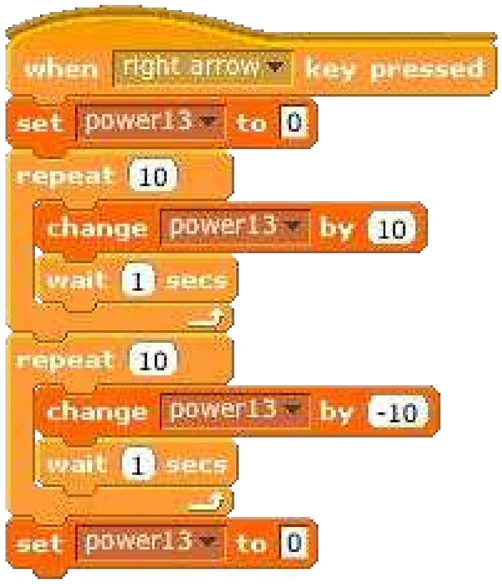
When green flag clicked [2] is a program that checks when the button is pressed on the Fish Dish. The button is on pin 26. It turns all the LEDs plus the Buzzer on when the button is pressed.

I used the **up arrow** key [5] to turn everything off and the space key [3] to turn the LEDs on and off. I also added a tricky code [4] to **pin13off** to confuse my brother.

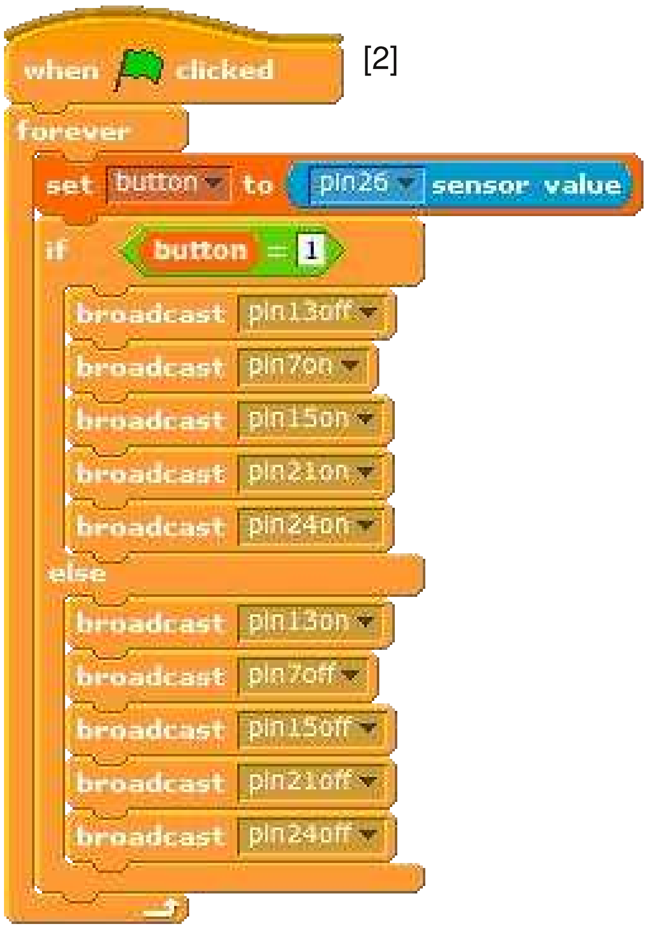
There are also some left over pins to add extra LEDs and I later added a red LED and 1k resistor on pin 13. The extra LED I put on pin 13 turns off when the button is pressed so I knew it worked.

If you add on a circuit to the pins that have the LEDs on then the LEDs tell you if your program works, so if your circuit does not work you can find the problem. I used a 1k resistor for my external circuit and perhaps the resistors on the Fish Dish could be 1k as well.

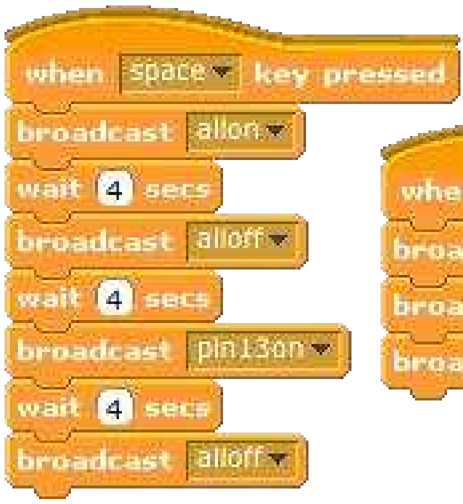
The Berryclip https://www.modmypi.com/berry-clip-raspberry-pi-add-on-board?filter_name=berryclip is nearly the same as the Fish Dish, but you cannot add extra LEDs. However, it is £3 cheaper and it comes with more LEDs. I recommend the Fish Dish for children aged 6+, who are learning about Scratch GPIO, but it should cost less.


[1]

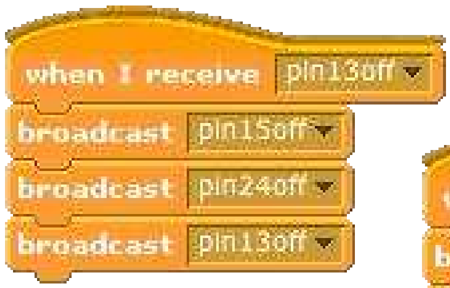
```
when right arrow key pressed
set power13 to 0
repeat 10
    change power13 by 10
    wait 1 secs
repeat 10
    change power13 by -10
    wait 1 secs
set power13 to 0
```


[2]

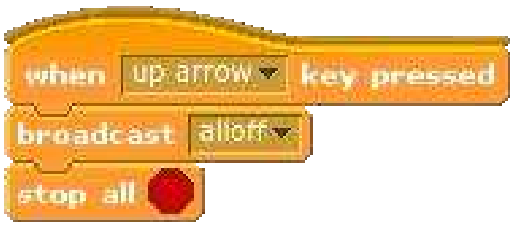
```
when clicked
forever
    set button to pin26 sensor value
    if button = 1
        broadcast pin13off
        broadcast pin7on
        broadcast pin15on
        broadcast pin21on
        broadcast pin24on
    else
        broadcast pin13on
        broadcast pin7off
        broadcast pin15off
        broadcast pin21off
        broadcast pin24off
```


[3]

```
when space key pressed
broadcast allon
wait 4 secs
broadcast alloff
wait 4 secs
broadcast pin13on
wait 4 secs
broadcast alloff
```


[4]

```
when I receive pin13off
broadcast pin15off
broadcast pin24off
broadcast pin13off
```


[5]

```
when up arrow key pressed
broadcast alloff
stop all
```

MouseAir

A Control Panel for Cat Entertainment

John Shovic

Guest Writer

Building a Control Panel

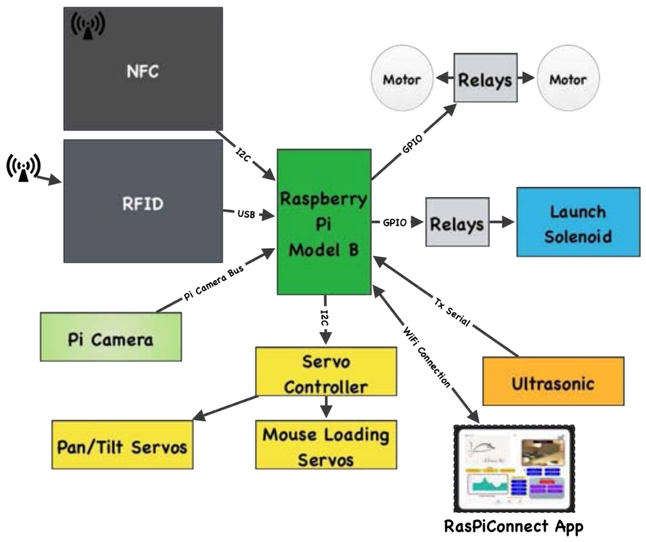
SKILL LEVEL : INTERMEDIATE

Introduction

I have always enjoyed building control panels for my projects. I used to laboriously build physical control panels with switches, meters and lights. Lots of lights. MouseAir is a complex project. 4 Servo Motors, 3 Sensors, 4 relays, 2 DC motors, one mother-of-all solenoids, one camera and one obnoxious cat, all working in sequence. All this to launch toy mice across the room.

I like to be able to change parameters, change modes and cause actions (such as entertaining a cat), whether sitting near the MouseAir launcher or across the world.

MouseAir System Diagram

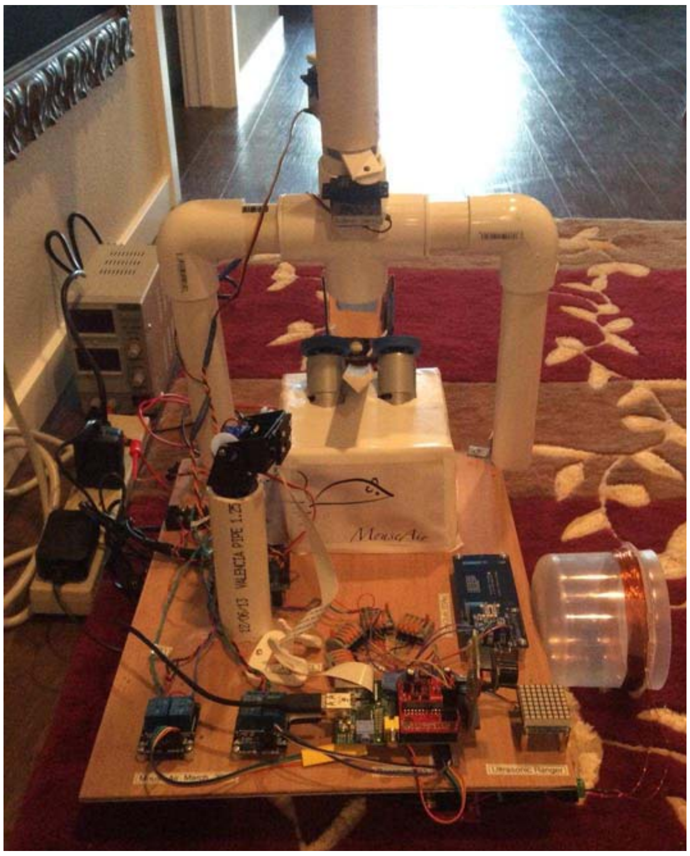


In this project at SwitchDoc Labs, I am using RasPiConnect (www.milocreek.com) to control MouseAir. RasPiConnect allows me to build multiple pages of controls, with graphs, webpages, pictures, streaming video (with a little more work!) and lots of lights and buttons on an iPad or iPhone without having to build and submit an app to the App Store. You build the controls on the phone or tablet and then modify a server on the Raspberry Pi to talk to the control panel. This is the second project for which I am using RasPiConnect. The first is Project Curacao, descripted in early issues of The MagPi.

Description of Mouse Air

The MouseAir system is built around a Raspberry Pi controlling all the devices necessary to launch the toys, connected to the iPad via a WiFi connection. We are using a Pi Camera on a pan / tilt structure to capture the cat events, examine the launching mechanism for jams, motion detection and even streaming video. It uses a solenoid to push a mouse between two rapidly spinning DC motors. Note the hand built 125KHz RFID antenna on the right side in the picture overleaf. See an early mouse launch at: http://www.switchdoc.com/2014/04/mouseair-prototype-video

The motors are a bit of an overkill. A properly loaded mouse can be shot 10 meters!



What Controls to Use

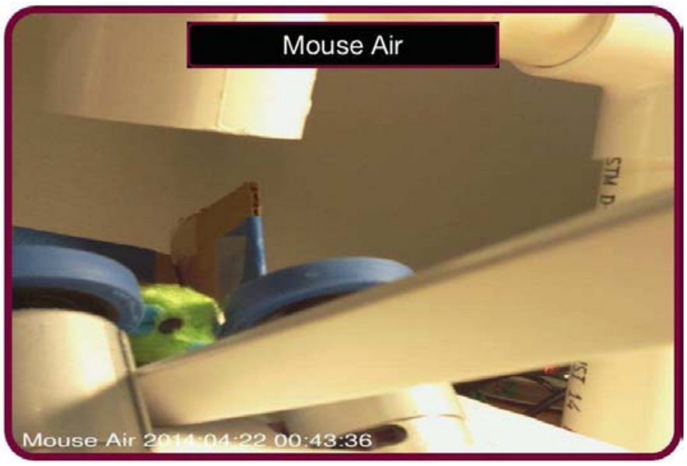
In designing a control panel using RasPiConnect, the first thing you have to do is choose which controls to use and where to place them on the iPad Screen. This is all done within the app on the iPad or iPhone. RasPiConnect has a collection of about 20 controls to choose from, all with special options and behaviours from meters to buttons to LEDs to constantly updated live graphs.

The first thing I did was decide what I wanted to control: which sensors are used for triggering, controlling and viewing the Pi Camera, be able to manually launch mice and manually control any part of the six step procedure to launch the mouse.

Every control in RasPiConnect has an input and response. The app sends an input to your Raspberry Pi (or Arduino, or both!) and the Raspberry Pi sends back a response.

Pi Camera display

For the camera display, I have chosen a Remote Picture Webview control. This control has a title and the response from the Raspberry Pi is in the form of an HTML page. It is very flexible and you can configure it for pictures or for streaming video with a little effort. Note the mouse peeking out.



Buttons for action

In RasPiConnect, there are two types of buttons: Action and Feedback Action. You use Action Buttons to do an action that does not require feedback (however, there are ways to provide feedback by refreshing other controls based on an Action Button tap). For example, "Take Picture".

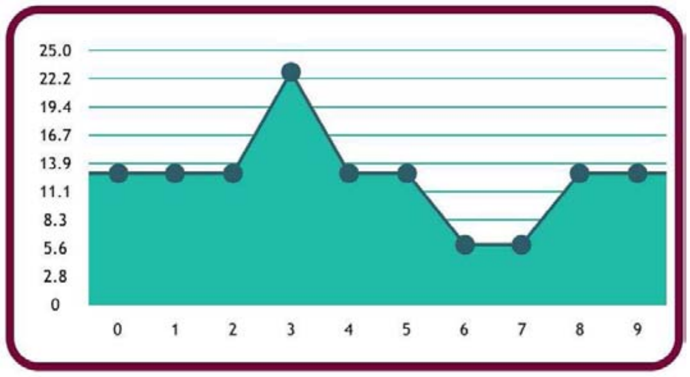
Feedback Buttons for Selections

Feedback Buttons are used for cycling through a set of options (such as "Off" and "On"). Tapping a Feedback Button sends an input to the Raspberry Pi (the current text in the button) and the response text from the Raspberry Pi replaces the button text. For example, tapping the "Ultrasonic Off" button will send "Ultrasonic Off" to the Raspberry Pi, turn the Ultrasonic sensor off and return the text "Ultrasonic On" to set up the button for the next tap. In Project Curacao, some buttons had 8 different states!

Live graphs for real time reports

The live controls are a new feature in the lastest version of RasPiConnect. They are a collection of controls that will periodically (interval set by the user) ask the Raspberry Pi for any new information and update the iPad screen. I decided to use a Complex Line Graph Live control to give a continuous display of what distance the ultrasonic sensor is reading (is the cat walking by?)

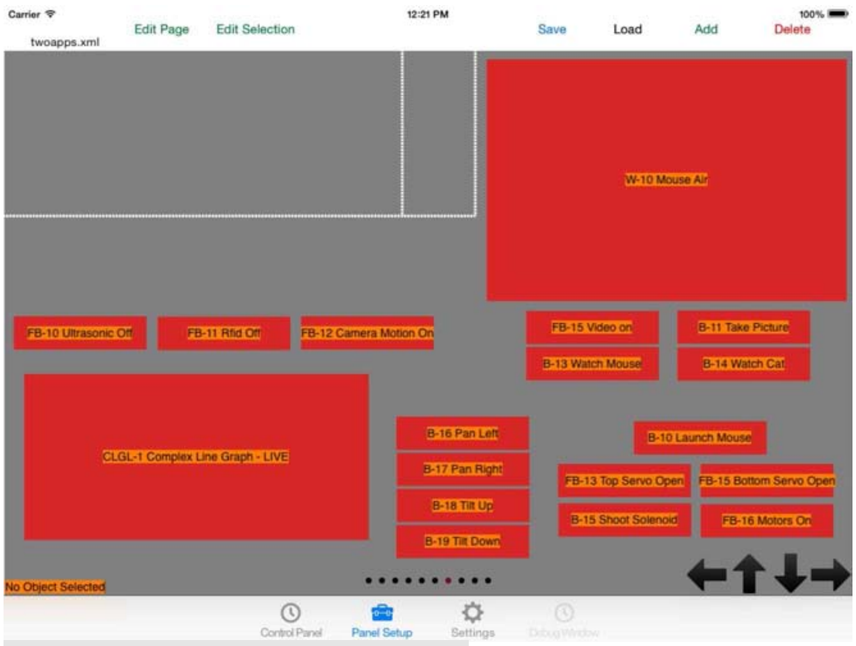
Each time the iPad app asks for an update of the graph, the Raspberry Pi returns a response of the list of data points to be graphed and what text to use for the x axis labels, I usually set this display to update every second. Note that RasPiConnect will only allow you to use this feature if you are connected to WiFi on the iPad. We don't want to run up the mobile data bill!



Live Ultrasonic Range

Configuring RasPiConnect on the iPad

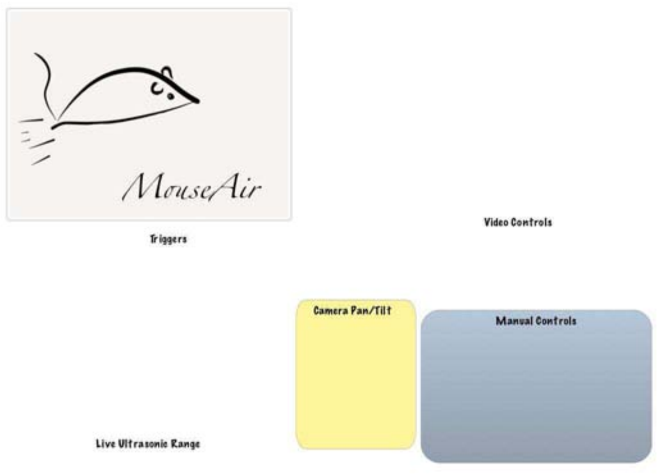
Each of the desired controls is placed on the iPad in the Panel Setup tab within RasPiConnect.



The design screen in RaspiConnect

Backgrounds

To finish off the control panel we add the MouseAir logo and accent text boxes onto the screen. To do this we build a custom background in the page (we used Grafio on an iPad to generate a JPG or PNG file) and then select this background graphic for the page.



The background image loaded into the app

Configuring RasPiConnectServer Software on the Raspberry Pi

The MouseCommand.txt file is first written by the RasPiConnectServer program and then read by MouseAir. When the command is complete, MouseAir writes "DONE" into the command file, telling RasPiConnectServer that it is finished and ready for the next command. Note that the RasPiConnect app keeps all commands in a queue and will not send another command until either a timeout occurs (programmable) or it gets a response from the Raspberry Pi.

The MouseAir Control Software

The MouseAir software operates in a loop, periodically checking for a new command from RasPiConnect, checking for triggers from RFID, Pi Camera motion and checking for an Ultrasonic trigger. The MouseAir software is available on http://github.com/switchdoclabs.

On the MouseAir side, the software for receiving commands from RasPiConnect is contained in `processcommand()`.

```
def processCommand():
  f =
open("/home/pi/MouseAir/state/MouseComman
d.txt", "r")
  command = f.read()  f.close()

  if (command == "") or (command ==
"DONE"):
    # Nothing to do
    return False

  # Check for our commands
  pclogging.log(pclogging.INFO, __name__,
"Command %s Received" % command)

  print "Processing Command: ", command

  if (command == "FIREMOUSE"):
    fireMouse()    completeCommand()
    return True
  if (command == "TAKEPICTURE"):
    utils.threadTakePicture("Picture
Taken -RasPiConnect Command")
    completeCommand()
    return True

...

def completeCommand():
  f =
open("/home/pi/MouseAir/state/MouseComman
d.txt", "w")  f.write("DONE")
  f.close()
```

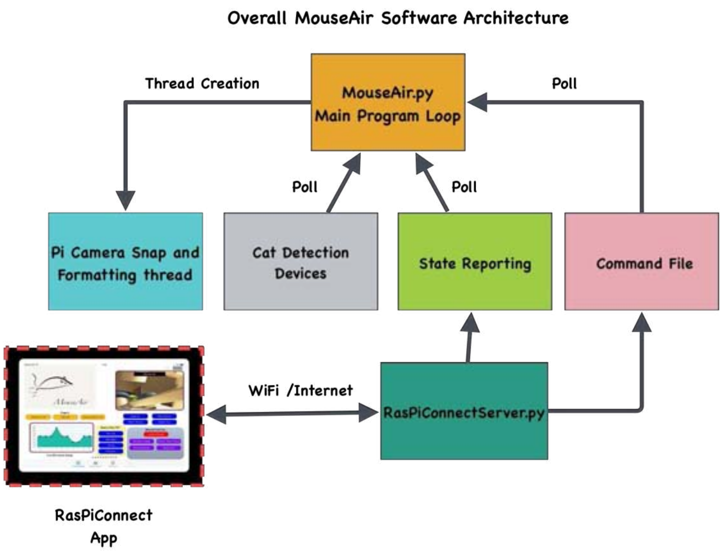
The RasPiConnectServer Software

RasPiConnectServer is a Python program provided for connection from a Raspberry Pi to the RasPiConnect app. Change the file Local.py (in addition config.py) to connect to the MouseAir software. MiloCreek provides full documentation for Local.py and the rest of the server at www.milocreek.com under the documentation link. Each button is pretty simple to do.

There are some other setup items such as setting URLs for your Raspberry Pi that are fully explained in the manual.

All of the software that you need to write is placed in Local.py. There is an example Local.py file provided in the server download. To illustrate how to write the interface, I will follow one entire button through the process.

I will use the Motors On button as an example. This button controls the DC motors that shoot the



mouse up in the air. It has two states, "On" and "Off". This makes it a perfect candidate for a Feedback Action Button.

When you add a control in RasPiConnect, you can set the control code (usually a unique identifier) for the button. By convention, each Feedback Action Button starts with "FB". Our motor control button a control code of "FB-16".

When you tap the button on RasPiConnect an XML message (or optionally a JSON or raw mode message) is sent from the iPad to the Raspberry Pi. The message is parsed by the RasPiConnectServer software and the results are presented to your customized code in the Local.py file. You don't have to deal with any of the parsing or handshaking, the libraries do all of this for us. The button is then presented to Local.py. We wrote one small routine to interface to the MouseAir.py command file.

```
defsendCommandToMouseAirAndWait(command):
```

The next section is the "money code", that is, the code where the functionality of the button is implemented.

```
  # object Type match
  if (objectType ==
FEEDBACK_ACTION_BUTTON_UITYPE):
    if (Config.debug()):
      print "FEEDBACK_ACTION_BUTTON_UTYPE
of %s found" % objectServerID

    # FB-16 -  turn motors on
    if (objectServerID == "FB-16"):
      #check for validate request
      # validate allows RasPiConnect to
verify this object is here
      if (validate == "YES"):
        outgoingXMLData +=
Validate.buildValidateResponse("YES")
        outgoingXMLData +=
BuildResponse.buildFooter()
        return outgoingXMLData
      # not validate request, so execute
      responseData = "XXX"
      if (objectName is None):
        objectName = "XXX"
      lowername = objectName.lower()
      if (lowername == "motors on"):
        print "set Motors On"
```

We now send the command to MouseAir.

```
        status =
sendCommandToMouseAirAndWait("MOTORSON")
        responseData = "motors Off"
        responseData =
responseData.title()
```

Note we have now "toggled" the button by sending "Motors Off" back to the app to set up the button for the next tap.

```
      elif (lowername == "motors off"):
        status =
sendCommandToMouseAirAndWait("MOTORSOFF")
        responseData = "Motors On"
        responseData =
responseData.title()
```

The default section is in case of a time out and the button becomes blank. In that case, we want the motors off.

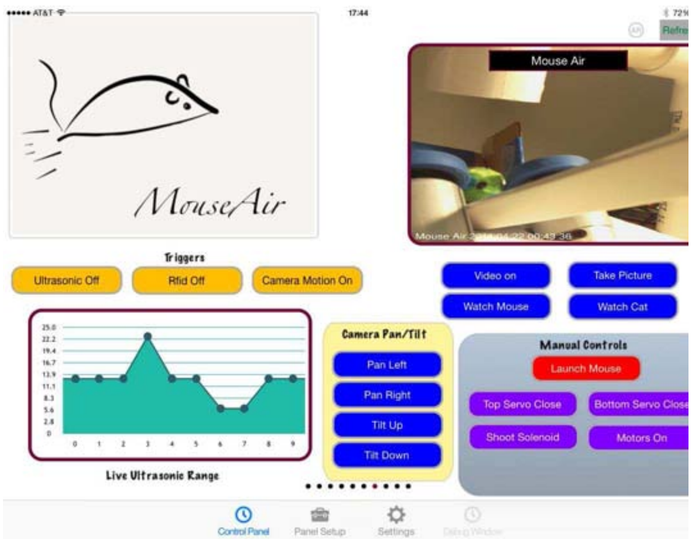
```
      # defaults to Motors off
      else:
        print "Motors Off"
        status =
sendCommandToMouseAirAndWait("MOTORSOFF")
        lowername = "Motors On"
        responseData = lowername.title()
```

FInally, the rest of the XML response is built. By the way, if you somehow screw up the XML, RasPiConnect just rejects it. There is error checking built into the App as well as checksum on each response.

```
      outgoingXMLData +=
BuildResponse.buildResponse(responseData)
      outgoingXMLData +=
BuildResponse.buildFooter()
      return outgoingXMLData
```

Looking at the code, you can see the command that is written to the MouseAir command file. The software then waits for the "DONE" and then sends the response (which is the text used for the button on the iPad. If you sent a "Motor On" command, the response to be sent back would be "Motor Off" and then the "Motor Off" would be displayed on the button on the app.

That is the complete cycle. This design pattern is used for all of the controls.



The completed final control panel for MouseAir

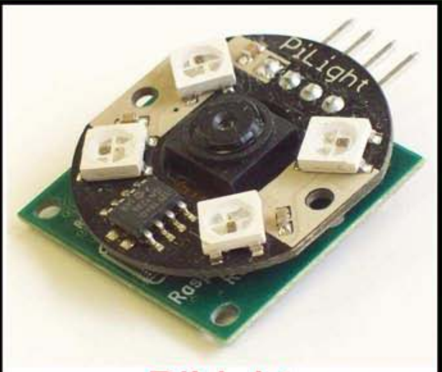
Conclusion

How to setup a control panel for your project is always a challenge. I like being able to control a project locally and across the Internet. MouseAir is a complicated project with a number of different things to adjust and to view. I found that RasPiConnect was a very good match and platform on which to build. I am already planning to use it on SwitchDoc Labs next project.

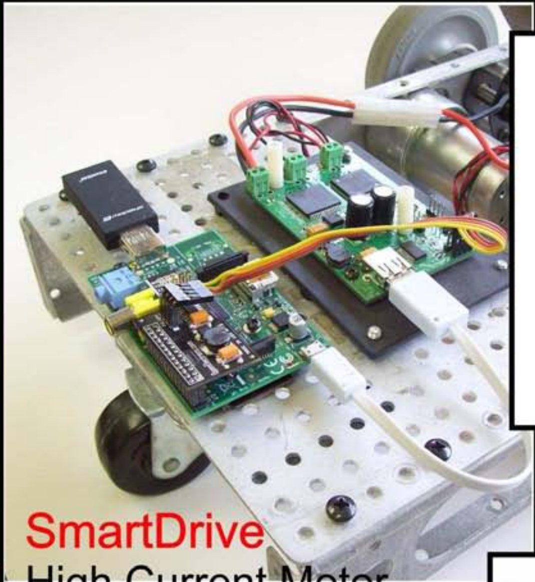
RasPiConnectServer is available on http://github.com/milocreek and the specific MouseAir RasPiConnect file (Local.py) is available on http://github.com/switchdoclabs.

For more information about MouseAir see the authors blog at www.switchdoc.com.

For more information about RasPiConnect see www.milocreek.com

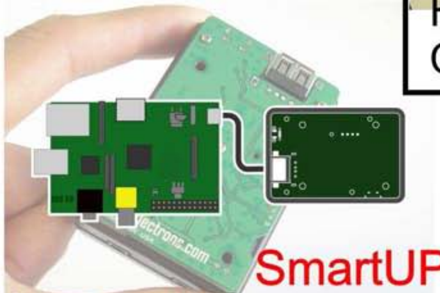


PiLight
Light for RPi Camera
(Infrared and Regular)



SmartDrive
High Current Motor
Controller for RPi

Pi-Pan
a Pan-Tilt for RPi Camera



SmartUPS
Uninterruptible power supply
for Raspberry Pi

PiServoController
6 Channel Servo Controller
for Raspberry Pi

OpenElectrons.com

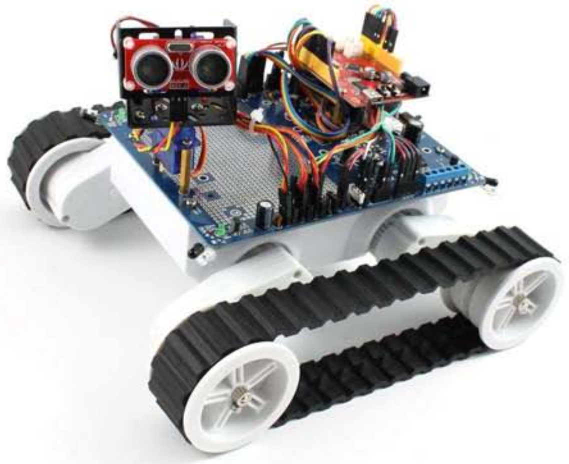
DAWN ROBOTICS

python powered

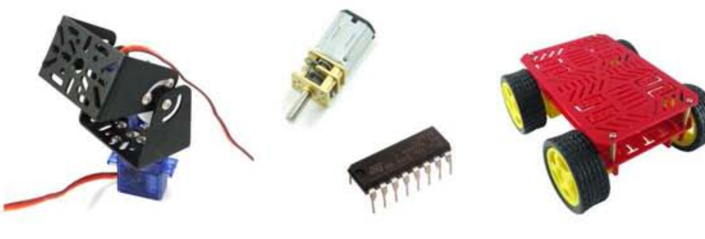
OpenCV

Raspberry Pi Camera Robot Kit

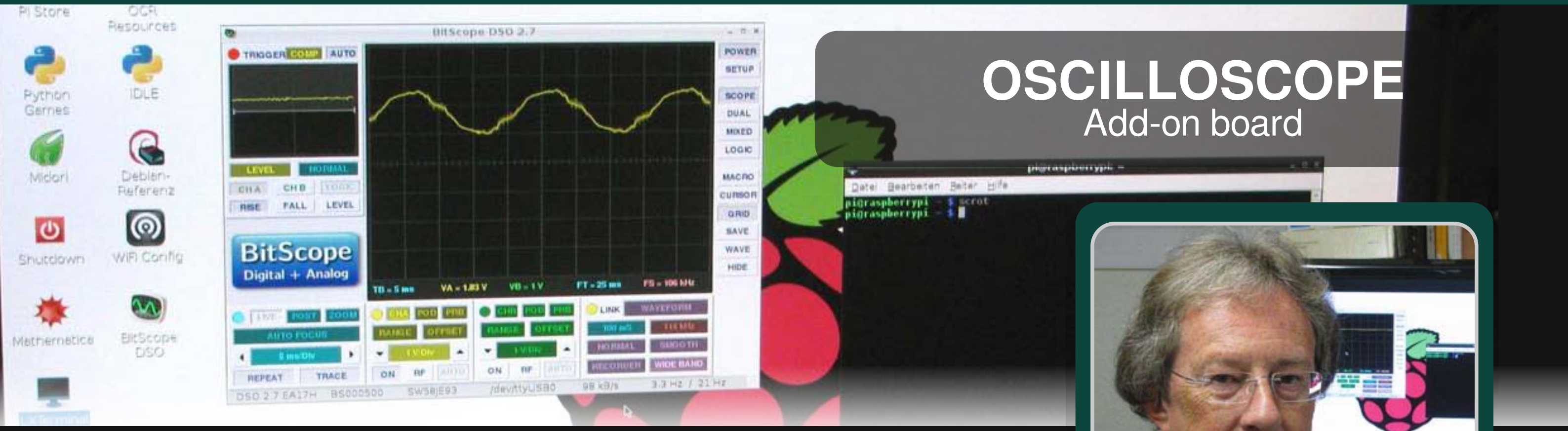
- Drive over WiFi using a Tablet, Smartphone or PC
- Programmable using Python and OpenCV for Computer Vision
- All software Open Source!



More robot kits and parts at
www.dawnrobotics.co.uk

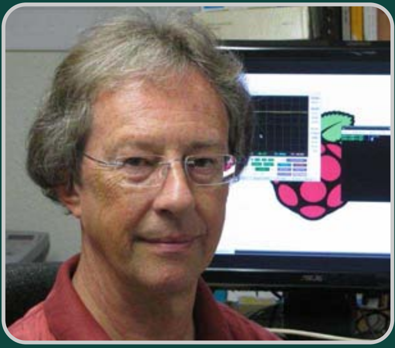


Find lots of robot
and Raspberry Pi tutorials at
blog.dawnrobotics.co.uk



Electronic Measurement? BitScope & RasPi!

SKILL LEVEL : BEGINNER



Karl-Ludwig

Guest Writer

An oscilloscope is only for professionals and rather expensive!? If you think this sentence is true then read on.

The BitScope Micro from the Australian manufacturer BitScope Designs is a small add-on board especially adapted to the Raspberry Pi, with which you can turn your Raspberry Pi into an oscilloscope, logic analyser, spectrum analyser and a waveform generator. In this miniseries, I am going to show you how to setup this dynamic duo and use its features to understand how electronic circuits work and debug them if they do not work.

First things first: what is an oscilloscope anyway? An oscilloscope is an electronic measurement device with which you can measure different electrical parameters such as voltages, frequencies, etc.. With a digital multimeter you are able to measure an electrical voltage and see a numerical representation of it in the display. However, an oscilloscope presents a graphical representation of the voltage over time. The oscilloscope does this by plotting measurements using a Cartesian co-ordinate system.

In this first part of the article we are going to setup the hardware, install the software and take our first graph.

Assuming you have your Raspberry Pi up and running, make sure that your Raspbian installation is up to date by typing:

```
sudo apt-get update
sudo apt-get upgrade -y
```

Next, install the BitScope software with the following procedure:

1) Download the BitScope DSO 2.7 package from the BitScope Download area from http://bitscope.com/pi/ and save it into the directory `/home/pi` on your Raspberry Pi.

2) When the download is complete, fire up your file manager, navigate to `/home/pi` directory and right click on the downloaded package and choose Open with from the context menu.

3) Select the custom command line tab, enter `sudo dpkg -i` and click OK.

4) When the installation procedure has finished, connect the BitScope Micros USB port to the Raspberry Pi as shown in Fig. 1. You will need a powered USB hub to do so, because normally both USB ports of the Raspberry Pi are in use for the keyboard and the mouse.

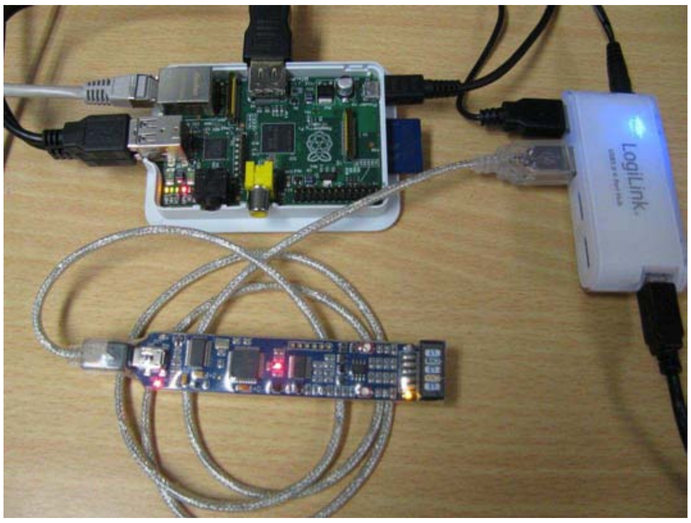


Figure 1: Connecting the BitScope Micro to your Raspberry Pi via USB

5) You should be able to start BitScope DSO from the main menu now and see its splash screen on your monitor. With your BitScope Micro connected click POWER on the right side of the splash screen.

With BitScope DSO up and running and the BitScope Micro add-on board connected, the only thing left before we start exploring the system is to familiarise ourselves with the user interface on the screen. Fig. 2 shows the main screen.

In the main display (1) you will see the results of the measurements in graphical form. This output is dependent on which buttons you choose on the right side (2). When measurements are taken is very important if you are working with an oscilloscope. Electronic engineers talk about triggering. (3) is called the trigger window, which we will leave at the default automatic mode for the moment. We will change the trigger behaviour with the trigger controls (4) at a later

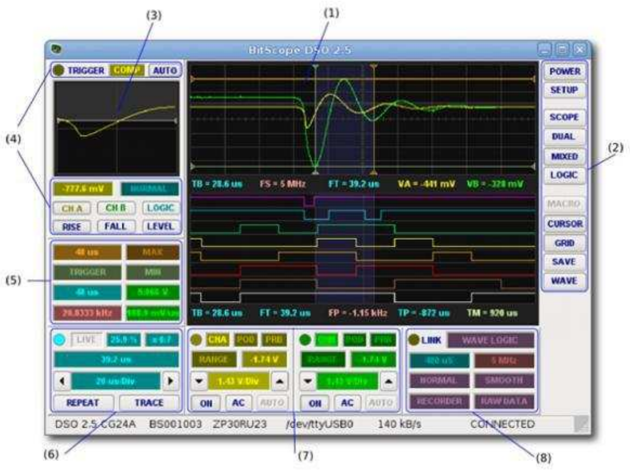


Figure 2: BitScope DSO software main screen elements (photo courtesy by BitScope Designs)

time. For exact measurements, BitScope Micro supports us with so called cursor measurements (5). With the time base control (6), we are able to zoom in and out of a graph and therefore see more or less details. The channel controls (7) let us influence the input, its source, range, vertical position and scaling. Last but not least the capture control (8) defines the capture sample rate, duration, frame rate and display modes.

First measurement

To check if your setup is ok and working, connect one of the test leads, which are included with the BitScope Micro, to the channel A pin (marked CHA (yellow), see Fig. 3):

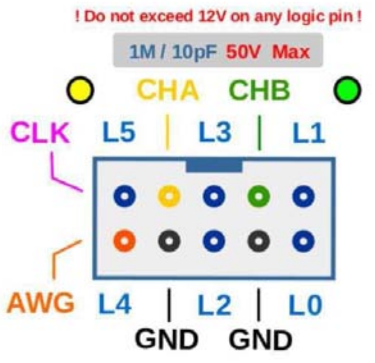


Figure 3: Pin layout of the BitScope Micro (photo courtesy by BitScope Designs)

Next change the time base (6) to 5msec/Div by clicking on the left arrow key until this value is displayed. 5msec/Division means that each square of the grid on the main screen (1) on the x-axis represents five milliseconds. Next look at the channel controls for channel A (7) and select 1V/Div. This means that each square of the grid on the main screen (1) on the y-axis represents one volt. Now press down the top of the grabber clip, such that you can see the tiny metal grabber coming out of the plastic on the opposite side and touch it with your finger. Do not be afraid nothing will happen, except if you let loose the top, then it will bite you. If you look to the main screen you will see a more or less perfect sine wave like the one in Fig. 4:

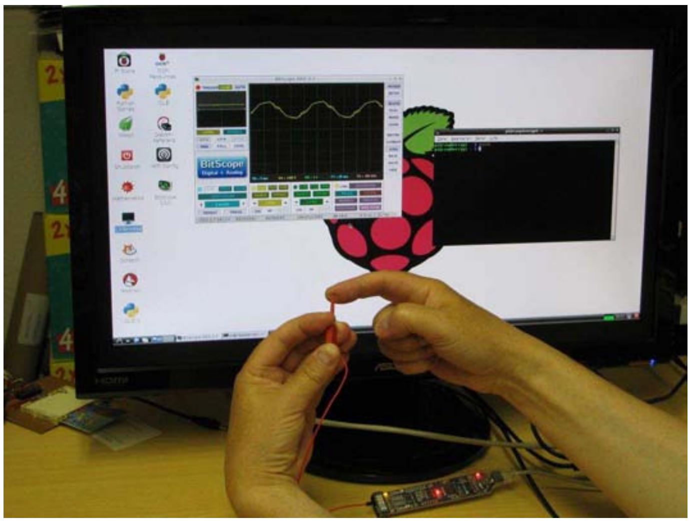


Figure 4: Touching the test lead with your fingertip

Where does this sine wave is come from? Take your finger from the metal grabber and it will disappear. Put it on again and the sine wave will reappear. No doubt, you are the source of the mysterious sine wave. How? Mains electricity is provided at each power outlet and has an alternating current with a frequency of 50 Hz (in Europe and 60 Hz in the USA). A wire that carries an alternating current behaves as a radio transmitter. Your body is receiving the signals from the transmitting wire. When you touch the metal grabber, the signal is transmitted to the BitScope Micro oscilloscope and onto the Raspberry Pi screen. Why am I so sure that the sine wave displayed has a frequency of 50 Hz? Well, look at Fig. 5:

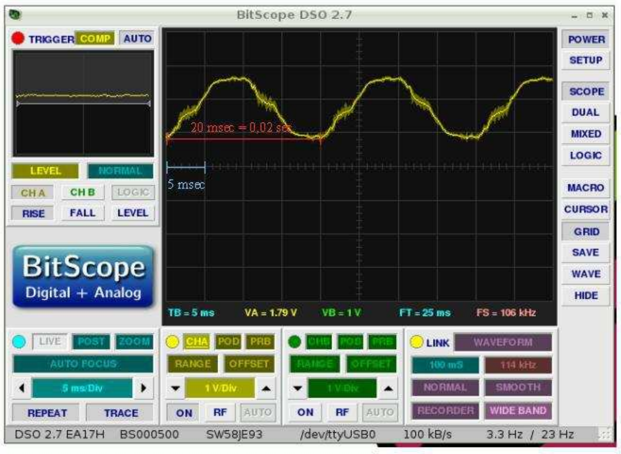
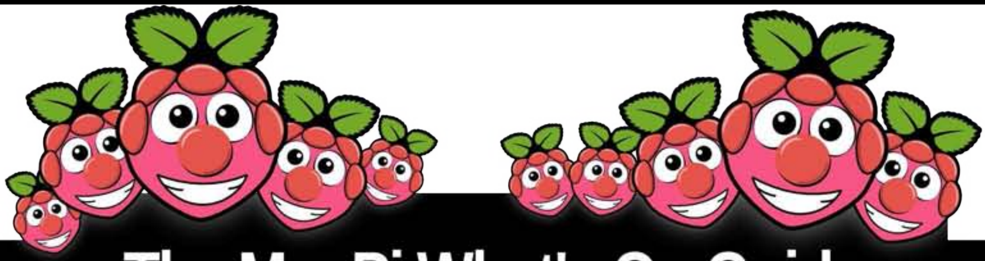


Figure 5: Determining the frequency of a signal

We selected 5msec/Div, where one cycle of the signal is 4 divisions long. 5msec*4 equals 20 msec (or 0.02 seconds. Since an oscilloscope cannot measure frequencies directly, we have to take the reciprocal of 0.02 seconds which is 50 Hz.

In this article we discussed what an oscilloscope is all about, set up the hardware and software, and checked the whole setup by injecting a 50 Hz sine wave with our fingertip.

Join me next month, when we start to delve into the fascinating field of electronic measurement with a digital storage oscilloscope. Yes, that is what our BitScope Micro is. For all those interested in actually turning their Raspberry Pi into a digital storage oscilloscope with the BitScope Micro add-on board, it is available from BitScope Designs in Australia (http://www.bitscope.com). In Germany, it is offered by BUTTE publishing company (http://www.BUTTE-verlag.de). In the UK, you can order the BitScope Micro at http://shop.pimoroni.com and in Switzerland at http://www.pi-shop.ch. A German translation of this article is available at http://www.BUTTE-verlag.de/.



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

Raspberry Jam Silicon Valley

When: Saturday 19th July 2014, 11.30pm to 4.30pm PDT
Where: Computer History Museum, 1401 N. Shoreline Blvd., Mountain View, CA 94043, USA

Open to all and free to attend whether you are new or experienced with the Raspberry Pi.
http://trivalleycoderdojo.wordpress.com/2014/04/17/raspberry-jam-silicon-valley

Raspberry Pi and DIY Gamer Bootcamp

When: Monday 21st to Tuesday 22nd July 2014, 09.30am to 3.30pm
Where: University of South Wales, Treforest Campus, Pontypridd, CF37 1DL, UK

A two-day workshop for 11 to 19 year olds from http://www.technocamps.com.
http://www.eventbrite.co.uk/e/12001716457

Intro to Raspberry Pi: Make a Cupcade

When: Wednesday 30th July 2014, 6.00pm to 9.00pm EDT
Where: Makery Bodega Pop Up, 195 Avenue C, New York, NY 10009, USA

At this workshop attendees can learn to construct a Cupcade (http://www.adafruit.com/product/1783) micro arcade cabinet for the Raspberry Pi. http://www.eventbrite.com/e/12049459257

Preston Raspberry Jam

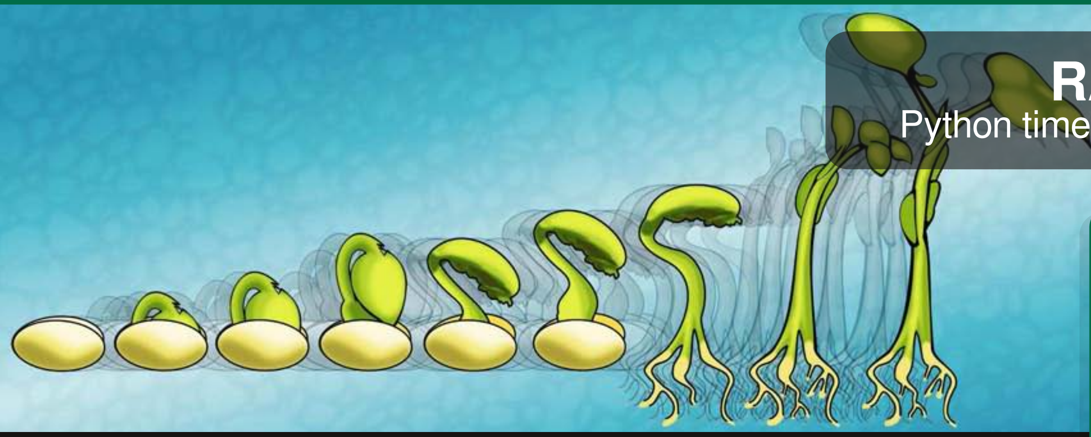
When: Monday 4th August 2014, 7.00pm to 9.00pm
Where: Media Innovation Studio, 4th Floor, Media Factory, Kirkham Street, Preston, PR1 2HE, UK

Come and see what others have done. Learn something new.
http://www.eventbrite.co.uk/e/10496195403

Peterborough Raspberry Jam

When: Saturday 9th August 2014, 11.00am to 5.00pm
Where: University Campus Peterborough, Park Crescent, Peterborough, PE1 4DZ, UK

The full day programme includes classroom activities on Scratch and Python, a maker area and lecture talks. http://www.eventbrite.co.uk/e/11727761049



RASPISTILL

Python time lapse & image processing

Tom Denton

Guest Writer

Create a time-lapse video with the Raspberry Pi camera

SKILL LEVEL : ADVANCED

On December 27th, 2013, a pair of Raspberry Pis with Raspberry Pi cameras were placed in a cottage in Ontario overlooking Georgian Bay. About four months later, they were found to have performed wonderfully, capturing over 40,000 photos at two minute increments. Although the shoot was planned for a few months, it was not expected to be as successful as it was. When the cameras were setup, the windows were quite frosted over. Therefore, the expected outcome was to see the back of an icy window. However, the windows stayed clear during the entire winter. The photographs taken captured ice forming and dissolving on the lake, deer passing like ghosts, magnificent storms and amazing sunrises. The images were compiled together to form the resulting time-lapse video given at:
http://inventingsituations.net/winter-on-georgian-bay

In this article, there are some code snippets that can be used to make a time-lapse video using Python and command line tools. In particular, the `raspistill` program is used to capture the images and the Python Image Library (PIL) and `mencoder` are used to create the final movie.

Setting up Raspberry Pi and camera

Follow http://www.raspberrypi.org/help/camera-module-setup/ to connect a Raspberry Pi camera to a Raspberry Pi. Then install the latest Raspbian image and type:

```
sudo raspi-config
```

Select

```
expand the filesystem
enable camera
advanced option
memory split (minimum 128 for gpu)
enable boot to desktop/scratch
Console
finish
reboot
yes
```

This will enable the camera, use all of the SD card for the Linux installation, turn off the boot to X and allow the ARM processor to use the maximum amount of memory. Next, make sure that Raspbian is up to date and install the libraries and programs needed for this project:

```
sudo apt-get update
sudo apt-get upgrade -y
sudo apt-get install -y python-imaging \
python-zmq python-matplotlib mencoder \
imagemagick
sudo reboot
```

Taking photos with raspistill

The easiest way to use the Raspberry Pi camera is via the `raspistill` command. It has a time-lapse option, which many people have used to good effect. For an extremely long time lapse, finer control over the images taken proved to be better than the automatic settings. As a result, a Python script was written to take photographs at regular intervals. Calculations were used to make sure that the images were of consistent quality. Image quality consistency was maintained by the manipulation of shutter speed, ISO and post capture alteration.

The two options that control the brightness of an image are the shutter speed and the ISO (International Standards Organization) setting. The shutter speed controls how long the camera collects light to create an image, where a longer shutter speed will create brighter images but will be more likely to have motion blur if there is movement in the frame. On the Raspberry Pi, the slowest shutter speed available is about 2.5 seconds; longer speeds will cause the camera to hang and not take pictures until the Raspberry Pi is rebooted. To be safe, the maximum shutter speed was set to two seconds. Shutter speeds in `raspistill` are measured in microseconds. Therefore, a shutter speed of 1000000 implies one second.

ISO setting controls the sensitivity of the sensor, within a range of 100 (low sensitivity) to 800 (high sensitivity). When the ISO setting is high, brighter images are taken, but they will also tend to be much more noisy.

The command `raspistill` can be used to take a photograph with a set shutter speed and ISO setting. The automatic settings should be avoided, apart from AWB (auto white balancing):

```
raspistill -awb auto -n -t 10 -ss 1000000 \
-ISO 100 -o mypic.jpg
```

Python can be used to run `raspistill` by using a subprocess call:

```
import subprocess
ss=1000000
iso=100
command='raspistill -n -t 10 '
command+='-ss '+str(ss)
command+=' -ISO '+str(iso)+' -o mypic.jpg'
subprocess.call(command , shell=True)
```

The goal for the project was to keep images at about the same brightness over the course of a day. The brightness of each image was measured. Then the shutter speed and ISO setting were adjusted. Since fast movements are not important for time lapse pictures, the ISO was kept at 100 as much as possible to ensure high quality images.

Calculating the average brightness

The Python Imaging Library (PIL) is great for manipulating images in Python:

```
import Image
im=Image.open('mypic.jpg')
pixels=(im.size[0]*im.size[1])
hist=im.convert('L').histogram()
br=float(sum([i*hist[i] for i in
range(256)]))/pixels
print br
```

This example opens an image and uses the image's histogram to compute the average brightness of the image. For a greyscale image, there are 256 possible pixel values. The histogram counts how many pixels there are for each value. The average brightness of the images is calculated by adding up all of the pixel brightness values and dividing by the number of pixels.

Adjusting the shutter speed

To retain the same image brightness, as the light increases or decreases, either the shutter speed or the ISO setting should be adjusted. The brightness from the previous image can be used to adjust the shutter speed:

```
delta=128-br
```

```
ss=ss*(1 + delta/128)
```

where 128 is the target brightness and ss is the shutter speed in the previous Python example. To reduce flicker in the time lapse video, the average brightness from the last ten or twenty images was used.

Since the camera used had an infrared filter and the light level was low during the night, a brightness threshold was used to avoid storing images in very dark conditions.

Installing the scripts

To install the scripts, type:

```
git clone https://github.com/sdenton4/pipic
cd pipic
chmod 755 timelapse.py
chmod 755 deflicker.py
```

The time lapse program can be run with the default settings:

```
./timelapse.py
```

or the available options can be listed:

```
./timelapse.py -h
```

When the time lapse program runs, it writes the pictures into the `~/pictures` directory.

To start the time lapse program when the Raspberry Pi boots type:

```
sudo nano /etc/crontab
```

and add

```
@reboot    pi    python /home/pi/timelapse.py
```

Then save the file and exit nano.

Making the movie File

After running the time lapse program for a long time, the photographs should be assembled into a movie. This can be done very easily with `mencoder`:

```
mencoder "mf://*.jpg" -mf fps=18:type=jpg \
-ovc lavc -lavcopts \
vcodec=mpeg4:mbd=2:trell:vbitrate=3000 \
-vf scale=512:384 -oac copy -o movie.avi
```

This command uses all of the jpg files in the present working directory and assembles them into a movie. A sound track can be added to the movie, using an existing audio file:

```
mencoder -ovc copy -audiofile MYMUSIC.mp3 \
-oac copy movie.avi -o movieWithMusic.avi
```

Post-processing

The movie can look a bit too grey. To create more contrast, PIL's image manipulation capabilities can be used. The greyscale histogram of a single image can be plotted using Python and `matplotlib`:

```
import Image
from matplotlib import pyplot as plt
im=Image.open('mypic.jpg')
hist=im.convert('L').histogram()
plt.plot(hist)
plt.savefig('myhist.png')
```

The `matplotlib` package can also be used to plot the histograms of many images at once:

```
import Image,glob
import numpy as np
from matplotlib import pyplot as plt
M=[]
for x in glob.glob('*.jpg'):
  im = Image.open(x)
  M += [im.convert('L').histogram()]
plt.pcolormesh(np.array(M))
plt.savefig('mymanyhist.png')
```

This example reads all of the .jpg files in the present working directory. In the resulting image each horizontal line of pixels is one image in the time lapse. The brightness of a pixel indicates how many pixels in the original image have a given intensity.

Some of the images captured were slightly grey. This implies that there is not very much contrast, because most of the pixel values are clustered in the middle of the histogram. In these cases, there were no completely black pixels and only a few completely white pixels. The image at the bottom left of this page illustrates is an example grey image.

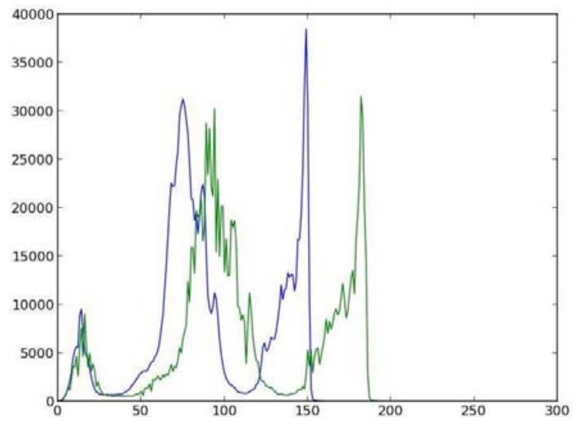
To improve the balance, the image histogram can be used histogram and stretched out to fill the whole range (from 0 to 256). The first step is to find the bounds a and b:

```
pixels=im.size[0]*im.size[1]
a=0
count=0
while count<0.03*pixels:
  count+=hist[a]
  a+=1
b=255
count=0
while count<0.05*pixels:
  count+=hist[b]
  b-=1
```

where a and b defined such that 97% of the pixels are brighter than a and 97% are darker than b. Now a and b can be used to stretch the histogram:

```
im3=Image.eval(im2, lambda x: (x-a)*255/(b-a))
```

where the eval function applies the function given within the parentheses to each pixel in the image and `lambda` is an easy way to make a simple function. For example, the following code snippet prints 12.



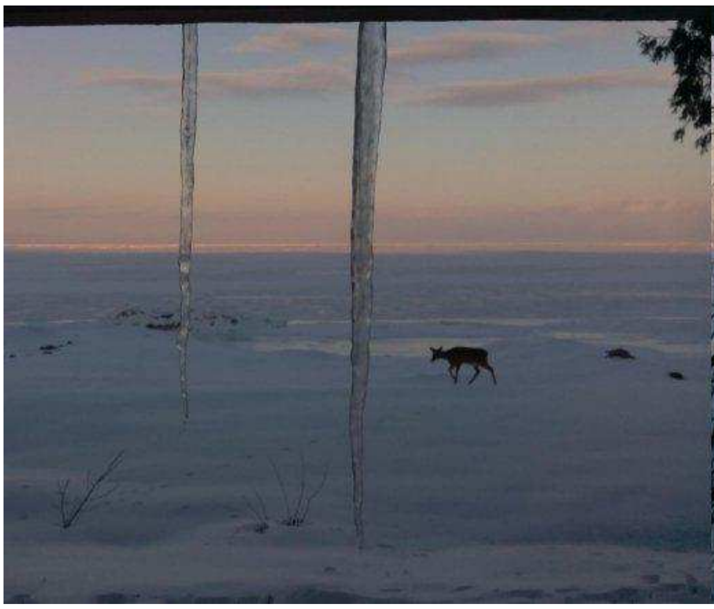
```
f=lambda x: x*3
print f(4)
```

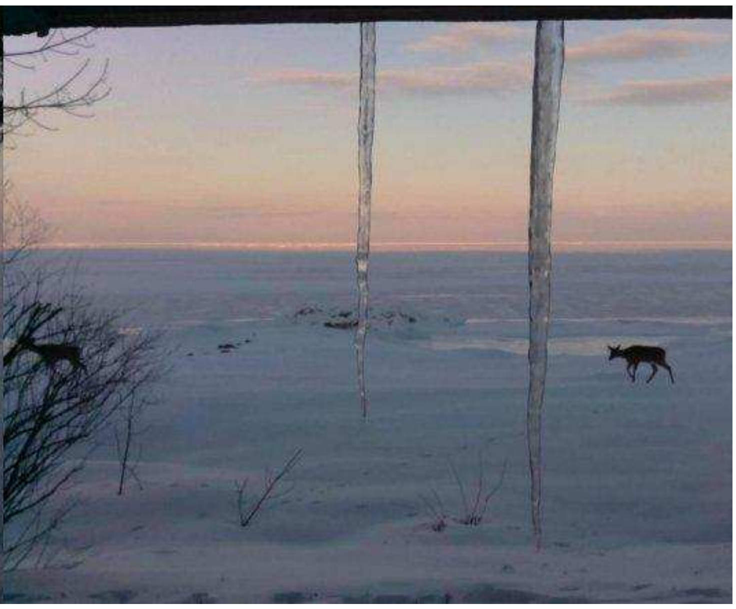
The histograms above are for the original (green) and levelled image (blue). The histogram for the levelled image is quite spread out. It is also very jagged, because the stretching removes many possible pixel values. Fortunately, this is not very visible in the levelled image. In practice, the bounds `a` and `b` were based on a rolling average from nearby pictures, such that there were no dramatic changes. The final levelled image is shown at the bottom right of this page.

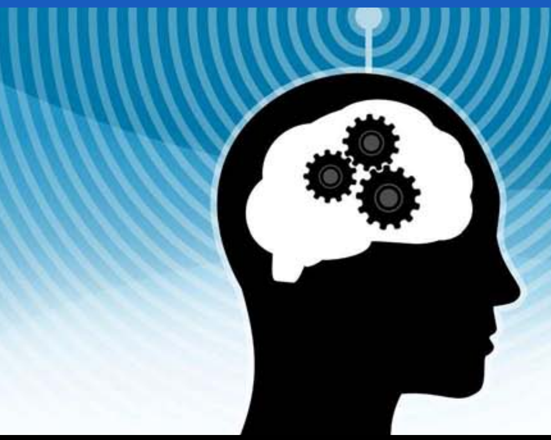
The techniques discussed in this section can be found in the `deflicker.py` script. Type:

```
./deflicker.py -h
```

for a list of available options. The script reads images that are in the present working directory.


Before


After



Richard Wenner

Guest Writer

Raspberry Spy Part 2: Understanding Wi-Fi

SKILL LEVEL : INTERMEDIATE

In the first part of this article we concentrated on turning your Raspberry Pi into an effective wireless sniffing tool by swapping the Ethernet port for a Wi-Fi dongle. Here we concentrate on understanding the basic structure of networks and then introduce a powerful set of network analyses facilities.

Networking

The Internet is often pictured as a group of multiply connected nodes all transferring small packets of information which may be instantly rerouted in the event of one or more of the nodes becoming congested or failing completely. In practice electronic failure is rare. The failure considered in the network design of the internet includes having any connection or node being blow to smithereens. After all, the Internet was initially a military communications system - they have a propensity to break things, or know folk who do!

This image of the Internet is a little like looking at the motorway system. Large, wide, fast, high capacity roads with few junctions (nodes) many kilometres apart. A motorway may pass metres from your house but you have no direct access to it. You may live in a cul-de-sac connected to a road, that leads to a B road connected to an A road that eventually connects to the motorway.

In this way you can see how the topology of the Internet is not the multiply connected nodes but looks more like a branch of a Christmas tree. Either way the critical component of any connection is location. Your location may be given by your 'address'. Once a recognised address is given the only other task needed to make a connection is to plan a 'route'. To make our investigations more incisive we need to understand how the Internet deals with addressing and routing problems.

IP Addresses

Real Internet addresses (known as IPv4 addresses) consist of 32 binary bits, for example:
01100000001010100000000000000001
This provides up to 4.3 billion addresses. This may sound large but the Internet began to run out of addresses long ago!

To make addresses easier to read, humans break the 32 bits into four 8-bit chunks - known as 'bytes' and then convert them to decimal values separated by dots thus:-

```
01100000  00101010  00000000  00000001
192.168.0.1   dotted decimal notation
```

Subnetting

An Internet Service Provider (ISP) may have been allocated a block of addresses in the range from 180.181.1.0 to 180.181.7.255.

The ISP has several large customers. Each needs (say) up to 250 addresses for their networks. To organise the addresses effectively and efficiently the ISP could allocate addresses thus:-

180.181.1.0	to	180.181.1.255	to site A
180.181.2.0	to	180.181.2.255	to site B
180.181.3.0	to	180.181.3.255	to site C
180.181.4.0	to	180.181.4.255	to site D

This division of addressing is called 'sub-netting'. (Remember Internet means interconnection of networks). For administration purposes let's refer to each network by it's base address - a network number.

One of the powerful aspects of the Internet is that it devolves sub-netting and we will see a practical example of this that effects you later in the article.

Subnet Mask

A clever and convenient form that is used to describe a complete range of addresses is the 'subnet mask'. The subnet mask is also a 32-bit binary number but if you consider examples you will discover that they have a pattern that look like a bar chart with 1's emerging from left side of the bit pattern e.g.

```
11111111111111111111111100000000
11111111111111111111111111111000
11111111111111111111111100000000
11111111111111110000000000000000
11111100000000000000000000000000
```

The zeros on the right hand side of the mask indicate which bits of an address may be changed. Note for later that the mask can also be described by the length of the 1's in the chain 255.255.255.0 is /24 in CIDR.

Take the second example above where only the lower three bits of an address may be changed and combine the network 192.168.0.0. A bitwise AND provides the network number:

```
01100000 00101010 00000000 00000000
11111111 11111111 11111111 11111000
```

Perform a bitwise OR with the twos compliment of the mask (a simple and fast digital calculation)

```
01100000 00101010 00000000 00000000
00000000 00000000 00000000 00000111
```

This reveals the highest address in the range i.e. 192.168.0.7. This is also known as the 'broadcast address' and every node on the sub-network listens to this address.

Applying all of this to our ISP set-up above. The ISP has a network number of 180.181.0.0 and subnet mask 255.255.248.0 The ISP then devolves the four network numbers to site A, B C and D with subnet masks 255.255.255.0

Gateway

One address in the remaining network range has to be the 'gateway'. This is where ALL data, not destined for the local network, is sent. It's the exit from the cul-de-sac metaphor used above. The local network treats the rest of the world as just one address.

DNS

The references here to binary values show how low level our reference to the Internet is. If we need to access http://www.bbc.co.uk our network has to resolve this named reference to an IP address. This is achieved by DNS, the Domain Name System. Our local network could keep a complete directory of all name/IP references, which would be a massive undertaking or we could defer to a special DNS service. Entering the IP addresses of these services in the file /etc/resolv.cof file simply resolves this problem. This is the list of servers that will be called upon to convert names to IP addresses.

This completes the minimum requirement for any Internet connected network but remember above where it was stated that the Internet has amost run out of addresses. This happened some time before the explosion in home networking. The solution to providing every house with its own Internet connected network is NAT 'Network Address Translation'.

NAT

Most home users will be connected to the Internet via some kind of modem. This provides Ethernet and/or Wi-Fi in the local area. From the Internet side it has been provided with a single IP network address (referred to as the WAN 'Wide Area Network Address'). NAT translates this address and maps it to a special private range of addresses allowing many devices to be connected inside the home. It is typical to use a mask of 255.255.255.0 providing 255 local addresses. Local users are deceived into believing they have a full IP but this deceit is provided by the modem.

To make connections even easier NAT is often used in conjunction with DHCP. This is the Dynamic Host Control Protocol. DHCP is used when the Raspberry Pi is first switched on.

Raspberry Pis are manufactured by the bucket load but every Raspberry Pi is not exactly the same. Each has been individually programmed with its own 'MAC address'. At switch on this address is used to call out on any connected network to appeal for local configuration details. DHCP, built into the modem/router responds to the appeal by providing the Raspberry Pi with an individual IP address, details of the local subnet mask, the address of the local gateway and finally the address of any DNS server. With all of these values in place the Raspberry Pi can join the global Internet community!

DHCP will loan each Raspberry Pi an IP address that is re-leased periodically as Internet activity continues. The router will maintain the IP address for the Raspberry Pi even after a period when the Raspberry Pi has been switched off. Should a device be switched off for too long the address can be given to another device. If instead the Raspberry Pi is permanently configured with an IP address this is called 'static' addressing (as opposed to 'dynamic').

Network Settings

We accessed and modified the key file in the raspberry Pi during the first part of this article. The contents should now make more sense.

By default the DHCP - dynamic setting will look something like this:-

```
auto eth0
   allow-hotplug eth0
   iface eth0 inet dhcp
```

To force a static IP address into your Pi enter:-

```
auto eth0
   iface eth0 inet static
       address 192.0.3.14
       netmask 255.255.255.0
       gateway 192.0.3.254
```

Basic Command Line Utilities

These commands may be entered directly from the command line:-

`ifconfig`

To check settings enter ifconfig .
The HWaddr is the individual MAC address referred to above. Every MAC address is unique
e.g.: `ifconfig`

`ping`

`Ping` is like a sonar pulse sent to detect another IP address.
e.g.: `ping 8.8.8.8`
to ping Google's DNS

`nslookup`

`nslookup` tests any nameserver
e.g.: `nslookup www.bbc.co.uk`

`tracert`

`traceroute` lists the routers used to connect a destination
e.g.: `tracert www.bbc.co.uk`

Network Monitoring Applications

With our network knowledge in place we can start to explore using simple tools.

Network Scanning

We need to install some further applications.

```
apt-get install nmap xsltproc elinks
```

Nmap may be used to carefully scan individual IP addresses or complete ranges of addresses. However, large scans can be very time consuming.

Perform a quick test first using:

```
sudo nmap -v -sL 192.168.1.1-16
```

to report on addresses 1 to 16 to check things are working (substitute your own local adress). Alternatively:

```
 sudo nmap 192.168.0.0/24
```

where /24 is the CIDR mentioned nomenclature mentioned above.

More complete scans with reporting may be achieved with:

```
sudo nmap -sC -sS -sV -O -oX view.xml \
192.168.0.0/24
xsltproc view.xml -o view.html
elinks view.html
```

Network Sniffing

Having identified the nodes, we can now look at the data going to and coming from them. This is called 'sniffing'. First obtain the application:

```
sudo apt-get install ettercap-text-only
```

Once downloaded set both `ec_uid` and `ec_gid` from 65534 to 0 in `/etc/etter.conf`. Next:

```
sudo ettercap -T -i eth0 -M arp:remote \
-V ascii -d //53
```

This command causes ettercap to log any packets on port 53 of any address. (Replace eth0 with wlan0 to tap the WiFi port).

Ports are used to distinguish between network services. Common ports include:-

20	ftp-data	File Transfer [Default Data]
21	ftp	File Transfer [Control]
22	ssh	Secure shell
23	telnet	Telnet
25	smtp	Simple Mail Transfer
37	time	Network Time
53	domain	Domain Name Server
80	www-http	World Wide Web HTTP
443	https	Secure http

Your screen could be flooded with details of naming activity on your network or empty. If nothing is seen try pinging an unknown name on a remote machine to confirm the sniffer is working. Press H for a menu or Q to quit the application. Now try:

```
sudo ettercap -T -i eth0 -M arp:remote \
-V ascii -d /192.168.0.1-16/80
```

to log all http traffic (port 80) on the first 16 addresses.

Conclusion

We have covered a good deal of ground in these two articles achieving a least a working system with example commands that may be used as solid introduction to further investigation. For further aid visit the following:

Nmap Security Scanner:
http://nmap.org

Wireshark, a network procol analyser:
http://www.wireshark.org

Pr0mpt, a training program that introduces Open Source and programming:
http://pr0mpt.me

The Ettercap Project:
http://ettercap.github.io/ettercap

Description of IPv4 addressing:
http://en.wikipedia.org/wiki/IPv4

IPv4 address exhaustion:
http://en.wikipedia.org/wiki/IPv4_address_exhaustion

FRESHLY ROASTED
A beginners guide to Java

Vladimir Alarcón
& Nathaniel Monson
Guest Writers

3 - An introduction to classes

SKILL LEVEL : INTERMEDIATE

This is the 3rd article in the Java programming series. The two previous articles can be found in Issues 14 and 16 of The MagPi. In this article, arrays, the basics of classes, objects and their methods are introduced. The article also covers the Java Class Library, which provides thousands of useful functions.

Classes, objects and methods

Programs written in Java are not normally written as one very long class. Instead, programs are typically divide into functions that are commonly referred to as "methods" of a class. Ideally, there should be a method for every task that a class performs. A program usually has several classes, each one in turn has several methods (tasks). For example, a class could be written that represents a car that has methods to move forward, turn left, and turn right:

```
public class Car {
  public void forward(double metres) {
    System.out.println("Forward " + metres);
    // Add more instructions...
  }

  public void turnLeft(double degrees) {
    System.out.println("Left " + degrees);
  }

  public void turnRight(double degrees) {
    System.out.println("Right " + degrees);
  }
}
```

The idea of defining a class with separate methods that have clearly defined purposes is that it is straightforward to understand which method to use without reading the code within the method. If someone else is developing another class that calls the `Car` class methods, the other developer can understand the function of each method just from its name. Building programs with logical structure is a useful skill in any programming language, leading to fewer bugs and more efficient programs.

A class method should include:

- its visibility: (`public` in this case means the other class can execute it); other visibilities are `private` (cannot be used by other classes), `protected` and `package` (discussed later in this article).
- the return type, which might be any Java type or `void` which implies no return type.
- the name of the method, which should indicate what the method does.
- in between parentheses, a list of parameters that are necessary for the method to perform its task. For example, "`double metres`" in the first method.
- Finally, in between curly braces, the Java code that is executed and that performs the requested task.

Once a class has been written with one or more methods, the methods can be called by instantiating an object of the class and then calling its methods. Each class is a blueprint from which many objects can be created: there may be a single class `Car` but a thousand car objects, where each of them is independent in memory. An object of the class `Car` can be created by using the instruction new:

```
public class CarDriver {
  public static void main(final String[] args) {
    Car car1 = new Car();
    car1.turnLeft(20.0);
    car1.forward(5.0);
    Car car2 = new Car();
    car2.turnRight(45.0);
    car2.forward(12.0);
  }
}
```

This program includes a `CarDriver` class that creates two objects of the class `Car` and then calls the methods of the objects. The new instruction creates an object, where the associated class constructor is given to the right of new. (Unlike C++, it is not necessary to delete the objects created with new since the Java garbage collector automatically deletes them when they go out of scope.) The syntax for calling the method of an object is the object name, a dot and then the method name with any parameters in parentheses:

```
    car1.turnLeft(20.0);
```

Challenge #1: Execute the program `CarDriver`. To do this put the `CarDriver` class in a file called `CarDriver.java` and the `Car` class in a file called `Car.java`. Then compile both classes as shown in the first Java article. Then run the program by typing: `java CarDriver`

Challenge #2: Add a third object of class `Car` called `car3`. Call its methods to move forward 10 meters, turn right 5 degrees and finally move forward 14 metres.

It is possible to create a class that has methods that can be called directly, using the class name instead of an object. These are called `static` methods and are explained later in this article. The `Math` class has several methods that are `static`. For example, to get the absolute value of the number -12 :

```
Math.abs(-12)
```

where `Math` is the class name and `abs` is the method name.

Organising classes and methods

There are a lot of heated discussions on how to decide how many classes should be written for a given program and which methods to write for each class. This is an advanced subject that may be skipped for now, but becomes more important when long (a thousand lines or more) programs are written. This discipline is called "Object Oriented Design". Some documentation on the most important principles of design, usually called "Design Patterns", can be found by searching the Internet.

The Java class library

In addition to the Java language itself, every Java installation comes with a comprehensive collection of classes that provide additional functionality. This collection of classes is called the Java class library and contains classes for managing files, network connections, processing images and sound, real time 3D, using the mouse and keyboard, using databases, showing web pages, and many others. Documentation can be found at: http://docs.oracle.com/javase/7/docs/api Open a browser and look at this page. Every class belongs to a "package". The list of packages is shown at the top left of the browser window. Clicking on a package name causes all the classes in it to be listed in the box below. Clicking on a class name will load the class details into the main area on the right.

The `Math` class, that was used in the previous section, is located in the `java.lang` package. Using the upper left box, scroll down and find the `java.lang` package and then click on this package. The box below will display many classes in six separate sections: Interfaces, Classes, Enums, Exceptions, Errors and Annotation Types. Scroll down through the classes listed and click on the `Math` class. Now the main area will show a description of the `Math` class, with three optional sections: Fields Summary, Constructor Summary (there are no constructors for the `Math` Class), and Methods Summary. Read through the methods section.

Challenge #3: Find the methods: `abs`, `sqrt`, and pow. Click on each one and see what each one does.

Challenge #4: Find the class `java.io.FileReader` that reads files and the class `java.util.ArrayList`.

Arrays

Arrays provide sequential storage for many variables or objects. A variable or object can be selected using the index of the element concerned. To store the price of three paintings, simple variables could be used:

```
int price1 = 500;
int price2 = 390;
int price3 = 640;
```

However, if the number of individual variables that are related becomes large, then carrying them around quickly becomes cumbersome. Instead of individual variables, an array can be used:

```
int prices[] = new int[3];  // create an array of type int with three elements
prices[0] = 500;  // Assign 500 to the first element
prices[1] = 390;  // Assign 390 to the second element
prices[2] = 640;  // Assign 640 to the third element
```

which has the secondary benefit that the index can be used in a loop structure. Using this array declaration and the assignment of the three values, the prices of paintings can be printed:

```
System.out.println("Price for #1 is"+ prices[0]);
```

Once declared, the values of the array elements can be assigned anywhere within the program. For example, the price of the 3rd painting can be increased:

```
prices[2] = 710;
```

The map generator

This program is composed of two classes: `MapGenerator` and `Terrain`. The `MapGenerator` class uses a `Terrain` object to draw items. The `Terrain` class is given below:

```
import java.util.Random;

public class Terrain {
  private Random r;
  private char[][] tiles;

  public Terrain(int mapNumber, int height, int width) {
    this.r = new Random(mapNumber);
    this.tiles = new char[height][width];
  }

  public void setArea(int fromH, int fromW, int toH, int toW, char symbol) {
    for (int x = fromH; x < toH; x++) {
      for (int y = fromW; y < toW; y++) {
        this.tiles[x][y] = symbol;
      }
    }
  }

  public void setRandomAreas(int height, int width, char symbol, int howMany) {
    for (int i = 0; i < howMany; i++) {
      int x = this.r.nextInt(this.tiles.length - height + 1);
      int y = this.r.nextInt(this.tiles[0].length - width + 1);
      this.setArea(x, y, x + height, y + width, symbol);
    }
  }

  public String show() {
    StringBuilder sb = new StringBuilder();
    for (int x = 0; x < this.tiles.length; x++) {
      for (int y = 0; y < this.tiles[x].length; y++) {
        sb.append(this.tiles[x][y]);
      }
      sb.append("\n");
    }
    return sb.toString();
  }
}
```

The `Terrain` class stores a map (of letters), using a two dimensional character array. A `Terrain` object is initialised as an empty container, but can be modified using its methods. The contents of the character array can also be returned as a string by calling the `show()` method.

The MapGenerator class is given below:

```
public class MapGenerator {
  public static void main(String[] args) {
    if (args.length != 3) {  // If less than three additional arguments are provided
      System.out.println("I need three parameters: "          + "map-number, height and width.");
      return;
    }
    int mapNumber = new Integer(args[0]);  // Create an integer version of the map number
    int height = new Integer(args[1]);  // Create an integer version of the map height
    int width = new Integer(args[2]); // Create an integer version of the map width
    Terrain land = new Terrain(mapNumber, height, width);
    land.setArea(0, 0, height, width, '.'); // ground
    land.setRandomAreas(1, 1, 't', 30); // trees
    land.setRandomAreas(height / 4, width / 4, 'T', 3); // woods
    land.setRandomAreas(1, 1, 'o', 20); // rocks
    land.setRandomAreas(height / 4, width / 5, ' ', 3); // lakes
    land.setRandomAreas(height / 6, width / 10, ' ', 12); // ponds
    land.setRandomAreas(1, 1, '$', 1); // treasure
    System.out.println(land.show());
  }
}
```

The `MapGenerator` class creates a `Terrain` object, adds objects to it and finally displays it. The map is initialised with a height and width. Then trees, water and treasure are added at random. Finally, the `show()` method is used to print the map on the screen.

Challenge #5: Compile the classes `MapGenerator` and `Terrain`. Then execute the `MapGenerator` program, providing the map number, the length and the width, e.g.: `java MapGenerator 6 15 38`

The result of the 5th challenge should be:

```
......t       ........................
....TTT               o.t.....t...t..t
..t.TTo               ................
....TTTTTTTTTTT       ................
.....oTTTTTTToT   .....t......t.......
......TTTTTTTTT   ....................
.........ot.......o..........o   ..t..
..o.....   ......t.o..........   ....t
...t....   ...t.......o.       TTTT$..
..t....t................       T   ...
..........   t......t...       T   ...
.o..t.....   ...........   .t.t..o....
.....t............   ...   ........
..t....   .....t..   ...   ....o...
.......   ....o........o..............
```

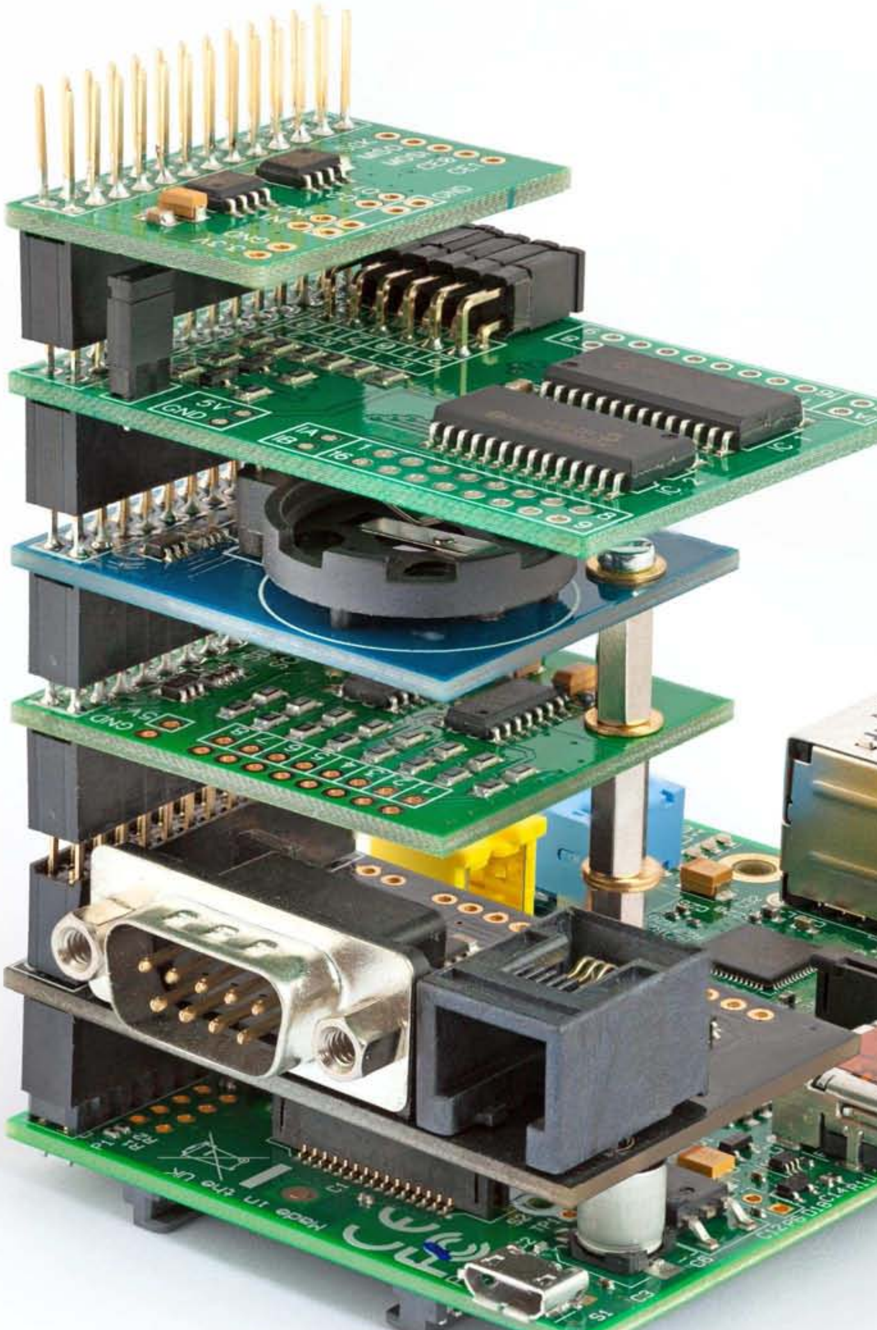
where the dots represent the land, the blanks represent water, the o characters represent rocks, T and t are trees and $ is the hidden treasure.

Challenge #6: Execute the MapGenerator program with different values to see different maps. For example:
`java MapGenerator 7 15 34`
Try other maps too and see how the MapGenerator class works.

Congratulations! You have learned the basics of arrays, classes and methods!

Expand your Pi

Stackable Raspberry Pi expansion boards and accessories



ADC-DAC Pi

2x 12 bit analogue to digital channels and 2x 12 bit digital to analogue channels.

IO Pi

32 digital input/output channels for your Raspberry Pi. Stack up to four IO Pi boards to give you 128 I/O channels.

RTC Pi

Real-time clock with battery backup and 5V I^{2}C level converter for adding external 5V I^{2}C devices to your Raspberry Pi.

ADC Pi

8 channel analogue to digital converter. I^{2}C address selection allows you to add up to 32 analogue channels to your Raspberry Pi.

Com Pi

RS232 and 1-Wire® expansion board adds a serial port to your Raspberry Pi. Ideal for the Model A to enable headless communication.

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.i
            raise StopIterati
        self.fact *= self.count
        if self.fact < 1:
            self.fact = 1
        self.count+=1
        return self.fact
```

Pocket Enigma Cipher Machine

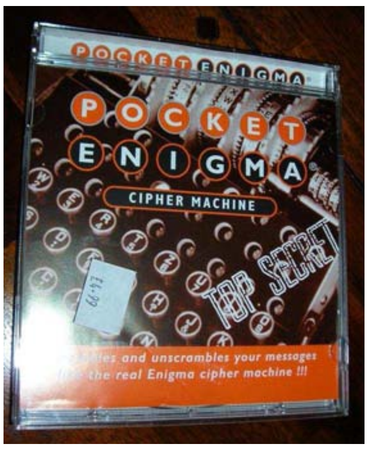
Ian Neill

Guest Writer

SKILL LEVEL : ADVANCED

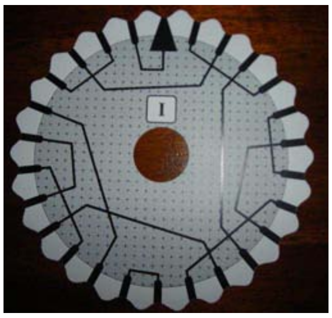
The Pocket Enigma Cipher Machine is a superbly designed toy that demonstrates some of the principles of a real Enigma Cipher Machine, as pictured on the cover of this issue. The Enigma Cipher Machine was used during World War two by the German armed forces, to send secrete messages to submarines and solders. I obtained my Pocket version from Bletchley Park [http://www.bletchleypark.org.uk/], but unfortunately it no longer appears to be on sale. It is made only of plastic and cardboard and is substantially simpler when compared with a real Enigma cipher machine! On the other hand, if you enjoy encryption and ciphers you will get a kick out of the Pocket Enigma. It is not too difficult to understand either.

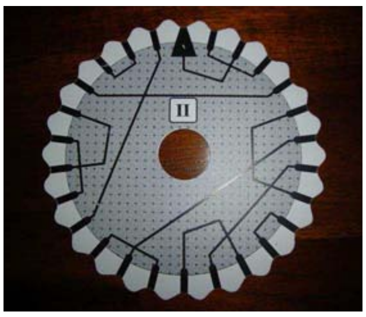
The Pocket Enigma Cipher Machine is not even close to an unbreakable cipher – it is a trivial cipher to break but it is fun. Therefore, do not use it to encrypt sensitive information. A full review of the Pocket Enigma Machine, including a detailed description and further reading, can be found at:
http://www.savory.de/pocket_enigma.htm



How does it work?

Each plaintext character is replaced with another character called the cipher text character. The cipher text character is chosen according to the connection between characters printed on the wheel, where there are two wheels to choose from.





In more detail, the algorithm follows:
1. Cipher wheel (1 or 2) is chosen.
2. The key character is chosen.
3. The start character is chosen.
4. The wheel is set to the key character and the start character is encoded.
5. The wheel is moved to the start character and the first message character is encoded.
6. The wheel is incremented by 1 position, and the next message character is encoded.
7. Repeat step 6 until the entire message is encoded.
8. The encoded message is arranged such that the encoded start character is separated from rest of the encoded message, which is arranged in blocks of, typically, five characters.

For the message to be successfully decoded by the recipient, they must already know the wheel number and key character that was used to encrypt the message.

Now for the limitations:
1. Only upper case characters can be encoded.
2. No punctuation can be encoded, apart from full-stops which are traditionally substituted with X.

With a bit of imagination the encoding algorithm can easily be modified. For example, more wheels could be used, or the increment could be varied or even reversed.

Python Pocket Enigma Cipher Machine

Use a text editor such as nano or emacs to create a new Python file called `Py-Enigma.py`. Then add:

```
#!/usr/bin/python

VERSION = "v1.0.0"
BUILDDATE = "01/06/2014"

# Global variables with starting values
selectedWheel = 1
pointerChar = 'D'
pointerInt = ord(pointerChar)-65
codeStartChar = 'J'
codeStartInt = ord(codeStartChar)-65
increment = 1
blocksize = 5
```

Py-Enigma.py (1/8)

to the top of the file. In this article each piece of the program is numbered and should be appended to this file, to create the final program. The `ord()` function returns the numerical value of an ASCII character. `ord('A')` returns 65, whereas `ord('B')` returns 66 etc.. Lower case characters have different numerical values, but since the cipher only has one set of characters, upper case characters are used throughout the program. The `selectedWheel` variable is used to select which wheel is used, `pointerChar` is the initial wheel settings and `codeStartChar` is the starting character. Integer values of these variables are also stored to simplify manipulating the wheels within the functions that follow. The `increment` is the amount that a wheel is turned after each character is encrypted and `blocksize` is used to split the encrypted string into pieces that are separated by spaces.

1. Analysis of the wheels

The wheels have no characters on them, just a lot of connections. One position has an arrow, or pointer, and is taken as the starting position (actually position 0). Looking at the pictures on the first page of this article, it is clear that the connections simply connect from one position to another. These connections indicate how one character should be substituted for another. The wheels can be summarised using two Python lists:

```
# Wheel definitions
wheel1 = [-1,3,-5,7,-3,2,3,-2,4,-3,-7,6,-4,1,-1,6,3,-6,2,-3,-2,-6,2,5,-2,1]
wheel2 = [2,2,-2,-2,-8,3,9,5,-3,1,-1,2,-5,-2,2,-9,-2,8,2,2,-2,-2,8,1,-1,-8]
```

Py-Enigma.py (2/8)

Add these two lists to the end of the Python file. Each list has 26 entries, since there are 26 characters around the outside of the wheel. The number in each entry corresponds to the joining line on the wheel, where a negative number implies moving to the left and a positive number implies moving to the right.

The Python version of the algorithm relies on the modulo (%) operator to stay within the A--Z character range. First, the character should be converted to an integer. Then the offset should be applied, using the modulo operator. For example, using `'A'` and the first wheel:

```
intValue = ord('A') - 65  # returns 0
intValue = intValue + wheel1[intValue] # returns -1
intValue = intValue % 26 # returns 25
charValue = chr(intValue + 65) # returns 'Z'
```

If the number is bigger than the 26 character range, then the modulo operator causes the number to become less than 26. This means that adding 1 to the value of `'Z'` returns `'A'`:

```
intValue = ord('Z') - 65  # returns 25
intValue = intValue + wheel1[intValue] # returns 26
intValue = intValue % 26 # returns 0
charValue = chr(intValue + 65) # returns 'A'
```

In both of these examples, the `chr()` function is used to convert an integer value back into the associated ASCII character. It helped me to visualise the modulo maths by imagining that it turned the alphabet into a circle.

2. Encrypting or decrypting a character

The Pocket Enigma algorithm states that the wheel should be moved 1 position clockwise after each message character is encoded. This means that a repeated character in the message is not encrypted as the same character. Append the code below to the end of the Python file.

```
# Encrypt or decrypt a single character
def transformChar(character, selectedWheel, pointer):
  character = character.upper()  # Ensure that the character is Upper Case.
  if(65 <= ord(character) <= 90):  # Only characters A-Z can be encrypted or decrypted.
    char_num = ord(character) - 65  # Convert ASCII to alphabetical position of the character.

    # Choose the offset for wheel one or two.  Then use the pointer value.
    if (selectedWheel == 1):
      offset = wheel1[(char_num - pointer)%26]  # Use mod with 26 to stay within circle
    else:
      offset = wheel2[(char_num - pointer)%26]  # Use mod with 26 to stay within circle

    # Convert alphabetical position of the character back to ASCII
    character = chr(65 + (char_num + offset)%26)  # Convert position back to ASCII character
  else:
    character = ''  # Ensure that nothing is returned if the character is not A-Z.
  return character
```

Py-Enigma.py (3/8)

This function takes an input character, the selected wheel number and the current pointer position. The pointer represents the position of the wheel and is substracted from the integer value of the character before it is used to find the offset. If a character that is not within the A--Z range is passed into the function, then it is ignored and no character is returned.

3. Encrypting a string

To encrypt a string, each character should be passed to the `transformChar()` function. Append the code below to the Python file.

```
# Encrypt a string
def encrypt(plaintext):
  pointer = pointerInt  # Set the wheel to the key character, using the global variable
  cipher = ''
  # Encrypt the Alpha Start character.
  cipher += transformChar(codeStartChar, selectedWheel, pointer)
  cipher += ' '
  pointer = codeStartInt # Set the wheel to the Alpha Start character.
  block = 0

  # Encrypt each letter in the plaintext string
  for o_char in plaintext:
    # Substitute '.' with 'X'
    if o_char == '.':
      o_char = 'X'

    # Encrypt this character
    e_char = transformChar(o_char, selectedWheel, pointer)

    # Do something if the character was encrypted ok.
    if len(e_char) > 0:
      block += 1
      if block > blocksize:
        cipher += ' ' # Add a space after a block of blocksize characters.
        block = 1   # Remembering the character that was blocksize+1.

      cipher += e_char  # Add the character to the result.
      pointer = (pointer + increment)%26  # Turn the wheel, using mod 26 to return to zero
  return cipher
```

Py-Enigma.py (4/8)

The function takes a string and returns an encrypted string. The pointer starts from the initial value (key character) set using the global variable `pointerInt`. Then the starting character is encrypted and appended to the encrypted string. The pointer value is reset to the starting character and then each character in the string is encrypted. To retain some punctuation, each `'.'` is replaced with `'X'`. The encrypted output is also split into fixed size blocks that are separated by spaces to help further disguise the words.

4. Decrypting a string

The connections on the wheels are bi-directional. Therefore, if a character is encoded as `'F'` and the wheel is in the same position, encoding `'F'` will return the original character. Consequently, the same encryption routine can be used to decrypt a string. Append the program at the top of the next page to the Python file. This function takes an encrypted string and returns a decrypted string. Notice that punctuation and spaces are not recovered during the encryption. Therefore, the person that receives the encrypted message will need to put those back in by hand.

```
# Decrypt a string
def decrypt(cipher):
  pointer = pointerInt  # Set the wheel to the key character.

  # Extract and decrypt the Alpha Start character.
  pointer = ord(transformChar(cipher[:1], selectedWheel, pointer))-65

  plaintext = '' # Output string with no characters
  # Decrypt each letter in the cipher.
  for e_char in cipher[1:]:
    # Decrypt this character
    o_char = transformChar(e_char, selectedWheel, pointer)

    # Do something if the character was decrypted ok.
    if len(o_char) > 0:
      plaintext += o_char   # Add the character to the result.
      pointer = (pointer + increment)%26   # Turn the wheel, using mod 26 to return to zero
  return plaintext
```
Py-Enigma.py (5/8)

5. Welcome, menu & input functions

To call the encrypt and decrypt functions, a text menu provides a simple interface. Add the code given below to the end of the Python file.

```
# Welcome message
def welcome(message):
  print(message)
  print("   Version, %s, %s" % (VERSION, BUILDDATE))

# Print the available menu options
def showMenu(min, max, quit):
  print("\n" + 30 * '-')
  print("      P y - E N I G M A")
  print("      M A I N - M E N U")
  print(30 * '-' + "\n")
  for i in xrange(min,max+1):
    if i == 1:
      print(" 1. Set Wheel      = %d" % selectedWheel)
    elif i == 2:
      print(" 2. Set Pointer    = %s" % pointerChar)
    elif i == 3:
      print(" 3. Set Code Start = %s" % codeStartChar)
    elif i == 4:
      print(" 4. Set Increment  = %d" % increment)
    elif i == 5:
      print(" 5. Set Block Size = %d" % blocksize)
    elif i == 6:
      print(" 6. Encrypt a Message")
    elif i == 7:
      print(" 7. Decrypt a Message")
    elif i == 8:
      print(" 8. Nothing Yet")
    elif i == 9:
      print(" 9. Nothing Yet")
    else:
      continue

  print("\n %d. Exit program\n" % quit)
  print(30 * '-')
```
Py-Enigma.py (6/8)

The first function is used to report the version and build date, whereas second function prints out the menu choices. To read values that correspond to the menu options three simple input functions are needed. Add the functions below to the end of the Python file.

```
def getValue(request="Enter choice: "):
  while 1:
    inputValue = raw_input(request)  # Print request and read the reply
    if len(inputValue) == 0:  # If the string has no length, report an error
      print("Error: no value given")
      continue
    return inputValue  # If the string has one or more characters, return the value

def getInteger(min,max,request,checkRange = True):
  while 1:
    inputValue = getValue(request)  # Print request and read reply
    try:
      intValue = int(inputValue)  # Try to convert the string to an integer
    except ValueError:
      print("Error: \"%s\" is not an integer" % inputValue)  # If it is not an integer
      continue
    if (intValue < min or intValue > max) and checkRange:  # Check the range if needed
      print("Error: \"%d\" is outside range [%d--%d]" % (intValue,min,max))
      continue
    return intValue  # Return the integer value.

def getCharacter(min,max,request):
  while 1:
    inputValue = getValue(request)  # Print request and read the reply
    if len(inputValue) != 1:   # A character is a string of length 1
      print("Error: \"%s\" is not a single character" % inputValue)
      continue
    charValue = inputValue.upper()  # Convert the string to upper case
    if ord(charValue) < ord(min) or ord(charValue) > ord(max):  # Check the range
      print("Error: \"%s\" is outside range [%s--%s]" % (charValue,min,max))
      continue
    return charValue  # return the character value
```

Py-Enigma.py (7/8)

These functions are used to read a value from the keyboard, read an integer value and read a character value, respectively. The functions prevent a string without any characters from being accepted, check the type of the input string and if it is within the allowed range of numbers or characters.

6. Finishing the program

To finish the program, a function is needed to call all of the other pieces of the program. Add the main function at the top of the next page to the end of the Python file. Then save the file and make it executable by typing:

```
chmod 755 Py-Enigma.py
```

Then run the program by typing:

```
./Py-Enigma.py
```

The program will print the menu and allow changes to the settings to be made. If the settings are updated, then the menu is printed again with the new values. The input functions are used to make sure that the settings do not go outside of their allowed range. The menu can also be used to encrypt or decrypt strings, where the result is printed on the screen. With the default settings of wheel 1, key character `'D'` and start character `'J'`, the message `"Attack at dawn."` becomes `"M UQXZI MGAZE DKS"`. Decoding this returns `"ATTACKATDAWNX"`.

```
# Main function
def main():
  global selectedWheel, pointerChar, pointerInt, codeStartChar, codeStartInt, increment, blocksize
  welcome("Py-Enigma - The Pocket Enigma Cipher Machine")
  menuMin = 1
  menuMax = 7
  menuQuit = 0

  while 1:
    showMenu(menuMin, menuMax, menuQuit) # Show the menu

    # Get the user choice, without checking the range
    userChoice = getInteger(0,0,"Enter choice [%d--%d]: " % (menuMin, menuMax),False)

    # Take action as per selected menu-option.
    if userChoice == menuQuit:
      break # Leave the while loop
    elif userChoice == 1:
      selectedWheel = getInteger(1,2,"Enter Coding Wheel [1 or 2]: ")
    elif userChoice == 2:
      pointerChar = getCharacter('A','Z',"Enter Pointer Position [A to Z]: ")
      pointerInt = ord(pointerChar)-65
    elif userChoice == 3:
      codeStartChar = getCharacter('A','Z',"Enter Coding Start Position [A to Z]: ")
      codeStartInt = ord(codeStartChar)-65
    elif userChoice == 4:
      increment = getInteger(-1,1,"Enter Increment [-1, 0 or 1]: ")
    elif userChoice == 5:
      blocksize = getInteger(1,10,"Enter Block Size [1 to 10]: ")
    elif userChoice == 6:
      plaintext = getValue("Enter Plaintext: ")
      print("Encryption: %s => %s" % (plaintext,encrypt(plaintext)))
    elif userChoice == 7:
      cipher = getValue("Enter Cipher: ")
      print("Plaintext: %s => %s" % (cipher,decrypt(cipher)))
    else:
      print("Error: \"%d\" is not a valid choice" % userChoice)

  print("\nGoodbye.\n")


#Run the program if it is the primary module
if __name__ == '__main__':
  main()
```

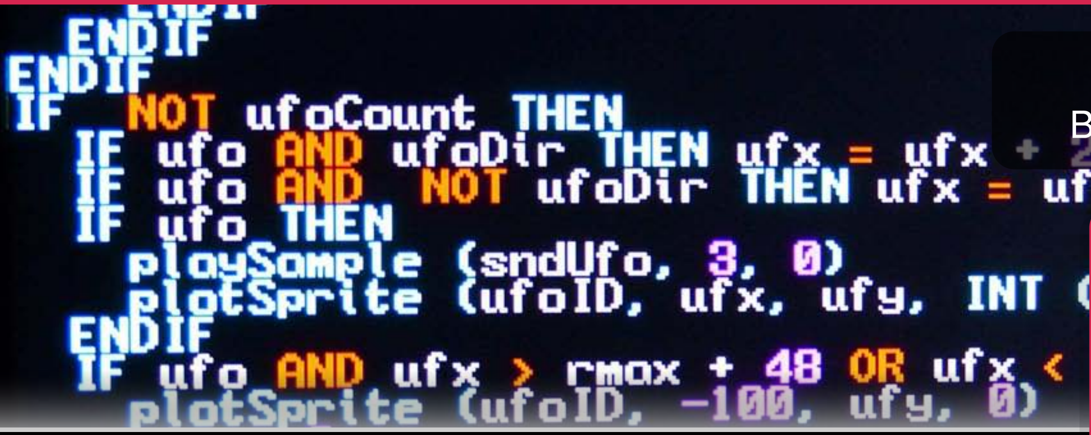
Py-Enigma.py (8/8)

The `main()` function starts by declaring the global variables as `global`. This is necessary to prevent Python from creating a local version of the same variable when a value is assigned. It is not needed if the values are only used. The welcome message is printed. Then in the `while` loop, the menu is printed. The users choice is read and checked against each of the menu options. Depending on the menu option, the required action is taken.

The `Py-Enigma.py` program is a Python version 2 program that cannot be used with Python3 without modification. The program was tested on the latest Raspbian image.

What next?

Well that depends... let me know what you think. All/any feedback is appreciated.



```
ENDIF
  ENDIF
ENDIF
IF NOT ufoCount THEN
  IF ufo AND ufoDir THEN ufx = ufx + 2
  IF ufo AND NOT ufoDir THEN ufx = ufx - 2
  IF ufo THEN
    playSample (sndUfo, 3, 0)
    plotSprite (ufoID, ufx, ufy, INT (
  ENDIF
  IF ufo AND ufx > rmax + 48 OR ufx <
    plotSprite (ufoID, -100, ufy, 0)
```

Jon Silvera

Guest Writer

Part 1: Back to BASIC

SKILL LEVEL : BEGINNER

Over the next few months The MagPi will publish a new series on learning BASIC. To whet your appetite, this month we will provide some background to the language and explain why it is still a great way to learn how to program.

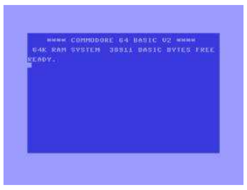
There's a lot of talk about programming at the moment. The UK Government has done a fair bit to get it back on the education curriculum to make sure UK schools are teaching kids how to program from an early start. In fact, from this September schools in England must teach the fundamentals of computing in early primary school, and text based programming from key-stage 3 onwards (age 12).

While there may be some debate about how programming is being introduced into the cirriculum, what is important is that something is being done.

In the beginning...

Roughly 2 million years ago computers like the Research Machines 380Z were bulky things with clunky keyboards, terrible graphics and sound. Actually, some of them didn't have clunky keyboards. Instead they had little rubber pads for keys that felt like a cat's tongue!

These machines were limited in so many ways. For example, the Sinclair ZX Spectrum had just one sound channel and the only noise it could make was a beep. In fact, the command it had for this was BEEP! `BEEP 1, 0` would make a C tone (0) for one second (1). It gets worse. While it was beeping it was thought that it couldn't do anything else, so to make a game with music playing was considered impossible.



```
**** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM  38911 BASIC BYTES FREE
READY.
```

The Commodore 64 had huge great borders around the screen which again was originally considered impossible to display anything on. So early games were displayed on the letterbox in the middle of the screen.

The BBC Micro originally had no hardware to display character graphics (sprites) so it was originally considered impossible to program games in anything other than text.

Then the wizards came. They made the Spectrum sing and dance; turned the BBC Micro into an arcade machine with almost perfect renditions of Pac Man, Defender and Asteroids; and the Commodore 64 became a graphical and audio tour de force. It really was the strangest time and of course it was not 2 million years ago, but just 30 years ago.

It all started back in the mid to late 1970's with companies like Atari and Apple. Atari decided to turn its console technology into something more sophisticated. The Atari 400 and 800 computers were ahead of their time in 1979. High resolution graphics, 8 KB of RAM, up to 256 colours and 4 channel sound. Apple launched the Apple I in 1976 and then soon after the Apple II. These were powerful machines, but they were also very expensive.

Compare this to the Sinclair ZX80 which was released in the UK, a year after the Atari 400, by the somewhat lesser known Clive (later to be Sir Clive) Sinclair. The Sinclair ZX80 had no sound, no graphics, no colour and had just 1 KB RAM.

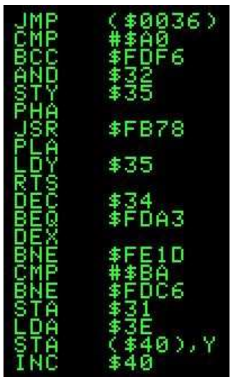
However, it was cheap... less than £100 cheap... and all things considered it can be held partly responsible for kick-starting the home computer industry in the UK.

A brain by any other name

It would be very unfair to credit any one company for the advent of the home computing revolution. It would be more considerate to attribute this to those responsible for designing and manufacturing the CPUs and custom silicon that were the bedrock of all the computers of the day. While many home computers were released between 1979 and 1984, there were only three commonly used CPUs (Central Processing Unit). The CPU is the brain of any computer and, in general, everything else within the computer either is controlled by or feeds information to the CPU.

The CPUs of the time were the Zilog Z80, the MOS 6502 and the less common Motorola 6800. Their hardware and software designers are the people responsible for the digital world we live in today.

While all these machines were designed by different companies with different specifications and configurations, they all had one major thing in common. They were all supplied with the programming language BASIC. (Ok, not every single one came with BASIC. The Jupiter Ace was supplied with Forth!)



```
JMP    ($0036)
CMP    #$A0
BCC    $FDF6
AND    $32
STY    $35
PHA
JSR    $FB78
PLA
LDY    $35
RTS
DEC    $34
BEQ    $FDA3
DEX
BNE    $FE1D
CMP    #$BA
BNE    $FDC6
STA    $31
LDA    $3E
STA    ($40),Y
INC    $40
```

The earliest computers were generally programmed in either bare metal mode, known as machine code, or in the slightly more friendly Assembly Language. Machine code is often cited as being the most complicated programming language ever devised, but it is actually one of the simplest and it certainly has the smallest number of instructions to learn. The complexity comes from having to know the hardware inside-out: how the memory is structured, the input/output system, the video capabilities, etc. We won't even go into interrupts, storage or sound.

Next came a few languages designed to perform specific tasks. COBOL, for example, was suited to business applications whereas Fortran was suited to mathematics and science.

Between the late 1950's and the early 1970's many programming languages were developed. A number of them have survived... or at least variations of them have. C is possibly the best example as this is still one of the most commonly used languages today.

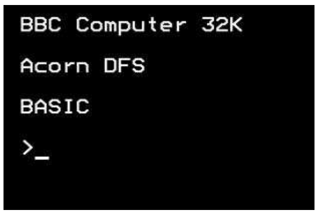
Welcome to BASIC

So what about BASIC then? BASIC (Beginner's All-purpose Symbolic Instruction Code) was written to simplify programming into a more understandable, less mathematically reliant language that could be used to develop a wide variety of applications.

First launched 50 years ago, in 1964, as Dartmouth BASIC by John Kemeny and Thomas Kurtz, its accessibility meant it went on to spawn many other versions over the next 30+ years. See http://www.dartmouth.edu/basicfifty for more info.

The first software product sold by Microsoft was a version of BASIC co-written by Bill Gates. It went on to ship with every IBM PC and many others for years to come. A number of Microsoft BASIC products are still incredibly popular. Visual BASIC and VB.NET are in common use today.

Back to the 1980's



```
BBC Computer 32K

Acorn DFS

BASIC

>_
```

In their heyday, the popular computers of the time - Apple, Commodore, Tandy, Sinclair, Atari and the Acorn

BBC Micro - were all supplied with BASIC built in. Switch on and you would soon see the "Ready >_" prompt or something similar. Type in something, press the ENTER key and see what happened. At first you would invariably get a "Syntax Error", but you soon picked up the user guide and started reading tutorials. The first thing to learn was,

```
10 PRINT "Hello world!"
20 GOTO 10
```

followed quickly by how to load a game from a cassette or, if you were very lucky, a disk drive!

Then came the magazines with game and program listings you could type in. Countless hours were spent entering these listings, often with very poor print quality, only to be bombarded with error after error when you came to run them.

The learning curve was steep, but fast, and before you knew it you were experimenting with your own programs and showing off to friends. If you had it in you, you would be applying for a programming job in a fledgling game publisher or trying to set up you own.

That an industry was born is not surprising, as over 50 million 8-bit home computers were sold globally. The exposure to programming grabbed the interest of many.

What happened next however was a bit odd. At the time, programming was seen as the next big thing and so it was taught in schools, done at home or at a friends place and so on. However, it reached a critical mass fairly quickly. Programmers went on to develop more powerful programming languages, gaming systems took over, small publishing companies grew into huge corporations and programming became something of an elite occupation. As the industry settled down, programming all but dropped off the curriculum.

Back to BASIC

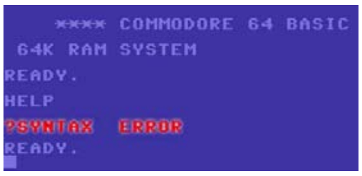
But 30 years later and now there is a huge wakeup call by many that has prompted the UK Government to realise that we do not have enough programmers available in the UK to supply demand. But this time around computers are "black boxes", with no easy way to learn about physical computing. Instead the entry point is a complex programming environment, developed by programmers for programmers. So where do we start?

At primary school children are being introduced to visual programming environments like Scratch that simplify the programming experience and do a great job of introducing computational concepts. They do not however offer the exposure to advanced programming that inspired so many of today's entrepreneurs the first time round.

In secondary school it looks like Python, another language derived from the early days, is being positioned as the language to best prepare students for the real world. However, BASIC is even easier to learn and the concepts learned with BASIC will provide a solid foundation for learning other languages.

BASIC for today and tomorrow

BASIC has continued to be developed over the years to keep it up-to-date and comparable to many other modern languages. Commands that were once considered bad practice (e.g. GOTO and GOSUB) along with line numbers are gone and replaced with structured programming techniques.



```
    **** COMMODORE 64 BASIC
 64K RAM SYSTEM
READY.
HELP
?SYNTAX  ERROR
READY.
```

The main attraction to BASIC is its instant accessibility: enter your program, RUN it, get an error, fix it and try again.

Thirty years ago those original computers ran BASIC slowly - it was its Achilles heel. You could just about do anything with it apart from do it quickly. That was why once you had mastered the basic principles you quickly advanced on to more complicated, compiled languages.

Today, BASIC runs on machines measured in gigahertz, not megahertz, and with gigabytes of RAM, not kilobytes. It is fast enough to program sophisticated games, especially when compared to the applications being published on today's mobile devices and tablets.

So if you are interested in learning to program but were wondering where to start, you will do yourself a favour by following our BASIC tutorials with your Raspberry Pi over the coming months.

http://www.fuze.co.uk

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 enquiries@themagpi.co.uk, 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.co.uk. We look forward to reading your comments.