ISSUE 22 - APR 2014

Get printed copies at themagpi.com

The MagPi

A Magazine for Raspberry Pi Users

140% More Solar Energy

Wyliodrin

Night Light

Laika Explorer

Automatic Garage

LEGO® Interfacing

Database Bootcamp

Scratch I/O Expansion

Three chances to win a Gertboard mount & 32GB SD card

ISSN 2051-9990

The MagPi

http://www.themagpi.com

The MagPi 22

Welcome to Issue 22 of The MagPi magazine.

We kick off this month's issue with an article on solar tracking. Nathan and Nicholas introduce their amazing project, the 'Reflective Solar Tracker', a solar cell with the brains of a Raspberry Pi, capable of chasing the sun to improve energy capture by up to 140%!

Andy Baker returns following his successful quad-copter series and this time he is featuring his intelligent night light. Andy describes building this project to reassure his son at night and banish those 'under-bed monsters'. It's a great mix of both the PiBow case and PiGlow add on-board.

Following its recent success on Kickstarter we examine the Laika Explorer, a digital Input/Output board for the Raspberry Pi with the genius additions of USB and expansion ports.

We welcome back Richard Wenner for his second article for The MagPi, where he continues his tutorial on using SQL by explaining how to insert and view stored data. Another welcome return is Philip Munts where he describes how to interface the Raspberry Pi to LEGO® Power Function motors. We also have more upcoming Raspberry Pi events from around the world and more book reviews to sink your teeth into.

To finish up, we go out with a bang with three great articles. There is a home automation tutorial on controlling your garage door over the internet, we show you how to remotely program your Raspberry Pi using the superb Wyliodrin platform and finally we show you how to control Input/Output devices using Scratch. What more Raspberry Pi themed goodness could you possibly want?

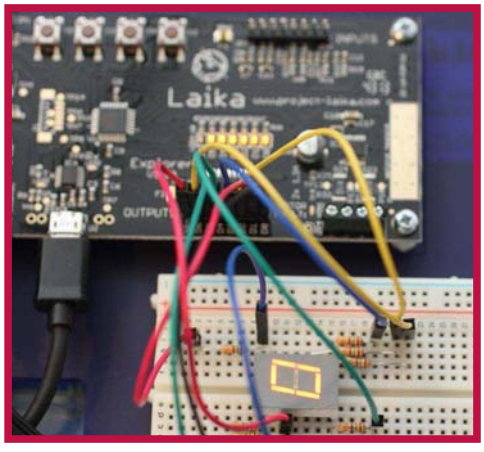
To follow our progress you can like us on Facebook at http://www.facebook.com/MagPiMagazine to keep up to date and give us more of your valued feedback.

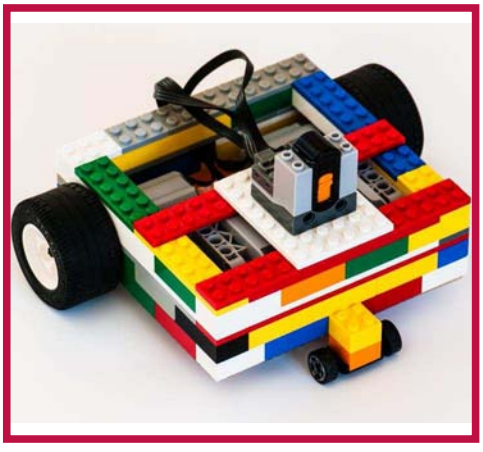
Chief Editor of The MagPi

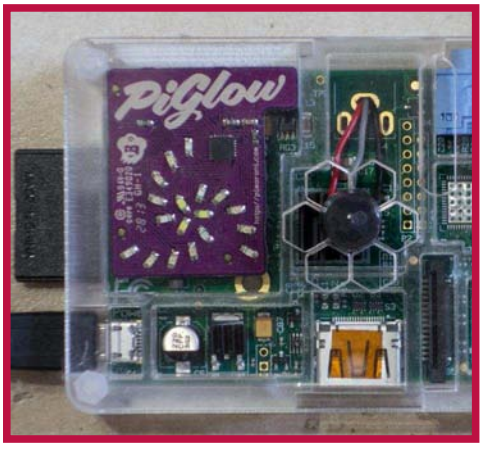
The MagPi Team

Ash Stone - Chief Editor / Administration
Les Pounder - Issue Editor
W.H. Bell - Layout / Administration
Bryan Butler - Page Design / Graphics
Ian McAlpine - Layout / Proof Reading
Matt Judge - Website / Administration
Aaron Shaw - Layout
Nick Hitch - Admin
Colin Deady - Layout / Testing / Proof Reading
Age-Jan (John) Stap - Layout
Sai Yamanoor - Testing
Claire Price - Layout
Shelton Caruthers - Proof Reading
Nigel Curtis - Proof Reading
Paul Carpenter - Testing
James Nelson - Proof Reading

Contents







Reflective solar tracking control system

Nathan D. Williams & Nicholas P. Truncale
Guest Writers

SKILL LEVEL : INTERMEDIATE

Mechanical tracking systems are available to reposition a variety of devices including radio telescopes, antennae, and even television satellite dishes. These systems are large and often unaffordable for the small project developer. We are team members of a collaboration including two universities and a non-profit sustainable energy entity whose goal is to develop and commercialise a new solar energy collection device.

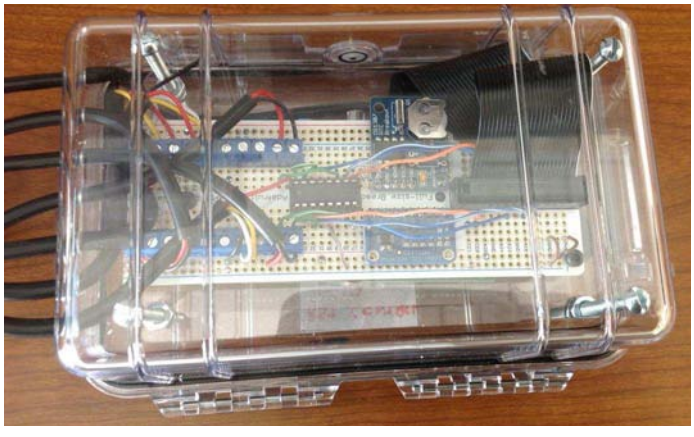
This article will describe the new patent pending device called the Reflective Solar Tracker (RST) and will showcase the control systems function. The control systems' main hardware component is the Raspberry Pi, whose primary algorithm uses astronomical data to reposition the RST and collect voltage data for power and energy calculations.

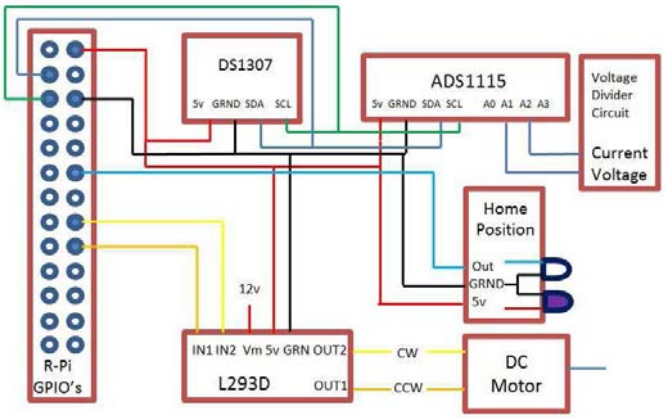
The Reflective Solar Tracker

This device utilises both reflected sunlight via Mylar panels and a rotating base platform to increase the energy density impinging on commercially available solar panels. The extra sunlight from reflection saturates the individual crystalline solar cells while the rotating base ensures the saturation takes place for a longer portion of the day compared to conventional stationary installations. The repositioning on the base platform is accomplished using a gear and worm screw turned by a low power DC motor controlled by the Raspberry Pi. The following describes the components for controlling a bi-directional DC motor, collecting voltage and current data from a solar panel installation, and the algorithm using astronomical data to guarantee the RST is always sun facing.

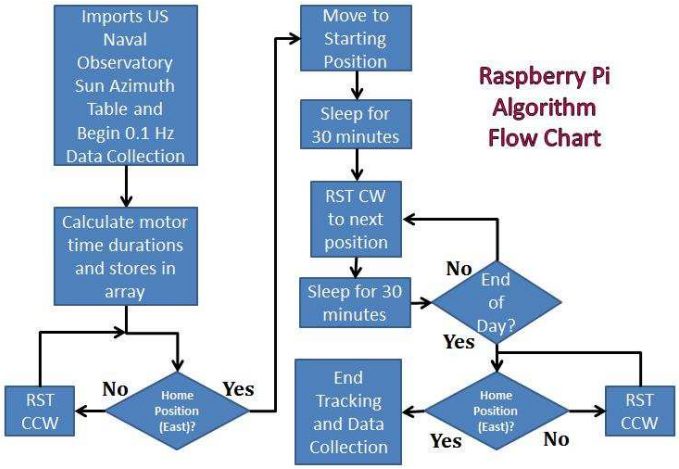
Hardware description

The hardware architecture for the Raspberry Pi control system functions with the use of products from http://www.adafruit.com specifically the DS1307 Real Time Clock, ADS1115 16-Bit ADC, L293D H-Bridge IC, IR home-position sensor and a 5200mAh power bank. The real-time clock is used to ensure the program executes daily at the proper time in case of power failure. To minimise the chance of failure, the Raspberry Pi is powered by a power bank, which is simultaneously being charged by a 5V 1A power adapter. The motor is directly controlled by an H-Bridge that is interfaced with the Raspberry Pi GPIO pins. In addition to the H-Bridge for motor control, an IR home-position sensor is used to ensure the motor is returned to the proper location at the end of every cycle. The hardware architecture also includes an analog-to-digital converter that allows us to collect the solar panel output voltage data throughout the day. We also collect the voltage across a 1Ω power resistor giving us the current. The connections of all of these components can be viewed in the provided block circuit diagram.





Tracking and data collecting algorithm



The primary algorithm, executed by a daily `cron` job, retrieves a local sun azimuth table containing the Sun's position in degrees relative to the eastern direction for the installation's geographic location in the current month. The hardware allows the motor to turn the base platform in both the clockwise (CW) and counter clockwise (CCW) directions. Beginning at 6.00am, the algorithm calculates the starting direction in reference to East and moves the base platform CW or CCW to this position. Throughout the rest of the day, the motor is turned on for a specific interval every half hour and moves the base platform CW, in the direction of the sun's trajectory, to the next sun facing position. The following snippets of code, which utilise the NumPy library, include the

`readDayData()` function that calculates the specific time durations from the imported sun azimuth tables. The durations are stored in the `motor_time[]` array.

Once the algorithm reaches the end of the imported file, the RST rotates CCW back to the home position so it is facing east the next day. For our data logging function, we used some of the code from the Adafruit ADS1x15 class combined with the Python threading class. The method itself is written as a thread that is kicked off in the main method. Once executed, the thread will open a new text file titled with the current date. Inside the file, the thread will log and timestamp data from inputs A1 and A2 every 10 seconds (0.1 Hz) until the daily tracking function ends and the thread stops.

```
import time, subprocess, threading, datetime, logging
import RPi.GPIO as GPIO
import numpy as np
from Adafruit_ADS1x15 import ADS1x15

# GPIO setup would go here

#Returns the 3 letter month abbreviation of current month
def getMonth():
  temp = subprocess.Popen(["date"], stdout=subprocess.PIPE)
  date_string = temp.communicate()[0]
  t1, month, t2= date_string.split(' ',2)
  return month

month_str = getMonth()
filepath = '/home/pi/RST/Months/' + month_str + '.txt'
rawData = np.loadtxt(filepath, dtype=(str, float), usecols=(0,2))

motor_const = 0.462 # Units of deg/sec based upon speed of motor

angle_deg = np.zeros((len(rawData),1), dtype=(float))
motor_dir = np.zeros((len(rawData),1), dtype=(int))
motor_time = np.zeros((len(rawData),1), dtype=(int))

def readDayData():
  z = 0
  while(z < len(rawData)):
    loop_time[z,0] = int(convTimeSec(rawData[z,0]))

    if(z == 0):
      if(float(rawData[z,1]) <= 90):
        angle_deg[z,0] = 90 - float(rawData[z,1])
        motor_dir[z,0] = -1
        motor_time[z,0] = angle_deg[z,0]/motor_const
      else:
        angle_deg[z,0] = float(rawData[z,1]) - 90
        motor_dir[z,0] = 1
        motor_time[z,0] = angle_deg[z,0]/motor_const
    else:
      angle_deg[z,0] = float(rawData[z,1]) - float(rawData[z-1,1])
      motor_dir[z,0] = 1
      motor_time[z,0] = angle_deg[z,0]/motor_const
    z += 1
```

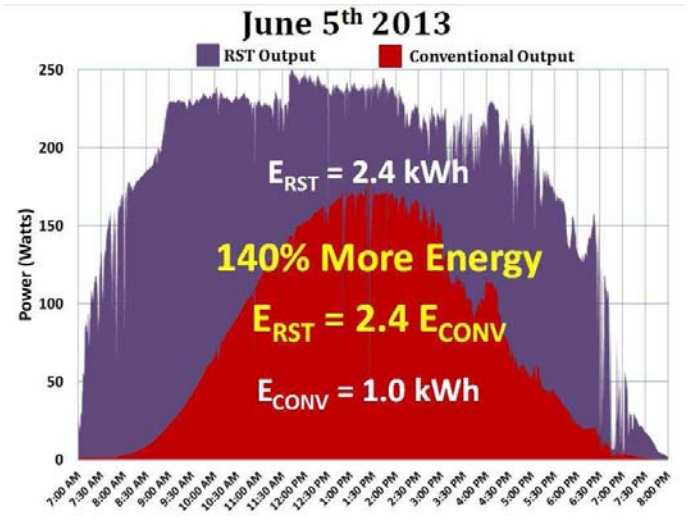
```
#Returns RST to home position
def returnHome():
  if(GPIO.input(homeSensor) == 0):
    logging.info(timestamp() + ': Returning to home position.')
    GPIO.output(hbEnable, 1)
    GPIO.output(motorLED, 1)
    GPIO.output(motorCCW, 1)
    print "Returning home."
    ...

class DataLoggingThread(threading.Thread):
  def __init__(self, threadID, stop_data_log, delay):
    super(DataLoggingThread, self).__init__(self)
    self.threadID = threadID
    self.delay = delay

  def run(self):
    ADS1015 = 0x00
    adc = ADS1x15(ic=ADS1015)
    logging.info(timestamp() + ': Running data logging thread.')
    while not stop_data_log.isSet():
      file = open('/home/pi/RST/Data/' + today + '.txt','a')
      voltage = adc.readADCSingleEnded(1, 4096, 250) / 1000
      file.write(timestamp() + '\t' + str(voltage))
      current = adc.readADCSingleEnded(2, 4096, 250)/ 1000
      file.write('\t' + str(current) + '\n')
      file.close()
      time.sleep(self.delay)
    logging.info(timestamp() + ': Exiting data logging thread.')
```

Reflective solar tracking in the field

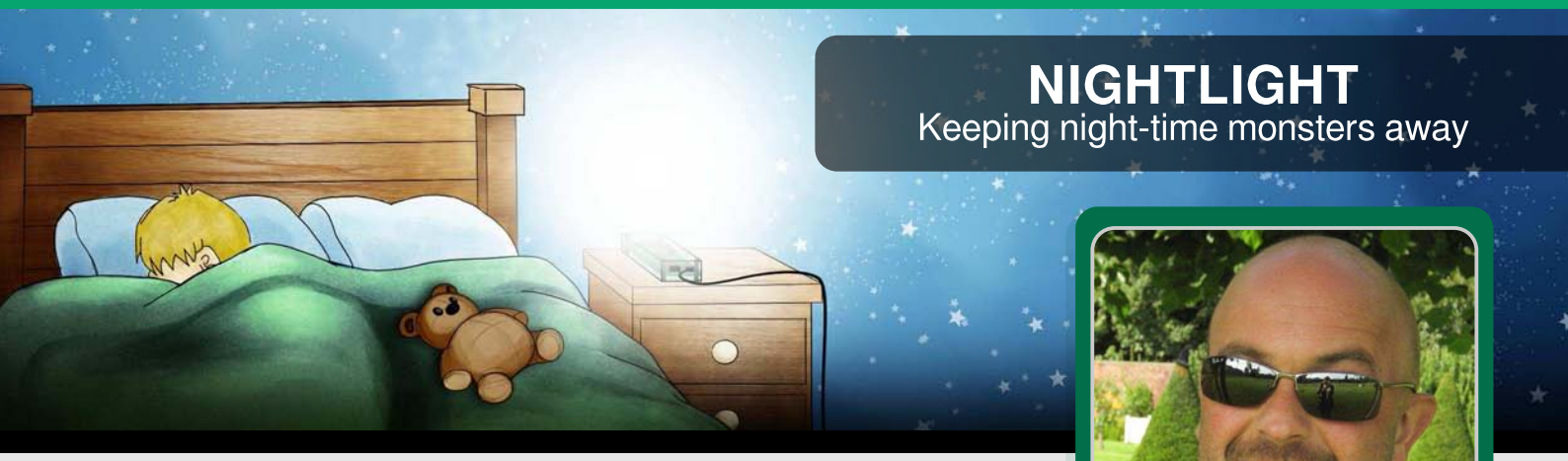
Using the collected voltage and current data from the solar panels, we multiply the values together to get the power output of the panels. Integrating these power values in kilowatts over the total amount of hours of sunlight, gives you the total energy output in kilowatt-hours (kWh) of your installation.

Our initial study compared the RST energy output versus a conventional stationary non-reflective installation. The plot on the right shows the power and energy comparison on a perfectly sunny day.

We recently sent two RSTs to two locations in Uganda where one will be used to power an electric water pump at a rural parish school and the other to St. Joseph's Secondary School as a science experiment where the students will



collect data with the Raspberry Pi as a project and send us the files for our study. We also gave a keynote presentation at the 2013 Energy Path conference about the RST and the Raspberry Pi control system. Anyone with questions regarding the control system or project may contact us via email at nathan.williams2@scranton.edu and nicholas.truncale@scranton.edu.



NIGHTLIGHT
Keeping night-time monsters away

Andy Baker

Guest Writer

PiBow + PiGlow night light

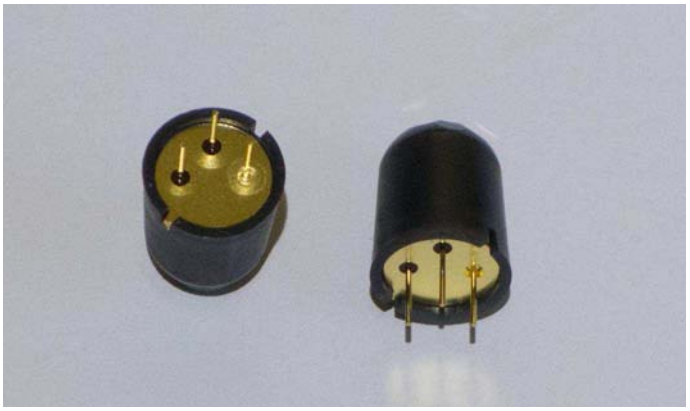
SKILL LEVEL : INTERMEDIATE

A few weeks ago I bought a PiGlow on a whim from Pimoroni, but did nothing with it until a few days ago when my son woke up early in the darkness, having had a nasty dream. That made me wonder whether I could make an intelligent night-light for him, that's dim when all's well, but lights the room in a soothing glow if he wakes disturbed?

Previously another project I'd started was a home alarm system; it reached proof of concept stage but then stalled. It used a Passive Infrared (PIR) motion detector: the perfect detector for under-bed monsters that go bump in the night.

So, down to business: NightLight is powered from a Blackberry wall-wart charger. When the power is switched on, NightLight starts as part of the boot sequence and the LEDs twinkle. When it detects motion the style of LED lighting changes to a random choice of four, lasting for five seconds after motion is no longer detected. My chosen styles of flashing perhaps aren't really suitable for a night light for kids, but they provide good examples with which you or your kids can tinker and make your own.

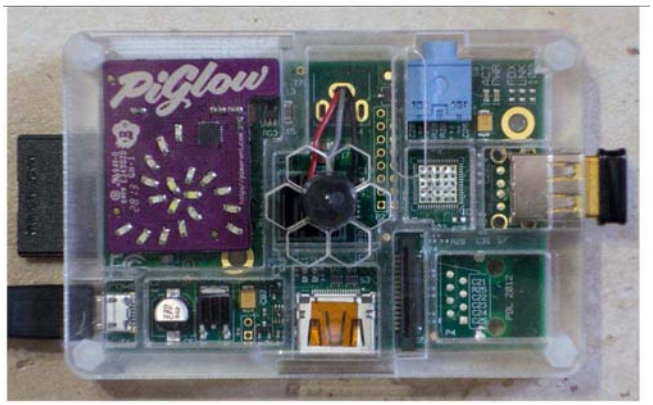
I've used one of my trimmed model A's plus a custom PiBow to house the night light, but that's not essential – a model A or B, with the PiGlow are all that's needed – case selection is up to you.

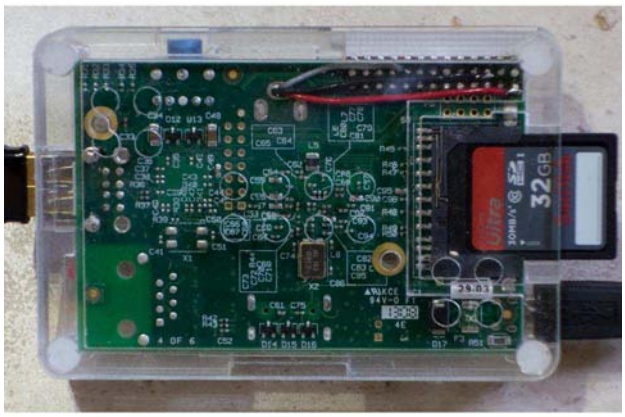


The PIR motion detector is available from Farnell / element14 at http://uk.farnell.com/jsp/search/productdetail.jsp?SKU=1373710.

Attaching it was a bit tricky for me – the PiGlow makes all the GPIO pins inaccessible using a normal IDC connector. The solution depends on your soldering skills. I attached wires to the PIR detector – red (VDD) to pin 1 on the left; grey (OUTPUT) to pin 2 in the center, and black (GND) to pin 3 on the right (see the top image on the next page).

I fed those wires beneath the Raspberry Pi through the holes of the removed AV socket; depending on what case you have, you may be able to find easier holes to use. The red wire is soldered to GPIO pin 1, the black wire is soldered to GPIO pin 6 and the grey wire is soldered to GPIO pin 18 (see the bottom image on the next page).





The movement detector could be replaced by a simple switch if you wish – simply connect one end of the switch to GPIO pin 1 (+3V3) and the other to GPIO pin 18 (GPIO input). Each press of the switch would trigger a new pattern exactly as though the motion had been detected.

The code was a little more complicated than I'd anticipated; the eighteen PiGlow LEDs each have their own number for use by the code, but the numbers do not correspond to any of their colours or positions on the PiGlow board. I ended up writing some mapping code allowing much simpler code to select LED position / colours. That also then allowed the LED pattern code to be cleanly separated and simplified from the rest of the code allowing the various LED patterns to be extended / customized by you and your kids.

The code is described below and is also available on GitHub as:

https://github.com/PiStuffing/NiteLite

```
from __future__ import division
import signal
import time
from smbus import SMBus
import RPi.GPIO as GPIO
import random
import math

# command register addresses for the SN3218 IC
used in PiGlow

CMD_ENABLE_OUTPUT = 0x00
CMD_ENABLE_LEDS = 0x13
CMD_SET_PWM_VALUES = 0x01
CMD_UPDATE = 0x16

class PiGlow:
 i2c_addr = 0x54 # fixed i2c address of SN3218
IC
  bus = None

 def __init__(self, i2c_bus=1):
  self.bus = SMBus(i2c_bus)

# first we tell the SN3218 to enable output
(turn on)

  self.write_i2c(CMD_ENABLE_OUTPUT, 0x01)

# then we ask it to enable each bank of LEDs
(0-5, 6-11, and 12-17)

  self.write_i2c(CMD_ENABLE_LEDS, [0xFF, 0xFF,
0xFF])

 def update_leds(self, values):
   self.write_i2c(CMD_SET_PWM_VALUES, values)
   self.write_i2c(CMD_UPDATE, 0xFF)

 def write_i2c(self, reg_addr, value):

# if a single value is provided then wrap it in
a list so we can treat

   if not isinstance(value, list):
       value = [value];

# write the data to the SN3218


self.bus.write_i2c_block_data(self.i2c_addr,
reg_addr, value)


LED_ARM_TOP = 0
LED_ARM_LEFT = 1
LED_ARM_RIGHT = 2

LED_COLOUR_RED = 0
LED_COLOUR_ORANGE = 1
```

```
LED_COLOUR_YELLOW = 2
LED_COLOUR_GREEN = 3
LED_COLOUR_BLUE = 4
LED_COLOUR_WHITE = 5

LED_PATTERN_TWINKLE = 0
LED_PATTERN_GLOW = 1
LED_PATTERN_SWELL = 2
LED_PATTERN_DROPLET = 3
LED_PATTERN_SNAKE = 4
```

Set up the LED spiral arm / colour mappings:

```
led_map = []
for arm in range(0,3):
 led_map.append([])
  for colour in range(0,6):
   led_map[arm].append(0)

led_map[LED_ARM_TOP][LED_COLOUR_RED] = 6
led_map[LED_ARM_TOP][LED_COLOUR_ORANGE] = 7
led_map[LED_ARM_TOP][LED_COLOUR_YELLOW] = 8
led_map[LED_ARM_TOP][LED_COLOUR_GREEN] = 5
led_map[LED_ARM_TOP][LED_COLOUR_BLUE] = 4
led_map[LED_ARM_TOP][LED_COLOUR_WHITE] = 9
led_map[LED_ARM_LEFT][LED_COLOUR_RED] = 0
led_map[LED_ARM_LEFT][LED_COLOUR_ORANGE] = 1
led_map[LED_ARM_LEFT][LED_COLOUR_YELLOW] = 2
led_map[LED_ARM_LEFT][LED_COLOUR_GREEN] = 3
led_map[LED_ARM_LEFT][LED_COLOUR_BLUE] = 14
led_map[LED_ARM_LEFT][LED_COLOUR_WHITE] = 12
led_map[LED_ARM_RIGHT][LED_COLOUR_RED] = 17
led_map[LED_ARM_RIGHT][LED_COLOUR_ORANGE] = 16
led_map[LED_ARM_RIGHT][LED_COLOUR_YELLOW] = 15
led_map[LED_ARM_RIGHT][LED_COLOUR_GREEN] = 13
led_map[LED_ARM_RIGHT][LED_COLOUR_BLUE] = 11
led_map[LED_ARM_RIGHT][LED_COLOUR_WHITE] = 10
```

Set up the LED number array:

```
leds = []
for led in range(0, 18):
 leds.append(0)
```

Set up the LED brightness array:

```
levels = [0, 1, 2, 4, 8, 16, 32, 64, 128]
```

Set up the shutdown handler:

```
def ShutdownHandler(signal, frame):
 global keep_looping
  keep_looping = False
```

Set up the PIR movement detection callback:

```
def PIRCallback(channel):
 global motion_detected_time
 global led_pattern
 global leds

 if led_pattern == LED_PATTERN_TWINKLE:
  led_pattern = random.randint(1, 4)
  motion_detected_time = time.time()
```

Set up the PIR movement detection:

```
GPIO_PIR = 18
GPIO.setmode(GPIO.BOARD)
GPIO.setup(GPIO_PIR, GPIO.IN, GPIO.PUD_DOWN)
GPIO.add_event_detect(GPIO_PIR, GPIO.RISING,
PIRCallback, 0)
```

Final steps of setup:

```
signal.signal(signal.SIGINT, ShutdownHandler)
piglow = PiGlow(1)
keep_looping = True
motion_detected_time = time.time() - 5.1

while keep_looping:
```

Drop back to the default LED pattern:

```
 if time.time() - motion_detected_time >= 5.0:

  led_pattern = LED_PATTERN_TWINKLE
  twinkle_count = 0
```

TWINKLE: 0: Random LED lit with random, decaying brightness:

```
 if led_pattern == LED_PATTERN_TWINKLE:

# dim all lit LEDs by one step in the levels
list
# first find the index into the brightness list
# This relies on the fact that values in the
levels
# list are all 2^n

  for led in range(0, 18):
   mant, level = math.frexp(leds[led])
   if mant == 0.0:
     level = 0
   if level > 0
     leds[led] = levels[level - 1]
```

```
# Add a random LED every 10 cycles with random
brightness

   if twinkle_count == 0:
     leds[random.randint(0, 17)] =
levels[random.randint(0, 8)]
     twinkle_count = (twinkle_count + 1) % 10

     piglow.update_leds(leds)
     time.sleep(0.1)
```

GLOW: 1; All LEDs glow at a low level:

```
   elif led_pattern == LED_PATTERN_GLOW:
    for led in range(0, 18):
     leds[led] = levels[4]
    piglow.update_leds(leds)
```

SWELL: 2; All LEDs brightness swelling up and down:

```
   elif led_pattern == LED_PATTERN_SWELL:
    for level in range(0, 8):
     for led in range(0, 18):
      leds[led] = levels[level]
     piglow.update_leds(leds)
     time.sleep(0.1)

    for level in range(8, 0, -1)
     for led in range(0, 18):
      leds[led] = levels[level]
     piglow.update_leds(leds)
     time.sleep(0.1)
```

DROPLET 3; Same colour sweeping up and down all the arms together at fixed brightness:

```
   elif led_pattern == LED_PATTERN_DROPLET:

     for colour in range(0, 5):
      for arm in range(0, 3):
       leds[led_map[arm][colour]] = 0x80

      piglow.update_leds(leds)
      for arm in range(0,3):
        leds[led_map[arm][colour]] = 0x00

      time.sleep(0.1)
      for colour in range(5, 0, -1):
       for arm in range(0,3):
        leds[led_map[arm][colour]] = 0x80

       piglow.update_leds(leds)
       for arm in range(0,3):
        leds[led_map[arm][colour]] = 0x00

       time.sleep(0.1)
```

SNAKE: 4; Light each arm sequentially, with LED brightness brighter at center:

```
elif led_pattern == LED_PATTERN_SNAKE:
 for arm in range(0, 3):
  for colour in range (0, 5):
   leds[led_map[arm][colour]] = levels[colour +
1]
  piglow.update_leds(leds)
  time.sleep(0.1)


 for colour in range (5, 0, -1):
   leds[led_map[arm][colour]] = levels[colour +
1]

 piglow.update_leds(leds)
 time.sleep(0.1)
 for colour in range(0, 6):
   leds[led_map[arm][colour]] = 0x00

# set all the LEDs to "off" when Ctrl+C is
pressed before exiting

for led in range(0, 18):
 leds[led] = 0x0
piglow.update_leds(leds)
```

For use as a night light, the last step is to run the code as part of the Raspberry Pi boot sequence so that it runs as soon as it's turned on at the wall. This consists of a bash script. This script needs installing by saving it in `/etc/init.d` and typing:

```
cd /etc/init.d
sudo chmod 755 nitelited.sh
sudo update-rc.d nitelited.sh defaults
```

In addition to boot-time startup, you can also manually start and stop the night light by typing:

```
sudo /etc/init.d/nitelited.sh start | stop
```

Finally, there's a video of it in operation on my site: http://blog.pistuffing.co.uk/?p=1748. It didn't take long to make this fun little project. It is relatively easy to tinker with and exploring the different light patterns and how they work is a great way to learn Python.

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

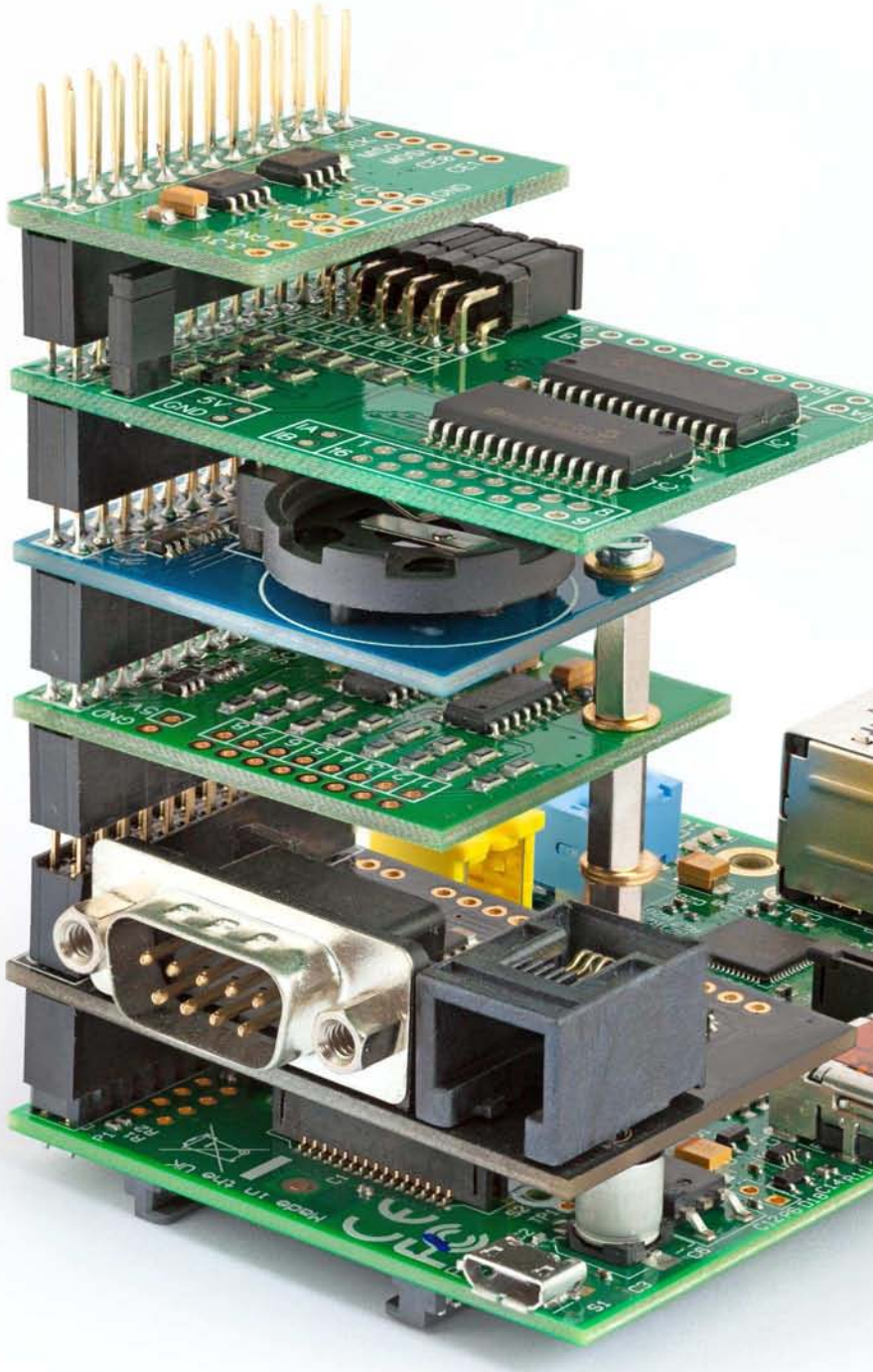
Real-time clock with battery backup and 5V I²C level converter for adding external 5V I²C devices to your Raspberry Pi.

ADC Pi

8 channel analogue to digital converter. I²C 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

Raspberry Pi® Starter Kits

Everything but the screen

http://shop.pimoroni.com

Starter Kit £75
Deluxe Starter Kit £120

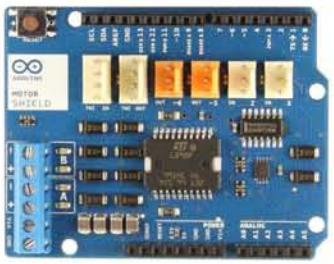
Pimoroni Gift Cards

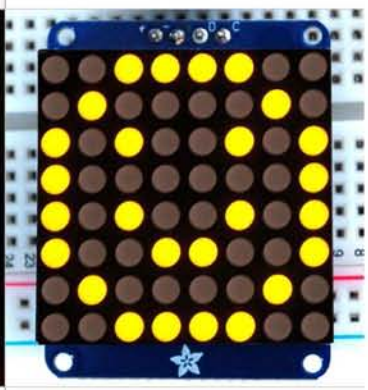
Choosing shiny things is hard!

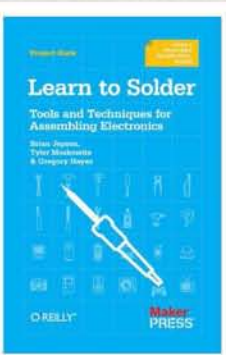
10 £25 £50

http://shop.pimoroni.com

PIMORONI





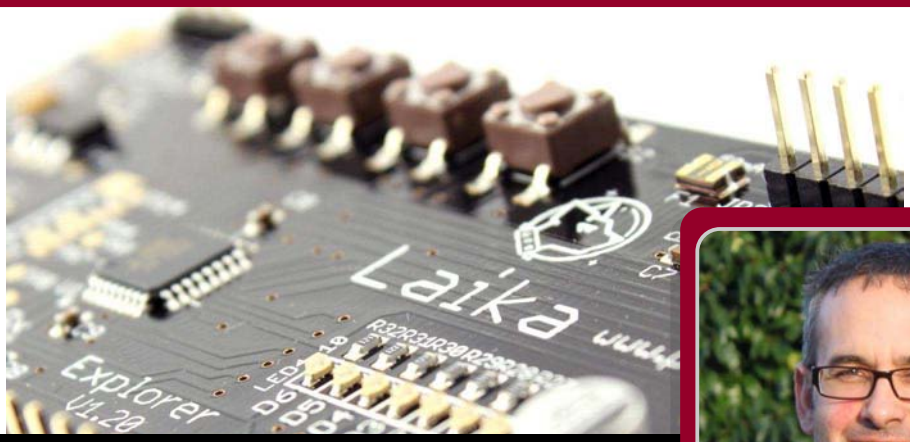


Wearables! Arduinos! Components! Dead Trees!

We now stock a range of shiny things you can make cool stuff with!

http://shop.pimoroni.com

PIMORONI



Introducing Laika Explorer and digital output electronics - Part 1

Andy Bakin

Guest Writer

SKILL LEVEL : BEGINNER

Introduction

The Laika Explorer is an I/O (Input/Output) device for your Raspberry Pi. It has seven digital outputs, four digital inputs, two analogue inputs and a dual 1.3Amp motor driver with PWM speed control. The PCB has four on-board switches and seven LEDs and can be controlled with Scratch, Python or C.

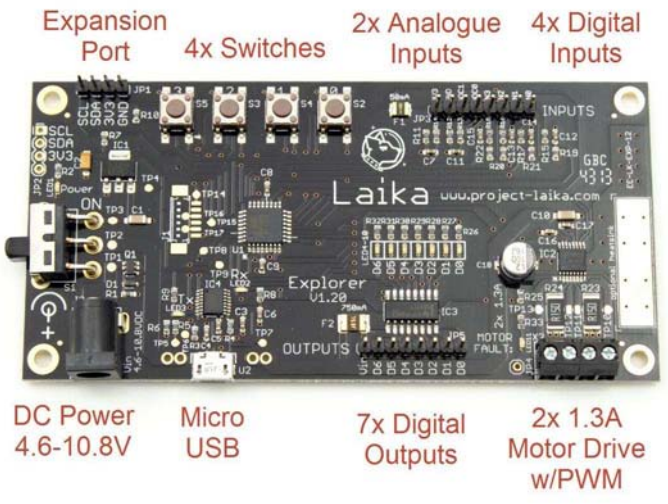


Figure 1.0 - The Laika Explorer V1.20

Okay, so another I/O device for the Raspberry Pi, albeit a very pretty one. Well, yes and no. The Laika system has two tricks up its sleeve: USB and an expansion port. Let us start with the USB. This simply means that it connects to your Raspberry Pi (or Linux computing device) using a USB cable and so frees up the GPIO port for any other devices you may already own. Secondly, the expansion port means that the core functionality can be expanded which is why Laika is a system - the Explorer is the first instalment of this system. Future plug-in modules will include high-current motor drives, I/O expansion, GPS, Bluetooth, Wi-fi, battery chargers, power supplies and many more. There is room for up to 32 modules, all of which can be controlled through the single USB connection.

After a successful Kickstarter campaign which reached its goal back in August 2013, the Laika Explorer has been put through production and is available now for £34.99 (~US$47), or as an Inventor's Kit for £74.99 (~US$102), plus postage. This project is more than a PCB design though. Its goal is to embellish the hardware with tutorials, guides, lesson plans, schemes of work and example applications. There is much to do but you can see the current state of this work on the project's website at: http://www.project-laika.com.

Start exploring

The idea behind Laika is to promote users getting hands-on with the hardware and electronics. This can be daunting at first with no background knowledge, but I am going to show you how easy this can be. If any of the terms in the opening paragraph confused you then don't worry: after following these tutorials you will have a good enough understanding of the technology to start thinking about designing your own wonderful projects.

Inputs and outputs

Hold this thought in your head: all a computer is, in very simple terms, is a device that takes a set of inputs, does something with them, and outputs something useful. What the computer does is controlled by a set of instructions – a computer program. Now think of Laika as a way of providing an easier way for you to interface your computer program to those inputs and outputs, or to put it another way, to interface to the real world. You are the most important part in this system because it is you who writes the instructions to control the I/O - to control the world!

Let's get digital

Digital means that data is represented in discrete values and for computers this is usually just two values. You may recognise this as binary (systems do exist with more than two states: trinary for example has three states) and you may think of these two states as 'on' and 'off', 'high' and 'low' or 'one' and 'zero': all are correct but remember that ones and zeros is how data is stored. For example, 'on' could be a state represented by a '1' as you might think, but could equally be represented by '0' if the hardware turns 'on' when it receives a '0'.

So when you are controlling the digital outputs you have two choices: 1 or 0. On the Explorer there are seven digital outputs. You might think of these outputs as being able to drive high and low, that is, when you turn off an output it goes to 0V and when you turn on an output it goes to some higher voltage. The former is true, but in the case of the Explorer, the latter is not.

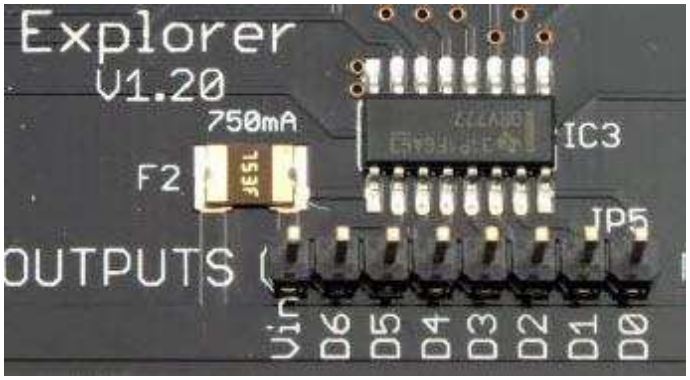


Figure 2.0 - Digital outputs on the Explorer

The reason for this is that we want to able to do something useful with our outputs and this usually means consuming a certain amount of current. Say for instance, you want to drive a relay which may take 50mA, then a typical microcontroller output is not capable of this and trying to accomplish it will likely damage the device. That's why you need to heed all the warnings about protecting the GPIO port on your Raspberry Pi from over-current. Most add-on GPIO modules promote the fact that they use buffering to prevent this damage.

To give us the extra power capability, a buffer is also used: you can see it in the picture above labelled as IC3. This device gives us over 100mA per channel which is plenty to drive most relays and even motors. There is a drawback: each digital output is only able to sink current. That means that when it is turned on, the output goes to 0V (a nice example of an 'on' being a 'low', not 'high'). This means that your device, LED, relay or motor, will always have positive + connected, and the negative is what is switched. See Figure 3.0 below.

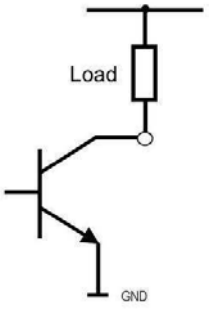


Figure 3.0 - Transistor switch

The symbol with an arrow is a transistor and you can think of it as a switch, switching the 0V (or ground/GND) to the load. The small circle is effectively the output pin which makes you see that it is not possible to drive the output high: only 0V and open/disconnected. It is possible to design circuitry to drive high as well as low but these come with cost implications.

Now that you have a better understanding of this port it is time to do something useful with it.

Scratch to digital outputs

Each output pin is labelled D0 to D6. These are the seven digital outputs and you can control these individually or by writing a single byte to update the whole port in one go. Each output pin has associated with it an on-board LED which means that you can test your output control without any other hardware.

In Scratch you can control the digital output port using binary, integer or hexadecimal. We will use binary as that is the raw form and is the right place to start if you are learning about computing.

To set the digital outputs in binary do the following:

Click on 'Variable' in the top left corner of Scratch.
Select 'Make a variable'.
Make the variable name `lk_exp_dout_bin` (this must be an exact match).

You should now see that Scratch has automatically produced the following commands:

Figure 4.0 – Setting up the digital output variable in Scratch

Move the 'set' command into the script area and set the 'to' value to '0000111'.

Figure 5.0 – Writing to the digital outputs in Scratch

Click the Green flag and voilà, the LEDs indicate the binary value.

Figure 6.0 - LED's

Here is an example of Laika controlling a 7-segment display with the digital outputs. See http://www.project-laika.com for a tutorial.

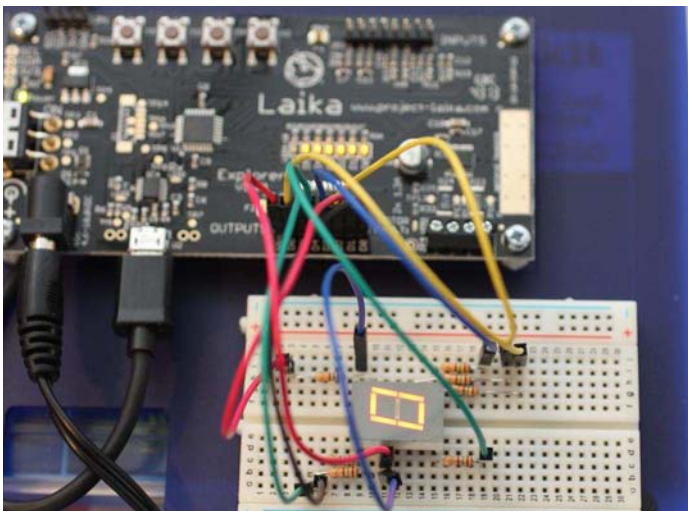


Figure 7.0 – 7-segment control

You may have noticed in Figure 2.0 that the digital output pins are accompanied by another pin labelled as 'Vin'. This stands for Voltage In and is connected to the input power of the Explorer board. So if you are using a 5V power adaptor to power your Explorer then Vin will also

be 5V. The Explorer is specified to operate between 4.6V and 10.8V, so you need to bear this in mind when connecting your load to the output pins.

EMF

No not the 1990's band with a hit song Unbelievable, but rather Electromotive Force – much more interesting I am sure you'll agree. You don't need to fully understand EMF, unless you have a physics exam coming up, but you should be aware of it. Remember this: pass an electric current through a piece of wire and you'll produce a magnetic field around it. When you remove the power that magnetic field collapses and causes some portion of energy to return to the source- this is known as back-EMF or counter-EMF, and can damage sensitive electronic components.

If all you are driving is LEDs then you can forget about back-EMF as there aren't enough coils of wire to produce enough energy to cause any damage- it is still there but is negligible. If, on the other hand, you are driving a motor or a relay which are effectively inductors made from coils of fine wire, then there will be a significant amount of back-EMF. So the amount of back-EMF is affected by the length of wire used? Yes! That is why you should always keep any wires carrying significant power as short as possible. Don't bundle all wires together, try not to coil them up too much as this may cause cross-talk where one wire induces interference on another and keep your signal wires away from your power wires and electrically noisy devices like motors. This is a very deep subject but as long as you have consideration you should be fine.

To protect the Explorer outputs from damage by back-EMF, each output is protected by a diode which allows the nasty EMF current to be dissipated through the diode and away from our delicate transistors. To take advantage of this feature you need to connect all loads that are controlled by the digital output pins to the Vin pin as this completes the circuit. You may never consider using anything other than Vin as your power source for the digital outputs, but you could run some LEDs from another power source if you wanted. You could run a motor or inductor too but you would have to provide your own diode protection in this case. Figure 8.0 shows an example of how this diode is wired.

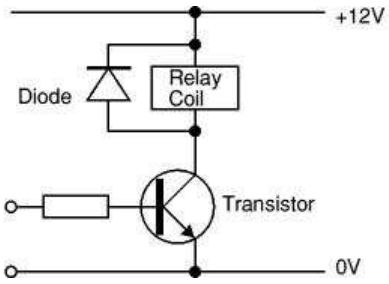


Figure 8.0 - Wiring the diode

Next time

In this article we have covered the digital outputs which means you can now drive relays, motors, LEDs, solenoids or whatever can be driven within the parameters. Next time we will look at the inputs, both analogue and digital. This leads us to creating a closed-loop system where the output is automatically controlled by values sensed on the inputs, thus opening up all sorts of possibilities.

In the meantime please see the website for more information:

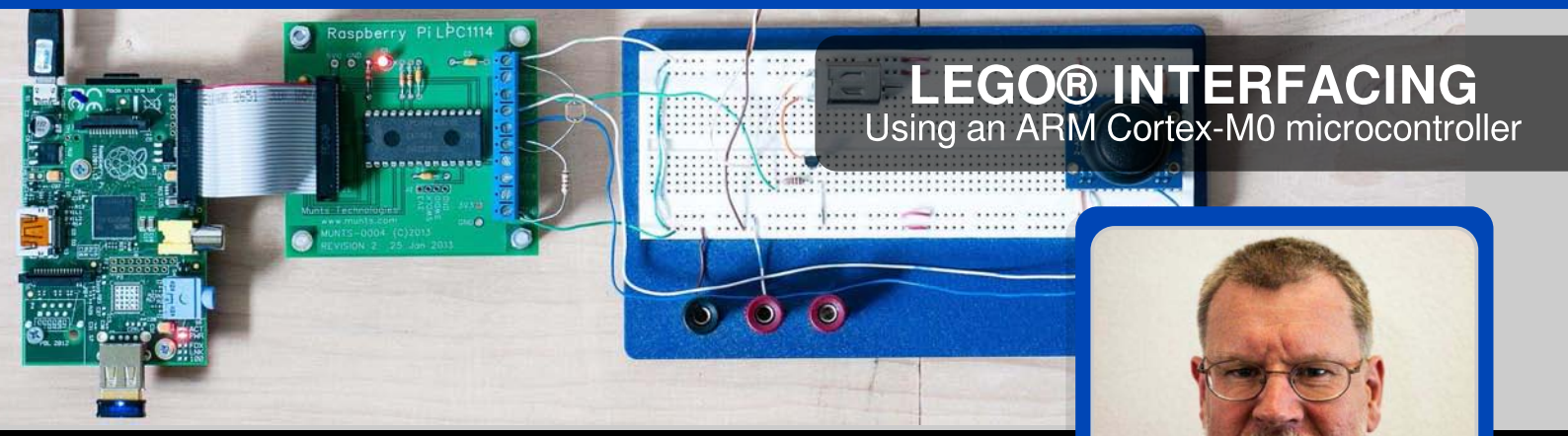
http://www.project-laika.com - main site

http://www.project-laika.com/about-laika
- introduction to the Laika platform

http://www.project-laika.com/tutorials-index
- a list of various tutorials offered for Laika

http://www.project-laika.com/Tutorials/laika-installation - installation instructions

Part 1:	Introducing Laika Explorer and digital output electronics
Part 2:	Analogue and digital inputs
Part 3:	Motor driving with PWM
Part 4:	Applications- bringing it all together



LEGO® Power Functions remote control

SKILL LEVEL : ADVANCED

Philip Munts

Guest Writer

Introduction

In The MagPi Issue #17, I introduced how to use the LPC1114 analogue inputs for sensing the position of an analogue joystick. In this article I will show how to control the LEGO® Power Functions motor with the LPC1114 I/O Processor and how to apply the joystick for controlling a LEGO® RPV (Remotely Piloted Vehicle).

LEGO® Power Functions remote control

LEGO® Power Functions is a system of motors and sensors for the LEGO® system. They can be used with either the traditional LEGO® blocks, or with the newer Technics mechanical components. The motors and sensors can be purchased separately or as part of a set, such as the LEGO® City train set #7938.

The Remote Control IR Receiver #8884 decodes infrared remote control messages in a special coding and sets the speed or position of up to two motors. The receiver is designed to work with the Remote Control #8879. We will be synthesising the infrared messages with firmware in the LPC1114 I/O Processor to control the IR receiver from the Raspberry Pi instead.

Figure 1 – Some LEGO® Power Functions components left to right: Motors, IR Receiver and Battery Box.

Infrared protocol

The IR receiver expects a fairly complex protocol. Fortunately LEGO® has published the protocol specification, which can be downloaded from the web page for the #8884 IR receiver. I have added a service to the LPC1114 SPI Agent Firmware that encodes messages for the IR receiver.

Setting up to generate LEGO® Power Functions RC messages is easy. First, configure one of the LPC1114 GPIO pins as an output, with the

```
SPIAGENT_CMD_CONFIGURE_GPIO_OUTPUT
```

service. Be sure to initialize the output state to OFF by setting the data field to zero.

Next, generate a message with the

```
SPIAGENT_CMD_PUT_LEGORC
```

service. The data field must be loaded with parameters encoded as follows:

```
Bits 0-7    Speed:     0-7
Bits 8-15   Direction: 0=Reverse 1=Forward
Bits 16-23  Motor:     0=All stop, 1=Motor A,
                       2=Motor B
Bits 24-31  Channel:   1-4
```

See the LPC1114 I/O Processor Expansion Board User Guide (http://munts.com/rpi-lpc1114/doc/UserGuide.pdf) for more information about the SPI Agent Firmware API (Application Programming Interface). The available services are described in detail there.

Infrared emitter circuit

To transmit commands to the IR receiver, you will need to drive an infrared LED (more properly: infrared emitter) from one of the LPC1114 GPIO pins. The GPIO pins can only supply 4 mA of current, which is too little to drive an infrared emitter efficiently. Infrared emitters can commonly be driven at 100 mA or more. In order to boost the output current we will use the ULN2003A, a current driver IC that has been around for many years but is still highly useful for this purpose.

The ULN2003A contains 7 amplifiers, or current boosters, each of which can sink 500 mA continuously, or even more for short pulses. Since the ULN2003A is a current sink (pull to ground) device, the cathode (minus) pin of the infrared emitter is connected to the ULN2003A output through a current limiting resistor, and the anode (plus) pin is connected to +5V. (Do not attempt to connect the infrared emitter to +3.3V; the voltage regulator on the Raspberry Pi cannot supply enough current.) See Figure 2 for the infrared emitter circuit.

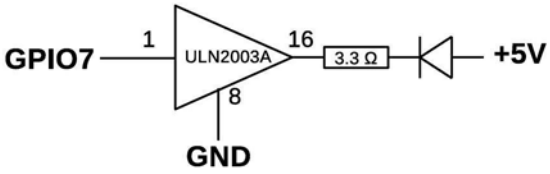


Figure 2 – Infrared emitter connection

A wide variety of infrared emitters are available, at either 850 or 940 nm wavelengths. Both wavelengths seem to work with the #8884 IR Receiver, but I infer from the colour of the filter that it may be optimized for 940 nm. Figure 3 shows a couple of emitters I tried.

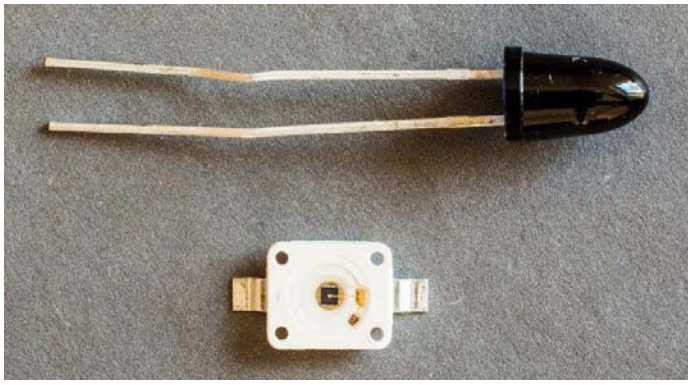


Figure 3 – Infrared Emitters – Top: SFH4545, Bottom: SFH4233

The SFH4545 (Digi-Key part number 475-2919-ND) is a 940 nm emitter rated at 100 mA current (pulse), and is representative of what you will find at Radio Shack or other hobbyist sources. This class of device should have a 33Ω current limiting resistor.

The SFH4233 (Digi-Key part number 475-2910-1-ND) is a very high power 940 nm emitter rated at 1 A current (pulse). This device should have a 3.3Ω current limiting resistor.

Warning: The combination of the ULN2003A, SFH4233 and 3.3Ω resistor are capable of generating serious heat if the GPIO pin is left in the ON state. They are capable of melting a solderless breadboard! (Examine the top of Figure 5 closely for an example!)

Python3 test program

Once the hardware is put together, you can test your setup with the Python3 test program `test_legorc.py`. It accepts an optional host name parameter and then 4 numbers (channel, motor, direction and speed). If the motor, direction and speed are all zero, a panic stop is issued (both motors are stopped immediately).

Show program usage

```
./test_legorc.py

Raspberry Pi LPC1114 I/O Processor Expansion
Board LEGO Power Functions RC Test

Usage: ./test_legorc.py [hostname] <channel:1-
4> <motor:0-2> <forward:0-1> <speed:0-7>
```

Channel 1, motor 1, forward, speed 1

```
./test_legorc.py localhost 1 1 1 1
```

Panic Stop

```
./test_legorc.py localhost 1 0 0 0
```

Python3 remotely piloted vehicle

For experimentation my son and I constructed a skid steer remote vehicle and wrote a Python program to control it using the analogue joystick. A skid steer vehicle has independent motors driving wheels or tracks on either side of the vehicle. Steering is accomplished by driving each side at a different speed and/or direction, as shown in Figure 4.

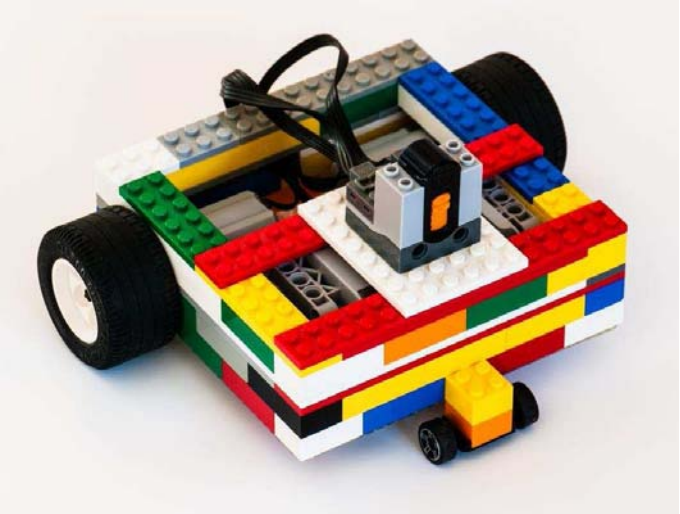


Figure 4 – Python3 remotely piloted vehicle

The Python3 RPV program `test_legorc_rpv.py` uses the analog joystick setup from the previous installment in The MagPi #17. The ULN2003A and the SFH4233 infrared emitter have been added to the breadboard, as illustrated in Figure 5.

Unlike `test_legorc.py`, which calls C functions in `libspiagent.so`, `test_legorc_rpv.py` uses XML-RPC. This simplifies the Python code somewhat.

Implementing skid steering in software from a single joystick proved tricky. A certain amount of heuristics (fudge factors) were required.

Useable range for the IR remote control system will vary depending upon the ambient environment and the placement of the infrared

emitter. In my office, with a low white ceiling and using the high power SFH4233, range and coverage are very good as the IR reflects well off the ceiling. At the Helsingborg Raspberry Jam, in an office building foyer with very high ceilings that were dull gray, range was only a few meters with reflected signal. In such circumstances, placing the emitter for line of sight communications instead of relying on reflections would help.

Links

LPC1114 I/O Processor Expansion Board support info, including how to buy one:
http://munts.com/rpi-lpc1114
LEGO® Power Functions home:
http://powerfunctions.lego.com
ULN2003 datasheet:
http://www.ti.com/lit/ds/symlink/uln2003a.pdf
Parallax 2-axis joystick information:
http://learn.parallax.com/KickStart/27800

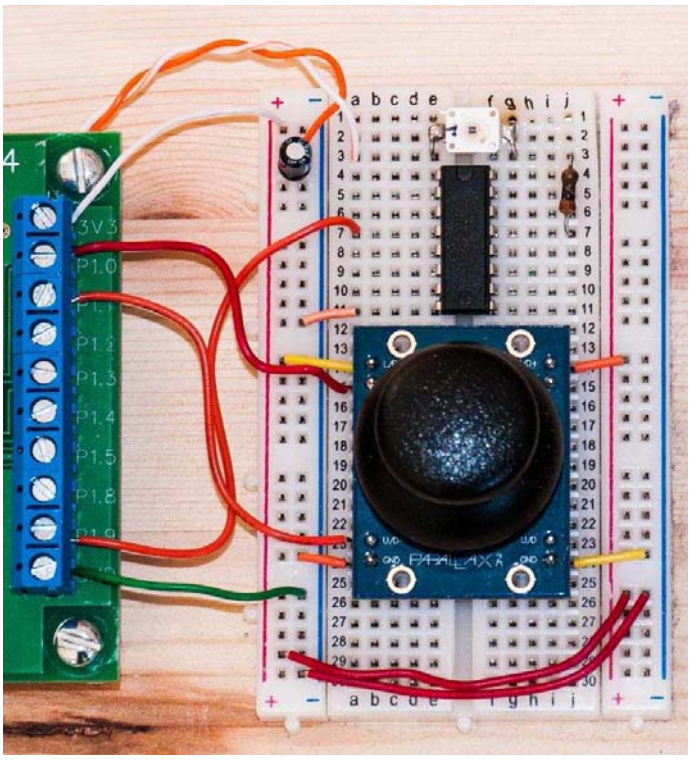
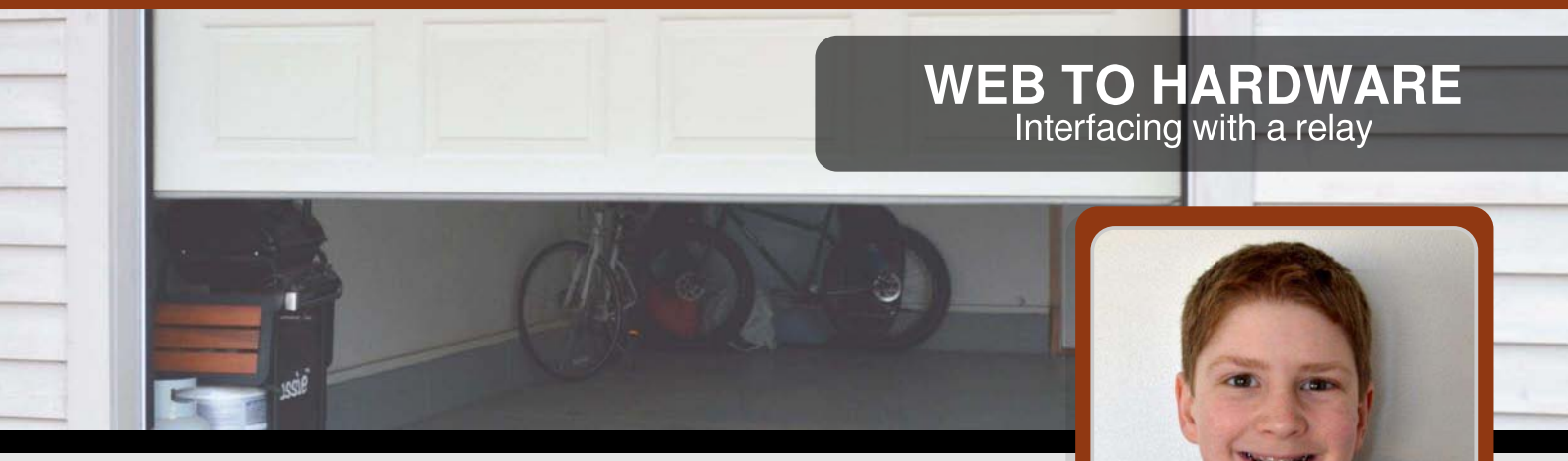


Figure 5 – Joystick and IR Emitter Hookup

CYNTECH COMPONENTS

Home of the PiHub, Pibrella, HDMIPi, Pi Case and a huge range of other Pi related accessories.

www.shop.cyntech.co.uk | sales@cyntech.co.uk



Garage door automation with WebIOPi

Lewis Callaway

Guest Writer

SKILL LEVEL : BEGINNER

Introduction

Have you ever opened your garage door, but then forgot to shut it later? In this article I will show you how to open and close your garage door over the internet, using a Raspberry Pi, a relay and WebIOPi. There is even a webcam attached to the Raspberry Pi, such that you can visually confirm that the door is really closed.

Before you start

These instructions assume that sshd is running on Raspbian. For older images, you can enable sshd using:

```
sudo raspi-config
```

and then enable SSH via `Advanced Options`.

Parts needed

You will need the following parts:
- Raspberry Pi Model B
- 12V relay
- Adafruit Small-Size Perma-Proto breadboard
- female jumper wires
- regular hookup wire
- 1N4001 diode
- webcam that does not require a powered hub

The tools needed are a soldering iron, screwdriver, wire strippers and wire cutters.

[Ed: Check the specification of your garage door and choose a relay that is appropriate for the voltage and current. It may be necessary to use a transistor/FET driver with the relay.]

Installing WebIOPi

The framework we are going to use to control the relay is WebIOPi. *[Ed: WebIOPi was first introduced in issues 9 and 10 of The MagPi.]* To install this package, from the command line enter:

```
wget http://webiopi.googlecode.com/files/WebIO
Pi-0.6.0.tar.gz
tar xvzf WebIOPi-0.6.0.tar.gz
cd WebIOPi-0.6.0
sudo ./setup.sh
```

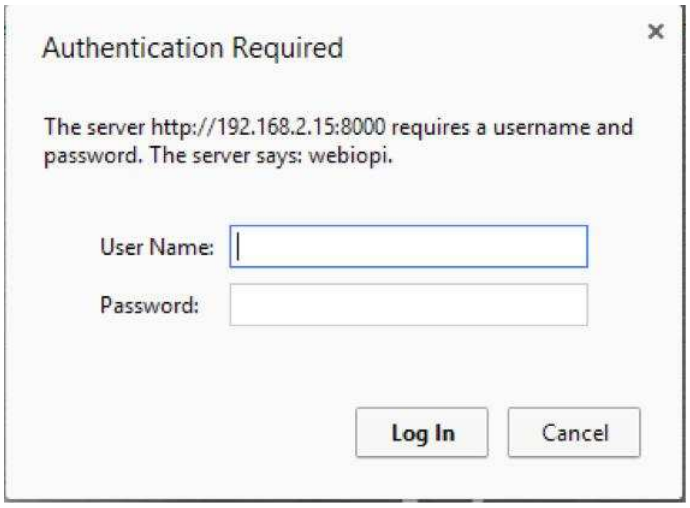
Then configure WebIOPi to start when the Raspberry Pi boots, and then reboot:

```
cd
sudo update-rc.d webiopi defaults
sudo reboot
```

Wait for a minute after rebooting the Raspberry Pi. Then open a web browser and enter the IP

address of the Raspberry Pi followed by the port number of WebIOPi which is 8000, e.g. http://192.168.2.15:8000.

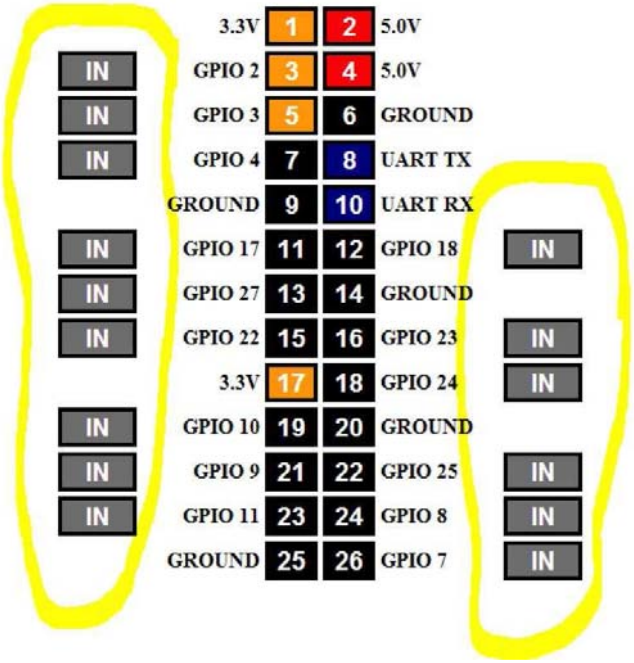
When connecting to the Raspberry Pi using the web browser, a login screen will appear.



The default login is:

```
User Name: webiopi
Password: raspberry
```

The next screen shows a virtual representation of the Raspberry Pi's GPIO pins. The relay can be connected between a ground pin and a GPIO pin. To activate the relay, click on the corresponding button to set it as an OUTput instead of an INput. Now when the GPIO pin is selected, it will send a voltage to the relay connected to it.



Configuration

To make the interface cleaner, WebIOPi can be configured to offer one button instead of many small ones.

Create folders to store the files:

```
mkdir -p garage/html
mkdir garage/python
```

The next step is to create a python file that helps the HTML file control the garage door opener.

```
nano garage/python/script.py
```

Now copy the code below into the new file.

```
import webiopi
GPIO = webiopi.GPIO
Garage = 17 # GPIO pin using BCM numbering
# setup function is automatically called at
# WebIOPi startup
def setup():
  # set the GPIO used by the light to output
  GPIO.setFunction(Garage, GPIO.OUT)

# loop function, repeatedly called by WebIOPi
def loop():
  # gives CPU some time before looping again
  webiopi.sleep(1)
```

Then press CRTL-X and then Y to save the file. WebIOPi will access this python script using an HTML page that contains the single button. Use nano to create a new HTML file,

```
nano garage/html/index.html
```

Then copy the HTML from the next page of this article into the file and save it.

```
<! DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Garage Control</title>
  <script type="text/javascript" src="/webiopi.js"></script>
  <script type="text/javascript">
    webiopi().ready(function() {
      // Create a "Light" labeled button for GPIO 17
      var button = webiopi().createGPIOButton(17, "Garage");

      // Append button to HTML element with ID="controls" using jQuery
      $("#controls").append(button);

      // Refresh GPIO buttons
      // pass true to refresh repeatedly of false to refresh once
      webiopi().refreshGPIO(true);
    });
  </script>
  <style type="text/css">
    button {
      display: block;
      margin: 5px 5px 5px 5px;
      width: 1280px;
      height: 720px;
      font-size: 100pt;
      font-weight: bold;
      color: white;
    }

    #gpio17.LOW {
      background-color: Black;
    }
    #gpio17.HIGH {
      background-color: Yellow;
    }
  </style>
</head>
<body>
  <div id="controls" align="center"></div>
</body>
</html>
```

To use the Python script and new HTML file, modify the WebIOPi configuration file:

```
sudo nano /etc/webiopi/config
```

Find the [SCRIPTS] section and add the following line:

```
garage = /home/pi/garage/python/script.py
```

Then find the [HTTP] line and add the following line:

```
doc-root = /home/pi/garage/html
```

Lastly, find the [REST] line and add the following lines:

```
gpio-export = 17
gpio-post-value = true
gpio-post-function = false
```

Then save the file and reboot the Raspberry Pi:

```
sudo reboot
```

Motion setup

The motion package can be used with a webcam, to check the garage door's status. While this article uses a USB webcam, the Raspberry Pi camera can also be used. There are specific instructions for using the motion package with the Raspberry Pi camera at: http://rbnrpi.wordpress.com/project-list/setting-up-wireless-motion-detect-cam/

To use UVC camera support, make sure the latest stable firmware is installed by updating the Raspbian installation:

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

Then install the motion package:

```
sudo apt-get install motion
```

Next, open the configuration file in nano:

```
sudo nano /etc/motion/motion.conf
```

Change `daemon off`, to `daemon on`. This will cause `motion` to run as a daemon process that runs in the background. The default resolution of 320x240 is a reasonable choice. Change `webcam_localhost` on to `off`. This allows the camera feed to be viewed from another device, instead of just from the Raspberry Pi. Now save the file.

To simply the setup, motion can be configured to start when the Raspberry Pi boots by editing:

```
sudo nano /etc/default/motion
```

In this file change,

```
start_motion_daemon = no
```

to:

```
start_motion_daemon = yes
```

Then save the file and reboot the Raspberry Pi:

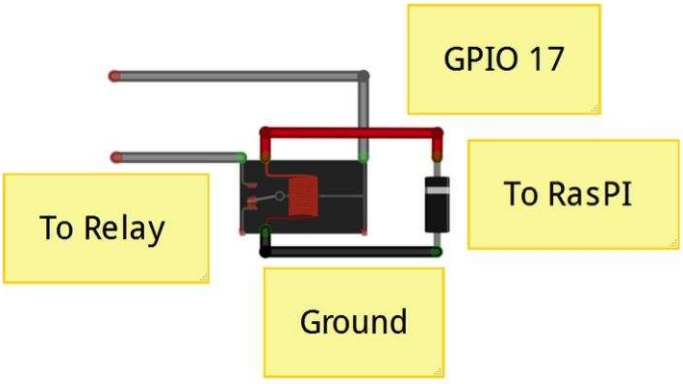
```
sudo reboot
```

Now that `motion` has been configured, plug in a Debian Wheezy compatible webcam. The webcam should then stream video to your Raspberry Pi's IP Address followed by `:8081`.

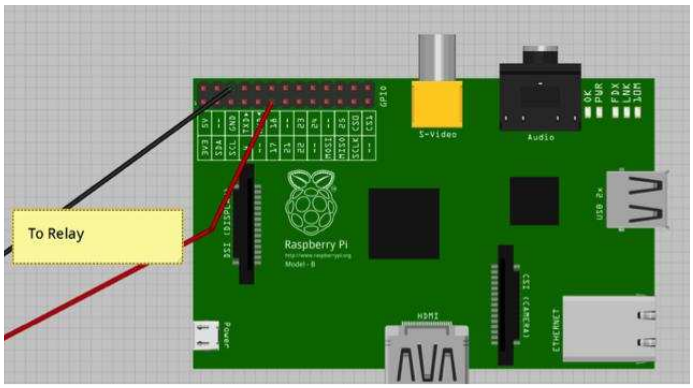
Hardware

The hardware in this project consists of a relay that allows the Raspberry Pi to switch a garage door opener on or off. To hookup the relay to the garage door opener, a Perma-Proto PCB was used. The relay was soldered directly to the Perma-Proto PCB, together with all the wires. To prevent an electrical short the bottom of the PCB was covered with electrical tape after the components were soldered on. A 1N4001 diode was placed across the relay coil pins, to avoid damaging the Raspberry Pi GPIO pin. **This is not optional**. If the diode is not used, it will eventually destroy the Raspberry Pi!

Use the datasheet for your relay to determine the pinout. There is an example below of how a relay can be connected. In the example, the Raspberry Pi connects to the coil with a diode in between the control (red wire) and ground (black wire) pins. The garage door is connected to the normally open pin and relay switch pin (two grey wires).

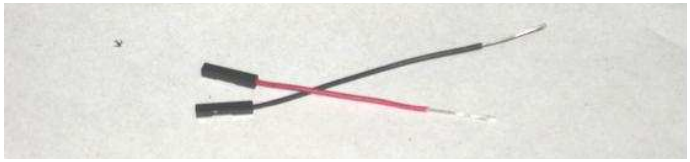


The coil on the relay should be connected to Ground and GPIO 17 on the Raspberry Pi using the female jumper wires.

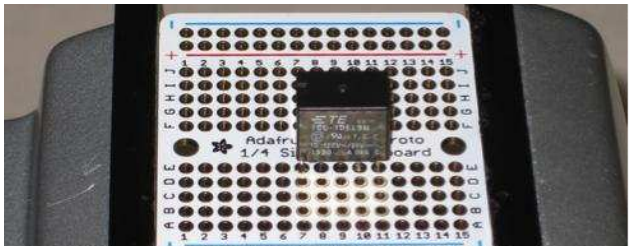


Soldering the Board

1. Cut and strip the two female jumper wires in half.



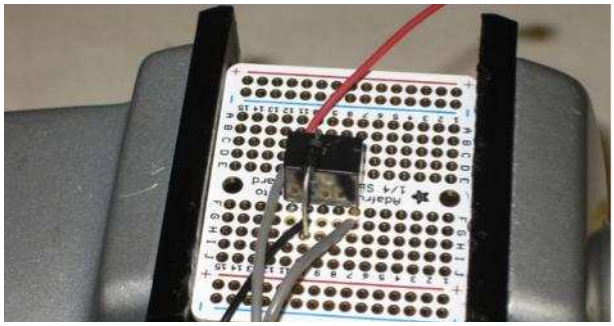
2. Solder the relay to the Perma-Proto board.



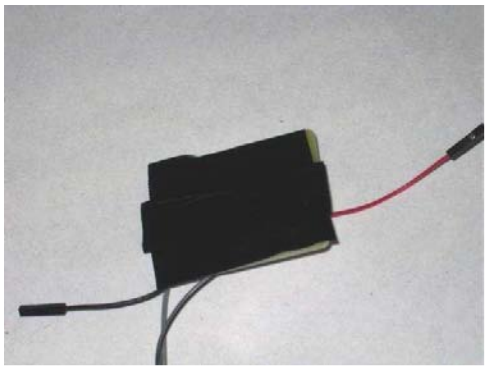
3. Next, solder the black female jumper wire to the coil on the relay and the red jumper wire to the coil on the relay. Then trim the excess wire from the bottom of the board.

4. To prevent the Raspberry Pi from being damaged, a 1N4001 diode is needed between the ground and control pins on the relay. The silver band on the diode needs to be connected to the control wire. Do not connect it to the ground.

5. Solder the wires that will connect to the garage door to the normally open pin and ground on the relay. Again, trim the excess wire after soldering.



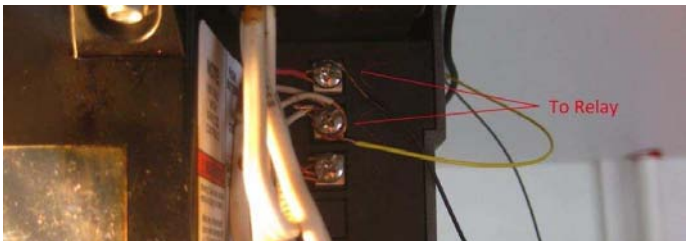
6. Lastly, cover the bottom of the board with electrical tape.



Once this is done, the PCB can be connected to the garage door opener.

Garage door opener connection

Be very careful when dealing with higher voltage circuits, such as a garage door opener. Before making any connections, make sure that the electrical power to the garage door is turned off. Then use a screwdriver to connect the wires from the relay to the garage door opener. If in doubt, consult the garage door opener manual, to check which wires to use.



There is a video of the system working at: https://www.youtube.com/watch?v=3glWNjPUQpI

Information on accessing the Raspberry Pi outside the private local area network is given at: http://www.wikihow.com/Set-Up-Port-Forwarding-on-a-Router

RasPiConnect

Connect your Raspberry Pi to the World!

Allows you to control virtually anything you connect to your Raspberry Pi from your iPad or iPhone.

- EASY to setup - no syncing required
- Buttons, gauges, webpages, webcam pictures and more!
- Exchange your panels with friends
- Supports multiple Raspberry Pis and multiple Arduinos
- Build your pages on your iPad/iPhone
- Ten pages of control panels
- Unlimited Controls
- Supports any computer that supports Python (windows, linux, etc.)
- *Now Allows Custom Backgrounds*

Available on the App Store

MiloCreek
Where Innovation Flows

Supports Arduino with in-app purchase

UPiS module

The **all-in-one** solution

A wealth of features in a single board:

- UPS function with Li-Po battery
- Versatile powering for RPi (7-18VDC)
- Hardware ON/OFF switch for the RPi
- Software-readable V/mA/°C sensors
- Battery backed-up real-time clock
- Timer controlled RPi ON/OFF
- RS232 and USB interfaces
- I^2C PiCO interface
- 1-wire and iButton input
- Basic I/O pin and relay outputs
- XTEA-based encryption of user software

available from distributors :

RASPPISHOP
Pi Supply
MODMYPI
NETTOP
PI-SHOP.CH
www.m.nu MATTEKNIK FÖR ALLA

πmodules
www.pimodules.com

Intelligent modules for your Raspberry Pi

Richard Wenner

Guest Writer

Inserting and viewing stored data

SKILL LEVEL : BEGINNER

In the article entitled DATABASE BOOTCAMP, way back in issue 8, on pages 26 to 28, I introduced you to (what I think) is the language most suitable for learning on computers and one rarely offered, the Stuctured Query Language or SQL.

I showed you how to install MySQL, until now the most popular database available and how to view databases, how to view data, how to create a database and how to create a table.

I promised you that the next time we would be looking at inserting data into a pre-populated table and how to sort and search by using logical expressions. I suggest that you read this previous article once more in order to refresh your memory.

In this issue I'll introduce you to this essential next step. In the earlier article we created a table named birdtable and than used the command `DESCRIBE` birdtable to reveal the description of this table:

Column	Content	Data-type
id	integer	int
first_name	text *(max 25 chars)*	char
town	Text *(max 30 chars)*	char
dob	date	date
birds	integer	int

I'll explain this table in conjunction with the next command, to shows you how to insert data:

```
INSERT INTO
birdtable (first_name,town,dob,birds)
values ("Fred","Epping",19571225,3);
Query OK, 1 row affected (2.53 sec)
```

So let's now examine the `INSERT INTO` command that can be entered on a single line but it has been laid out here for more clarity. It should be clear to see if you execute a `SELECT` that the table called birdtable now has a row (or record) containing:

Fred in first_name
Epping in town
19571225 in dob (date of birth)
ie 25th December 1957 and
3 goes into birds

Note the way data is entered. Text is surrounded by quotation marks, the date is entered in the yyyymmdd format although you could change this but for the moment we will use this format. The numbers (integers) are entered directly as numbers. Each value is separated by a comma. Also note that we put a semicolon (;) at the end of the command. This is what the table was programmed to accept in the

`CREATE TABLE` command in issue 8.

Comparing the SQL command to the list of data entered to the table of columns may leave you confused: what is the "id" column for and why is it not populated? I'm glad you asked!

Suffice it to say that we created a column "id" that will automatically increment each time a new entry is added to the table. This will result in the first row in the table having an id=1 , the second row id=2, the third row id=3, and so on.

The column "id" is not something that we need to worry about after we create the table, as it is all automatically calculated within MySQL.

Try adding a few more lines of data of your own. Top-tip: it is quicker to press the up arrow key in order to recall and edit the previous line and press enter again. This saves typing the whole line, particularly as the rest of the command remains unchanged.

Also note that MySQL confirms each entry with a line about the number of rows affected.

Confirm your table contains the values you have added by entering:

```
SELECT * FROM birdtable;
```

Two useful database functions are **sorting and searching**. Let us try some examples.

Viewing the data in a sorted order by entering:

```
SELECT * FROM birdtable
ORDER BY first_name;
```

To see everything sorted by first_name:

```
SELECT * FROM birdtable
ORDER BY birds DESC;
```

To see the same list but this time by the descending number of birds.

Viewing the data - searching

In the `SELECT FROM` command we used the Asterisk (*) as a wild card meaning all records or a substitute for zero or more characters in a record. There are two other wild cards that we can use: the percent (%) symbol and the underscore (_). Where you can only use the asterisk as a wildcard (or truncation) in a full text search, the % (match zero or more characters) and _ (match exactly one character) are only applicable in LIKE-queries.

The `LIKE` keyword tells MySQL that the named column must match the given pattern. In a database with a long text string for instance, we could write:

```
SELECT text_string FROM database_name
WHERE text_string LIKE "%Raspberry Pi%";
```

We will touch on the `LIKE` command a little bit later.

We will try these wildcards in the database by entering:

```
SELECT * FROM birdtable
WHERE (first_name = "Fred");
```

To see everything where the name is Fred.

```
SELECT * FROM birdtable
WHERE (first_name LIKE "Fr%");

SELECT * FROM birdtable
WHERE (first_name LIKE "Fr_d");
```

To catch many options (if they existed): Frad Frbd Frcd Frdd Fred Frfd Frzd etc. The underscore limits your search to just that character while the percentage replaces any number of characters.

```
SELECT * FROM birdtable
WHERE (first_name like "%r%");
```

All first names with an r in them.

```
SELECT * FROM birdtable
WHERE (birds >3);
```

All records with more than three birds will show.

Logical words in SQL commands

Two logical words that can be included in SQL commands are `AND` and `OR`. `AND` means that both conditions of a statement must be met and `OR` insists that either one or the other condition must be met.

Try commands like:

```
SELECT * FROM birdtable
WHERE (first_name like "%r%")
  AND birds >3;
```

To find names with the letter r in them and birds greater than 3.

`OR` means either condition needs to apply. Try commands like:

```
SELECT * FROM birdtable
WHERE (first_name = "Fred") OR birds < 3;
```

To find Freds or entries with birds less than 3.

All this is nice but how about being more selective in choosing our parameters?

In every example above we have printed out all of the fields for each selected record. We can be more selective by typing:

```
SELECT town, dob from birdtable
WHERE (first_name like "%r%")
  AND bird > 3 ORDER BY bird;
```

Notice here how the `ORDER BY` option can be added to build even smarter SQL commands and to retrieve the data in the order we want.

Conclusion

There is a lot to be gained by testing these commands on your Raspberry Pi. Working in text mode as we have been doing in this article is a rapid way of learning, testing, building and repairing databases as well as being a fundamental computing skill.

MySQL has always had a rather steep learning curve when you want to go from straight forward to complex SQL queries, especially when they involve joins.

We are dealing with databases and can not afford to make mistakes. This adds to the pressure and manipulating these databases should therefore always be done on a copy, away from the real thing until we feel secure in doing what we want to do.

There are many good books on the market that deal with SQL databases. One that comes to mind is the excellent work of Kevin Yank. Kevin has published a number of books through the SitePoint publisher. His line of approaching SQL almost makes the experience a gradual stepping up to the rank of intermediate SQL professional.

His book is: PHP & MySQL: Novice to Ninja 5th edition (Build Your Own Database Driven Web Site Using PHP and MySQL).
http://www.amazon.co.uk/Kevin-Yank/e/B0034Q3XD8/

The World Wide Web is also a good source of MySQL teachings and many of these texts will help you in your efforts to conquer the SQL database.

For example, w3schools.com host a PHP and MySQL Introduction:
http://www.w3schools.com/Php/php_mysql_intro.asp

The MySQL Reference Manual contains a tutorial and is available at:
http://dev.mysql.com/doc/refman/5.5/en/tutorial.html

A supporting video by the author is also available online: http://pr0mpt.me

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

Torbay Raspberry Jam

When: Saturday 12th April, from 1.00pm
Where: Paignton Library and Information Centre, Great Western Road, Paignton, TQ4 5AG, UK

The second Torbay Raspberry Jam. Scratch, Python, Minecraft. http://dcglug.drogon.net/future-jams.

Raspberry Pi at CERN

When: Saturday 12th April, 9.30am to 4.30pm (Central European Time: France)
Where: Route de Meyrin, 1211 Geneva, Switzerland

Come and discover in the presence of experts. This workshop is intended for children (from 6yrs), their parents, teachers and all interested members of the public. http://www.eventbrite.fr/e/3914624748

PyCon 2014: Young Coders

When: Saturday 12th - Sunday 13th April, 9.00am to 4.00pm (PDT)
Where: Palais des congrès de Montréal, 201 Viger Ave West, Montréal, Québec, Canada

A free tutorial exploring Python games programming using simple data types, comparisons and loops in Pygame. http://www.eventbrite.com/e/10565294079

Wakefield Acorn & RISC OS Computer Show

When: Saturday 26th April, 10.30am to 4.30pm
Where: The Cedar Court Hotel, Denby Dale Road, Calder Grove, Wakefield, West Yorkshire WF4 3QZ

The North's Premier RISC OS Show. Now in its 19th year. A full day of theatre presentations and exhibitors stands. http://www.wakefieldshow.org.uk/index.php

Raspberry Pi: Kids' "Beta" Tech Camp

When: Sunday 27th April, 2.00pm to 4.00pm (EDT)
Where: Funutation Tekademy LLC, 23715 Mercantile Road, #215 Beachwood, OH 44122

Working in teams of two, campers will use the Raspberry Pi to do computer animations, design games, and make simple window applications. http://www.eventbrite.com/e/9981423707

Programming the Raspberry Pi from a web browser using a visual language

Miruna Tataru & Ioana Culic
Guest Writers

SKILL LEVEL : BEGINNER

Imagine making your own radio or creating disco lights using a Raspberry Pi, which you control from a web browser. Wyliodrin gives you this possibility. You can write, modify and run programs in real time, no matter where in the world your Raspberry Pi is located.

We all face the inconvenience of building our first Raspberry Pi based devices - installing software and repeatedly plugging and unplugging the Raspberry Pi from the device (e.g. a remote controlled car) and connecting it to a screen until it finally does what it is supposed to do. SSH is an option, but it requires firewall configuration for remote access.

With Wyliodrin, you can install your Raspberry Pi directly onto the device you want to build and program it from your web browser (we recommend using Chrome, Firefox or Safari). This way, programming is possible regardless of the Raspberry Pi's location and without the need to connect to it directly. The code is stored on the Wyliodrin servers, so you are now able to use any computer to do the programming.

Wyliodrin comes from the old Gallic words “wylio” and “drin”, which summarises what the platform does: it monitors and handles the treatment/control of the Raspberry Pi.

What languages can you use?

One of our goals with Wyliodrin is to bring students closer to engineering; to building things. With the Raspberry Pi, building electronic gadgets has become much easier. One of the major problems encountered in making the embedded field more accessible is the popular C programming language. Don’t get us wrong, C is very good for projects on microcontrollers. But the Raspberry Pi can accomplish more sophisticated tasks than a microcontroller. For beginners, the C language is hard to access online resources as they need to know about network sockets and low level programming.

We think there is a need for high level languages, but with the power of C. With Wyliodrin, we have integrated the C/C++ GPIO libraries into several programming languages so that you are able to program in the language that you know. The available languages include Pascal, C#, Javascript, Python, Shell, PHP, Perl and Objective-C. This is very good for education as teaching electronics in school becomes easier. Students can use their favourite programming language to build things.

Starting a project from scratch can be difficult. We always take examples and modify them in

order to learn. Wyliodrin allows you to start a new project using existing examples. This way you can see it working and then start modifying.

How to setup and use Wyliodrin

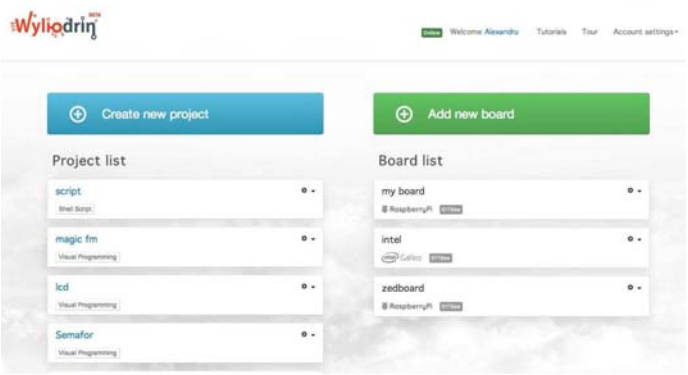
Wyliodrin is currently available for the Raspberry Pi. It provides an online IDE, accessible from http://www.wyliodrin.com. All you need to do is sign up, download the Raspberry Pi software from Wyliodrin, write it to an SD card and start building projects.

The Wyliodrin software on the SD card is the latest Raspbian Linux with Wyliodrin's server and development libraries installed. You can use the image directly, or you can download the source code and build it on your own Linux distribution. You can find it at https://github.com/Wyliodrin.

First, logon to the Wyliodrin website at http://www.wyliodrin.com using either your Facebook, GitHub or Google account .

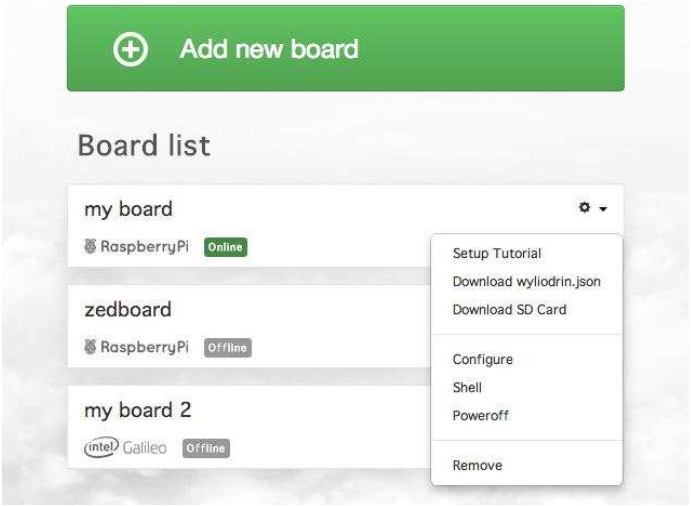
You are presented with the Projects page, where you can create a new project and add the board you want to develop on (i.e. a Raspberry Pi). But, before doing that you should follow the short virtual tour and you can study the tutorials from http://projects.wyliodrin.com/wiki.

Click on the "Add new board" button to add your Raspberry Pi. You need to give it a name. Next, you will be asked about the networking setup. In order to use Wyliodrin you will need to connect your Raspberry Pi to the internet so it can



connect to Wyliodrin's servers. An Ethernet connection is recommended. Plug in a network cable and it should work. If you have Wi-fi you will need to attach a USB Wi-fi dongle to your Raspberry Pi and follow configuration steps on the website. Wyliodrin will then display some instructions on how to connect your Pi.

1. You need a 4 GB SD card.
2. Download the Wyliodrin SD card image from http://projects.wyliodrin.com/images/raspberrypi.
3. Write the image to the SD card. For Windows, you can use Win32DiskImager (http://sourceforge.net/projects/win32diskimager). For an Apple Mac computer you can use PiWriter (http://sourceforge.net/projects/piwriter). For Linux you can use the dd command. For more information please read our tutorial at https://www.wyliodrin.com/wiki/boards_setup/raspberrypi.
4. Copy the JSON configuration file to your SD card. This file is specific to your Raspberry Pi and network. It contains your Raspberry Pi identification and network setup. You can download it from your Wyliodrin account, from the pull-down menu for your Raspberry Pi.



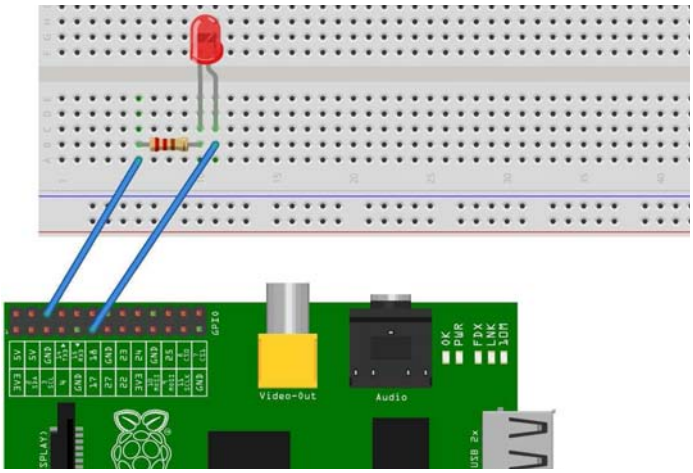
Copy the downloaded file to the root of the SD card. Make sure the file is named `wyliodrin.json`.
5. Power up your Raspberry Pi and wait for it to appear online on your Wyliodrin webpage.

Let's now make our first project using Wyliodrin.

Blinking LED

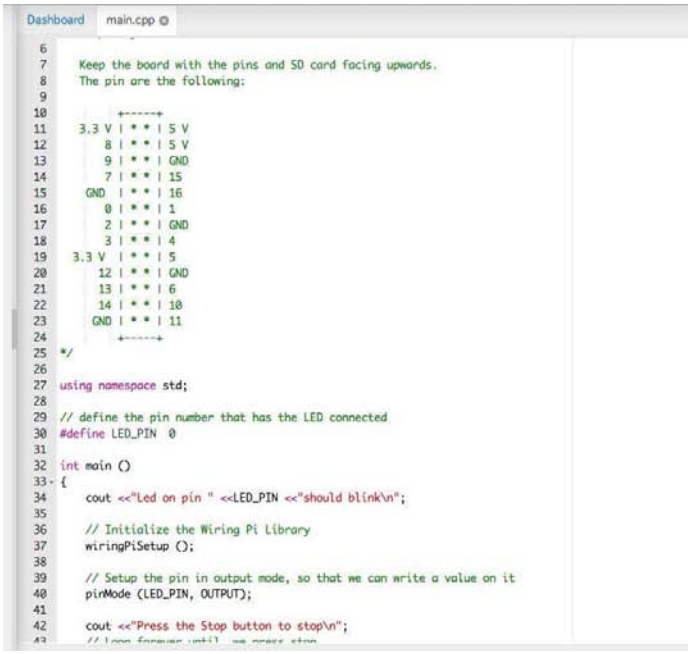
For the electronics setup you need the following basic components:
• breadboard
• LED
• 220Ω resistor (or similar, e.g. 470Ω)
• 2x jumper wires
• Raspberry Pi

We have to wire the components, keeping in mind that each GPIO pin on the Raspberry Pi acts like a software programmed power source. In Wyliodrin's tutorial "Raspberry Pi Pins Layout" we can see how to connect the components to the Raspberry Pi using WiringPi pin numbering.



The next step is to implement the project. Go to the Projects page and click on the “Create new project” button. Give it a name, a description and choose a programming language from Visual Programming, Shell script, C, C++, C#, Java, Javascript, Objective-C, Pascal, Perl, PHP or Python.

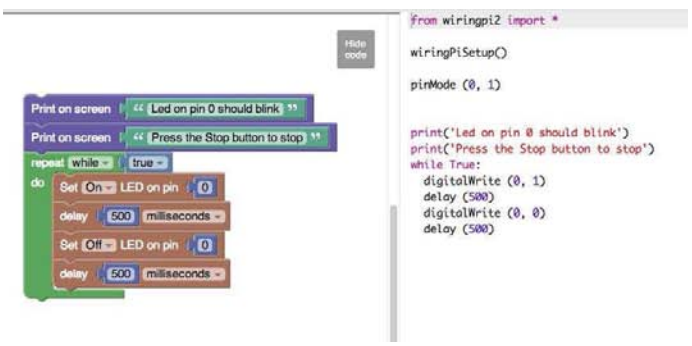
The "main" file was implicitly created together with the project. There you can write the code in the chosen programming language and then you will run it on the board. For example, below we have the code for making a LED blink using C++.



By choosing `LED Blink - C++` when creating the new project, you can use the sample code provided by Wyliodrin. You only have to click on the "Run" button and choose your Raspberry Pi in order to make the LED blink! All this time, the Raspberry Pi had no connection to the computer we wrote the program on.

For beginners who have not learnt a programming language, they can use Visual Programming. This lets you add blocks using drag-and-drop and Wyliodrin will automatically write the code. Visual Programming used by Wyliodrin is based on Google Blockly, a language like Scratch but for electronics. Wyliodrin implemented pins, LEDs and button blocks, to let you run several applications.

Below is the Visual Programming blocks needed to make a LED blink using the Raspberry Pi, plus the generated Python code.

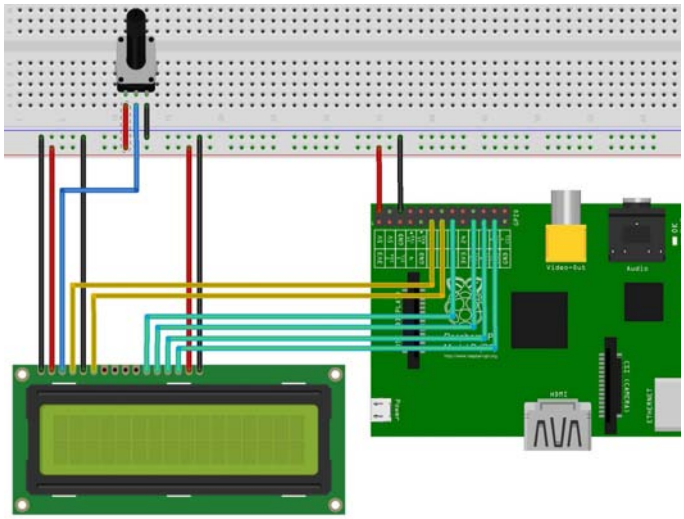


Music and an LCD

Our next Wyliodrin project shows you how to build an internet radio complete with an LCD. We will use the LCD to display a VU meter plus the name of the song that is currently playing. For the electronics setup you need the following components:

- breadboard
- 16x2 LCD screen (other sizes can be used)
- 10kΩ potentiometer
- 14x jumper wires
- Raspberry Pi

The wiring diagram is shown below.



Wiring up the LCD can be done in two ways; we will use the one that requires the least number of wires. You have to connect power to the LCD. Please be careful. The pin layout of the LCD may vary from model to model, so take a look at its data sheet. Our LCD is a 5V one with 16 columns and 2 rows. Its data sheet can be found at http://robofun.ro/docs/RC1602B-BIW-CSX.pdf Connect the ground pin (V_{SS}) to the ground of the Raspberry Pi and the 5V pin (V_{DD}) to the 5V pin on the Raspberry Pi. The VO pin on the LCD is used to set up the contrast, so connect it to the wiper of the 10kΩ potentiometer. Also wire the backlight pins (LED - and LED +) to the ground pin and 5V pin on the Raspberry Pi.

You also need to connect the RS, E, DB4, DB5, DB6 and DB7 pins to the Raspberry Pi. You can connect them to any of the Raspberry Pi GPIO data pins. The R/W pin is connected to ground as we will only write data to the LCD.

Let's make a new project and write the software. Click on the "Create new project" button and choose `Music - Visual Programming`, so that you can drag and drop blocks and Wyliodrin will write the Python code for you. This way you have the music player set up. Once you have created the project, click it to edit it.

As you can see above, you already have the blocks for playing music from an online radio station. The default station does not show the title of the song, so please change it to another. Click on the block "Load audio stream from" and enter another address. *[Ed: See The MagPi issue 21, page 24 for some examples, e.g. http://pub6.sky.fm:80/sky_tophits.]* Let's test if it plays. Click on the "Run" button and don't forget to connect a speaker to your Raspberry Pi.

Now let's modify the program so that it displays the song title and then shows a VU meter. First, you need to initialise the LCD. Go to the LCD blocks on the left toolbar and drag the "Init LCD" block and place it between the "Play audio stream" and "repeat" blocks. You will see you need to set the LCD pin numbers. For our LCD, the pins are RS = 0, E = 2, Data1 = 3, Data2 = 12, Data3 = 13 and Data4 = 14. They are the same as in the wiring diagram. Another parameter is the number of columns and rows. Our LCD is 16 columns and 2 rows.

Let's display the internet radio station information. Inside the "repeat" block, after the "delay" block, put a "Reset Position on LCD" block. The LCD works like a normal screen: once you write to it, it moves the cursor ahead. As we want to display the name at the beginning, you need to add a "Reset Position on LCD" block.

After this, place a "Print on LCD block". Inside this block, replace the text block with the "Audio stream Address Title" block, which is found in the Multimedia blocks on the left toolbar. This block receives a parameter with the variable of the stream. By default, this is called `audio`. Change it to `music` (the same variable used for the other Multimedia blocks).

Click the "Run" button and you should see the station name or song title (depending on what the station broadcasts) on the LCD. Some stations do not provide this information so don't worry if nothing is written.

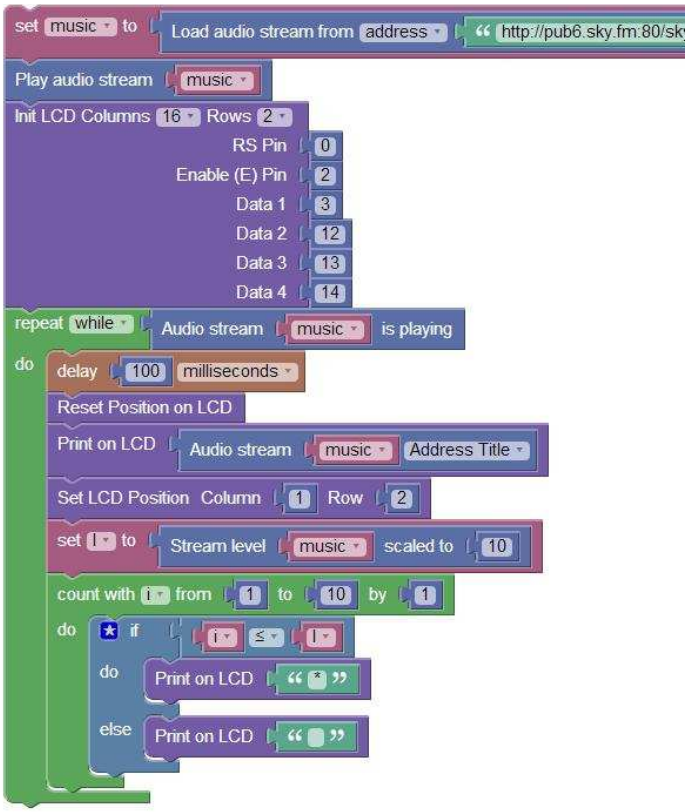
The VU meter is a measure of how loud the music is. We want to display this in the form of * (asterisks) on the LCD. For this, you need to use the "Stream level" block from the Multimedia blocks. First, set a variable called `l` to the value returned by the "Stream level" block. To do this, drag the "Set variable to" block from Variables and place it after the "Print on LCD" block. Create a new variable called `l`. Connect to this block the "Stream level" block. Select the `music` stream and set the scale to `10`. The scaling will tell the block to return a value between 0 and 10.

Next we need to display the asterisks. Set the LCD position to the beginning of the second row Place the "Set LCD Position" block before the "Set variable to" block and set the row to 2. From the Loops blocks add a "count with" block and set its variable to `i` and make it count from 1 to 10 by 1. This means that the content of this block will be repeated for every value of `i` from 1 to 10.

Inside this block we display on the LCD either a * or a space using the following rule. If the variable `l` (which stores the current level) is less than `i`, then we display a * otherwise we display a space. For this we use a Logic block called "if". Drag an "if" block inside the "count with" block. The "if" block has a star on it. If you click it, a small window will appear and we can set up the block. We need to drag an "else" block inside the "if" block. Once you have done this, you are ready to write on the screen. Connect to the "if" block the "less or equal" Logic block. The condition we want is `i <= l`. We need to drag in the `i` and `l` blocks from the Variables block.

The "if" block has two parts: "do" and "else". The "do" part will be executed if the condition is true or the "else" part will be executed if the condition is false. In the "do" part, place a "Print on LCD" block with the text *. On the "else" part, place a "Print on LCD" block with a space.

Before you run the program, change the "delay" block to 100 milliseconds for a faster update of the VU meter. The final program is shown below.



Future perspectives

We are working on implementing graphics, which will allow users to monitor their Raspberry Pi in real-time. We also plan on introducing more blocks for the Visual Programming language. We would like to ask you to provide feedback. This is very important for us. If you would like new features or think that we should do something differently, please don't hesitate to contact us.

You can read more about the project at http://www.wyliodrin.com, on the Wyliodrin Facebook and Google+ pages and also on Twitter @wyliodrin.

Adding custom devices to RpiScratchIO

W. H. Bell

MagPi Writer

SKILL LEVEL : ADVANCED

Scratch is a very useful programming language that children and students can pick up quickly. However, one feature lacking from the current version of Scratch is Input/Output (I/O) support. Thankfully, Scratch supports network messages that can be exchanged with local or remote devices. The RpiScratchIO package, which was intoduced in Issue 20 of The MagPi, was written to provide a general protocol for I/O devices and minimise the overhead of adding new devices to Scratch. This package uses the network broadcast message system in Scratch, via the scratchpy Python library. The RpiScratchIO package implements a general protocol that can be used to connect any I/O device to Scratch. Additional documentation for RpiScratchIO can be found at :
https://pypi.python.org/pypi/RpiScratchIO/
The purpose of this article is to demonstrate how to add a user defined I/O device to Scratch using RpiScratchIO.

Installing RpiScratchIO

This article requires RpiScratchIO version 0.1.5 or greater. If RpiScratchIO has not already been installed, then follow the instructions in Issue 20 of The MagPi to install the package. If RpiScratchIO is already present then update the package by typing:

```
sudo pip install RpiScratchIO --upgrade
```

I/O devices in RpiScratchIO

An I/O device is anything that reads or writes data. This could be a LINUX command or an expansion board. RpiScratchIO expects that an I/O device has a list of input and output channels associated with it. For simple devices, this could imply one output channel only. The channels on the device do not have to be numbered sequentially, to allow for physical pin numbering or another non-sequential system. A channel can function as an input or as an output device or both.

Each I/O device is described by a simple Python class that has to implement a constructor and can optionally implement config, read and write member functions. When config, read or write commands are sent from Scratch, the RpiScratchIO package calls the `config`, `read` or `write` functions of the associated I/O device. If

one of these functions has not been implemented and it is called by Scratch, then a warning message is printed to warn the user that the function has not been implemented.

Setting the PYTHONPATH

For a user defined Python class to be included in RpiScratchIO, it should be saved in a file that is within a directory that is in the PYTHONPATH. The PYTHONPATH is an environmental variable that can be set in the same way as any other Bash environmental variable,

```
export PYTHONPATH=$HOME/user-devices
```

Rather than risk overwriting previous configuration or continuously appending to the PYTHONPATH, a little more Bash can be used:

```
user_devices=$HOME"/user-devices"
if [[ -z $PYTHONPATH ]]; then
  export PYTHONPATH=$user_devices
elif [[ $PYTHONPATH != *"$user_devices"* ]]; then
  export PYTHONPATH="$PYTHONPATH:$user_devices"
fi
unset user_devices
```

Similar to the PATH variable, the format is that multiple directories are separated by colons. In this example, the `$HOME/user-devices` directory is where the Python files that contain the user defined devices are stored. This script could be added to the end of the `~/.bashrc` file or sourced beforehand.

Adding an I/O device

Create a `BasicDevice.py` file in the `$HOME/user-devices` directory. Then append to this file,

```
import RpiScratchIO
from RpiScratchIO.Devices import GenericDevice

class BasicDevice(GenericDevice):
  def __init__(self,deviceName_,scratchIO_,connections_):

    # Call the base class constructor
    super(BasicDevice, self).__init__(deviceName_,scratchIO_,connections_)

    # This device has a single input channel
    self.inputChannels += [0]

    # Add other code here if needed...
```

This represents a device that has one input channel. All device classes must inherit from the `GenericDevice` class in the RpiScratchIO package, either directly or via one of the other base classes. The constructor of the class must contain the four default arguments, the device name, the pointer to the `ScratchIO` object and a list of connections associated with the device. Other arguments can be passed to the constructor after the default arguments. The other thing to note about this constructor is that the single input channel number is appended to the `self.inputChannels` list. This list and the `self.outputChannel` list are defined in the `GenericDevice` base class. Therefore the derived class should append to the lists. To use the example

BasicDevice class in RpiScratchIO, create a test.cfg configuration file that contains:

```
[DeviceTypes]
test = from BasicDevice import BasicDevice; BasicDevice()

[DeviceConnections]
test = something
```

The user defined device class is imported, instantiated and associated with the name test. The default arguments are automatically passed to the constructor of the BasicDevice class. Since the device connections are not used in the example device class, any dummy value can be assigned to test. To use this class, create a Scratch program that broadcasts config, read and write messages:

Then enable the remote sensor connections, by right clicking on the "sensor value" text at the bottom of the "Sensing" tool palette. (When the current Raspbian version of Scratch starts, remote sensor connections are not enabled. To enable them, disable and then re-enable them. Refer to the article in Issue 20 for more information.) Now start RpiScratchIO, using the configuration file.

```
RpiScratchIO test.cfg
```

Then write a program to test the test sensor value and send read, write and config commands. In this example, the sensor value from the device will always be zero. Calling the read, write and config functions will cause warning messages to be printed on the screen. These warning messages are printed by the base class versions of the config, read, and write functions. These functions can be overridden be implementing functions with the same input arguments as those given in GenericDevice.

Pinging a computer

The ping command was discussed in Issue 21 of The MagPi. ping can be used as an I/O device with one input channel, where the input channel is a selected result from the ping command. The steps to create a device class that uses a ping command are the same as for the previous example. First, create a file called PingHost.py in the $HOME/user-devices directory,

```
import subprocess,string
import RpiScratchIO
from RpiScratchIO.Devices import GenericDevice

class Ping(GenericDevice):
  def __init__(self,deviceName_,rpiScratchIO_,connections_):

    # Call the base class constructor
    super(Ping, self).__init__(deviceName_,rpiScratchIO_,connections_)
```

```
  # This device has a single input channel
  self.inputChannels += [0]

  # The default host, in case no host is configured
  self.hostname = "localhost"

#-----------------------------

def config(self,argList):
  nargs = len(argList)
  if nargs == 0:
    print("WARNING: \"config\" in device %s expects at least one argument." % self.deviceName)
    return None

  # Set new host name to ping
  self.hostname = argList[0]

#-----------------------------

def read(self,channelNumber):

  # Setup the ping command to run once
  ping = subprocess.Popen(
    ["ping", "-c", "1", self.hostname],
    stdout = subprocess.PIPE,
    stderr = subprocess.PIPE
  )

  # Set the default value to indicate something went wrong
  avgRoundTrip = -999

  # Run the ping command
  out, error = ping.communicate()

  # Parse the standard out and error
  if string.find(error,'unknown host') != -1: # LINUX specific
    avgRoundTrip = -2
  elif string.find(out,'100% packet loss') != -1: # LINUX specific
    avgRoundTrip = -1
  else:
    for line in string.split(out,'\n'):
      # This search string is Linux specific
      if string.find(line,'rtt min/avg/max/mdev') == -1:
        continue
      frags = string.split(line,' = ')
      if len(frags) != 2:
        print "WARNING: badly formatted string"
        continue
      values = string.split(frags[1],'/')
      if len(values) < 2:
        print "WARNING: badly formatted values string"
        continue
      avgRoundTrip = float(values[2])
```

```
    # Send the value back to Scratch
    self.updateSensor(channelNumber,avgRoundTrip)

    # Since there might be a significant delay, send Scratch a trigger message.
    self.broadcastTrigger(channelNumber)
```

The next step is to create a configuration file to associate the class with Scratch. Then launch RpiScratchIO, to read the new configuration file. Once RpiScratchIO is running, the `ping` sensor will be available in Scratch.


```
[DeviceTypes]
ping = from PingHost import Ping; Ping()

[DeviceConnections]
ping = localhost
```

In the configuration file, the connection passed to the `Ping` class is the default computer to `ping`.

In the example Scratch program, a sprite with two costumes was created, where one of these costumes was named `ping` and the other was named `idle`. Then broadcast commands were added to test each of the functions and the response of the `Ping` device.

Try pressing the spacebar. If the `b` key is pressed, then a `badhost` name is used. This will produce a sensor error value of `-2`. In the case of this example program, the `192.168.2.1` address was a router that was firewalled. This caused a sensor error value of `-1`. In the case of the `localhost`, the round trip time reported by ping is quite quick. Try swapping the IP addresses for some other machine on the network.

When the `ping` command runs it waits until a reply is received or it reaches the timeout value. This means that `ping` can be a bit slow to respond, if it takes a long time for the network packet to return. Rather than polling the output of the `ping` command, the `Ping` device sends a `trig` message back to Scratch to indicate a new sensor value is available. This means that Scratch is idle until `ping` has finished.

Next time

The next tutorial will include some examples of writing output results and interfacing with hardware devices. Until then, try inheriting from the `SpiDevice` base class and connect to an SPI device. There is an example of how to do this in the `SpiDevices.py` file in the RpiScratchIO package.

METAL CASE
FOR PI & CAMERA
USE OUTSIDE!

DESIGNED FOR

AS SEEN ON KICKSTARTER

FROM £26!

TRI-POD MOUNT
THERMAL COOLING
REMOVABLE SIDE PANELS

THE PI FITS

PERFECTLY

AVAILABLE IN BLACK, SILVER, RED & GREEN
BUY 2 AND MIX & MATCH!

NEW

WEATHER SHIELD

USE OUTSIDE

IN THE RAIN

VESA MOUNT

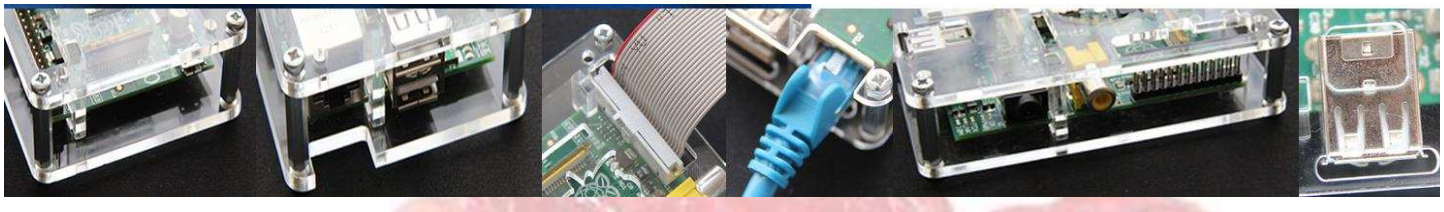
FOR A MONITOR

PiCE RASPBERRY PI
CAMERA ENCLOSURE

BY EDVENTURE www.ed-venture.biz

USE CODE: MAGPITHX FOR 10% DISCOUNT!

APRIL COMPETITION



Once again The MagPi and PC Supplies Limited are proud to announce yet another chance to win some fantastic Raspberry Pi goodies!

This month there are THREE prizes!

Each of the three winners will receive a Gerboard mount and a 32GB SDHC Memory card. The prefect accessories for a serious Raspberry Pi project.

For a chance to win this month's competition visit **http://www.pcslshop.com/info/magpi**

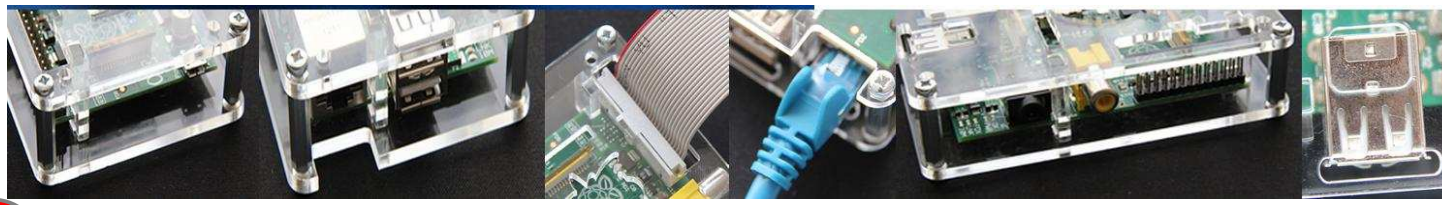
Closing date is 20th April 2014.
Winners will be notified in the next issue.

To see the large range of PCSL brand Raspberry Pi accessories visit **http://www.pcslshop.com**

March's Winner!

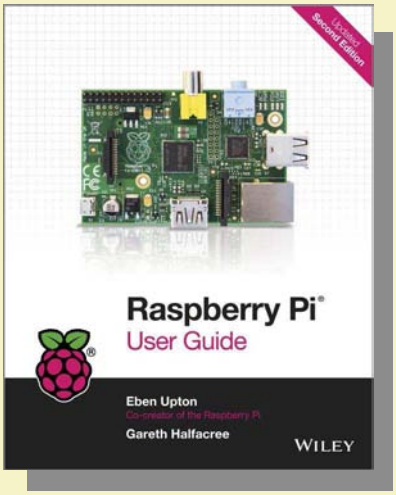
The first prize winner of a Pi NoIR camera board, 32GB SD card, VESA mount case and a gigabit network hub! **Scott Lindsay (East Kilbride, Scotland)**
The second prize winner of a Cyntech Geek case, GPIO breakout board, breakout cable set and a 16GB SD card! **Lee Murphy (Wakefield, England).**

Congratulations. We will be emailing you soon with details of how to claim your prizes!



Raspberry Pi User Guide (2nd Edition)

Eben Upton and Gareth Halfacree
Wiley



The Raspberry Pi has been a success beyond the dreams of its creators. Their goal, to encourage a new generation of computer programmers who understand how computers work, is well under way.

Raspberry Pi User Guide 2nd Edition is the newest edition of the runaway bestseller written by the Raspberry Pi's co-creator, Eben Upton, and tech writer Gareth Halfacree. It contains everything you need to know to get the Raspberry Pi up and running, including how to connect up a keyboard, mouse and other peripherals, installing software, basic configuration and Linux system administration, using the Raspberry Pi as a web server or media centre, learning programming languages Scratch and Python and, of course, how to use the GPIO pins to interface with electronics projects.

Unlike the previous edition, this book now includes coverage of the Camera Board, the introduction of the Pi Store, NOOBS and much more. Raspberry Pi User Guide 2nd Edition is the perfect companion for getting the most out of the computing phenomenon that is the Raspberry Pi.

MagPi readers are able to receive a 30% discount off the RRP. To claim, purchase from www.wiley.com and enter the promo code **VBH02** when prompted.

Please note: this discount is only valid from the 1st April to the 31st May 2014 on the two book titles listed here.

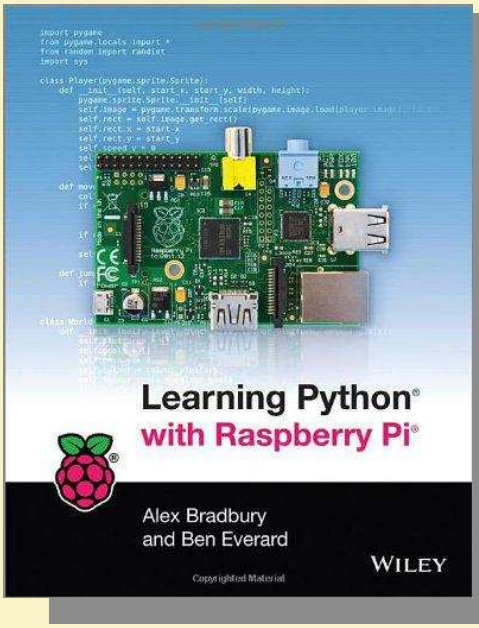
Learning Python with Raspberry Pi

Alex Bradbury and Ben Everard
Wiley

Python is one of the world's top programming languages. In addition to being used by the likes of Google, Spotify, YouTube and the BBC, it is also the language of choice for the Raspberry Pi.

Co-authored by one of the lead software developers for the Raspberry Pi Foundation, Learning Python with Raspberry Pi is the must-have companion to the Raspberry Pi User Guide (above) and an indespensible resource for anyone wishing to learn how to program on the Raspberry Pi.

Assuming no prior programming experience whatsoever, it covers all of the bases for novice programmers and first-time Python developers. Following a brief introduction to programming and coding in general and Python in particular, authors Alex Bradbury and Ben Everard cut to the chase with clear, step-by-step guidance on configuring and getting started, using variable, loops and functions, learning 3D graphics programming, using PyGame, programming Minecraft, and using the GPIO port.



Featuring easy-to-follow hands-on instruction and packed with working examples and useable source code, Learning Python with Raspberry Pi gets you up and running with the knowledge and skills you need to write your own programs in Python.

Feedback & Question Time

I just wanted to let you know that I really enjoyed Martin Hodgson's Python port of Strongholds of the Dwarven Lords that was in the recent issue of The MagPi. I plan on sharing it with some of my more advanced gifted middle school students. Please continue to include Python programs like this in your magazine. It's nice to be able to do cool things with the Raspberry Pi without needing to purchase hardware or having the needed knowledge to make it work. I hope to see more of Martin's submissions in future issues!

Thanks!
Jason Kibbe
CAMS North & South Gifted Support

Really enjoyed this issue - thank you very much!

I particularly liked the type-in, oh the memories... Anyway, as per normal, it did not work first time - never had one that did way back then either on my Amstrad. I have enjoyed fixing it.

Keep up the good work.

Kind regards,
Ian Neill

After taking the plunge and purchasing my very first Raspberry Pi, I stumbled upon your great magazine, As a complete novice to the world of the Raspberry Pi, The MagPi magazine is an excellent resource. The variety of articles, guides and project ideas are an inspiration; there is really something for everyone!

Many thanks,
Caron Jones

Good day. I am Ober Choo from Cytron Technologies, Malaysia.

We are one of the Raspberry Pi resellers in Malaysia. Personally I love your magazine especially as it is open!

Regards,
Ober Choo
Cytron Technologies

If you are interested in writing for The MagPi, would like to request an article or would like to join the team involved in the production, please get in touch by emailing the editor at:

editor@themagpi.com