ISSUE 29 - DEC 2014

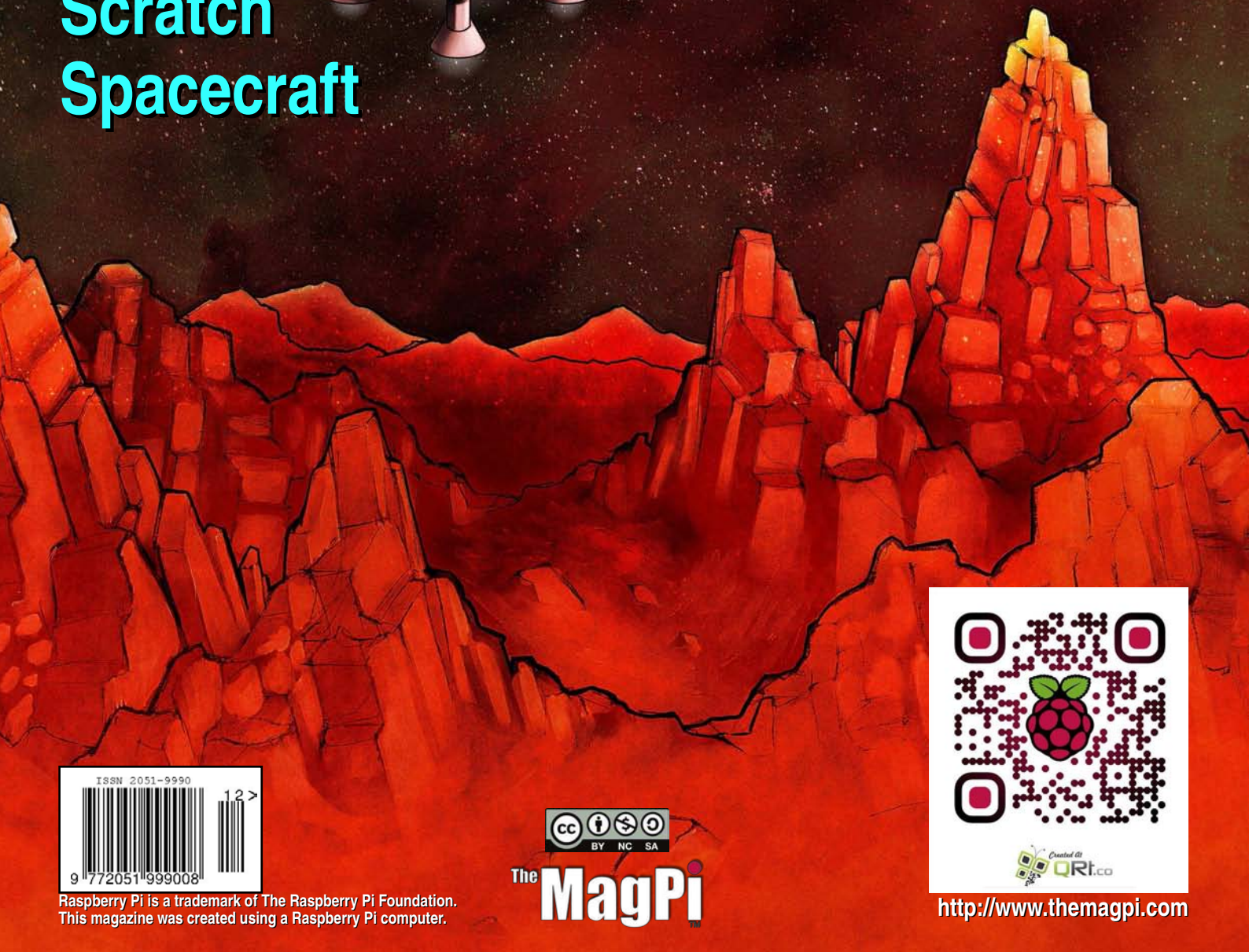
Get printed copies at themagpi.com

The MagPi

A Magazine for Raspberry Pi Users

Temperature Sensors
External Storage
Project Curacao
BASIC Robotics
Traffic Light
Python GUI
A+ Review
Using Git
OpenCV

Scratch Spacecraft



ISSN 2051-9990

The MagPi

http://www.themagpi.com

Welcome to Issue 29 of the MagPi, packed with the usual mixture of hardware projects and programming articles, providing lots of avenues for invention during December.

With the Christmas holidays drawing near, what could be better than some new Raspberry Pi hardware. For all those looking forward to building a high altitude capsule or autonomous submarine, the Model A+ provides many great features for a very low power budget. Dougie Lawson presents a whistle-stop tour of the A+, comparing it to other Raspberry Pi Models.

On the subject of robots, computer vision can provide an image cognition solution within many autonomous robotics projects. Derek Campbell sketches out more features of OpenCV (open source computer vision) image recognition software.

The Raspberry Pi is ideally suited as the hub of a sensor array or control unit, since it can be used to propagate information via a web server or other remote protocol. In this Issue, John Shovic's presents his final article in the Project Curacao remote monitoring series, David Bannon demonstrates how to build and read a simple array of digital temperature sensors, and Brian Grawburg introduces his traffic light extension board.

When developing software or projects, it is important to retain unique files that are part of the build. In this Issue, Alec Clews continues his series on software repositories and using Git, and William Bell discusses the basics of adding external storage to the Raspberry Pi.

Computer programming enables the Raspberry Pi to be used within many different applications. This month, Jon Silvera discusses how to drive a robotic arm with FUZE BASIC, William Bell presents a simple space arcade game in Scratch and Paul Sutton introduces Python graphical user interfaces (GUIs).

The MagPi will be taking a short break over Christmas and the first Issue of 2015 will be published at the start of February.

Merry Christmas and best wishes for 2015.

Chief Editor of The MagPi

The MagPi Team

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

Contents

4 **TRAFFIC LIGHT**
Simulating a bi-directional traffic light

8 **PROJECT CURACAO**
Part 6: Upgrades on the Beach

14 **NEW MODEL A+**
Introducing the latest Raspberry Pi hardware

18 **INTRODUCING OPENCV**
Part 2: Computer Vision on the Raspberry Pi

22 **DIGITAL TEMPERATURE SENSOR**
Logging temperature with 1-wire sensor

26 **FUZE BASIC**
Part 5: Using FUZE BASIC to control a robot arm

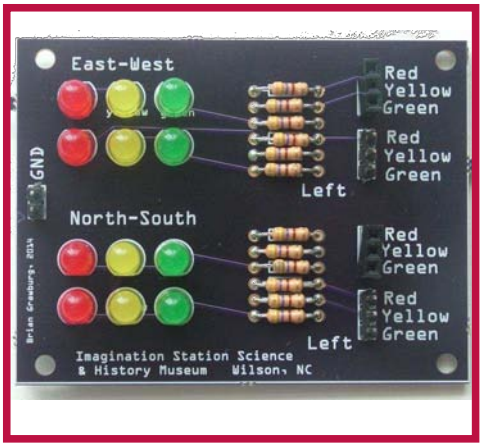
31 **EXTERNAL STORAGE**
Part 1: File systems, partition tables and rsync

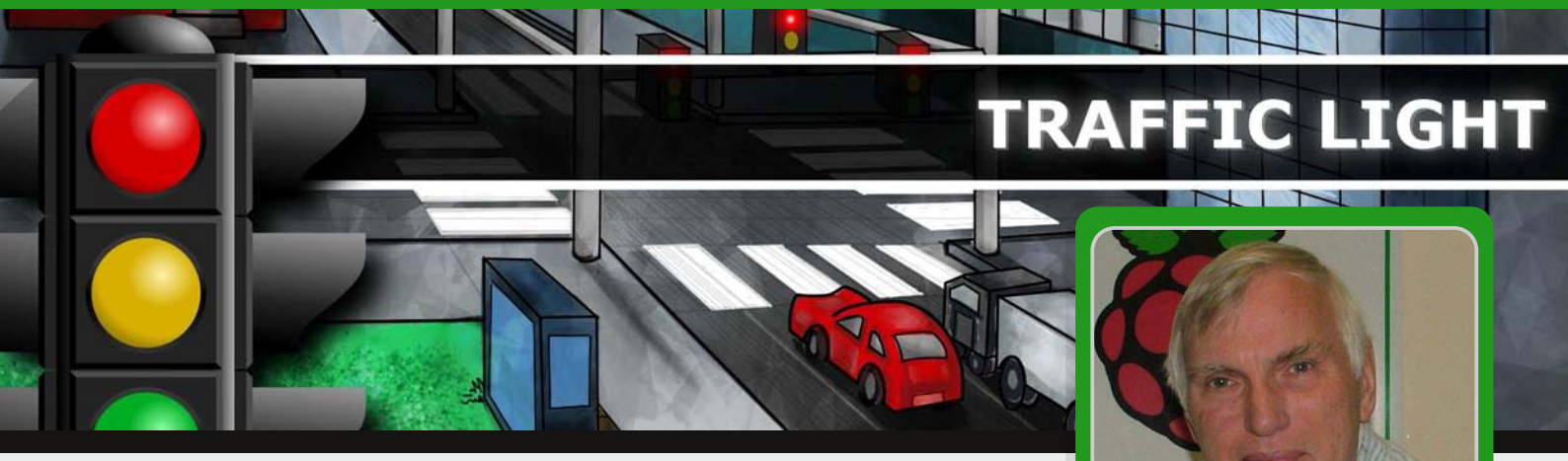
36 **VERSION CONTROL**
Version control basics using Git - Part 3

40 **SCRATCH PATCH: GOING BALLISTIC**
Learning to land on Mars

43 **THIS MONTH'S EVENTS**
Manchester, Lagos, Northern Ireland, Glasgow, Saarbrücken

44 **PYTHON PIT: MAGIC 8 BALL**
Creating a GUI with Python's Tkinter





Simulating a bi-directional traffic light

Brian Grawburg

Guest Writer

SKILL LEVEL : BEGINNER

The Imagination Station Science Museum in North Carolina (USA) offers several Raspberry Pi and Python classes and camps. This article is a shortened version of a project given to the students after they've completed about 20 hours a preliminary exposure to the Pi and Python. A previous project introduced the use of an 8 port I/O expander (MCP23008) and provided sufficient background in binary and hexadecimal numbering to enable them to complete this project on their own. New to this project was the CD74AC02 Quadruple 2-input NOR gate IC and the use of two MCP23017 chips (16 I/O ports each).

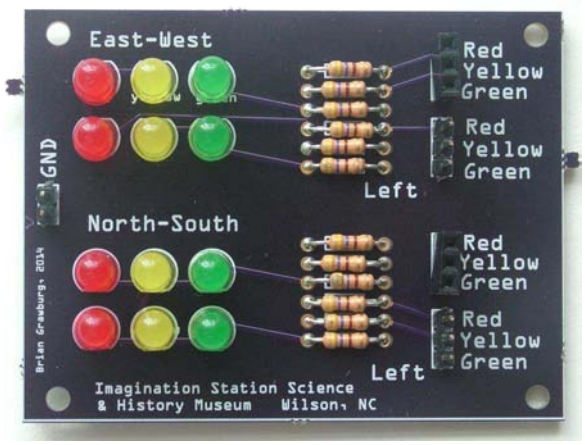
Overview of the Project

A typical traffic signal at a major intersection in the U.S. has a red-yellow-green light and often a left-turn signal, also with red-yellow-green lights/arrows. Although many intersections are asymmetrical as regards turning lanes and timing, for this project I limited the number to two symmetrical directions – North (representing north-south) and East (representing east-west).

Like a real traffic signal, the Python code turns on the red for one direction while the green and then yellow are on for the opposite direction; then both reds are on for a short time simultaneously until the cycle starts over.

Most intersections with a separate left-turn lane don't usually activate the left-turn lights if there are no cars in the lanes. For this project momentary push-buttons simulate a car in the lane to activate both left-turn lanes.

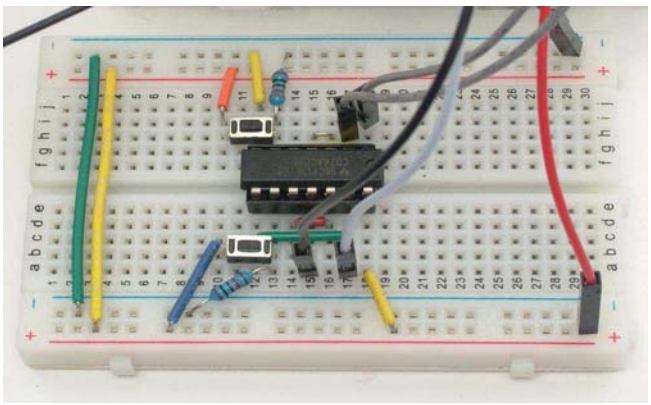
The MCP23017s could be fitted to a breadboard and then wired back to the Pi's GPIO pins with jumpers, however I decided to use a pre-fabricated Protect Your Pi board from MyPiShop.com to make the connections much easier. In addition, the board can be reused for a variety of projects.



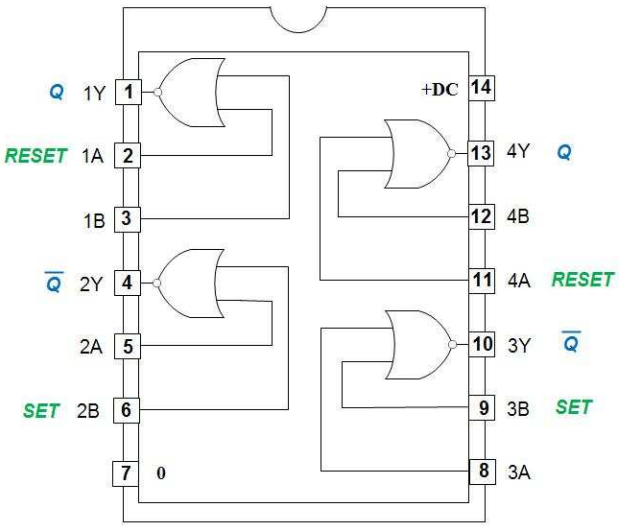
The project uses 12 LEDs and 12 resistors; I made PCBs for the students to use because I felt it was much easier to work with a PCB than a breadboard. The CD74AC02 and the pushbuttons, however, are on a breadboard.

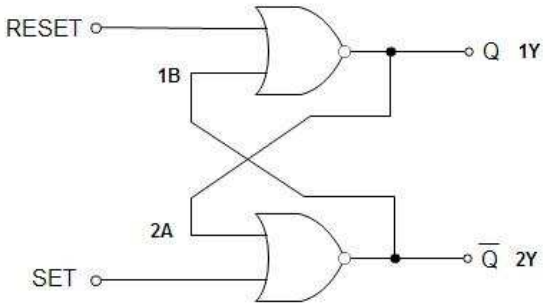
About the CD7AC Chip

One of the more frustrating aspects of this project was trying to incorporate the left-turn option into the Python code. No doubt there was a way to do it but I couldn't figure it out (I saw one possible way, but it required a lot of code). Here's where being part of a user group can really make a difference. I'm active in the Triangle Embedded Devices group (www.triembed.org) and posted a series of emails to the group; got a great answer that I'm using here.



The "problem": As the lights were cycling through I wanted to push a button to simulate a car pulling into the left turn-only lane which then activates something to tell the program "there's a car that wants to turn so when you're finished





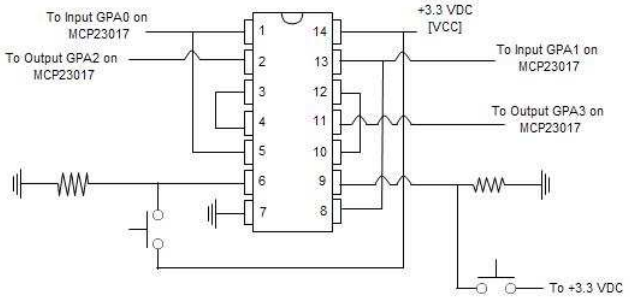
with the normal cycle run the left turn cycle." Back in 2010/2011 I took a year-long course at a local community college on programming PLCs (Programmable Logic Controllers). One of the basic functions is latching an output, meaning that when it is set high it would remain high until specifically set low by the program. That's what I was trying to do within the Python code.

The CD74AC02 contains four independent 2-input NOR gates as diagrammed below (NOR is an abbreviation for NOT OR, two options in logic construction). I needed to make a latching Set-Reset flip flop: a logic circuit that has two inputs and one output. Latch circuits can be either active-high or active-low.

Active-high circuit: Both inputs are normally tied to ground (LOW) by a pull-down resistor and the latch is triggered by a momentary HIGH signal on either of the inputs.

Active-low circuit: Both inputs are normally HIGH, and the latch is triggered by a momentary LOW signal on either input.

In an active-high latch when the SET input goes HIGH, the output also goes HIGH. When the set input returns to LOW, however, the output remains HIGH. The output of the active-high latch stays HIGH until the RESET input goes HIGH. Then, the output returns to LOW and will



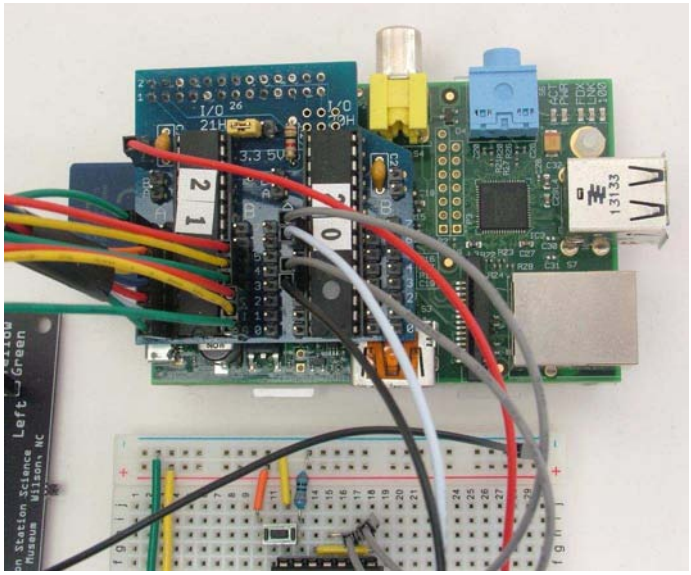
go HIGH again only when the SET input is triggered once more.

The latch remembers that the SET input has been activated. If the SET input goes HIGH for even a moment, the output goes HIGH and stays HIGH, even after the SET input returns to LOW. This is what happens when one of the pushbuttons is depressed. The output returns to LOW only when the RESET input goes HIGH; this is written into the Python code.

The project is an active-high circuit. This option seems more intuitive to me.

The Protect Your Pi Board

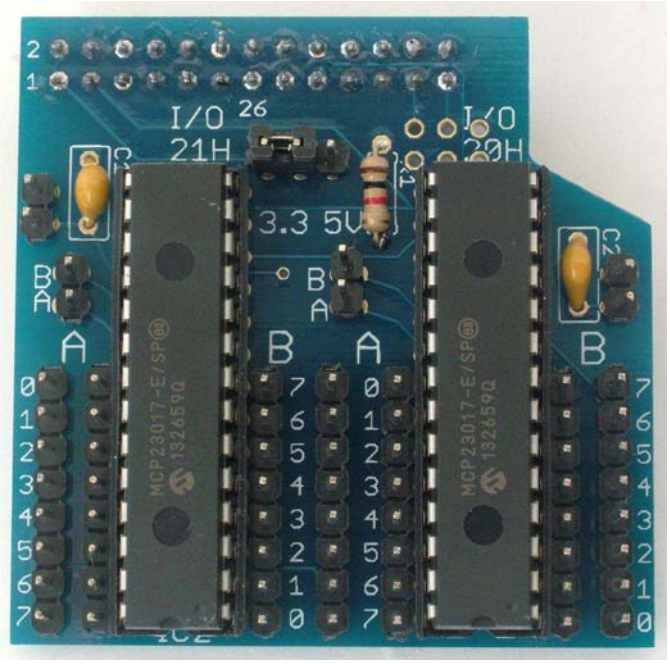
This project requires a minimum of 16 ports; 14 outputs and 2 inputs. The board can be constructed as a shield to attach directly to the Pi (as shown below) or attached with an external ribbon. If space is a limiting factor the



shield option is a good idea. However, I prefer to attach the boards to the Pi using a ribbon cable while I work on the code and component layout.

The CD74AC02 is powered by the Protect Your Pi board (3.3 VDC). Ground from the LED board and the breadboard is also connected to this board.

The left MCP chip (banks A and B) takes care of the outputs to the LEDs and the right chip (bank A) takes care of the inputs from the two pushbuttons and the RESET output from the Python code to the CD74.



The Python code I suggested to the students followed the pattern they had already encountered in previous classes. It was straighforward and easy to understand. A future large-scale project will be to recreate a small portion of the street that runs past the Imagination Station Science Museum with several intersections and a train crossing. The ultimate goal being to synchronize the lights for morning traffic (into town) and evening traffic (away from town).

The Code.

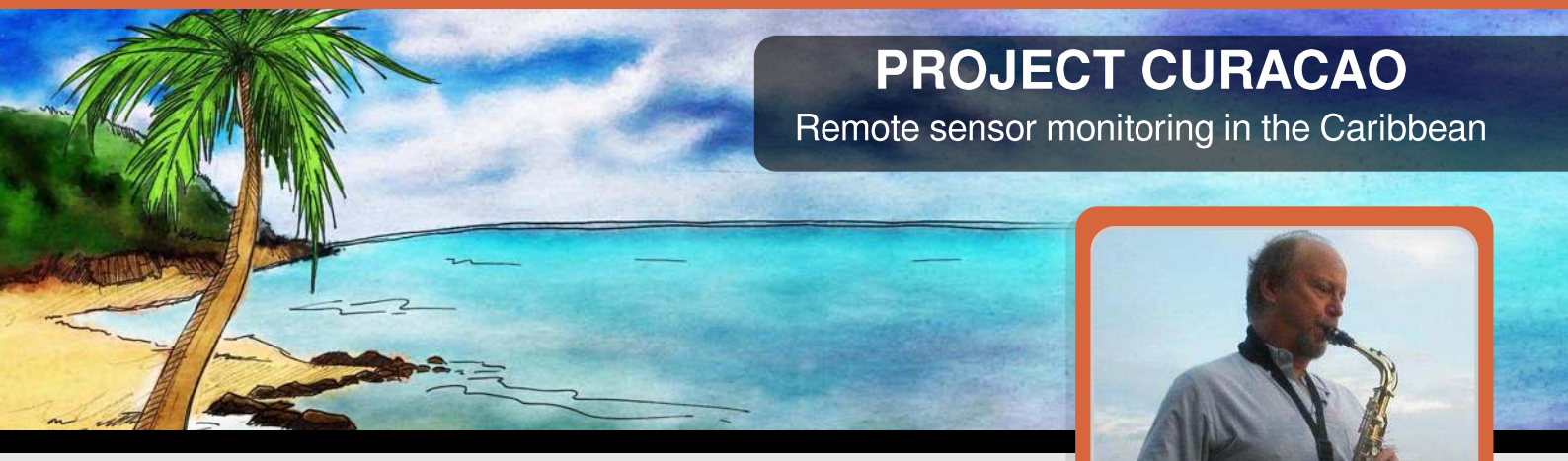
The following page shows a sample Python script that can demonstrate the system. Much of the script is about defining labels that refer to the various ports or values we later poke into those ports. Using labels (or constants, variables) like this is always a good idea as it makes the actual code much easier to read and modify.

The full project manual, including the Python code, can be downloaded from the Triangle Embedded Devices Web page: http://triembed.org/blog/?page_id=658.

Brian Grawburg is from the Imagination Station Science & History Museum, Wilson, NC (USA)

```
#!/usr/bin/env python
import smbus
import time

bus = smbus.SMBus(1)

address1   = 0x21
address2   = 0x20
IODIRA     = 0x00
IODIRB     = 0x01
GPIOA      = 0x12
GPIOB      = 0x13
OLATA      = 0x14
OLATB      = 0x15
light_on   = 5         # Straight lights
both_red   = 1
left_on    = 4         # Left lights

io_setting = [
     [0,0,0],          # Reset: all off, no delay
     [0x30,0x24,light_on],   # Straight East Green, North Red, L-turns Red
     [0x28,0x24,light_on],   # Straight East Amber, North Red
     [0x24,0x24,both_red],   # East & North Red, L-turns Red
     [0x84,0x24,light_on],   # North Green, East Red
     [0x44,0x24,light_on],   # North Amber, East Red
     [0x24,0x09,left_on],    # Both L-turns Green, Straights Red
     [0x24,0x12,left_on]     # Both L-turns Amber
]

def cycle(index):
     bus.write_byte_data(address1,OLATA,io_setting[index][0])
     bus.write_byte_data(address1,OLATB,io_setting[index][1])
     time.sleep(io_setting[index][2])

bus.write_byte_data(address1,IODIRA,0x00)
bus.write_byte_data(address1,IODIRB,0x00)
bus.write_byte_data(address2,IODIRA,0x03)

cycle(0)

bus.write_byte_data(address2,OLATA,0x00)

try:
     while True:
          buttonPressed=bus.read_byte_data(address2,GPIOA)
          if buttonPressed == 0:
                    cycle(1); cycle(2);
                    cycle(3); cycle(4);
                    cycle(5); cycle(3)
          else:
                    cycle(6); cycle(7); cycle(3)
                    bus.write_byte_data(address2,OLATA,0x18)
                    bus.write_byte_data(address2,OLATA,0x00)

except KeyboardInterrupt:
     cycle(0)
     bus.write_byte_data(address2,OLATA,0x00)
```

Sample code to demonstrate the Traffic Light System



PROJECT CURACAO

Remote sensor monitoring in the Caribbean

Part 6: Upgrades on the Beach

John Shovic

Guest Writer

SKILL LEVEL : INTERMEDIATE

What is Project Curacao?

This is the sixth and final part of a series discussing the design and building of Project Curacao, a sensor filled project that hangs on a radio tower on the island nation of Curacao. Curacao is a desert island 12 degrees north of the equator in the Caribbean. This article will show the upgraded sensor suite, higher reliability design and replacement of the ill-fated wind turbine.

Project Curacao is designed to monitor the local environment unattended for six months. It operates on solar power cells and communicates with the designer via an iPad App called RasPiConnect. All aspects of this project are designed to be monitored and updated remotely (with the current exception of the Arduino Battery Watchdog). It was first deployed in March 2014 and has been featured in five previous articles in The MagPi Magazine that covered the Raspberry Pi Model A, the various subsystems (Power, Environmental Sensor and Camera), the software running on the Raspberry Pi and Arduino Battery Watchdog, and the first months of operation.

First six month results

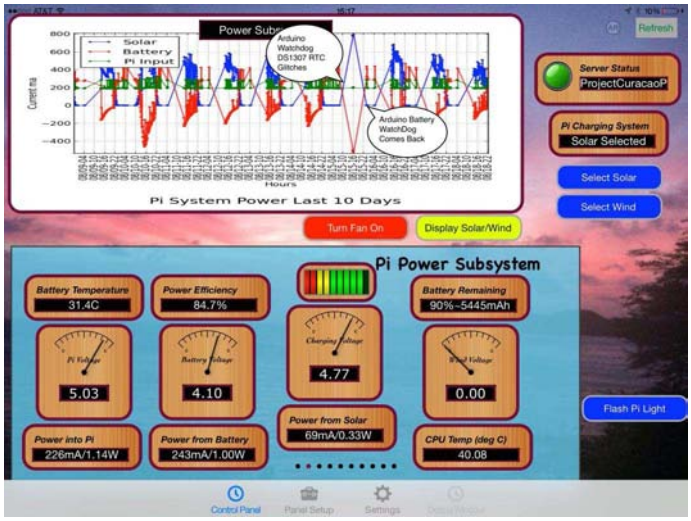
The box was deployed in mid-March 2014, and I didn't see the box again until September 30, 2014.

Overall, I was pretty pleased with the results. The box operated normally most of the time but would glitch about once a month when the DS1307 Real Time Clock would lose where it was and the Battery WatchDog Arduino would lock up and not know what time it was. This happened three times, roughly on a monthly basis.

Glitches in the Sun

A major glitch in the Project Curacao Solar Power control system happened in late August, detected by my trusty friend the RasPiConnect Control Panel. The Solar Power system was working very well providing full charges to the Raspberry Pi but monitoring and debugging the system from 3,500 miles away was not getting any easier. When I designed Project Curacao, I spent a lot of time thinking about what might happen and how I could

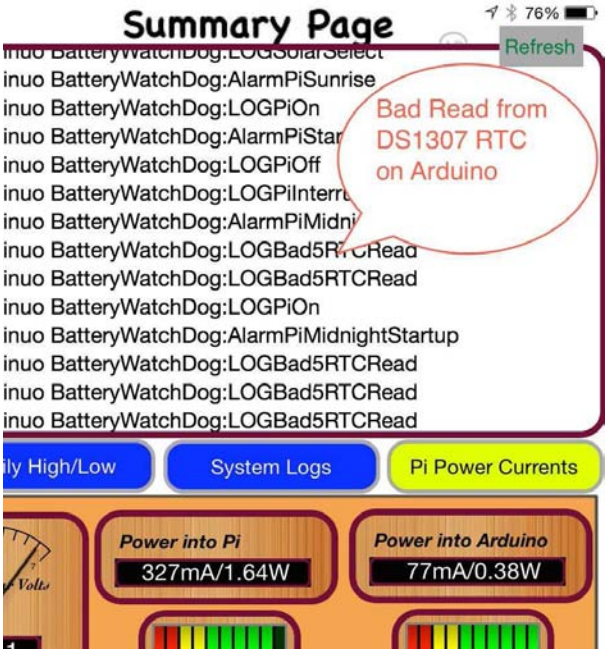
work around issues. So far, I seem to have made pretty good decisions, but a bad one I did make is coming back to haunt me. The issue is with the Battery WatchDog Arduino. It is supposed to run all the time and never need to be rebooted. The issue revolves around the DS1307 Real Time Clock connected via the I2C bus to the Arduino. The DS1307 is flaky. Because I sleep the Arduino about 70% of the time to save power, the Arduino Battery Watchdog is completely dependent on reading the DS1307 when it wakes up to set the time. I saw the DS1307 glitch a couple of times during development and just improved the filtering.



These screen shots are taken from the RasPiConnect (http://www.milocreek.com) based Project Curacao Control Panel.

The Arduino Project Curacao software (3000 lines) is robust and doesn't hang or bomb (known to me), but it is vulnerable to the DS1307 I2C Real Time Clock going south, which it appears to do about once a month. Heat? Electrical Noise? Unknown. I saw it do it a couple of times during design and added filtering to the Arduino power supply (a big capacitor from +5V to Ground) and it did not do it again. However, after six weeks in the hot sun, the DS1307 flaked out and lost the time again. It has happened three times in the last four months. Most recently, the Arduino recovered itself as can be seen in the log to the right where the DS1307 RTC fails and then recovers later. Sometime after this, the Arduino managed to reset to the default threshold values and time values for starting up and shutting down the Raspberry Pi, the temperature and voltage control thresholds. This meant that the Raspberry Pi stayed on a lot longer, hence the battery went lower.

These results and glitches made replacing the DS1307 a top priority. If you would like to see a set of benchmarks of various Real Time Clocks, check out http://www.switchdoc.com/2014/09/real-time-clock-comparison-results/. I really did my homework on what to replace the DS1307 with. I chose the temperature compensated DS3231.



The upgrade

We planned for the upgrade for months. We dug through the data coming back from the Raspberry Pi, stared at the RasPiConnect graphs looking for behaviour clues and worried about structural faults. We finally came to the conclusion that we would do the following four things:

- Add a new wind turbine and stiffen the rack as well as eliminating the pop-out problem (see Article 5 on Project Curacao in The MagPi issue 24).
- Add new weather instruments and a way to store data at night.
- Add more solar panels, but not a sun tracker (a new set of products and articles are coming out at http://www.switchdoc.com about sun trackers for the Arduino / Raspberry Pi in early 2015).
- And most importantly make the Arduino more reliable by replacing the DS1307 and adding an external WatchDog Timer.

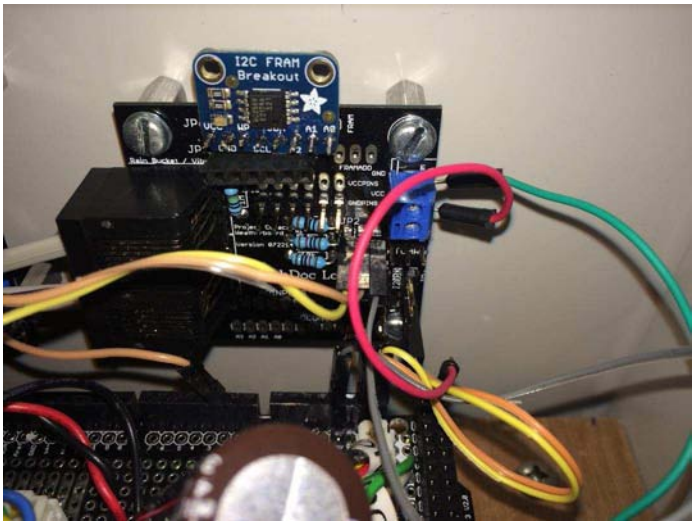
The wind turbine WeatherRack

After the great Wind Storm of March 2014 in Curacao (which really wasn't that much of a wind storm - 35 MPH, it popped the turbine out of the

tube and then the turbine destroyed itself completely), we decided that we had to make the improvements to have a longer lasting wind power turbine. The turbine only lasted one week in March 2014 after I left. Now it has been up for three weeks and we are keeping our fingers crossed. We replaced the wind turbine, added a cable to keep it from popping out, stiffened the mount and added wind speed, direction and rain measurement to Project Curacao. Below is a picture of the new turbine and WeatherRack.

Adding more weather sensors using WeatherPiArduino

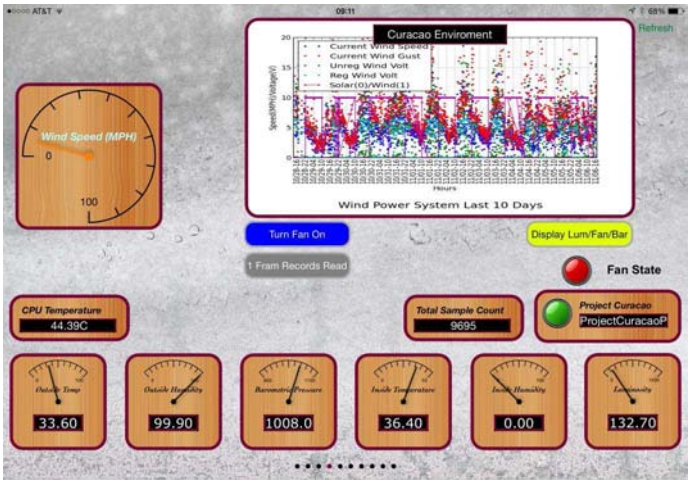
Putting more weather sensors in the box was problematic. Since building the box we have made the move of not building prototype boards by hand. We make PC Boards using the excellent PCB services of DFRobot and TinySine. Both of these guys rock. Fast and cheap with very good quality. Out of this weather sensor addition was born the WeatherPiArduino board (available at http://www.switchdoc.com) for interfacing between the Weather Meter from Sparkfun or Argent Data Systems and an Arduino or Raspberry Pi. The new version of the WeatherPiArduino just released to manufacturing adds an I2C temperature sensor / barometric pressure sensor and has the sockets to connect to the Weather Meter the neccessary pullups for the weather sensors and the 3.3V/5.0V I2C and GPIO level shifters. It also has three sockets on the board for the RTC / FRAM and ADC talked about in the next section. The installed WeatherPiArduino Board (sans the DS3231 since it covers up a bunch of good stuff) is below.



Adding more storage on the Arduino and the new DS3231 RTC

The WeatherPiArduino board has sockets for a DS3231 (our RTC of choice) from Amazon, an Adafruit 256Kbit FRAM for nighttime data storage and an Adafruit ADS1015 4 channel A/D converter (all I2C devices) that we ended up not using when we eliminated the vibration sensor due to lack of time to finish the design. We wrote an Arduino software library for this board and the sensors available at http://github.com/switchdoclabs/SDL_Weather_80422.

The DS3231 has performed flawlessly in three weeks of operation and the FRAM busily is storing weather information during the time that the Raspberry Pi is off and then transferred by the Arduino Battery WatchDog when the Pi wakes up in the morning. 24 hour coverage now of wind, rain and wind turbine performance. The wind turbine is still an underperformer but now we have a lot more data to prove that. RasPiConnect presents the 24/7 weather data coming to us from the Raspberry Pi.

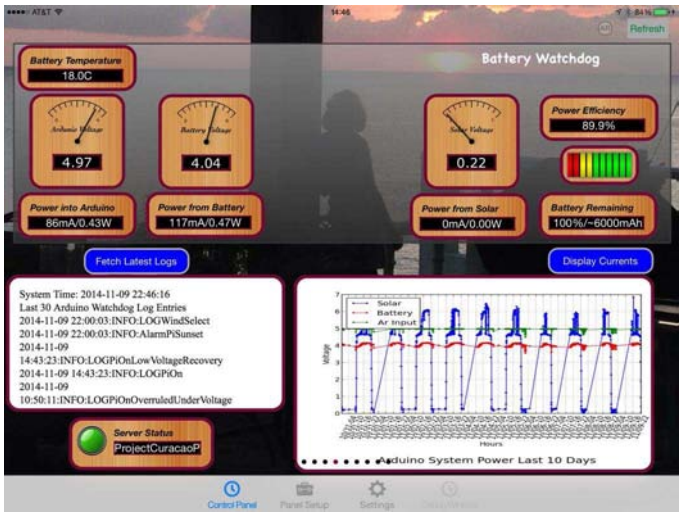


Adding more solar panels

First the good news. The Adafruit solar panels survived 6 months in the sun and rain in excellent condition. No trace lifting (a Curacao expert on solar cells, Brett Ruiz, told us that trace lifting and seal breaking is a big problem on the island) and they were perfectly clean. Our last test of solar panels and lights was 3 metres lower and 10 metres closer to the ocean. Those lasted less than three months. We made sure no one cleaned the panels during the six months. The picture of the 6 month old solar cells is below. Those are clouds reflecting in the panels, not dirt.

We had a power problem with the Arduino Battery Watchdog. On a very cloudy week, it would lose battery power and then lock up in some odd state and never come back up. We needed more power (although adding the external WatchDog timer also fixes the issue) so we added a new solar panel to the Arduino and added a new solar panel for the Raspberry Pi. About 70% more power for the Arduino and about 25% or so more for the Raspberry Pi. These numbers are less than you would expect because of the angles of the panels to the 12 degree tropical sun. Angle to the sun makes a huge difference:
http://www.switchdoc.com/2014/10/solar-power-raspberry-piarduino-new-results/

Now the power RasPiConnect screen looks really good. The graph below shows the voltage from the solar cells goes above 5V on a daily basis which indicates the battery is fully charged. The Raspberry Pi still does not get enough energy to run 24/7, but since the voltage goes over 5V on the Pi solar cells on most days, it's clear that the limitation is how much we can store in the battery, and not the cells themselves.



SwitchDoc.com is releasing a third generation improved solar cell charger and sun tracker called SunAir in the late fall of 2014. SunAir is the sum total of all our Project Curacao experience and really nails what is needed for this kind of system. Oh, it charges your phone too, which is helpful in Curacao where the mains power goes away sometimes.

The BIG reliability improvement - a WatchDog to watch the Arduino

Computers, like Project Curacao, sometimes lose their way. A power glitch, RFI (Radio Frequency Interference - curse of hanging on an amateur radio tower see http://www.switchdoc.com/2014/11/16-days-breeze-wind-power-raspberry-pi-arduino/), hanging peripherals, or just plain bad programming can cause your small computer to hang causing your application to fail. It happens all the time. How often do you have to reboot your PC? Not very often, but once in while your Mac or PC will freeze meaning you have to power cycle the computer. Raspberry Pi's will sometimes freeze because of a task not freeing up sockets or consuming other system resources. Arduinos sometimes freeze because of brownouts on the power line or a short power interruption or because of running out of system resources such as RAM and/or stack space, a very limited resource in an Arduino. Sometimes even programmers make mistakes.

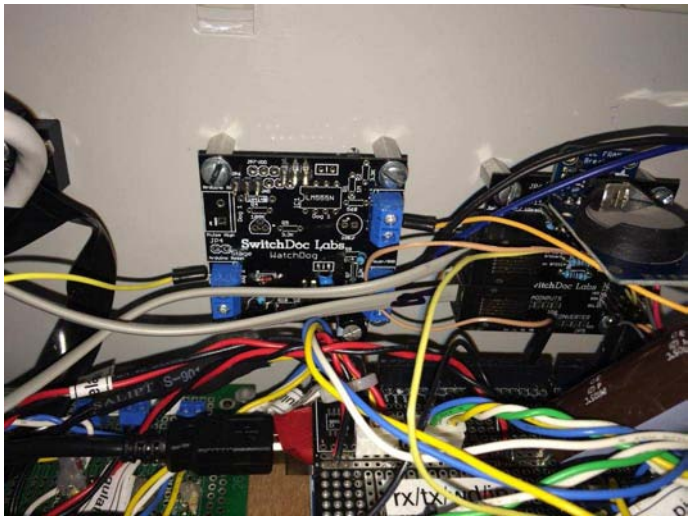
In small computers, you give your device the chance to recover from faults by using what is called a WatchDog Timer (WDT). A WDT is an

electronic timer that is used to detect and recover from computer malfunctions. If the computer fails to reset the timer (also called “patting the dog”) on the WDT before the end of the timer, the WDT signal is used to initiate either corrective actions or simply to reboot the computer.

We talked about a Deadman Switch in The MagPi article about the Arduino Battery WatchDog (The MagPi issue 20) and said that we would probably rue the day we did not stick an external WatchDog timer on the Arduino. Well, we did rue the day. We had failures from brownouts that hung the Arduino (not to mention the DS1307 problems). Does the Arduino have an internal WatchDog Timer? Yes, but it is disabled on some Arduinos and an internal timer does NOTHING for the kind of hangs that we are experiencing. It will help make your software more reliable, but does little in the case of a Solar/Wind powered system.

We ended up building a 555 timer based external Dual WatchDog Timer and building a board for Project Curacao. This is a 212 second WatchDog Timer that is compatible with the Arduino and Raspberry Pi. Here is version 1 of the board in Project Curacao. Only one side of the WatchDog board is populated for Project Curacao. We took what we learned with this board and made version 2.

Version 2 is professionally manufactured with a range of 30 to 240 seconds of timeout and with the right kind of connectors and diodes to make it work well with both the Arduino and the Raspberry Pi. The Dual WatchDog Timer product specification, theory of operation and the product is available on http://www.switchdoc.com.



The results of adding the WatchDog were invaluable during the 28MHz radio contest on the tower where the WeatherRack is hung. The Arduino rebooted on a regular basis (You could see it in the log) from the Radio Frequency Interference (RFI) coming from the close antennas. The cable from the box to the tower is about 15 meters which turns out to be very close to 3 times the wavelength of 28MHz, so we got substantial voltage on the lines. According to two of the radio gods, the princely Dr. Geoff Howard and the saintly Jeffery Maass (amateur radio experts in their own right), the voltage on my lines should be somewhere between 2 and 4 volts that close to the antennas which could seriously interfere (and did) with Project Curacao. Shortly after the end of the contest, all problems went away and the box has been working perfectly for 14 days with no Arduino reboots at all. We haven't had a brownout yet, but we did test that while we were down there by running the battery down by hand. Below is the box in it's final resting place with the WeatherRack on the tower in the background.

You can see the live hourly data and picture from Project Curacao at:
http://milocreek.com/projectcuracaographs.

What's Next?

At this point, we are planning to visit the box again in June of 2015. We will analyze the logs and the corrosion of the parts and solar cells, and carry the box gently home in our luggage. What are we going to do next with the box? Maybe bury it with a copy of all the MagPi articles written about it or more likely use all the parts in new projects! Or maybe write one more update if the powers that be would like that. This has been a large project (started in June of 2013) and we want to thank all of the people and companies that helped make this possible. More on Project Curacao at www.switchdoc.com.

More discussion on Project Curacao at: http://www.switchdoc.com

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.

Serial Pi

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

ADC Pi

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

1 Wire Pi

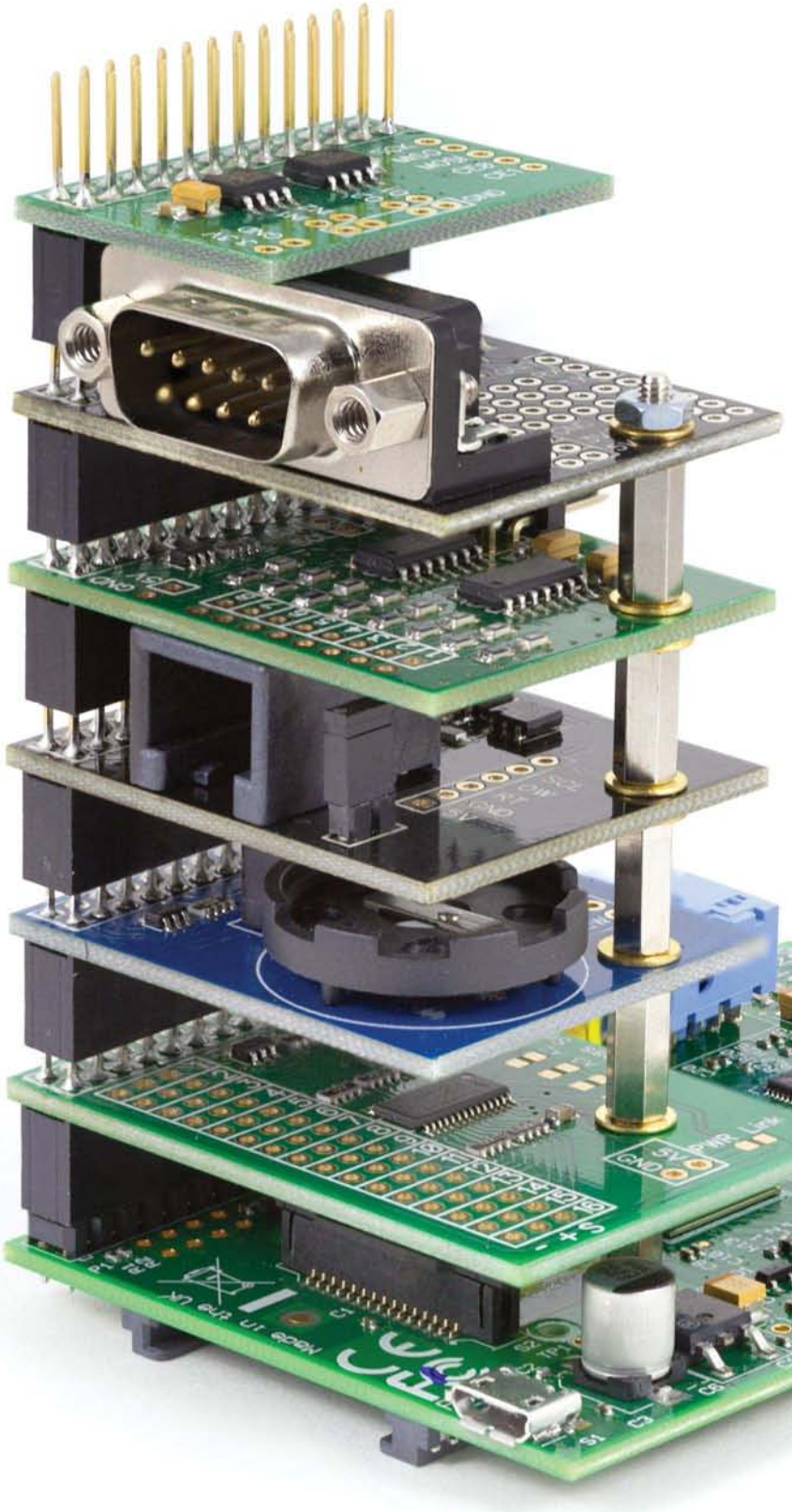
1-Wire® to I²C host interface with ESD protection diode.

RTC Pi

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

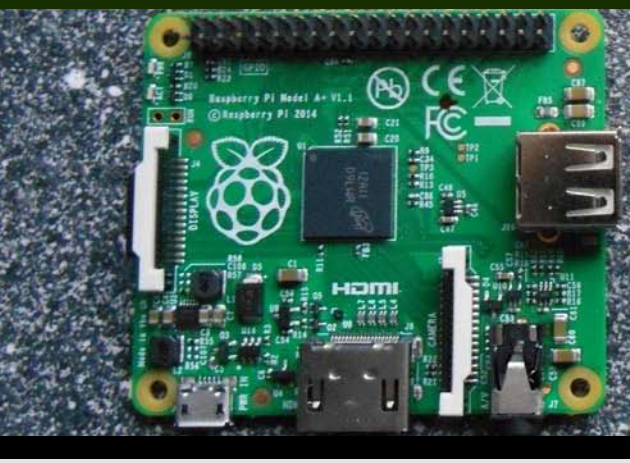
Servo Pi

16-channel, 12-bit PWM controller suitable for driving LEDs and radio control servos.



AB electronics UK

www.abelectronics.co.uk



NEW MODEL A+
Ready for embedded applications

Dougie Lawson

MagPi Writer

Introducing the latest Raspberry Pi hardware

After much speculation, the Raspberry Pi Foundation officially released the Model A+ Raspberry Pi on the 10th of November. The Model A+ is an upgraded version of the Model A, which is intended for embedded applications. Similar to the Model A, the Model A+ has 256MBytes of memory on top of the SoC (System on Chip) and a single USB socket. The Model A+ mirrors the other Model B+ interfaces, including the longer GPIO connector and combined audio and analogue video jack. The Model A+ is only 65mm long and weighs just 23g, which is a significant reduction over the Model B+ length 85mm and weight 41g and the previous Model A. The processor is the standard Broadcom BCM2835 SoC that is present on all current and previous Raspberry Pi products.

Pricing

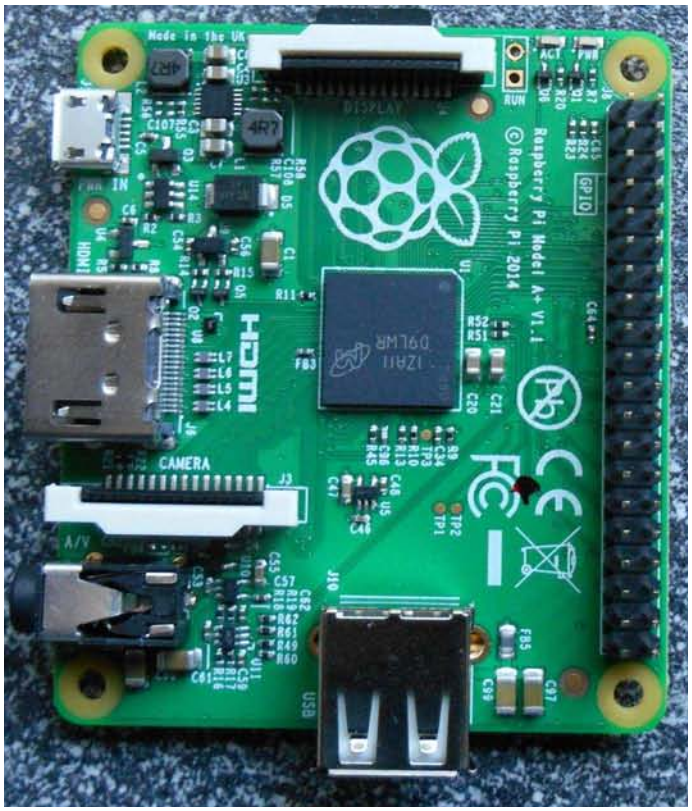
The Model A+ is a powerful computer that is 20mm smaller than the length of a credit card and has a retail price of only $20 (excluding taxes and shipping). This is $5 cheaper than the previous Model A Raspberry Pi.

Layout

The first thing that stands out about the Model A+ are the mounting holes. They have the same spacing as the mounting holes on the B+, such that HAT (Hardware Attached on Top) boards can be securely fitted. The GPIO connector is located along the same edge as the Model B+ and also has 40-pins. The left hand 26-pins are associated with the same connections as the 26-pin header on the Model A and version 2 Model B Raspberry Pis.

Connections

The top of the Model A+ is shown below. The 40-pin GPIO header is on the right-hand side, running down



the full length of the board. The USB port is located at the bottom of the board. On the left-hand, there is the micro-USB power connector at the top, the HDMI connector in the middle and the combined 3.5mm audio and analogue video jack at the bottom.

The DSI (display interface) and CSI (camera interface) are on the top of the board and are labelled as DISPLAY and CAMERA respectively.

Turning the board over there are some components and a lot of test points. The test points are used for

quality control during manufacture. Similar to the Model B+, the A+ has a microSD card slot. Unlike the SD card on the Model A, the microSD only requires a little extra space around the edge of the board.

So all in all the A+ is a superb tiny computer. It has a lot of processing power in an incredibly small package. It is going to be the Raspberry Pi of choice for anyone who needs the lowest weight and smallest footprint (and it fits in an Altoids tin). As usual with each of the new Raspberry Pis; Alex Eames (http://raspi.tv) has done some power usage measurements. (Alex's complete results can be found on his website.) Alex has run a sequence of four tests on a Model B, Model B+ and the new Model A+. The conclusion of these tests is that the Model A+ uses 70% less power than the old Model A. This implies that those interested in launching their Raspberry Pi to 40km under a helium balloon not only have a lower payload mass, but will get more run-time before their batteries go flat. It also means that I can run my Model A+ from mains electricity for a whole year for less than one pound. I'm sure we'll see lots of robot projects that are based around the Model A+, rather than the other models.

Software

The NOOBS, NOOBS lite and the other OS images from http://www.raspberrypi.org/download will all work on a Model A+. The process of writing an image to the microSD card is the same as for an SD card. The only requirement is that the card writer should have a slot for a microSD card. For this review, the Raspbian image was used to test the network configurations and various extension boards.

Networking

Similar to the Model A, the Model A+ does not have a RJ45 Internet connection. Therefore, networking can be provided either using a USB WiFi dongle, USB wired Ethernet dongle or ENC28J60 SPI Ethernet board.

To configure the Model A+ and initialise the networking, a USB hub is needed to connect a keyboard and mouse. Once the A+ has been deployed within a robot or similar remote autonomous project, the USB is probably not going to be needed. Therefore, only having one USB port reduces the size and power consumption of the A+.

For the average model A+ user, I recommend purchasing either a Tenda W311M Wireless-N150 USB WiFi dongle (RT5370 chipset) or an Edimax EW-7811Un dongle (rtl8188cus chipset). The drivers for both of these dongles are already present within the Raspbian distribution. Remember that without a working network, it is more difficult to install a special driver that has to be downloaded from the Internet.

Checking the Model type

Linux provides access to system information via the /proc virtual file system. The CPU type can be printed by typing `cat /proc/cpuinfo`
For a Model A+, this will return something similar to:

```
processor       : 0
model name  : ARMv6-compatible processor rev 7 (v6l)
Features        : swp half thumb fastmult vfp edsp
java tls
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xb76
CPU revision    : 7

Hardware        : BCM2708
Revision        : 0012
Serial          : 000000005b592f7f
```

where the `Revision` number is used to store the Model information and a `Revision` number of `0x0012` corresponds to a Model A+. (The Serial number is unique for each Raspberry Pi.)

Testing peripherals

There are a lot of Raspberry Pi peripherals on the market. Many of these add-on boards are designed to fit the Model B and may rely on the placements of the Model B components or the 26-pin header dimensions to allow direct connection. However, with some adapters or slight additions of supporting material it is normally possible to get peripherals running as normal.

The first peripheral that I tested was a Fish Dish. (A description of the Fish Dish is given in Issue 25 of The MagPi.) The Fish Dish can be easily connected to the A+ and fits the GPIO pins with the edge of the Fish Dish overhanging the new pins on the A+.

The ModMyPi 8X8 matrix was next in my test list. The matrix board fits, but is a little wobbly. This can be made more secure with some insulated packing.

Next up was my PiFaceCAD. This board is longer than the A+, and the end of the PiFace board hangs over the USB plug. The PiFaceCAD worked as expected and could be secured with a bit more tape.

Following on from my Arduberry article in Issue 28 of The MagPi, it made sense to run a test on it too. It had the same wobble, but was easily connected and ran without any problems.

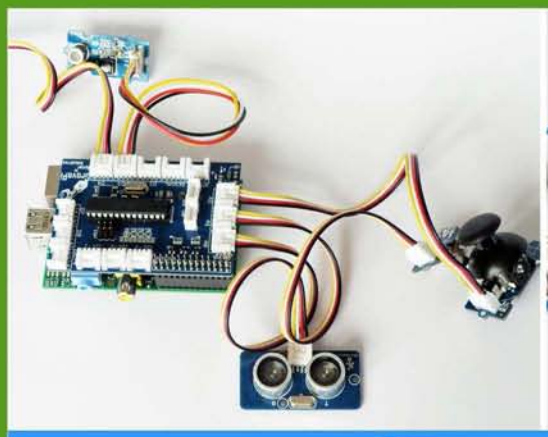
The Wolfson audio board was designed to run on a Model A or a second generation Model B Raspberry Pi, since it is dependent on the P5 header that is not present on the Model A+ or B+ Raspberry Pis. I will continue to use the audio board with my Model B.

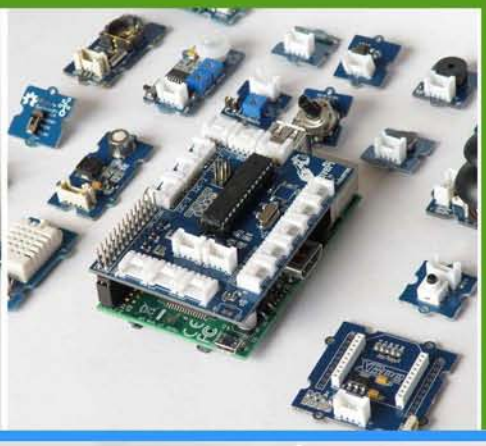
The final peripheral to be tested was my Quick2Wire GPIO expander board. The GPIO expander is designed with a 26-pin female connector on the end of a ribbon cable that should be connected to a 26-pin GPIO header. A 26-pin female connector cannot be directly connected to a 40-pin header, such as present on the Model A+ or B+, since the additional pins interfere with the connector. The easiest way to solve this problem is to make a new ribbon cable that has a 40-pin IDC connector on one end and a 26-pin IDC connector on the other. Making ribbon cables with IDC connectors is covered in Issue 3 of The MagPi.

The community has been adapting to the presence of the combined audio and analogue jack on the Model B+, since there is no common standard for connections of this sort to a four pole 3.5mm jack. The four pole connector on the Model B+ is the same as on the A+. Thankfully, Matt Hawkins has provided an excellent write up at http://www.raspberrypi-spy.co.uk/2014/07/raspberry-pi-model-b-3-5mm-audiovideo-jack/ on this subject.

Summary

The Model A+ has lots of similarities to the older Model A, but more similarities to the newer Model B+. The primary new features are the 40-pin header that provides access to extra GPIO pins and the second I^2C interface, the whole package is 20mm shorter than the A, B or B+ boards and weighs less than Models A, B and B+. The Model A+ is perfect for projects that require the lowest power consumption and smallest form factor.



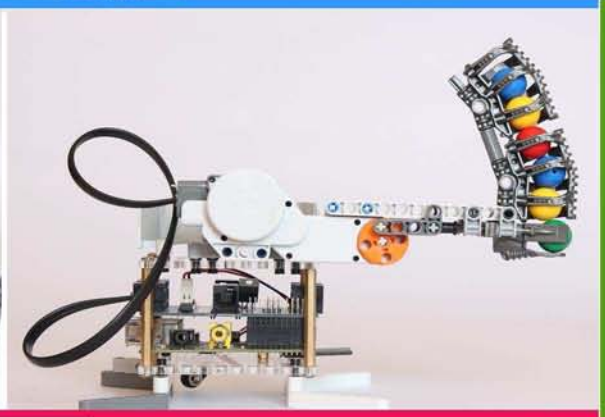


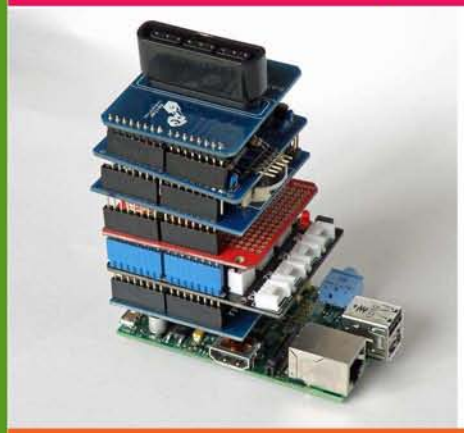
GrovePi

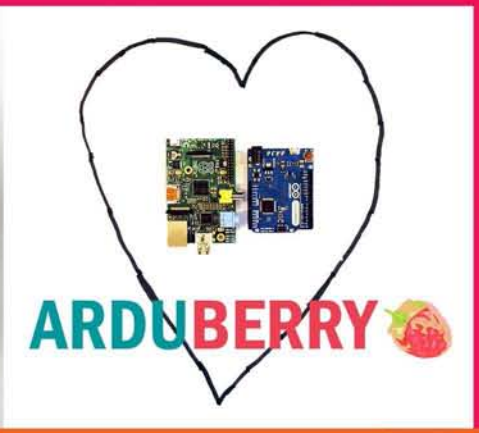
Connect Hundreds of Sensors to your Raspberry Pi

BrickPi

Turn your Raspberry Pi into a LEGO© Robot





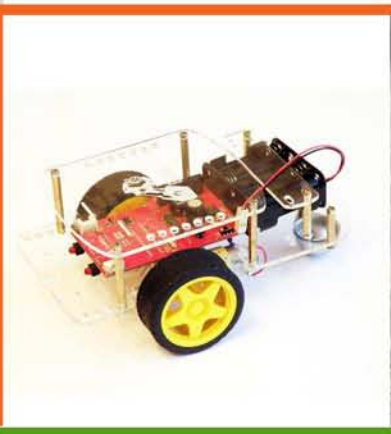


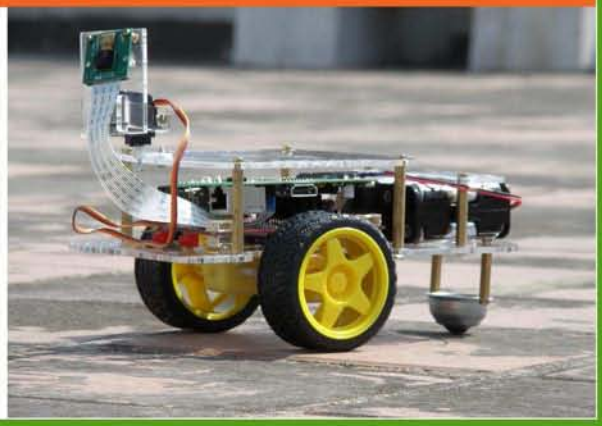
Arduberry

Unite the Raspberry Pi and Arduino

GoPiGo

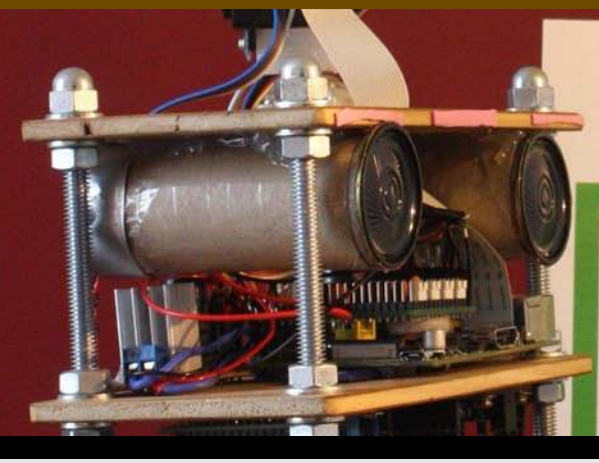
Turn Your Raspberry Pi into a Robot





dexterindustries.com

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



Computer Vision on the Raspberry Pi - Part 2

Derek Campbell

Guest Writer

SKILL LEVEL : INTERMEDIATE

Last time we saw how PiTeR, the terrestrial robot, can identify symbols he sees in his environment, even when the symbols are at an angle to the ideal symbol being used as comparison - as might happen if the robot arrived at the symbol slightly off-axis from it.

This article does not cover any new uses of OpenCV, but instead shows how what we got working last time is integrated into PiTeR's control programs to allow him to navigate autonomously using the information provided by the OpenCV algorithms.

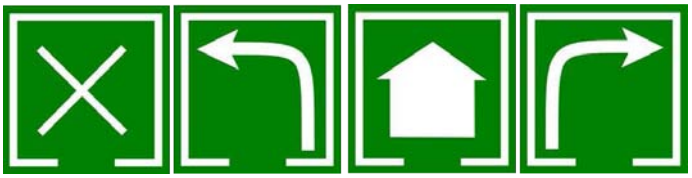
Take me for a drive

We want to enhance our example to distinguish between the different symbols and act on them. PiTeR needs to turn and drive based on what he is seeing. Because the actions and control logic are different systems, we will deal with turning and driving separately.

PiTeR balances on two wheels. How he does this will be covered in a later article, but you can find out more now on his website https://github.com/Guzunty/Pi/wiki/PiTeR-and-OpenCV. He drives forward or backward when both wheels turn in the same direction and at the same speed. When the wheels rotate at different speeds, he turns. If he is standing still and the wheels rotate in opposite directions, PiTeR turns on his own axis.

Now let's think about what we want to get PiTeR to do. We want him to look in his visual field for a symbol of a particular colour. What we want to do next depends on how far away from the symbol he is. If he is far from the symbol, we want him to drive towards it. If he is close we want him to recognise the symbol and act on it.

You may recall from the last article PiTeR recognises the following symbols:

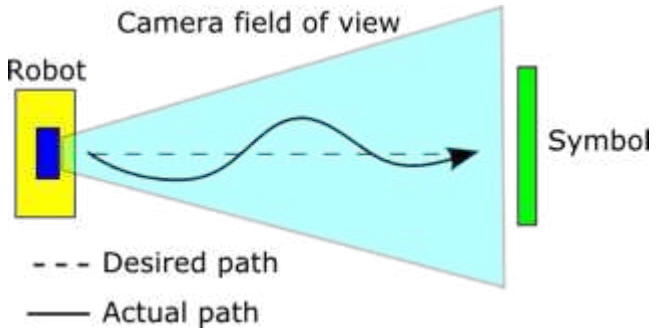


Robot tracking symbols:
Turn Back, Turn Left, Home and Turn Right.

It is clear what we want to do for the "Turn" symbols, but what about "Home" and "Turn Back"? In the case of "Turn Back", we want him to turn around and go back the way he came. In the case of "Home", he is at the end of the trail and has reached his goal. When this happens we will make him do a little victory dance!

First, we need to decide if we are near or far from the symbol. This is easy to do. From last time you will recall that we get a rectangle locating the symbol patch in PiTeR's visual field. All we need to do is to compute the area of the rectangle. If it is less than a certain size we will drive forward, if it is more, we will stand still and figure out which symbol it is.

While we are driving, we will look at the position of the patch in the camera's field of view. We will input small turn commands so that the symbol is kept as close to the centre of our view as possible. In this way, PiTeR will correct himself as he drives, always moving towards the symbol.



Once we are close enough, we will use the 'good match' (`knnMatch`) feature in OpenCV, as we discussed last time. The symbol with the greatest number of good matches will be the one we act on.

Threading

Before we go off and start work on this, there are a couple more things to think about. Once we start driving around, we cannot go off into a loop and forget about everything else. PiTeR constantly exchanges information with the person controlling him; for example, they might decide to intervene while he is doing his automated actions. If he starts to ignore his human, he will be branded a rogue robot!

He also needs to exchange information with his wheel control system in order to drive and, equally important, to know how far he has gone and slow down when necessary.

To support these needs, PiTeR needs to execute his long running algorithms, like symbol and colour patch detection, independently of the control loop. Only when the image processing is complete do we look at the results in the main drive loop. This keeps PiTeR responsive to human control.

One of the things that is so great about a longer term project like a robot is that you never know where it is going to take you. To allow PiTeR to continue controlling things and do vision processing, we need to learn about Python threads. To create a new thread, we create a Python class which utilises the `Thread` class of the `threading` package:

```
class SymbolFinder(threading.Thread):
```

To use threading, we need to define two additional methods,

```
def __init__(self):
  super(SymbolFinder, self).__init__()
  # rest of our initialisation here
```

and

```
def run(self):
  # loop repeating the colour patch search
```

The `__init__` method is special. It is called a constructor. When you create an instance of `SymbolFinder`, the `__init__` method gets called for you automatically. To use threading, we must call the `Thread` constructor in our own constructor with the `super` command.

Now, before entering the main loop, we create an instance of `SymbolFinder` and call the `start` method, like this:

```
symFinder = symbolFinder.SymbolFinder()
symFinder.start()
```

The `start` method is inherited from the `Thread` class. It returns almost immediately, but by the time it does, whatever we put in the main loop will be running independently on its own

operating system thread. Cool, we are now doing two things at once, albeit a little more slowly than if we were doing each alone.

PiTeR also uses the same technique for scheduling actions like speech, LED illumination and autonomous driving. By using separate threads, PiTeR appears to be able to do several things at once, which is what we would expect of a good robot.

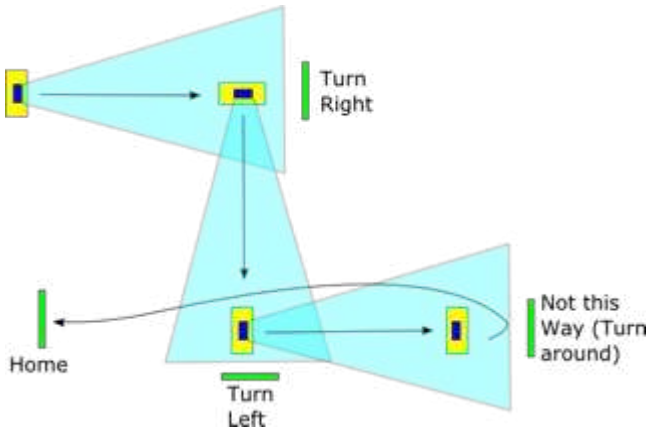
Which symbol did we find?

The next thing we need to enhance is symbol recognition. Our previous script read just one example symbol and looked for a match with it. We now need to recognise four different symbols. We need to adjust the script to read all four symbols and compute the key image features for each one. As before, we will do this at start up time because it only needs to be done once for each reference image.

We will also need to adjust our matching algorithm to loop over the four sample symbols and compute the good matches for each one.

The script looks like this:

```
# Repeat
#   Find a patch
#   If the patch is too small:
#     drive towards it.
#     if the patch is not in the centre, turn
while driving to bring it towards the centre
#   else:
#     crop the image to the size of the patch
(actually we will crop a bit bigger than this)
#     loop for each symbol
#       compute the good matches for the
symbol
#       remember the symbol with the most good
matches
#     if the matched symbol is a 'Turn Left'
or 'Turn Right' command
#       turn left or right 90 degrees
#     else if the matched symbol is 'Turn
Back'
#       turn through 180 degrees
#     else if the matched symbol is ‘Home’
#       do a dance and exit the outer loop
```



In all cases except the "Home" case, PiTeR will find the next patch in the trail. It will be further away, so he will drive to it and the whole outer loop will be repeated until he reaches the "Home" symbol. Yay, PiTeR can do cub scout trail tracking!

More computer vision tricks

Following symbols is not the only thing PiTeR can do. OpenCV also has facilities for detecting and recognising faces. To do this we use the OpenCV `CascadeClassifier` class. This can be programmed to look for the classic shape of a human face. This is the same method that modern cameras use to choose where to focus. If one is found in the field of view, we command PiTeR's head servos to try to centre the face in the field of view. This gives him the ability to follow you as you move around.

If more than one face is found, we choose a face at random and centre that. This gives the uncanny impression that PiTeR is looking at the different people in a crowd (and in a way, I suppose he is).

Coming up...

We can take this one step further and have PiTeR say "Hello!" to individual people he recognises. To do this, we need to train him with still photos of the people we want him to recognise. We will discuss face recognition in the next article.

Don't just wait for Santa...

Impress him!

Build a Christmas DIY project

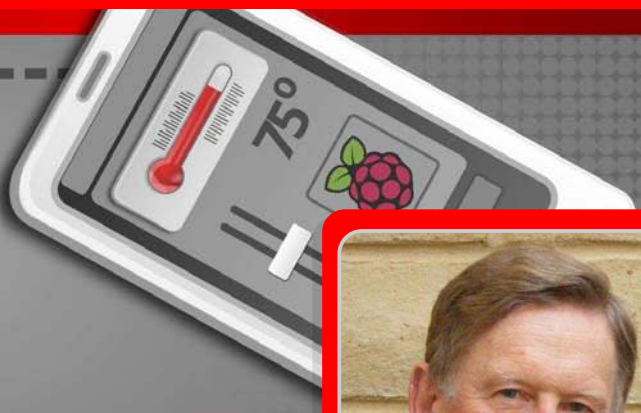
with Wyliodrin and get your present!

for more details go to

www.wyliodrin.com/XmasHack

www.wyliodrin.com

python C++ node.js Perl Java ARDUINO php



Logging temperature with 1-wire sensor

David Bannon

MagPi Writer

SKILL LEVEL : BEGINNER

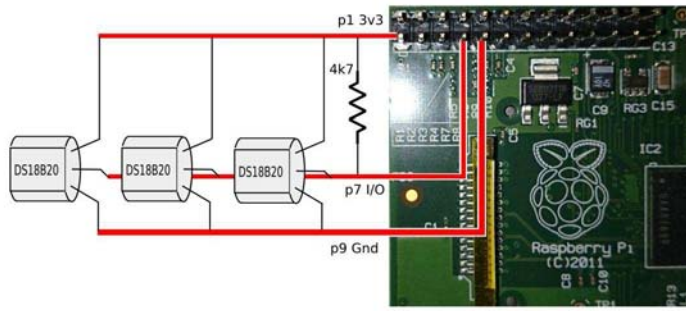
The Raspberry Pi is particularly suited to logging data and temperature is a very popular thing to log! This article describes how to measure temperature using the DS18B20 1-wire digital sensor from Dallas Semiconductor. The DS18B20 sensor is widely available from many different suppliers, as a component or packaged within a sealed unit, e.g.: http://www.adafruit.com/product/381
The sensor requires a 4.7kOhmn resistor to connect it to the Raspberry Pi.

The DS18B20 sensor is relatively cheap and is supported by the Linux kernel present in the current Raspbian distribution for the Raspberry Pi. The data recorded from it is available directly as a temperature value. Without additional calibration, it is sufficiently accurate for most applications. It can measure temperature from -55 to 125degC, with a resolution of 9 to 12 bits implying an accuracy of 0.5degC over much of the range.

If you are happy with a soldering iron, making your own sensors using the TO92 packages is easy, cut the DS18B20's leads so there is only about 5mm remaining, solder a lead to each, mix up some epoxy and fill the volume around your solder joints. A sensor made like this is smaller and possibly faster to respond that the more robust commercial ones.

The real beauty of the DS18B20 sensor is that it is very easy to use on the Raspberry Pi! While it is called a V1-wireWinterface, in practice it needs two wires and over long runs three wires are more reliable. The three wires correspond to a ground or common connection, a 3V3 supply and the signal output from the sensor. To make life easier, it is possible to connect many sensors in parallel using the three connection wires. Therefore, an array of DS18B20 sensors uses very few GPIO connection pins. It is possible to use long cables with the sensors and each sensor draws very little current, adding only a small additional loading to the Raspberry Pi.

The signal wire is used to both to read and write to the sensors, where each sensor has its own unique identifier (ID) number. The `w1-gpio` and `w1-therm` Linux kernel modules hide all the complexity and are pre-installed in both Raspbian and Arch Linux for the Raspberry Pi.



Read the data sheet of the DS18B20 carefully, checking which wire is which. Then connect the 3V3 volt input wire to pin 1 on the Raspberry Pi GPIO. If in doubt, refer to http://elinux.org/RPi_Low-level_peripherals for the GPIO pin mapping. Connect the signal to pin 7 and ground to pin 9. Make sure that a 4.7kOhm is connected as shown in the diagram above. The connection can be achieved using a breadboard or by soldering the component to the other wires. Then power up the Raspberry Pi and open a LXTerminal. The Linux kernel modules are not loaded by default. To load them type:

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
```

If there are no error messages, then the modules should have been loaded successfully. When the kernel modules are loaded they create directories in a virtual file system, where there is one instance for each connected sensor (and one for a master). For example, typing `ls -L /sys/bus/w1/devices`

```
28-000004749871  28-001414af48ff
28-0014153fc6ff  w1_bus_master1
```

In this example, three sensors were connected. If you see only the master and not the serially numbered sensor directories, check your wiring. The temperature value can be printed by typing:

```
cat /sys/bus/w1/devices/28-000004749871/
w1_slave
```

which returned a temperature of 22.062degC.

```
1 01 4b 46 7f ff 0f 10 02 t=22062
```

The sensor values can also be read using a simple Python program:

```
#!/usr/bin/env python
import os
dev_dir = '/sys/bus/w1/devices/'
devices = os.listdir(dev_dir)
devices.remove('w1_bus_master1')
for dev in devices:
  f = open(dev_dir + dev + '/w1_slave', 'r')
  content = f.readlines()
  temp_start = content[1].find('t=')
  print content[1][temp_start + 2:] + dev
  f.close()
```

Make this program executable by typing chmod 755 `test.py` . Then run the program by typing `./test.py` . The program prints the temperature readings in milli-degC.

The files in the virtual file system are created and maintained by the Linux kernel modules and may not exist when the Linux installation is being updated or if the modules have not been loaded. Therefore, there is a more complete Python program on the next page. The program initialises, identifies connected sensors and displays temperature readings. To continue to read the temperature values, the `read_temps()` function should be called many times within a loop.

The program contains three functions, such that pieces can easily be used for other programs. The `load_modules()` function checks to see if the kernel modules have been loaded and attempts to load them if necessary.

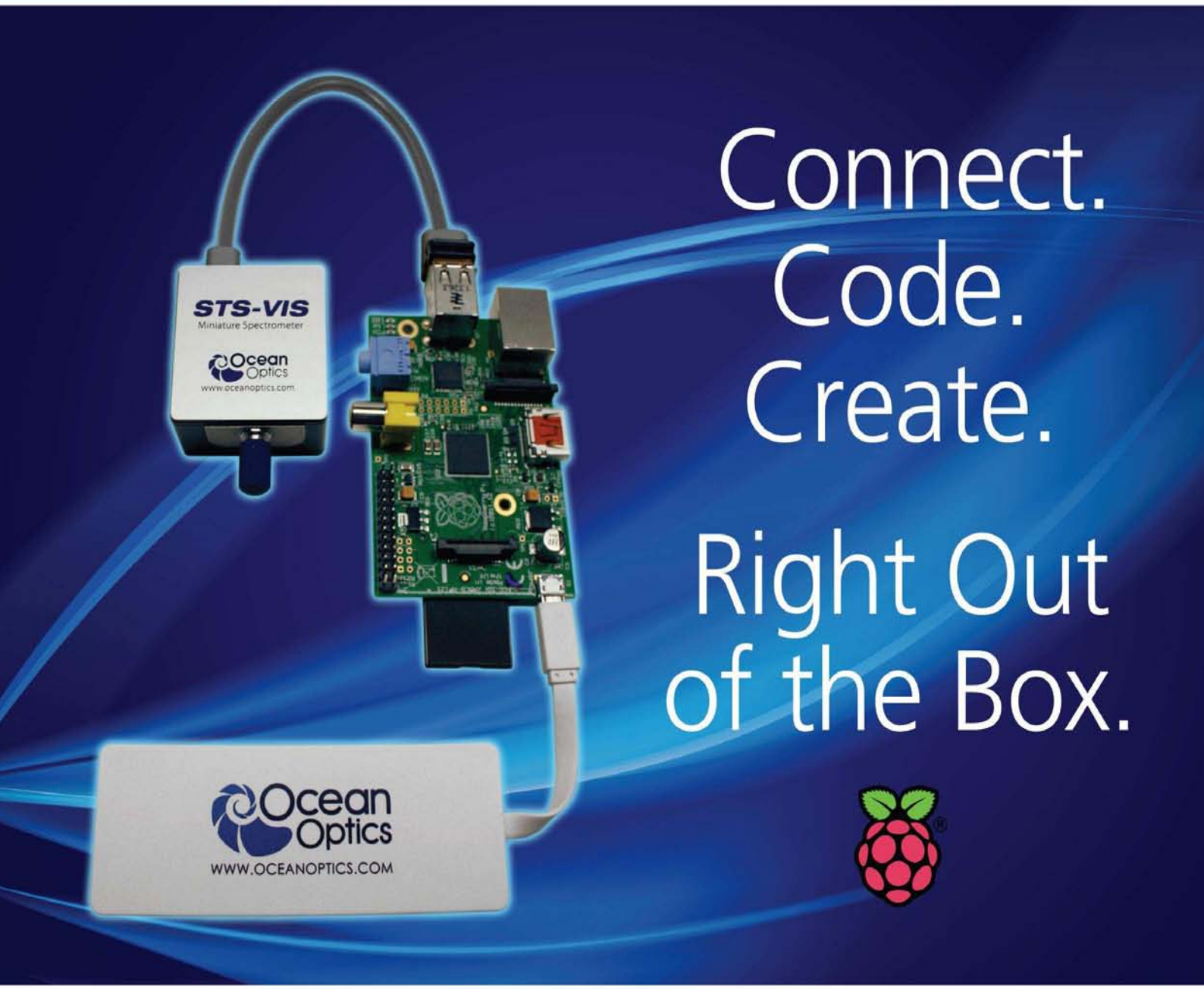
The `read_temp_lines()` function accesses the virtual files that are associated with the temperature sensors and returns the values stored in the files. Finally the `read_temps()` function parses the text output and returns the temperature values.

The `read_temps()` function can be called when the kernel is updating its sensor file. If this happens it waits for a tenth of a second and tries again. The function can also cope with a situation where a sensor has been unplugged.

Save the program as `temperature.py`, make it executable and run it by typing:

```
sudo ./temperature.py
```

```
#!/usr/bin/env python
import os, time, sys
dev_dir = '/sys/bus/w1/devices/'

def load_modules():
  first_time = 0
  if os.getuid() == 0:    # Only root can load the mods
    if os.system('modprobe --first-time -q w1-gpio') == 0:
      first_time = 1
    if os.system('modprobe --first-time -q w1-therm') == 0:
      first_time = 1
    if first_time:     # wait a bit for the devs to be populated
      time.sleep(5)
    return first_time
  else:
    if os.system('modprobe -q w1_gpio') == 256 or os.system('modprobe -q w1_therm') == 256:
      print 'sorry, modules not loaded and we are not root'
      sys.exit(1)


def read_temp_lines(dev):
  # Note that the read process is slow, bit less than
  # a second per device.
  try:
    f = open(dev_dir + dev + '/w1_slave', 'r')
    content = f.readlines()
    f.close()
  except IOError:
    lines = ['no file']
  return content

def read_temps():
  # Build a tuple with data from each device we know
  # about, set the ones that are not present to 0.0
  results = []
  for dev in devices:
    content = read_temp_lines(dev)
    if content == ['no file']:
      results.append(0.0)
      continue
    while content[0].find('YES') == -1:
      # try again if dev was not ready
      time.sleep(0.1)
      content = read_temp_lines(dev)
    temp_start = content[1].find('t=')
    if temp_start != -1:
      temp_string = content[1][temp_start+2:]
      results.append(float(temp_string) / 1000.0)
  return results

load_modules()
devices = os.listdir(dev_dir)
devices.remove('w1_bus_master1')
print read_temps()
```



Connect. Code. Create. Right Out of the Box.

Powered by Raspberry Pi, the STS Developers Kit is the best way to integrate spectral sensing into your application.

Measure color more accurately than the human eye, monitor solar UV levels, prototype compact medical self diagnostic tools or monitor crops from a UAV. The STS Developers Kit makes it easy with integrated device drivers and an easy to access API to get going quickly.

Ocean Optics

www.oceanoptics.com | info@oceanoptics.com | **US** +1 727-733-2447 **EUROPE** +31 26-3190500 **ASIA** +86 21-6295-6600

Part 5: Using FUZE BASIC to control a robot arm

Jon Silvera

Guest Writer

SKILL LEVEL : BEGINNER

In this article I am going to show you how to use FUZE BASIC to control a robot arm. The robot arm that we are using is available in the UK from Maplin Electronics (Code: N35DU or A37JN). Worldwide it is known as the OWI robotic arm. Don't forget you need the USB version. The robot arm is also bundled with the FUZE keyboard in kit T2-R and the new BBC Micro styled T2-SE-R kit. For more details visit http://www.fuze.co.uk.

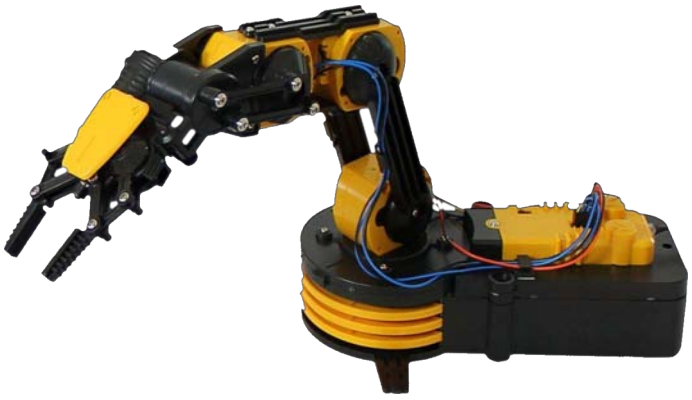
Before you attempt this project I am going to assume that you have followed the FUZE BASIC articles in issues 26, 27 and 28 of The MagPi. If not I recommend you take the time to read these articles and get familiar with FUZE BASIC.

"Wir sind die Roboter"

Please setup your Raspberry Pi and connect the robot arm to one of the available USB ports. It is best to connect the robot arm before running FUZE BASIC and make sure it is also switched on.

Double click the FUZE BASIC icon to begin. As you will have come to expect, FUZE BASIC will leap into action and present you with the Ready> prompt.

First of all straighten the robot arm so it is not all folded up. Do not worry if the arm clicks here and there. This is just the gears clicking and nothing actually breaking! Your robot arm should look something like the picture below.

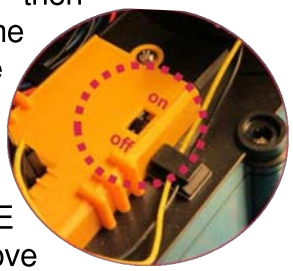


Warning: The following commands will set the robot arm moving as soon as you press <Enter>. If you do not type the next command the arm will go as far as it can and start clicking - you should enter the `ArmBody (0)` command to stop it.

Type in the following commands, pressing <Enter> after each one:

```
ArmBody (1)
ArmBody (-1)
ArmBody (0)
```

If at this point you get an error stating “Unable to find Robot Arm” or similar then exit FUZE BASIC with the `Exit` command. Unplug the robot arm and reconnect it again. Also please make sure that the robot arm is switched on. Start FUZE BASIC and try the above again. If at this point it still does not work, seek help from an adult - or if you are an adult then seek help from a child!



The first known use of the word Robot comes from the Czech Čapek brothers and was used in Karel Čapek’s science fiction play “Rossum's Universal Robots”. The original meaning of the Czech word "Robota" is ”drudgery” or “slave labour”.

Assuming everything worked correctly, did you notice something? Yep, FUZE BASIC has support for the OWI/Maplin robot arm built into the language!

Now try these other control commands. In each case x can be either -1, 0 or 1. The exception is `ArmLight (x)` where x can only be 0 or 1. Try out the different values for x and see what happens:

```
ArmShoulder (x)
ArmElbow (x)
ArmWrist (x)
ArmGripper (x)
ArmLight (x)
```

A useful trick to know at this point is that you can repeat the last command by pressing the <Up> arrow key and then just edit the number. Remember, you still need to press <Enter>.

Time to write some code

Let’s put some of this new found knowledge into action. Press <F2> to enter the FUZE Editor. If there is another program listed, then make sure it is not needed and then press <F12> to clear it. Enter the following lines of code:

```
CLS
PROC ResetArm
END

DEF PROC ResetArm
  ArmBody (0)
  ArmShoulder (0)
  ArmElbow (0)
  ArmWrist (0)
  ArmGripper (0)
  ArmLight (0)
ENDPROC
```

Press <F3> to run the program. You will be prompted for a file name. Name it something like “RobotArm”, for example.

The purpose of this code is to make sure that the robot arm can be instructed to switch everything off, so absolutely nothing will happen when you run the program, but we will use this code a lot later.

Edit the program to add the following code at the end:

```
DEF PROC DisplayInstructions
  CLS
  FONTSCALE (2, 2)
  INK = Red
  PRINT “We are the ROBOTS!”
  INK = White
  HVTAB (0,2)
  PRINT “Press”
  PRINT
  PRINT “1 or 2 for Body left & right”
  PRINT “3 or 4 for Shoulder up & down”
  PRINT “5 or 6 for Elbow up & down”
  PRINT “7 or 8 for Wrist up & down”
  PRINT “9 or 0 for Gripper open & close”
  PRINT “Enter to turn the Robot light on”
  INK = Red
  PRINT
  PRINT “Space to stop movement & switch light
off”
ENDPROC
```

We also need to add a call to PROC DisplayInstructions. Add this to the start of the program. The grey text is what you should already have:

```
CLS
PROC ResetArm
PROC DisplayInstructions
END
```

You should have something similar to the picture below.

```
PROC ResetArm
PROC DisplayInstructions
END

DEF PROC ResetArm
ArmBody (0)
ArmShoulder (0)
ArmElbow (0)
ArmWrist (0)
ArmGripper (0)
ArmLight (0)
ENDPROC

DEF PROC DisplayInstructions
CLS
fontScale (2, 2)
INK = Red
PRINT "We are the ROBOTS!"
INK = White
hvTab (0, 2)
PRINT "Press"
PRINT
PRINT "1 or 2 for Body left & right"
PRINT "3 or 4 for Shoulder up & down"
PRINT "5 or 6 for Elbow up & down"
PRINT "7 or 8 for Wrist up & down"
PRINT "9 or 0 for Gripper open & close"
PRINT "Enter to turn the Robot light on"
INK = Red
PRINT
PRINT "Space to stop movement & switch light off"
ENDPROC
```

When you press <F3> to run the program, you should see something like the following picture.

```
We are the ROBOTS!

Press

1 or 2 for Body left & right
3 or 4 for Shoulder up & down
5 or 6 for Elbow up & down
7 or 8 for Wrist up & down
9 or 0 for Gripper open & close
Enter to turn the Robot light on

Space to stop movement & switch light off

* F2 -> Edit or ESC:
```

PROC, FONTSCALE and HVTAB

We have introduced some new commands that deserve a brief explanation.

The PROC command, as used in PROC DisplayInstructions and PROC ResetArm, is short for Procedure. The command tells the program to jump to the part of the program labelled DEF PROC "procedure name"; in this case DisplayInstructions and ResetArm.

The end of the procedure is defined by the ENDPROC or End Procedure command at which point the program will return to where it was called from.

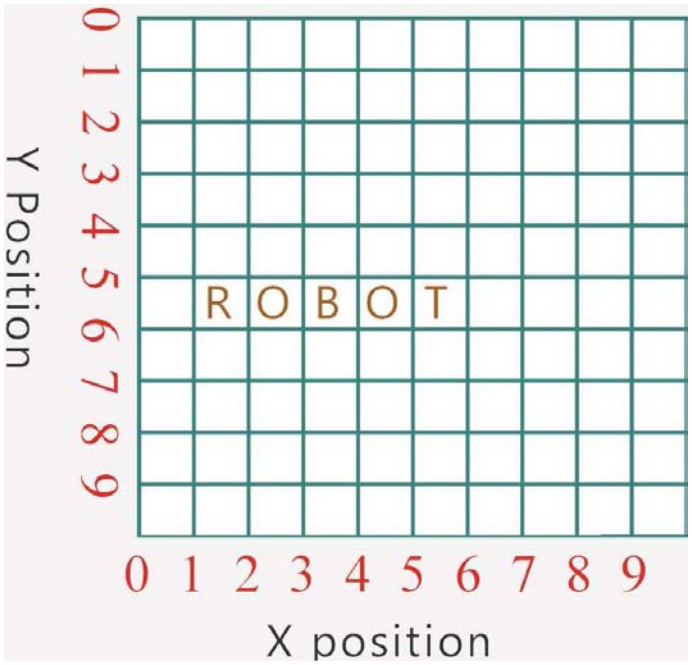
Procedures help keep a program tidy as we can place routines and functions away from the main program. They also allow us to reuse the same routine many times with a single command. The ResetArm procedure for example can be used at any point to turn everything off just by calling PROC ResetArm. It is important to grasp this concept as we will be using it later.

FONTSCALE is very straightforward. (1, 1) is normal size whereas (3, 3) is three times width and height and (2, 4) is double width but four times the height. You can experiment with this in Direct mode.

HVTAB is also very simple to grasp once explained. H is for Horizontal and V is for Vertical. The command positions the text cursor at a specified position on the screen so that the next PRINT command will place the text at that position on the screen.

Look at the example commands below:

```
HVTAB (1, 5)
PRINT "ROBOT"
```



Notice that the (0, 0) co-ordinates are in the top-left corner when plotting text characters and the size of the grid depends on the width and height of the characters.

However, from the previous articles you will also know that this is different for graphics. When plotting graphics the (0, 0) co-ordinates are in the bottom-left corner and the size of the grid is based on the number of pixels along the width and height of the screen.

Take action

We now add the main loop to the start of the program. This will make the robot arm respond to our commands.

There is a lot to add here so please be careful to copy it exactly. Once again you already have the code in grey:

```
CLS
PROC ResetArm
PROC DisplayInstructions

CYCLE

Key = Inkey

SWITCH (Key)
  CASE 49
    ArmBody (1)
  ENDCASE
  CASE 50
    ArmBody (-1)
  ENDCASE
  CASE 51
    ArmShoulder (1)
  ENDCASE
  CASE 52
    ArmShoulder (-1)
  ENDCASE
  CASE 53
    ArmElbow (1)
  ENDCASE
  CASE 54
    ArmElbow (-1)
  ENDCASE
  CASE 55
    ArmWrist (1)
  ENDCASE
  CASE 56
    ArmWrist (-1)
  ENDCASE
  CASE 57
    ArmGripper (1)
  ENDCASE
  CASE 48
    ArmGripper (-1)
  ENDCASE
  CASE 32
    PROC ResetArm
  ENDCASE
  CASE 13
    ArmLight (1)
  ENDCASE
ENDSWITCH

REPEAT

END
```

INKEY, SWITCH and CASE

There are more new commands here. First the Inkey command. This is a very useful command and one that you will use over and over again.

For example, we can use Inkey to pause any program to wait (LOOP) for a key to be pressed:

```
PRINT “Press any key to continue”
CYCLE
REPEAT UNTIL Inkey <> -1
```

If no key is being pressed the value of Inkey is -1. Whenever a key is pressed its ASCII (American Standard Code for Information Interchange) code value is stored in Inkey. So the above loop will repeat until Inkey is not equal to -1.

This also means we can check if a specific key is pressed. For example the value of the <Space Bar> is 32, so we could change the above to:

```
PRINT "Press the Space bar to continue"
CYCLE
REPEAT UNTIL Inkey = 32
```

This time the program waits specifically for the <Space Bar> key to be pressed and everything else is ignored.

Here are a few more Inkey codes, just in case you need them:

```
48 - 0 49 - 1 50 - 2 51 - 3 52 - 4 53 - 5
54 - 6 55 - 7 56 - 8 57 - 9 65 - A 66 - B
67 - C 68 - D 69 - E 70 - F 71 - G 72 - H
73 - I 74 - J 75 - K 76 - L 77 - M 78 - N
79 - O 80 - P 81 - Q 82 - R 83 - S 84 - T
85 - U 86 - V 87 - W 88 - X 89 - Y 90 - Z
32 - Space Bar       13 - Enter
```

In our program we store the value of Inkey (the ASCII code value of any key pressed) in the variable Key.

The remaining code is much easier than it looks. The SWITCH command checks the value stored in Key and, depending on the value, performs the command(s) in the relevant CASE section.

So if <1> is pressed, the ASCII code value is 49 (see reference chart above) and therefore the command ArmBody (1) is executed.

The Three Laws of Robotics

The science fiction author Isaac Asimov wrote hundreds upon hundreds of books, articles and short stories about science and robotics. He introduced the idea of programming a set of rule, or laws into all robots to protect humankind.

The stories written around these laws are extremely popular.

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.

2. A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.

3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

Asimov later went on to add a new law to precede these - the "Zeroth Law", which focuses on humanity as a whole rather than the individual.

0. A robot may not harm humanity or, by inaction, allow humanity to come to harm.

There are many debates as to whether we should implement a similar set of rules into modern day robots. What do you think?

Challenges

Now that you have seen how easy it is to control the robot arm using FUZE BASIC, why not use this new knowledge and try these two challenges.

1) Practice moving different parts of the robot around in Direct mode? Remember <F2> switches between Direct mode and the Editor.

2) Write a new program to repeat a series of robotic movements. Use the WAIT command to determine how far each movement goes.

Part 1: File systems, partition tables and rsync

W. H. Bell

MagPi Writer

SKILL LEVEL : BEGINNER

It is a good idea to backup any unique files that are not part of a software package or standard installation script, by saving them to an external or remote storage area. This allows the freedom to experiment with a Raspberry Pi software installation in the knowledge that files can be recovered if something goes wrong.

When developing software, it can be helpful to use an external repository such as Git. To make sure that the files are safe, each change can then be pushed to the external repository. The external repository can also be downloaded onto another computer for recovery or assessment by a teacher. More information about Git and repositories is provided within the Version Control series in Issues 27, 28 and 29 of The MagPi.

Instead of using an external repository, it is possible to back up a complete SD card. This type of backup operation was discussed in Issue 10 of The MagPi. The complete backup of an SD card is not as quick as using an external repository, but allows the posibility of saving larger files. While files can be recovered from a SD card image by re-installing a SD card or by mounting the image on another disk, it is not very convienent for single file recovery and does not allow direct file access on non-Linux computers.

When developing software or other applications, unique files are often kept within a users home directory, whereas the other files on an SD card are installed by using a package manager and a set of installation instructions. This means that most of the SD card can be reinstalled by using a basic image such as Raspbian and a script that contains package installation commands (apt-get, pip install, etc.). The rest of this artcle assumes that the latest Raspbian Linux image has been installed on the client Raspberry Pi. However, the instructions should be valid for other Linux distributions too.

File system considerations

There are several file systems that could be used to store files from a Raspberry Pi. However, some are more suitable than others. For example, a USB memory key is typically formatted using the FAT32 file system. While accessable on many different operating systems, this file system does not allow Linux file permissions to be preserved. However, file creation time can be retained. For this reason, the FAT32 file system is not suitable to be used as a home directory, but could be used to backup files.

New external hard disks are often formatted with the NTFS file system. The EXT4, ZFS or OSX file systems are more appropriate for saving files from a Raspberry Pi, since these file systems allow the file permission information to be preserved in the backup directory structure.

Mounting a USB key

USB memory keys are cheap and can be connected directly to a Raspberry Pi, via a USB hub or from a remote Raspberry Pi or other computer on the same network. They provide an inexpensive backup solution, but can fail if used for a large number of file writing operations.

When a USB device is attached to a Raspberry Pi that is running Raspbian it will cause a message to be printed in the `/var/log/messages` file. While the kernel in Raspbian recognises a large number of USB devices, it does not automatically mount storage devices.

Before attaching a USB key, open an LXTerminal and type:

```
sudo tail -f /var/log/messages
```

Then connect a new USB key to the Raspberry Pi, either directly or via a USB hub. When the USB key is attached a message similar to the one at the bottom of this page will be printed on the screen. Once the message has been printed, press <CTRL>-<C> to exit the `tail` command.

In the example `tail` output below, `sda` is the device name. There is only one partition on the device `sda`, which is called `sda1`. If there were more partitions on the device then they would be visible in the output message. If more than one USB storage device is connected, then the second device will be called `sdb`.

A storage device cannot be directly mounted and needs to have at least one partition. A new USB key or hard disk will typically have only one partition. To use the partition within Linux it has to be mounted. A mounted partition provides storage that is partially in memory and partially on the storage medium. This means that the partition is not as likely to become fragmented, but also means that data can be lost if the storage device is removed without unmounting the device.

The USB key in the example `tail` output can be mounted as the `/mnt` directory by typing:

```
sudo mount /dev/sda1 /mnt
```

Once this command has been typed, the size of the USB key and the remaining space can be determined by typing:

```
df -h /mnt
```

After some files have been copied to or from `/mnt` the USB file system can be safely unmounted either by shutting down the Raspberry Pi or by typing:

```
sudo umount /mnt
```

If an application is still using `/mnt` then the command will fail and report that the file system is busy.

```
Nov  7 14:04:54 raspberrypi kernel: [56548.652450] usb 1-1.2: new high-speed USB device number 4 using dwc_otg
Nov  7 14:04:54 raspberrypi kernel: [56548.753515] usb 1-1.2: New USB device found, idVendor=0781, idProduct=5566
Nov  7 14:04:54 raspberrypi kernel: [56548.753550] usb 1-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
Nov  7 14:04:54 raspberrypi kernel: [56548.753565] usb 1-1.2: Product: Firebird USB Flash Drive
Nov  7 14:04:54 raspberrypi kernel: [56548.753579] usb 1-1.2: Manufacturer: SanDisk
Nov  7 14:04:54 raspberrypi kernel: [56548.753592] usb 1-1.2: SerialNumber: 4C532000041220118090
Nov  7 14:04:54 raspberrypi kernel: [56548.759966] usb-storage 1-1.2:1.0: USB Mass Storage device detected
Nov  7 14:04:54 raspberrypi kernel: [56548.772703] scsi0 : usb-storage 1-1.2:1.0
Nov  7 14:04:55 raspberrypi kernel: [56549.773846] scsi 0:0:0:0: Direct-Access     SanDisk  Cruzer Slice     1.26 PQ: 0 ANSI:
5
Nov  7 14:04:55 raspberrypi kernel: [56549.777593] sd 0:0:0:0: [sda] 31266816 512-byte logical blocks: (16.0 GB/14.9 GiB)
Nov  7 14:04:55 raspberrypi kernel: [56549.780002] sd 0:0:0:0: [sda] Write Protect is off
Nov  7 14:04:55 raspberrypi kernel: [56549.780757] sd 0:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesn't
support DPO or FUA
Nov  7 14:04:55 raspberrypi kernel: [56549.800289]  sda: sda1
Nov  7 14:04:55 raspberrypi kernel: [56549.806209] sd 0:0:0:0: [sda] Attached SCSI removable disk
Nov  7 14:04:55 raspberrypi kernel: [56549.841487] sd 0:0:0:0: Attached scsi generic sg0 type 0
```

External hard disks

The procedure for mounting an external hard disk drive on a Raspberry Pi is exactly the same as the USB key example. The only difference is that external USB hard disk drives are often initially formatted using the NTFS file system.

Modifying the partition table

Each storage device has a partition table. The partition table contains the dimensions of each partition and a corresponding flag for each partition type. For the partition table to be modified all of the partitions of the given device must be unmounted.

Using a USB key as an example, the partition table can be accessed using the `fdisk` command. Enter:

```
sudo fdisk /dev/sda
```

From the `fdisk` prompt, the partition table can be printed by pressing "p" at the prompt. In this case there is only one FAT32 partition that spans the USB key. (In the `fdisk` command `/dev/sda` should be chosen to match the device, as discussed earlier in this article).

For the next part of this example an EXT4 file system will be used. Therefore, the file system type of the first partition should be changed. The partition system id can be changed to match the EXT4 file system by pressing "t" at the `fdisk` prompt and entering "83":

```
Command (m for help): t
Hex code (type L to list codes): 83
```

The changes made can then be saved to the storage device by pressing "w":

```
Command (m for help): w
```

The `fdisk` command provides information on each of the possible commands and also has an associated manual page. To read this, open another terminal window and enter:

```
man fdisk
```

Creating an EXT4 file system

File systems can only be rebuilt when a file system partition is unmounted. Unlike the `fdisk` command, only the partition that is to be formated needs to be unmounted. To create an EXT4 file system on the USB key enter:

```
sudo mkfs.ext4 /dev/sda1
```

A list of possible file system creation commands can be found by entering `man mkfs`.

Once the EXT4 file system has been created, it can be mounted:

```
sudo mount /dev/sda1 /mnt
```

While the EXT4 file system cannot be read on OSX or Windows, it provides the advantage that the file permissions and ownership information is preserved. A USB formatted with EXT4 can be accessed using another Linux computer, which is not necessarily a Raspberry Pi.

Introducing Rsync

The `rsync` command provides an efficient way of copying files between directories, different storage devices or to another computer.

To test the `rsync` command and understand some of the file system issues, enter the following commands:

```
cd
mkdir -p important/private
touch important/textFile
touch important/private/textFile
touch important/private/exeFile
chmod 700 important/private
```

This will create a directory structure and three test files. The permissions of the `private` directory are chosen to be tighter than the default file mask and only allow the user to access the `private` directory. To print the file permissions and ownership information type:

```
ls -al important
```

Now that the test directory structure has been created, it can be mirrored to the EXT4 file system on the USB key. Enter:

```
sudo rsync -av important /mnt/
```

Note: the `rsync` command is sensitive to the trailing forward slash (/) character. This command will create a new directory called `important` on the USB key. If the command is typed again, it will check the file time stamps and permissions and only copy files if the files on the local disk do not match the files on the USB key. If there are additional files on the USB key that are not present within the `important` directory, then they will not be affected by this mirroring command.

To mirror the files and delete any files on the USB key that are not present in the local copy of the important directory, enter:

```
sudo rsync -av --delete important /mnt/
```

The action of typing this command can be checked beforehand by using `-avn` rather than `-av`. When the `-n` flag is used, `rsync` will just print what it would have done. Files can be mirrored back from the USB key to the local disk by swapping the two directories:

```
sudo rsync -av /mnt/important/ important/
```

More `rsync` commands can be found in the documentation by entering `man rsync`.

Rsync to FAT32

If a USB key is formatted with the FAT32 file system, then files cannot be mirrored to it using `rsync` with the `-a` flag. While the ownership and file permissions information cannot be preserved on a FAT32 partition, the time stamp information can be preserved.

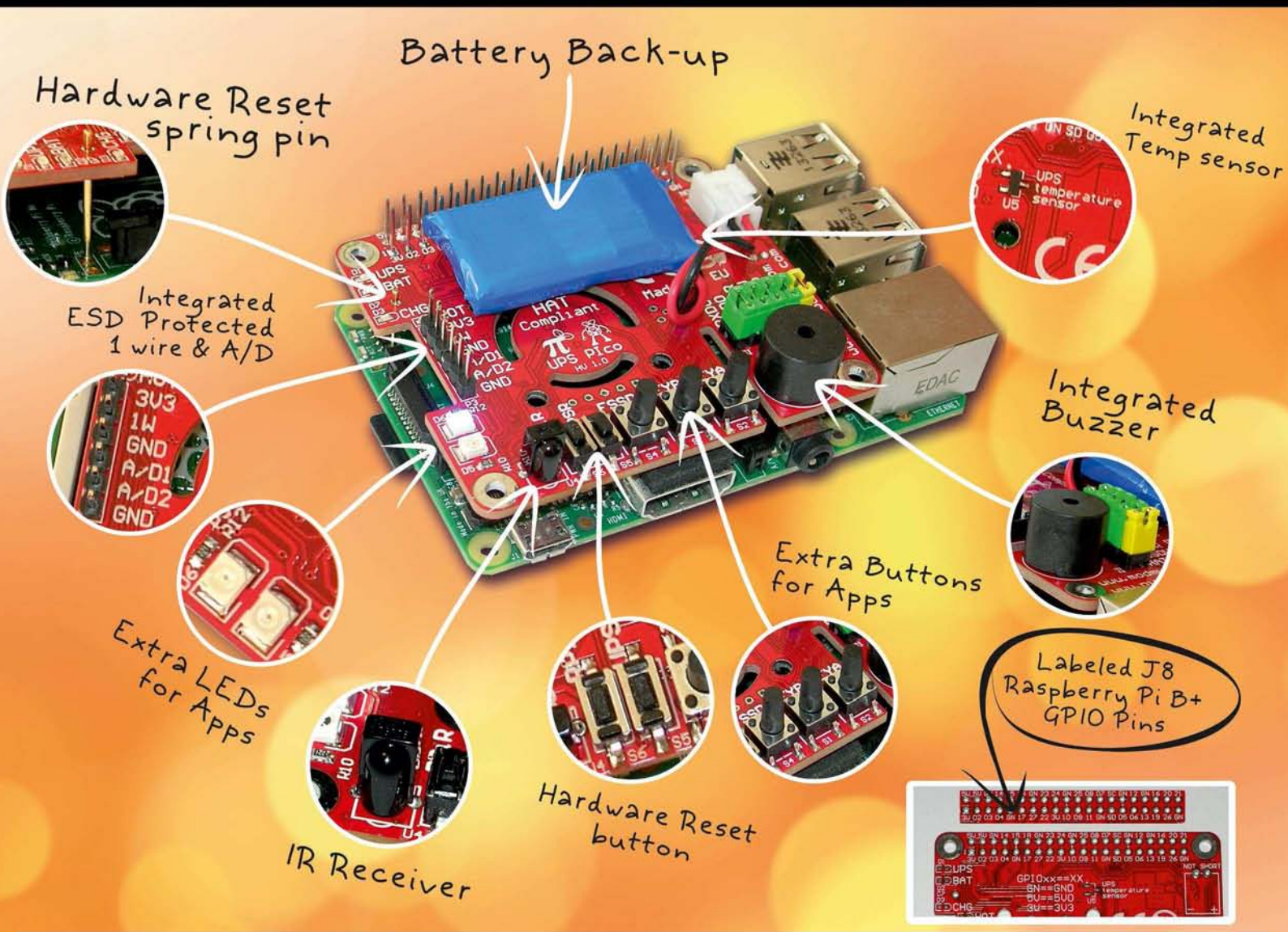
Files and directories can be mirrored to a FAT32 or NTFS file system on a USB key or external hard disk by using the `-rltv` flag combination. For example:

```
sudo rsync -rltv important /mnt/
```

UPS PIco

Uninterruptible Power Supply with Peripherals & I²C Control interface

Protect your Power
Protect your Pi



The all new UPS PIco offers an intelligent, uninterruptible power solution to the innovative Raspberry Pi computer.

- HAT Compliant
- No Additional External Power Required
- Designed for the Raspberry Pi® B+ and A+
- Integrated LiPO Battery with intelligent charger
- 5V 2A Power Backup
- Additional LEDs & Buttons for user applications
- Battery Backed RTC
- Integrated buzzer
- Integrated ESD-Protected 2 Channel A/D
- Integrated ESD-Protected 1-Wire Interface
- XTEA Based Cryptography
- 2 Level Watch-dog Functionality
- Raspberry Pi ® Hardware Reset Button
- IR Receiver Interface
- Labeled J8 Raspberry Pi B+ GPIO Pins
- PWM FAN interface
- Safe Shutdown on Power Outage
- 5 - 10 Mins Power Back-Up
- Upgradable Battery for 8 Hours Use
- Fits Inside Most Existing Cases

Version control basics using Git - Part 3

Alec Clews

Guest Writer

SKILL LEVEL : BEGINNER

Introduction

In Issue 28 of The MagPi, I explained what happens when a change is made to a document or program, plus I described what a commit ID is and the use of branches. We also touched on Git graphical tools.

This month we will focus on merging branches, I will explain more git graphical tools plus how to work with other people's code.

Merging

Let's look again at the current structure of our commit tree.

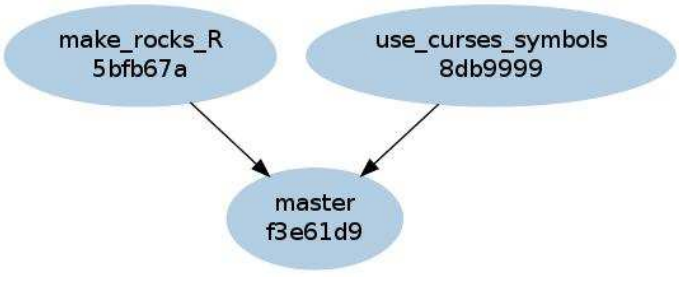


Figure 1: The current repo history with three branches and one commit on each branch.

At some point we need to bring both our changes, which we are now happy with, back into the master branch. This will make them part of the default code and we can make new changes on top of that. This process is called **merging**.

The concept is simple enough, but it is important to remember that we have three branches in this example, `master`, `make_rocks_R` and `use_curses_symbols`. Each branch has only one commit.

Fast-forward merging

The first step is to merge `make_rocks_R` into `master`. Notice that this operation is not commutative. So `make_rocks_R` merged into `master` is not the same as `master` merged into `make_rocks_R`. Make the current branch `master`. Enter:

```
cd ~/snakes
git checkout master
```

You should see the following output:

```
Switched to branch 'master'
```

Now merge from `make_rocks_R` into the current branch. Enter:

```
git merge make_rocks_R
```

Author photo courtesy of Jack Cotton

You will see something similar to:

```
Updating 5bfb67a..f3e61d9
Fast-forward
 game/snake.py |    4 ++--
 1 file changed, 2 insertions(+), 2
deletions(-)
```

Notice the phrase **Fast-forward**. This is because `master` has no changes of its own since `make_rocks_R` was created. In this case, all that happend was that the master pointer was moved up the graph until it pointed to the HEAD of `make_rocks_R`. In a minute we will create a merge that cannot be fast-forwarded.

The repo graph now looks like...

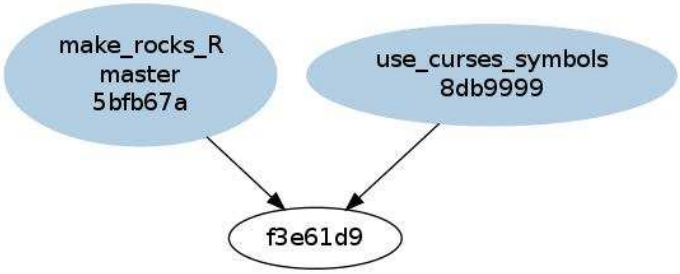


Figure 2: The repo history after our first merge.

Merging with conflicts

Now let's perform a more complex merge using `use_curses_symbols`. First let's check we are on the correct branch, `master`. Enter:

```
git branch
```

You will see the following:

```
  make_rocks_R
* master
  use_curses_symbols
```

Now enter:

```
git merge use_curses_symbols
```

This results in the following output:

```
 Auto-merging game/snake.py
 CONFLICT (content): Merge conflict in
game/snake.py
 Automatic merge failed; fix conflicts and
then commit the result.
```

Now we are getting a conflict, which means that Git cannot automatically merge the two versions. This is because we have changed the same line in both branches.

The `git status` tells that we have a half complete commit with some instructions on what to do next. Enter:

```
git status
```

```
 # On branch master
 # Unmerged paths:
 #   (use "git add/rm <file>..." as
appropriate to mark resolution)
 #
 #       both modified:      game/snake.py
 #
 no changes added to commit (use "git add"
and/or "git commit -a")
```

Let's see what our conflict looks like. Enter:

```
git diff
```

```
 diff --cc game/snake.py
 index f3e61d9,8db9999..0000000
 --- a/game/snake.py
 +++ b/game/snake.py
 @@@ -47,9 -47,9 +47,13 @@@ def
add_block(scr, width, height)
               empty = False

         if empty:
 -         # if it is, replace it with a
"Y" and return
 +         # if it is, replace it with a
"R" and return

 ++<<<<<<< HEAD
  +        scr.addch(y, x, ord("R"),
curses.color_pair(2))
 ++=======
 +         scr.addch(y, x, ord("Y"),
curses.color_pair(curses.COLOR_GREEN))
 ++>>>>>>> use_curses_symbols
           return

  def snake(scr):
```

Again we can ignore most of the this report. What is interesting is the text between `<<<<<<<`, `=======` and `>>>>>>>`. The markers are

inserted by Git to show the line that is different in each version.

To fix this we only need to edit the file `snake.py` and edit the text between the two markers (including the markers) to be what we want. Once this is complete try `git diff` again:

```
git diff
```

```
 diff --cc game/snake.py
 index f3e61d9,8db9999..0000000
 --- a/game/snake.py
 +++ b/game/snake.py
 @@@ -47,9 -47,9 +47,9 @@@ def
add_block(scr, width, height)
                empty = False

         if empty:
 -         # if it is, replace it with a
"Y" and return
 -
 -         scr.addch(y, x, ord("R"),
curses.color_pair(2))
 +         # if it is, replace it with a
"R" and return
 -        scr.addch(y, x, ord("Y"),
curses.color_pair(curses.COLOR_GREEN))
 ++        scr.addch(y, x, ord("R"),
curses.color_pair(curses.COLOR_GREEN))
           return

 def snake(scr):
```

It will probably take a little while to verify that this report shows we have completed the change. Once we are happy, (and we should also probably do a test as well), then we can add and commit the change. Enter:

```
git add .
git commit -m "Merged in Rocks being 'R'"
```

```
 [master b499f56] Merged in Rocks being
'R'
```

So `master` has now got a new commit (compared to the previous merge where it was able to "reuse" the HEAD commit on another branch i.e. the fast-forward). The new commit contains both sets of changes.

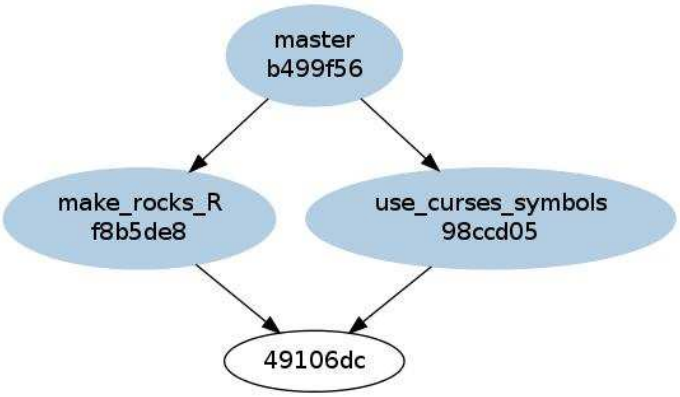


Figure 3: The repo history after our second merge.

The example merge we just completed required us to edit the merge halfway through. Life is usually much simpler as Git can perform the edit for us if the changes do not overlap, the commit is then completed in a single merge command.

Rebase

Git also has a `git rebase` command which allows us to bring branches together in very convenient ways. However, we do not have enough space to discuss that in this article but later I will suggest some online resources for you to use. I recommend getting familiar with all the great things `git rebase` can do.

Graphical helpers

Previously I mentioned the `git gui` program that provides a GUI interface to most of the commands we have been using so far (e.g. `init`, `add`, `commit`). Another program that I use a lot is `gitk` which provides a nice list of the all the commits and is easier to browse than the `git log` command. Use the `--all` parameter to see all the branches in the current repo.

Difftool

As we have already seen, the output from running the `git diff` command is not always obvious. Fortunately Git provides the `git difftool` command to display side by side differences in the GUI. A variety of third party tools are supported and I generally use `kdiff3` which works across Linux, OS X and Windows.

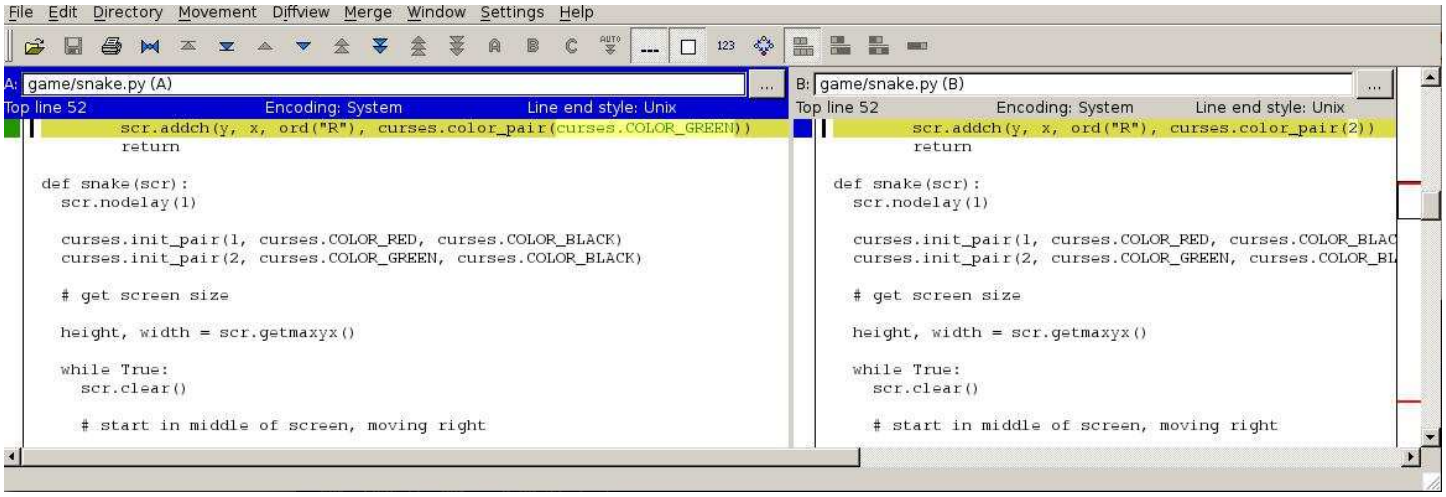


Figure 4: Running the git difftool command.

```
sudo apt-get install kdiff3-qt
```

To see the difference between the `master` and `make_rocks_R` branches, enter:

```
git difftool master make_rocks_R
```

```
 merge tool candidates: opendiff kdiff3
tkdiff xxdiff meld kompare gvimdiff
diffuse ecmerge p4merge araxis bc3 emerge
vimdiff

 Viewing: 'game/snake.py'
 Launch 'kdiff3' [Y/n]:
```

Press <Y> and the above screen should appear.

Wrap up

Working with other people's code

I hope to cover this topic in a lot more detail in future articles when we use services like GitHub or BitBucket.

However, before we wrap up it is probably worth introducing the `git clone` command. This is identical to `git init` in that it creates a new repository. But it then copies the contents of another repository so that you can start working on it locally. For instance if you want to get a copy of this article to improve, enter:

```
git clone https://github.com/alecthegeek/
version-control-basics.git
```

```
 Cloning into 'version-control-basics'...
```

Ignoring files

By default, every time the `git status` command is used Git reminds us about all files that are not under version control. However in most projects there are files we do not care about (e.g. editor temporarary files, build time object files, etc). Create the file `.gitignore` in the top project directory and list all the files you want to ignore.

Note: You should still check the `.gitignore` files into your repo along with the other files.

Further reading and help

We have covered the following Git workflows:

1. Creating a new repo
2. Adding code to the repo
3. Making changes and using the index
4. Creating branches to keep changes separate
5. Using merge to bring our changes together

I had to skip over a few things so please make sure you use the following great resources to improve your knowledge.

A great jumping off point for Git is the web site http://git-scm.com. This contains links to software, videos, documentation and tutorials.

Additionally, "Pro Git" (http://progit.org) is a highly recommended online book. Also watch the "Introduction to Git" video with Scott Chacon of GitHub (http://youtu.be/ZDR433b0HJY).

Thanks to Matthew McCullough from Github for his help with this article.

Learning to land on Mars

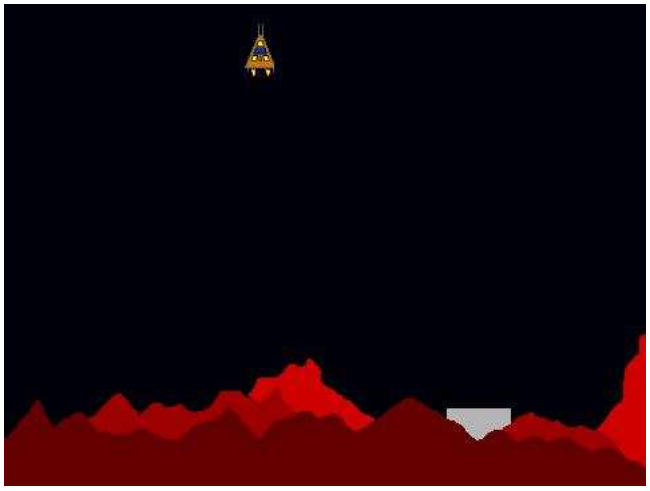
W. H. Bell

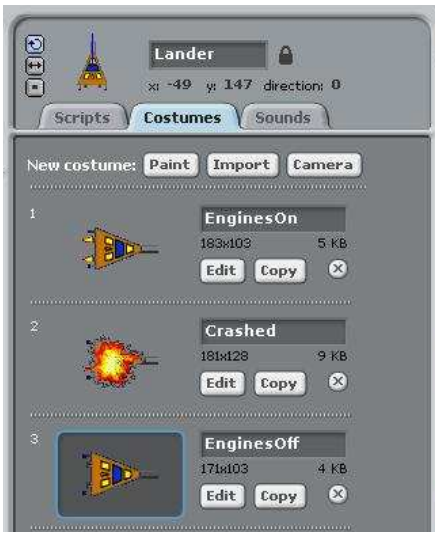
MagPi Writer

SKILL LEVEL : BEGINNER

Adding natural physics processes to games can make them more real and thrilling to play. In Issue 17, a simple projectile motion game was introduced. The game included gravity and initial velocity. These concepts can be used for other types of games too.

This month's game introduces the simulation of a spaceship landing on a remote planet, where the landing craft obeys the normal laws of physics. In this case, the Lander has rocket motors to change its direction.





The idea of the game is to land the spacecraft on the landing pad. If the spacecraft lands on the surrounding ground, then it will explode. If the spacecraft does not touch down on the landing platform correctly, then it will explode too. The spacecraft must be travelling slowly when it touches down or it will be destroyed.

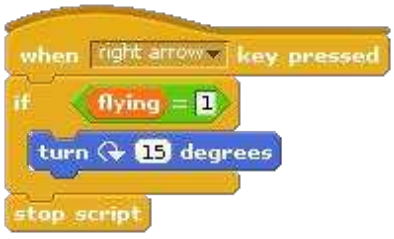
The Lander sprite has three costumes, to show the Lander with its engines on, when crashed, and when the engines are off. The costume with the engines off was copied and then modified to form the other costumes. The costumes were drawn on their sides, such that an angle of zero degrees corresponds to the spacecraft pointing straight upwards. (More details of the Scratch system of angles are given in Issue 17.)

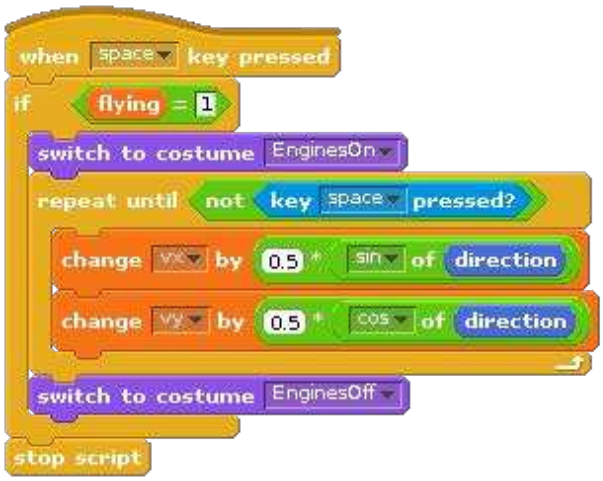
The Stage

Two backgrounds were created for the Stage, where the size of the landing pad and the amount of surrounding red rock was chosen to be different. Once the background images had been drawn, the Lander sprite was resized to match landing pad by the right clicking the mouse on the Lander sprite. The colours used for the surrounding ground and the landing pad for the two backgrounds were chosen to be the same, to allow tests based on the colours of the sprite and the background. Then a simple script was written to select a random Stage background when the green flag is pressed.

Controlling the Lander

The Lander is controlled with the left and right arrow keys and the space bar. To prevent the controls from moving the lander when the game is not taking place, a local variable was created called **`flying`**. If the **`flying`** variable is set to one, then the controls will change the angle of the Lander sprite.



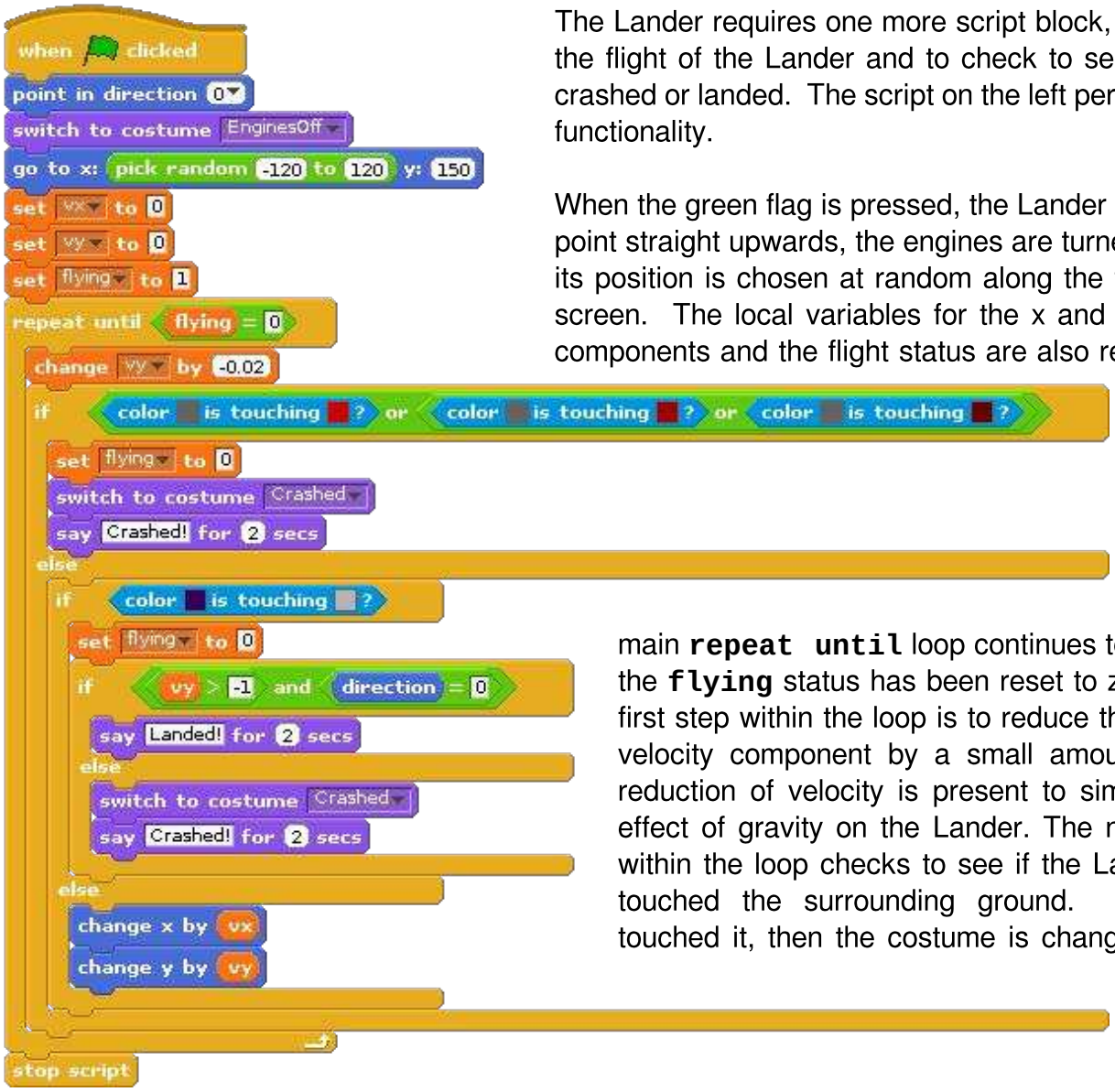


Tapping on the left arrow causes the Lander to turn 15 degrees to the left and tapping on the right arrow causes the lander to turn 15 degrees to the right.

The space bar is used to fire the thrusters, to slow down the Lander and manoeuvre it to the landing pad. When the space bar is pressed, the costume of the Lander changes to the version that shows the engines burning. The costume continues to show the engines burning, while the space bar is pressed. When the space bar is released, the costume changes back to show the engines as being off.

While the space bar is pressed the velocity of the Lander is increased, according to its current direction. The sine and cosine operators are used to calculate the increase of the x and y velocity components using the direction, where the velocity components are stored in local variables **vx** and **vy** respectively. The direction is zero, when the Lander points straight upwards, 180 when it points straight downwards, positive when it points to the right and negative when it points to the left.

The Lander in flight



The Lander requires one more script block, to control the flight of the Lander and to check to see if it has crashed or landed. The script on the left performs this functionality.

When the green flag is pressed, the Lander is reset to point straight upwards, the engines are turned off and its position is chosen at random along the top of the screen. The local variables for the x and y velocity components and the flight status are also reset. The main **`repeat until`** loop continues to run until the **`flying`** status has been reset to zero. The first step within the loop is to reduce the vertical velocity component by a small amount. This reduction of velocity is present to simulate the effect of gravity on the Lander. The next block within the loop checks to see if the Lander has touched the surrounding ground. If it has touched it, then the costume is changed to the crashed version and the **`flying`** status is set to zero. If it has not touched the surrounding ground, then the program checks to see if the landing feet have touched the landing pad or not. If they have touched the landing pad, then the vertical velocity component and the direction of the Lander is checked. If the touch down is not successful, the Lander will be shown as crashed. If the landing is successful, then the Lander will say "Landed!". While the Lander is not in contact with the ground, the position of the Lander is updated using the velocity components. In this program, the speed of the loop regulates how fast time is passing in the simulation.

To make the game harder or easier, try changing the reduction of the vertical velocity in the loop (the effect of gravity) or try changing the effect of the thrusters. The lander could be given a limited amount of fuel, by counting the number of times the space bar is pressed.

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

Manchester Raspberry Jam

When: Saturday 6th December 2014, 10.00am to 5.00pm
Where: The Shed, John Dalton West, Chester Street, Manchester, M1 5GD, UK

This event is suitable for anyone – professional or novice, adult or child – you don't need any prior knowledge or experience. http://www.eventbrite.co.uk/e/14124943085

Lagos Raspberry Jam

When: Thursday 11th December 2014, 12.00pm to 6.00pm (WAT)
Where: 294 Herbert Macaulay Way, Sabo, Yaba, Lagos, Nigeria

The CcHUB Raspberry Jam is a rare gathering of hackers from within and around Lagos, aimed at demonstrating the evolving maker movement in Nigeria. http://cchubnigeria.com/rpi/jam

Northern Ireland Raspberry Jam 7

When: Saturday 13th December 2014, 1.00pm to 5.00pm
Where: Farset Labs, Linfield Industrial Estate, Belfast, BT12 5GH, UK

The sessions are aimed at complete beginners with more complicated challanges and for those with already some Raspberry Pi experience. http://bit.ly/farsetjam7

Glasgow Raspberry Pi Day

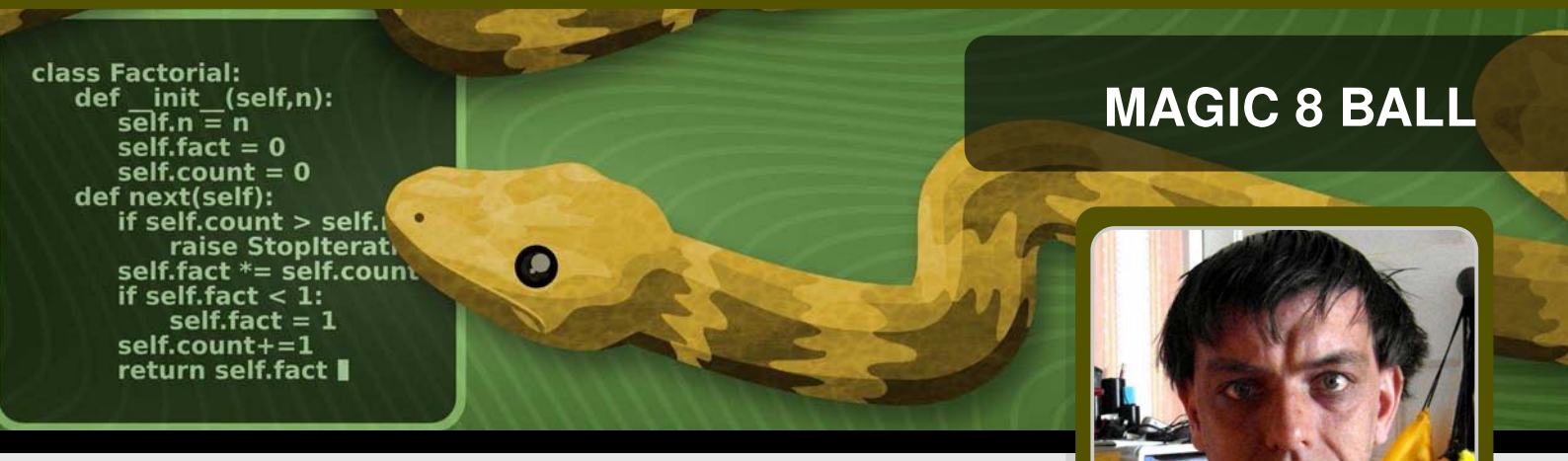
When: Saturday 17th January 2015, 10.00am to 4.00pm
Where: John Anderson Building, 107 Rottenrow, Glasgow, G4 0NG, UK

A Raspberry Pi computing event for Gloucestershire and beyond. Deliberately family friendly, with a portion of tickets reserved for children. http://phys.strath.ac.uk/raspberrypiday/

Pi and More 6

When: Saturday 24th January 2015
Where: Campus Alt-Saarbrücken, Goebenstraße 40, 66117 Saarbrücken, Germany

Pi and More is the first and largest German Raspberry Jam, with up to 150 attendees.
Event details at https://piandmore.de/



Creating a GUI with Python's Tkinter

SKILL LEVEL : BEGINNER

Paul Sutton

Guest Writer

A while back, I had never really touched graphical programming. However after reading through the book Raspberry Pi in Easy Steps I decided to have a go myself. As I have started to write my own graphical programs I wanted to see if I could recreate a Magic 8 Ball.

Once you are familiar with creating programs in Python that run from the console the next step is to create programs that will run in a window (noting that not all programs are suitable to run in this way). We are going to use Tkinter, which according to the Python Wiki https://wiki.python.org/moin/TkInter is Python's de-facto standard GUI (Graphical User Interface) package.

We are going to build up our GUI step by step so that you can see exactly what is needed, starting initially with a few concepts in Tkinter before moving onto our Magic 8 Ball. But first, it is back to the command line to install Tkinter with the following command:

```
sudo apt-get install python-tk
```

With that done we can create the start of our Magic 8 Ball. The following program produces a Window on the screen and sets various attributes:

```
#!/usr/bin/env python
from Tkinter import *

window = Tk()
window.title('GUI Tkinter 1')
window.geometry("300x250") # w x h
window.resizable(0,0)

window.mainloop()
```

The first line tells the the interpreter to use Python. This line is only essential if you want to run the program without prefixing it with the python command. We then import all functions from the Tkinter module. At this point we can define a window, which will have a title, a size and whether it will be resizable or not. Lastly, `window.mainloop()` displays the window, prevents the program from immediately exiting and enables us to interact with it through the GUI.

Using Tkinter's grid

We are now going to add a label to the window we have created. To do this we need to add an extra two lines of code immediately before `window.mainloop()`.

```
#define labels
box1 = Label(window,text="Entry 1: ")

#place the label in the window object
box1.grid(row = 1, column = 1,
  padx = 5, pady = 5)
```

We are using the `grid` method to place the label on the screen. Tkinter supports an invisible, customisable, grid upon which screen elements can be placed. Each element can be one or more rows wide and one or more columns tall.

Adding a button

Now that we are able to add objects to the window we can add more interactive components such as buttons. Replace the lines that we entered, above, with the following above `window.mainloop()`:

```
def btn1():
  print ("button pressed")

btn_tog2 = Button( window,
 text ='button1', command=btn1)
btn_exit = Button( window,
 text ='exit',command=exit)

btn_tog2.grid(row = 1, column = 1,
 padx = 5, pady = 5)
btn_exit.grid(row = 2, column = 1,
 padx = 5, pady = 5)
```

First we define a function that can be called (for the moment, ours will just print out some text to the terminal). Note that the lines contained within the function must be indented. Next we create two buttons: the first names the function which will be called when clicked (`btn1`), while the second calls `exit` to close the program. exit is a function built in to Python and will appear in a different colour in most editors (including nano and Idle). Lastly, we position both interface elements onto the grid, one to a row, both in the first column. We apply a bit of spacing (using `padx` and `pady`) to ensure that the buttons do not touch (this is good practice in user interface design to reduce the chance of the user accidentally clicking the wrong button).

Most importantly, functions must be defined before you call them from within your program.

Improving usability

Now that we are able to add a label and a button to a window, we can start to make our application more user friendly. While buttons have in-built labels the purpose of this lesson will simply be about adding both (for example: for those times when you want to use an image for a button and have descriptive text next to the image). The button has adjacent text with a description of what the button does.

```
#define functions for button(s)
def btn1():
  print ("button pressed")

#create button objects
btn_tog2 = Button(window,
 text ='button1', command=btn1)
btn_exit = Button(window,
 text ='exit',command=exit)

#place button objects
btn_tog2.grid(row = 1, column = 2,
 padx = 5, pady = 5)
btn_exit.grid(row = 2, column = 2,
 padx = 5, pady = 5)

#define labels
button1 = Label(window,
 text="click button")
button2 = Label(window,
 text="exit program")

#place labels
button1.grid(row = 1, column = 1,
 padx = 5, pady = 5)
button2.grid(row = 2, column = 1,
 padx = 5, pady = 5)
```

I have placed the objects explicitly on the window: column 1 has labels, while column 2 has buttons. I have also tried to name the objects logically in the code to make debugging easier.

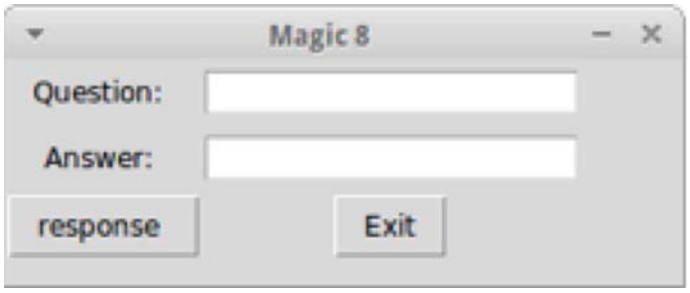
We are also commenting (# comment) as we go to explain what the code does.

Creating the Magic 8 Ball

Now that we can add objects we can add other things too.

We need to modify our program to add three things: a text box for the user to enter their question, an area to show the response, and the logicial code to generate that response. This will take text input and then generate a random response. This gives us the components of a question and answer program. The response part of the program was created by Tom Brough. The full listing is:

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

import random
from Tkinter import *

def response():
  response_phrase =
   random.choice(RESPONSES)
  #print response__phrase
  # clear prev output
  circletext2.delete(0, END)
  circletext2.insert(0,str(
   response_phrase))

#set up
window = Tk()
window.title('Magic 8')
window.geometry("300x100") #wxh
window.resizable(0,0)

RESPONSES = ["It is certain",
  "It is decidedly so",
  "Without a doubt",
  "Yes definitely",
  "You may rely on it",
  "As I see it yes",
  "Most likely",
  "Outlook good",
  "Yes",
  "Signs point to yes",
  "Reply hazy try again",
  "Ask again later",
  "Better not tell you now",
  "Cannot predict now",
  "Concentrate and ask again",
  "Don't count on it",
  "My reply is no",
  "My sources say no",
  "Outlook not so good",
  "Very doubtful"]

#define and place labels
box1 = Label(window, text="Q: ")
box2 = Label(window, text="Answer: ")

box1.grid(row = 1, column = 1,
 padx = 5, pady = 5)
box2.grid(row = 2, column = 1,
 padx = 5, pady = 5)

#define entry box
circleVar = StringVar()
circletext = Entry(window,
 textvariable=circleVar)

#define output box
circleVar2 = StringVar()
circletext2 = Entry(window,
 textvariable=circleVar2)

#display boxes
circletext.grid(row = 1, column = 2)
circletext2.grid(row = 2, column = 2)

#define and place buttons
response = Button( window,
 text ='response', command=response)
exitbtn = Button( window,
 text ='Exit', command=exit)

#place buttons
response.grid(row = 4, column = 1,
 padx = 1, pady = 1)
exitbtn.grid(row = 4, column = 2,
 padx = 1, pady = 1)

#display window
window.mainloop()
```



Let's look at the program in more detail. Firstly, we import the Python random module. This lets us hand over the task of selecting which response to show over to the program and means that we are unlikely to see the same response twice. We then define a function that calls up the random response from an array of values, known as a list in Python. In our function, `response_phrase` is a variable that stores a single randomly selected response. I have commented out the `print` line as this was used for testing, however you may wish to uncomment it while testing your programs. In the final version of any program it is recommended to remove debug lines of code such as this so that they do not get accidentally uncommented later, resulting in unexpected behaviour.

We now go on to define the window as before, but you will notice I have made the window size 300 x 100. This makes the window a better fit around the program buttons and text entry / output boxes.

We type in the responses we want and store these in a list called `RESPONSES`. You will note that we do not define the list inside the function call. If we did, then the list would be recreated each time the function is called, which would slow (albeit, in this case by not much) the program.

The line `circletext2.delete(0, END)` clears the output text box from the first character. This keeps the program tidy. You don't strictly need it but it keeps things in good order. The next line does all the work, and inserts the response variable response_phrase as a string.

Once this is done we implement the GUI, incorporating all of the elements that we require.

Improving Magic 8 Ball

Now we have the basic program working, we can look in to making a few enhancements. At present if you do not type in anything or you type in a sequence of numbers you still get a response, which clearly isn't very helpful. We can modify the `response()` function in our program to take account of this. I have added code to check if what the user enters is actually text. However using the function `isalpha()` doesn't work here as it seems to detect the spaces in your question and throws up an error. To get round this I did something slightly different:

```
def response():
  msg = "error : must be a
   text value"
  i = circletext.get()
  y = i.isdigit()
  q_length = len(i.get())

  if y == True or q_length == 0:
    circletext.insert(0,(msg))
  else:
    response_phrase =
     random.choice(RESPONSES)

    # clear prev output
    circletext2.delete(0, END)

    # insert response
    circletext2.insert(0,str(
    response_phrase))
```

Here, we detect if the text in the text box is numeric with `i.isdigit()` and if it is zero characters in length with `len(i.get())`. If either is true then we display an error: the text entered (or not) is clearly not a question.

Lastly, a suggestion for further improvement: how would you provide a function for the user to trigger an action to clear both the input and output boxes? This is but one of several ways in which the program can be improved and I encourage you to use this article as the basis for your own inventiveness.

Paul can be found at: http://www.zleap.net

PRINT EDITION AVAILABLE WORLDWIDE

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

Americas

adafruit INDUSTRIES

EMEA

MODMYPI

AUTHENTIC SWAG EDUCATING SINCE '12

The Pi Hut

Pi Supply

AsiaPac

AusPi Technologies Australia

Have Your Say...

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

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