Build a Raspberry Pi cluster computer

Think Raspberry Pi computers are no match for their bigger cousins? Think again, because there's strength in numbers

MAKER

PJ Evans

PJ is a writer, developer, and Milton Keynes Raspberry Jam wrangler. Unlike a cluster, he can only do one thing at a time.

@mrpjevans

Raspberry Pi computers are famously cheap and cheerful. Great for playing around with and the odd little project, right? Well, sometimes.

However, our little friend is a surprisingly powerful computer and when you get lots of them working together, amazing things can happen.

The concept of computer 'clusters' (many computers working together as one) is nothing new, but when you have a device as affordable as Raspberry Pi, you can start to rival much more expensive systems by using several in parallel. Here, we'll learn how to make a cluster computer from a lot of little computers.

You'll Need

- 4 × Raspberry Pi 4 computers **magpi.cc/rpi4**
- Cluster case **magpi.cc/XtUbVm**
- Ethernet switch **magpi.cc/fzcGEA**
- Multi-port USB PSU **magpi.cc/AekkAr**
- 4 × USB C cables **magpi.cc/UcmFAM**
- 4 × Ethernet cables **magpi.cc/LsAnrA**

01 Cluster assemble!

A cluster of Raspberry Pi computers can start with as little as two and grow into hundreds. For our project, we're starting with a modest four. Each one, known as a 'node', will carry out part of our task for us and they all work in parallel to produce the result a lot quicker than a single node ever could. Some nice 'cluster cases' are available, and we start here by assembling our Raspberry Pi 4B computers into a four-berth chassis. Many different configurations are available, including fan cooling.

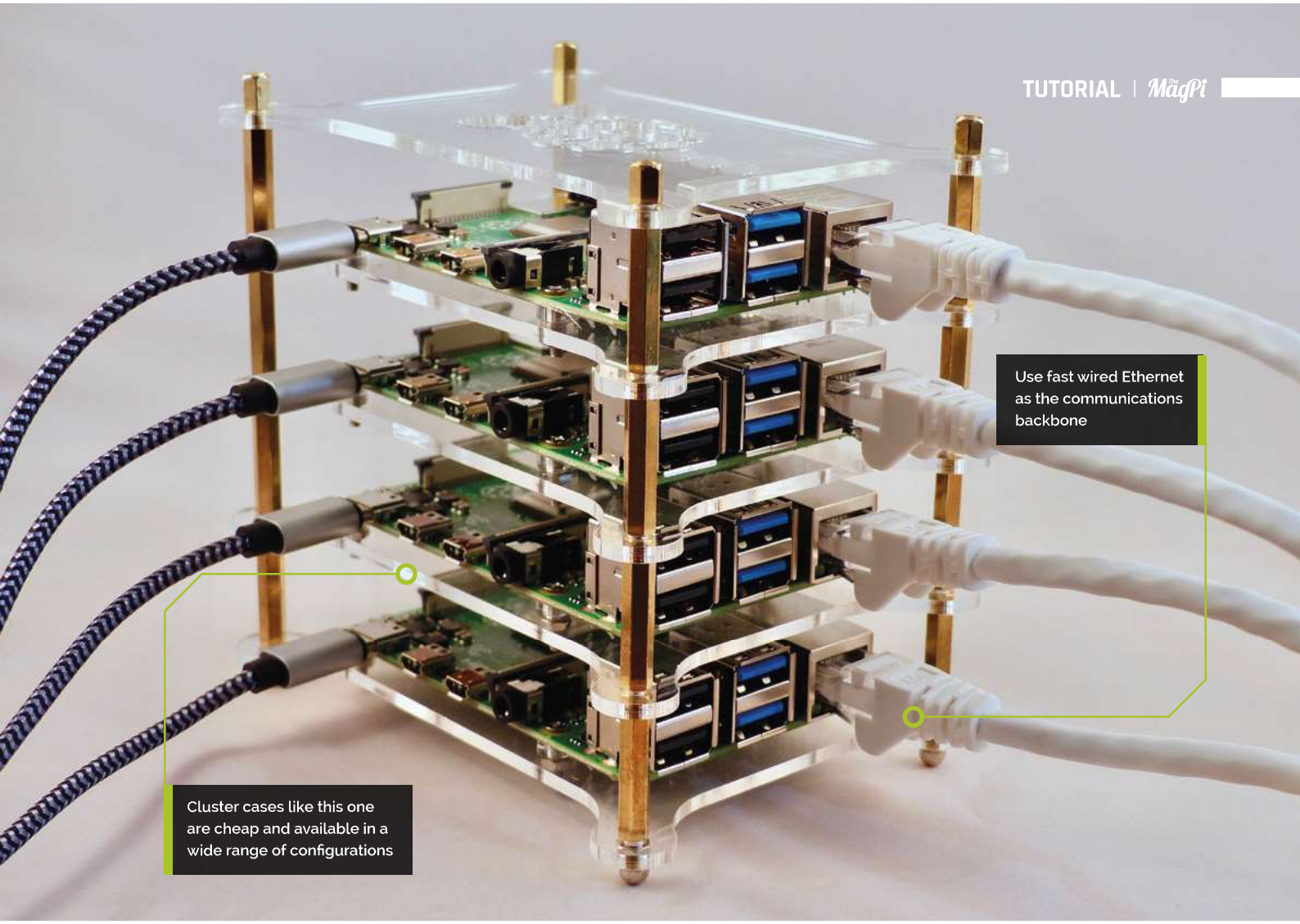
▲ A dedicated inexpensive switch will speed up communications. Raspberry Pi 4 computers can take advantage of full-bandwidth Gigabit Ethernet

02 Power up

Consider the power requirements for your cluster. With our four nodes it's not going to be ideal to have four PSUs driving them. As well as being ugly, it's inefficient. Instead, track down a good-quality, powerful multi-port USB charger that is capable of powering your chosen number of computers. Then all you need are the cables to link them and you're using a single mains socket. USB units are available that can handle eight Raspberry Pi computers without breaking a sweat. Do be careful of the higher demands of Raspberry Pi 4.

03 Get talking

A cluster works by communication. A 'master' node is in charge of the cluster and the 'workers' are told what to do and to report back the results on demand. To achieve this we're using wired Ethernet on a dedicated network. It's not essential to do it this way, but for data-intensive applications it's advisable for the cluster to have its own private link-up so it can exchange instructions without being hampered by wireless LAN or other network traffic. So, in addition to wireless LAN, we're linking each node to an isolated Gigabit Ethernet switch.



04 Raspbian ripple

We're going to access each node using wireless LAN so the Ethernet port is available for cluster work. For each 'node', burn Raspbian Buster Lite (**magpi.cc/raspbian**) to a microSD card, boot it up, and make sure it's up to date with `sudo apt -y update && sudo apt -y upgrade`. Then run `sudo raspi-config` and perform the following steps:

- Change the 'pi' user password.
- Under 'Networking', change the hostname to nodeX, replacing X with a unique number (node1, node2 etc.). Node1 will be our 'master'.
- Enable WiFi if desired.
- Exit and reboot when prompted.

05 Get a backbone

The wired Ethernet link is known as the cluster's 'backbone'. You need to manually enable the backbone, as there is no DHCP server to help. We're going to use the 10.0.0.0 subnet. If your regular network uses this, choose something different like 192.168.10.0. For each node, from the command line, edit the network configuration:

```
sudo nano /etc/dhcpcd.conf
```

Go to the end of file and add the following:

```
interface eth0
static ip_address=10.0.0.1/24
```

For each node, replace the last digit of '10.0.0.1' with a new unique value: 10.0.0.2, 10.0.0.3, and so on. Reboot each node as you go. You should be able to ping each node – for example, from 10.0.0.1:

```
ping 10.0.0.2
```

06 Brand new key

For the cluster to work, each worker node needs to be able to talk to the master node without needing a password to log in. To do this, we use SSH keys. This can be a little laborious, but only needs to be done once. On each node, run the following:

```
ssh-keygen -t rsa
```

This creates a unique digital 'identity' (and key pairs) for the computer. You'll be asked a few

Top Tip

Load balancing

Clusters are also useful for acting as a single web server and sharing traffic, such as Mythic Beast's Raspberry Pi web servers.

Top Tip

Fault tolerance

Certain cluster types, such as Docker Swarm or Kubernetes, allow individual nodes to fail without disrupting service.

questions; just press **RETURN** for each one and do not create a passphrase when asked. Next, tell the master (node1, 10.0.0.1 in our setup) about the keys by running the following on every other node:

```
ssh-copy-id 10.0.0.1
```

Finally, do the same on the master node (node1, 10.0.0.1) and copy its key to every other node in the cluster.

07 Install MPI

The magic that makes our cluster work is MPI (Message Passing Interface). This protocol allows multiple computers to delegate tasks amongst themselves and respond with results. We'll install MPI on each node of our cluster and, at the same time, install the Python bindings that allow us to take advantage of its magical powers. On each node, run the following:

```
sudo apt install mpich python3-mpi4py
```

Once complete, test MPI is working on each node:

```
mpiexec -n 1 hostname
```

You should just get the name of the node echoed back at you. The `-n` means 'how many nodes to run this on'. If you say one, it's always the local machine.

▲ Instead of a row of wall-warts, use a multi-port USB charger to power your cluster nodes, but make sure your choice of unit has enough amps

08 Let's get together

Time for our first cluster operation. From node1 (10.0.0.1), issue the following command:

```
mpiexec -n 4 --hosts 10.0.0.1,10.0.0.2,10.0.0.2,10.0.0.4 hostname
```

We're asking the master supervisor process, `mpiexec`, to start four processes (`-n 4`), one on each host. If you're not using four hosts, or are using different IP addresses, you'll need to change this as needed. The command `hostname` just echoes the node's name, so if all is well, you'll get a list of the four members of the cluster. You've just done a bit of parallel computing!

> "Now we've confirmed the cluster is operational, let's put it to work"

09 Is a cluster of one still a cluster?

Now we've confirmed the cluster is operational, let's put it to work. The **prime.py** program is a simple script that identifies prime numbers. Enter the code shown in the listing (or download from **magpi.cc/EWASJx**) and save it on node1 (10.0.0.1). The code takes a single argument, the maximum number to reach before stopping, and will return how many prime numbers were identified during the run. Start by testing it on the master node:

```
mpiexec -n 1 python3 prime.py 1000
```

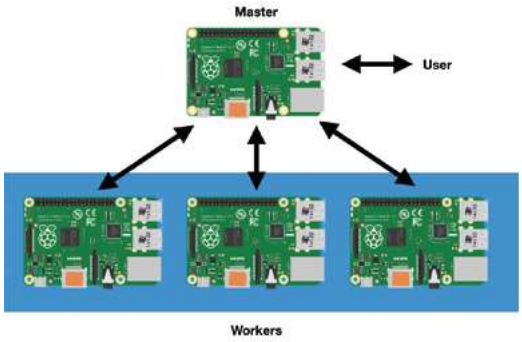
Translation: 'Run a single instance on the local node that runs **prime.py** testing for prime numbers up to 1000.'

This should run pretty quickly, probably well under a second, and find 168 primes.

10 Multiplicity

In order for the cluster to work, each node needs to have an identical copy of the script we need to run, and in the same place. So, copy the same script to each node. Assuming the file is in your home directory, the quick way to do this is (from node1):

```
scp ~/prime.py 10.0.0.x:
```



▲ Our cluster works by assigning a master node. The master assigns tasks to its member nodes and waits for them to report their results.
Credit: *Raspberry Pi illustrations by Jonathan Rutheiser*

Replace `x` with the number of the node required: `scp` (secure copy) will copy the script to each node. You can check this has worked by going to each node and running the same command we did on node1. Once you are finished, we are ready to start some real cluster computing.

11 Compute!

To start the supercomputer, run this command from the master (node1):

```
mpiexec -n 4 --host 10.0.0.1,10.0.0.2,10.0.0.3,10.0.0.4 python3 prime.py 100000
```

Each node gets a 'rank': a unique ID. The master is always 0. This is used in the script to allocate which range of numbers each node processes, so no node checks the same number for 'primeness'. When complete, each node reports back to the master detailing the primes found. This is known as 'gathering'. Once complete, the master pulls all the data together and reports the result. In more advanced applications, different data sets can be allocated to the nodes by the master ('scattering').

12 Final scores

You may have noticed we asked for all the primes up to 1000 in the previous example. This isn't a great test as it is so quick to complete. 100,000 takes a little longer. In our tests, we saw that a single node took 238.35 seconds, but a four-node cluster managed it in 49.58 seconds – nearly five times faster!

Cluster computing isn't just about crunching numbers. Fault-tolerance and load-balancing are other concepts well worth investigating. Some cluster types act as single web servers and keep working, even if you unplug all the Raspberry Pi computers in the cluster bar one.

prime.py

DOWNLOAD THE FULL CODE: magpi.cc/EWASJx

Language: **Python 3**

```
001.  from mpi4py import MPI
002.  import time
003.  import sys
004.
005.  # Attach to the cluster and find out who I am and how big it is
006.  comm = MPI.COMM_WORLD
007.  my_rank = comm.Get_rank()
008.  cluster_size = comm.Get_size()
009.
010.  # Number to start on, based on the node's rank
011.  start_number = (my_rank * 2) + 1
012.
013.  # When to stop. Play around with this value!
014.  end_number = int(sys.argv[1])
015.
016.  # Make a note of the start time
017.  start = time.time()
018.
019.  # List of discovered primes for this node
020.  primes = []
021.
022.  # Loop through the numbers using rank number to divide the work
023.  for candidate_number in range(start_number,
024.                                end_number, cluster_size * 2):
025.
026.      # Log progress in steps
027.      # print(candidate_number)
028.
029.      # Assume this number is prime
030.      found_prime = True
031.
032.      # Go through all previous numbers and see if any divide without remainder
033.      for div_number in range(2, candidate_number):
034.          if candidate_number % div_number == 0:
035.              found_prime = False
036.              break
037.
038.      # If we get here, nothing divided, so it's a prime number
039.      if found_prime:
040.          # Uncomment the next line to see the primes as they are found (slower)
041.          # print('Node ' + str(my_rank) + ' found ' + str(candidate_number))
042.          primes.append(candidate_number)
043.
044.  # Once complete, send results to the governing node
045.  results = comm.gather(primes, root=0)
046.
047.  # If I am the governing node, show the results
048.  if my_rank == 0:
049.
050.      # How long did it take?
051.      end = round(time.time() - start, 2)
052.
053.      print('Find all primes up to: ' + str(end_number))
054.      print('Nodes: ' + str(cluster_size))
055.      print('Time elasped: ' + str(end) + ' seconds')
056.
057.      # Each process returned an array, so lets merge them
058.      merged_primes = [item for sublist in results for item in sublist]
059.      merged_primes.sort()
060.      print('Primes discovered: ' + str(len(merged_primes)))
061.      # Uncomment the next line to see all the prime numbers
062.      # print(merged_primes)
```

Part 04

Add sensors to a low-cost robot

Make a robot react to the world with sensors. Fascinating behaviours emerge with only a bit of code and electronics!

MAKER

Danny Staple

Danny makes robots with his kids as Orionrobots on YouTube, and is the author of *Learn Robotics Programming*.

orionrobots.co.uk

Over the last few issues we've built a low-cost wheeled robot. Without any sensors, it doesn't respond to the world and drives into walls.

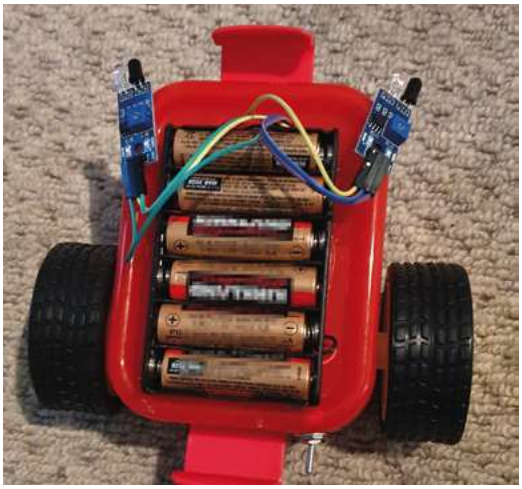
The robot's Raspberry Pi has many GPIO pins left to add inexpensive sensors for it to interact with its surroundings. This tutorial shows how to use a couple of similar sensor types, put them in useful places, wire them in, and write the code. We'll touch on the trade-offs and limitations, learn how to calibrate the sensors, and make test tracks for line following.

You'll Need

- 2 × Obstacle avoidance modules **magpi.cc/paTuQW**
- Mini breadboard **magpi.cc/CuBDyB**
- 2 × TCRT5000 sensor modules **magpi.cc/TtfhiF**
- Jumper wires **magpi.cc/pPnpZL**
- Plastic standoffs and screws **magpi.cc/Cixtpr**
- 5 mm pitch terminal block **magpi.cc/jeThnM**
- Black insulating tape
- Plain white paper

01 Meet the sensors

The two types of sensor in this tutorial both detect reflected infrared light (IR). Obstacle sensors detect objects close enough by a brightness level. Line sensors detect how light/dark the floor is below them.



▲ The robot with sensors part-fitted. See how the cables go through a port in front of the batteries

The sensor models chosen are cheap and easily found online, but being IR sensors, they can be dazzled and confused by bright sunlight and some fluorescent lights.

These sensors output digital (on/off) signals, with a dial adjusting the level to go from off to on. They come on carrier boards as 3.3V compatible modules, making them easier to connect to Raspberry Pi.

02 How to use sensors

Obstacle sensors should be front-most, so they don't detect the robot itself. By using two sensors facing forward and slightly to either side, a robot can decide which way it needs to turn to avoid an obstacle.

Line sensors should be under the robot to detect if it has gone off track. A single sensor could sweep across and back over a line, but two sensors, wide enough to go either side of a line, make for a smoother system. They can sense when the line is not in the middle, and which way the robot needs to turn to correct it.

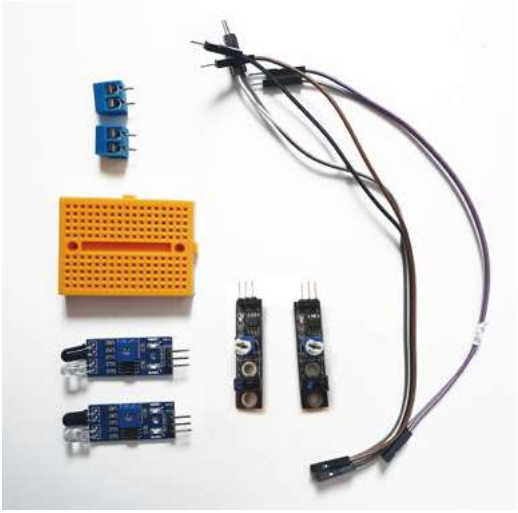
03 Planning and sketching

Test-fit the obstacle sensors on the top of the robot, facing forward and slightly outward.

Sketch the top of the robot showing where to attach the sensors (**Figure 1**). 2.5 mm bolts work for this. Add a 10 mm hole for wires to go through, clear of other features.

Next, test-fit line sensors under the robot, sticking out of the front and about 5 to 7 cm apart.

Finally, sketch the bottom of the robot with line sensors and another wire hole. You may be able to use the threads from the standoffs for Raspberry Pi to hold the line sensors if they are long enough.



▲ The components needed include terminal blocks on a breadboard, obstacle sensors, line sensors, and jumper wires

04 Drilling holes

Take off the top of the robot, disconnecting the power wires from the motor controller. Use safety gear for all drilling.

Depending on the line sensor placement, you may need to detach the robot's Raspberry Pi and castor before drilling holes in the base of the robot.

Using your sketches, measure out and drill the holes for the sensors. For the cable holes, drill small holes and enlarge them. Remove any excess plastic. Replace your Raspberry Pi and castor, but don't bolt the sensors in or reconnect the batteries just yet.

05 Power distribution

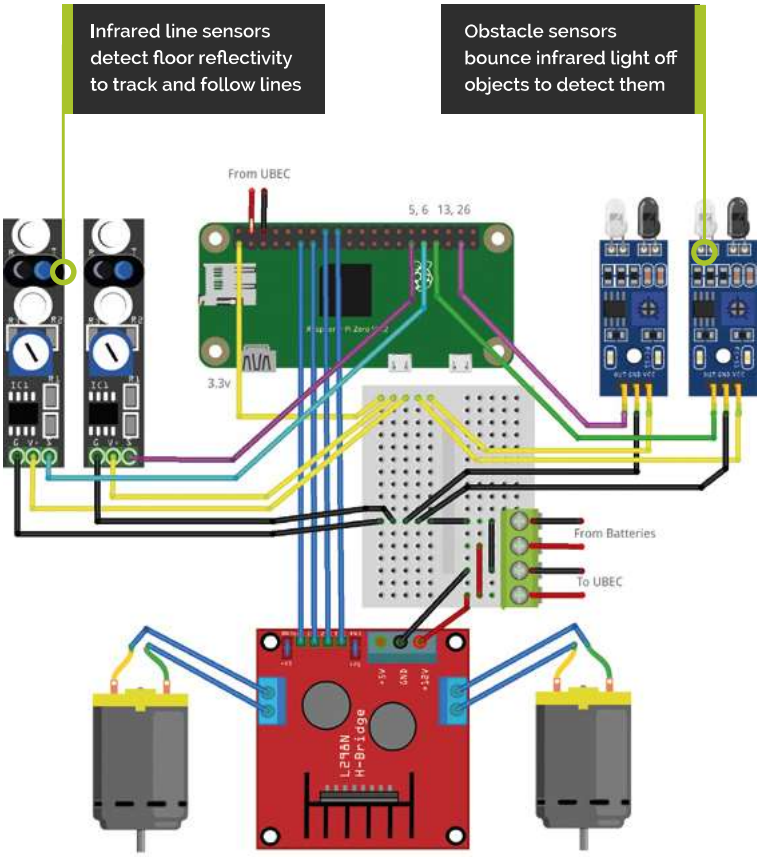
The breadboard gives the sensors access to power rails. 5mm terminal blocks fit into the breadboard, simplifying connecting the batteries and UBEC. Slot two of the blocks together to make a block of four and pop it into the breadboard, as shown in **Figure 2** (overleaf).

Add breadboard internal wiring connections – pre-cut jumper wires do a tidy job of this. Popping the breadboard lightly into the robot, make the connections to Raspberry Pi 3.3V and motor board.

The jumper coming from the battery red (positive) wire on the breadboard is a suitable place for an optional power switch.

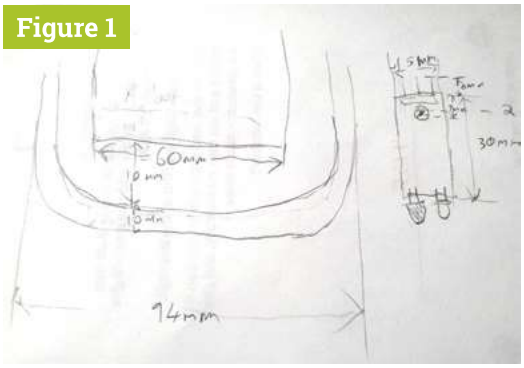
06 Wiring the sensors

Push a four-way male-female jumper wire through each cable port on the top and bottom for power connections. Use these to wire the sensor VCC/V+ pins to the 3.3V breadboard row and the GND/G pins to the ground row.



> 5mm terminal blocks fit into the breadboard, simplifying connecting the batteries and UBEC

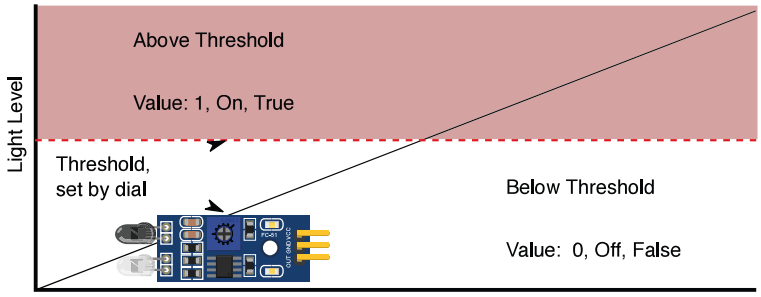
Figure 1



▲ **Figure 1** A start on a sketch of the top and the sensor dimensions. Be prepared to sketch a couple of times, adding more features

Warning!
Wear goggles

Please use safety goggles and a desk clamp for drilling.



▲ As the light detected goes up and crosses a threshold set by the dial, the sensor will output a logic 1. Below this it's 0

Sensor digital output pins are labelled Do, DOUT, or S. Wire this pin to a free GPIO pin with a female-female jumper wire. The code uses pins 5 and 6 for the line sensors facing the floor and pins 13 and 26 for the obstacle sensors. Mount the sensors, reassemble the robot, and turn it on.

07 Calibrating the sensors

The sensors may have two LEDs: one is for power and always on, and the other for sensing. Take the robot out of direct sunlight. Ensure nothing is in front of the sensors for a couple of metres.

The sensors have a screwdriver dial on them to adjust the light level they switch on at, which translates to the distance.

obstacle_avoid.py

DOWNLOAD THE FULL CODE:
magpi.cc/QkkmqW

› Language: **Python 3**

```
001. from signal import pause
002. import atexit
003. import gpiozero
004. from gpiozero.tools import scaled, negated
005.
006. robot = gpiozero.Robot(left=(27, 17), right=(24, 23))
007. left_obstacle_sensor = gpiozero.DigitalInputDevice(13)
008. right_obstacle_sensor = gpiozero.DigitalInputDevice(26)
009. # Ensure it will stop
010. atexit.register(robot.stop)
011.
012. robot.right_motor.source = scaled(
     left_obstacle_sensor, -1, 1)
013. robot.left_motor.source = scaled(
     right_obstacle_sensor, -1, 1)
014.
015. pause()
```

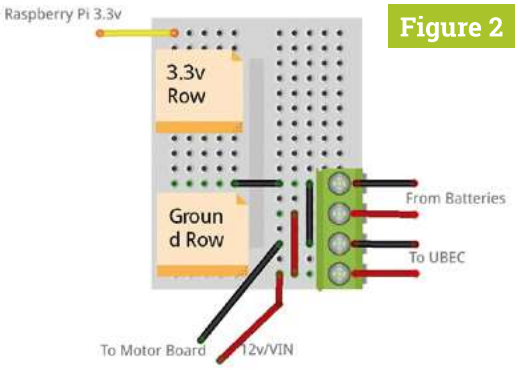


Figure 2

▲ **Figure 2** The power board wiring prepares a power rail for 3.3 V and a rail for GND to connect the sensors to

Put a reflective obstacle about 10 cm in front of a sensor. Turn the dial slowly to a point the LED turns off. Turn it a tiny way back and it should be on. Wave the obstacle back and forth and watch the LED changing.

08 Sensors in GPIO Zero

GPIO Zero sensors have a value for code to read their state, which a loop could use to set motor values. However, there is a different way to use it.

GPIO Zero has a smart source/value system. A sensor input device can be a 'source' for an output device like a motor, sending a continuous stream of data. So sensor inputs can be virtually wired to affect output, eliminating the loop.

The source/value system has tools to manipulate the data like scaling, negating, or delaying it. Mathematical functions like a sine wave can also be sources. See **magpi.cc/hPmwrv** for more ways to use this.

> "A sensor input device can be a 'source' for an output device like a motor, sending continuous data"

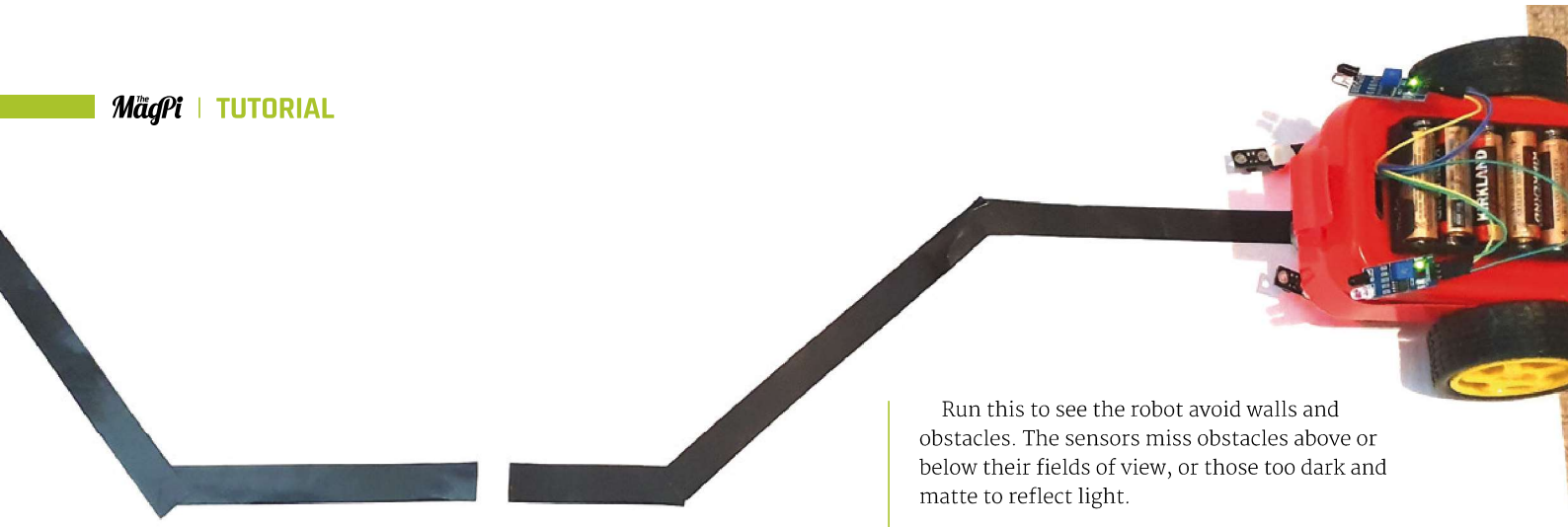
09 Driving with sensors

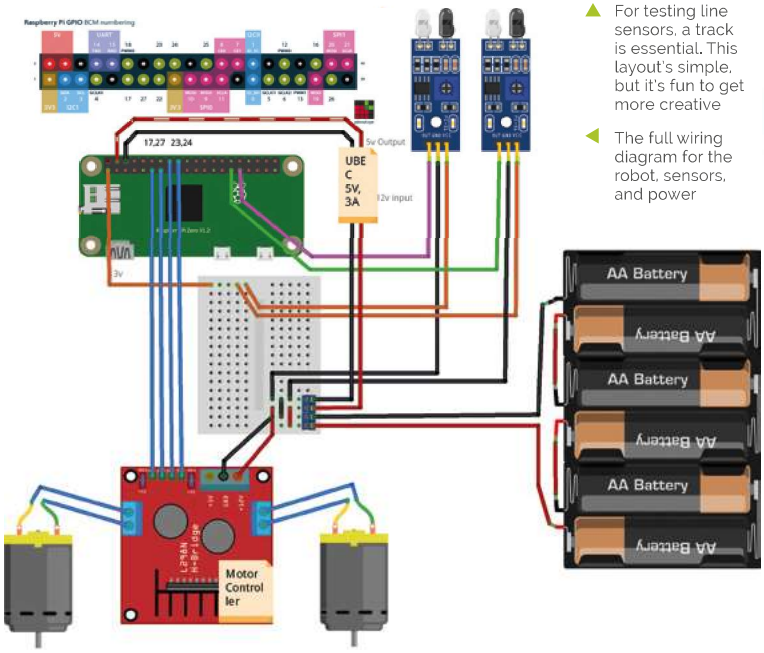
The **obstacle_avoid.py** code makes a `DigitalInputDevice` for each sensor.

Line 10 registers `motor.stop` with the 'atexit' system, guaranteeing when the code stops, for any reason, the motors stop too.

Lines 12 and 13 wire the sensors to the opposite motors, so the robot will turn away from any detected object.

The code scales the sensor values 0 (obstacle detected) and 1 (clear) into motor speed values of -1 (reverse) and 1 (go forward).





▲ For testing line sensors, a track is essential. This layout's simple, but it's fun to get more creative

◀ The full wiring diagram for the robot, sensors, and power

follow_line.py

› Language: **Python 3**

```
001. from signal import pause
002. import atexit
003. import gpiozero
004. from gpiozero.tools import scaled, negated
005.
006. robot = gpiozero.Robot(left=(27, 17), right=(24, 23))
007. left_line_sensor = gpiozero.LineSensor(5)
008. right_line_sensor = gpiozero.LineSensor(6)
009. # Ensure it will stop
010. atexit.register(robot.stop)
011.
012. robot.left_motor.source = scaled(negated(left_line_
     sensor), -0.3, 0.4)
013. robot.right_motor.source = scaled(negated(right_line_
     sensor), -0.3, 0.4)
014.
015. pause()
```

Run this to see the robot avoid walls and obstacles. The sensors miss obstacles above or below their fields of view, or those too dark and matte to reflect light.

10 Line following

Using sheets of plain white paper (A4 to A1) and black tape (about 20 mm wide), make a single line along the middle of a sheet.

Create a small calibration square of about 40 mm, and put a strip of tape across one end.

Make some curved sections on other paper sheets; keep the turns to less than 45 degrees, and the lines no closer than a robot width apart.

The robot drives forward until the sensor encounters a line. You can use this to make a crossing, by leading tracks to a gap from both horizontal and vertical directions

11 Calibrating line following

Calibrate the line sensors in a similar way to the obstacle sensors. It may be easier to detach the sensor for calibration and reattach it afterwards depending on where the dial is.

Using the calibration square, hold a white area about 2 cm from a line sensor. Slowly turn the sensor dial until the LED changes – wave the paper between black and white to observe the LED changing, and adjust if needed.

You'll need to recalibrate for different lighting or surface conditions.

12 Line following code

The **follow_line.py** code is similar to obstacle avoiding.

Lines 7 and 8 set up GPIO Zero line sensors (based on digital input) on the correct pins.

The line sensors straddle the track. When the sensor detects white (sending 0), it's not crossing the line and so the motor goes forward.

Lines 12 and 13 connect the sensor output so a motor reverses when its sensor crossed the line onto black tape (sending 1). The input source is negated to make 0 the move forward condition.

So that the robot responds to the track before driving past it, scale the source data to go from -0.3 to 0.4.

Part 01

Hack GraviTrax with Raspberry Pi

Make spectacular marble runs by triggering sounds and animations from a Raspberry Pi

MAKER

Mike Cook

Veteran magazine author from the old days, writer of the Body Build series, plus co-author of *Raspberry Pi for Dummies*, *Raspberry Pi Projects*, and *Raspberry jPi Projects for Dummies*.

magpi.cc/TPaUfT

GraviTrax is a construction system to make your own marble runs of any complexity. From the famous toy maker Ravensburger, it is aimed at the STEM market as well as kids of all ages. It has lots of extension packs for more basic parts and new fun features. There is only one thing missing, until now: an interface to Raspberry Pi.

We have always wanted to do a marble run project, so when we saw the GraviTrax system we were excited because it had all the ingredients we needed. Also, it is built up on a grid of hexagon spacing, and we are suckers for hexagons. The standardised parts, and wide availability, meant we could make our own parts, or modify existing parts, to feed back events into the Raspberry Pi and have them trigger sounds or animations. We can also trigger lights to make our run more exciting. In this first part, we will concentrate on detecting the balls.

01 The GraviTrax system

When you get your starter set, you have to prepare the cardboard base by pushing out hexagons to leave holes to mount the tiles in.

Do not discard these hexagons, because we are going to use them in our project. If you have already discarded them, then you'll need to cut out a hexagon from either cardboard of the same thickness or 3 mm plywood. We made two types of sensor: a standalone one, and ones where you have to modify an existing part. Just like the GraviTrax system, our approach is modular and you can make as many of each sensor type as you like.

02 How do they work?

All the sensors in this part use light to detect the presence of a ball – through breaking a beam, detecting reflected light, or simply the presence or absence of ambient light. As such, the circuits are all very similar as shown in **Figure 1**; the only difference is the physical sensor used, and the current limiting resistor value on the LED. They are connected to the Raspberry Pi by a length of three-wires-wide ribbon cable, with a pull-up resistor going on the board that connects to the GPIO pins. In later parts of this tutorial series, we will look at making a distribution board for them all.

▼ **Figure 1** A universal schematic of the optical sensors

Figure 1

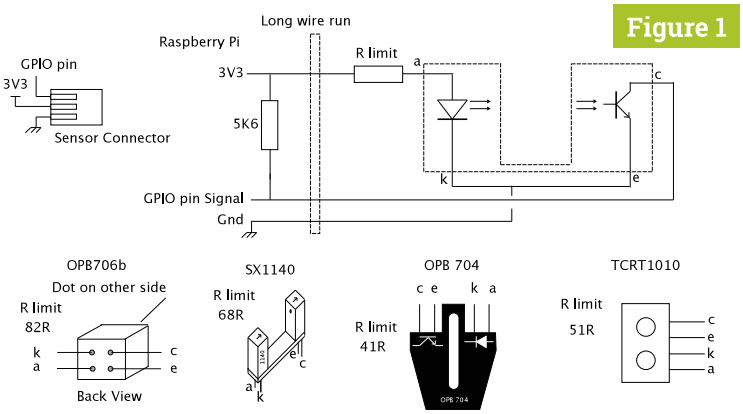
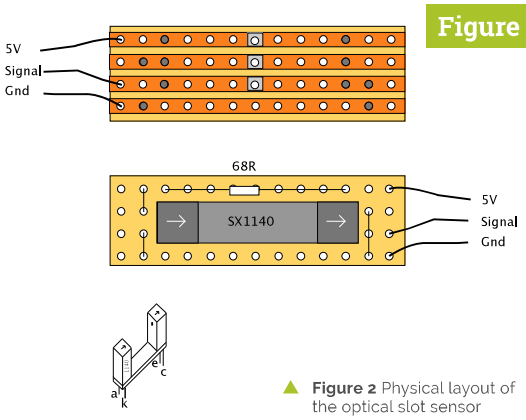


Figure 2



▲ **Figure 2** Physical layout of the optical slot sensor

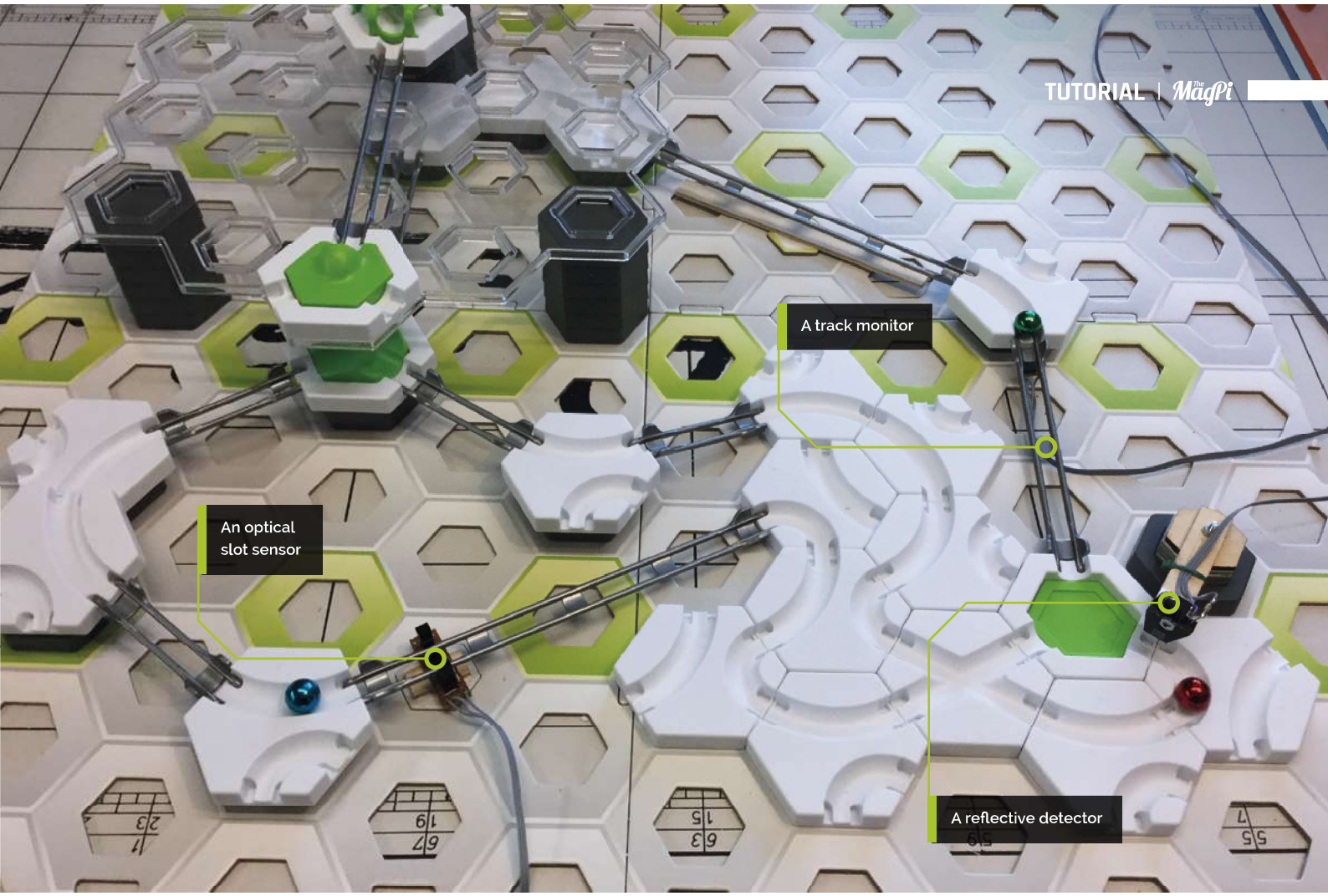
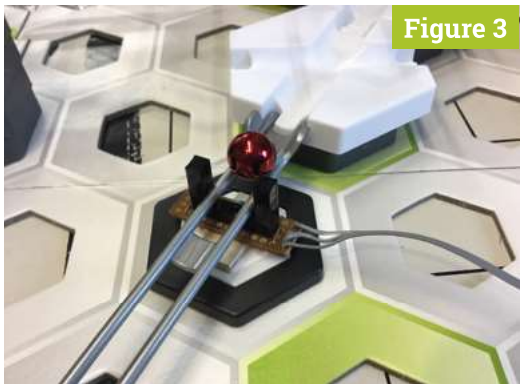


Figure 3



▲ **Figure 3** The optical slot sensor in action

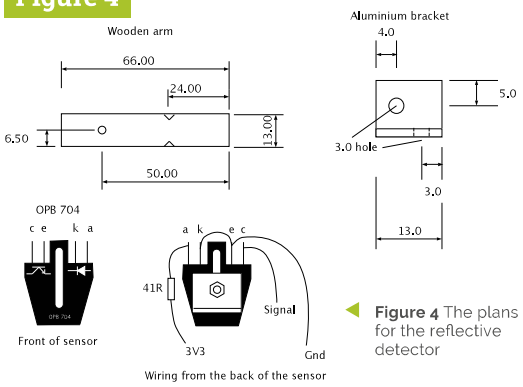
03 Optical slot sensor

The first sensor we will look at is an optical slot sensor. This fits under a track and the height can be adjusted by using the small or large height tiles. Basically, this is a cardboard hexagon with a small piece of 3 mm plywood stuck on it and the electronics glued to that, going from point to opposite point on the hexagon. The physical construction of the electronics board is shown in **Figure 2**, with the whole assembly shown in **Figure 3**. It is best to glue the electronics in place with a track running through it, between two pieces, so you can get it aligned precisely.

04 Reflective detector

The previous sensor needs to be placed under a track, whereas this one can be placed in an adjacent space and it looks downwards. It is capable of detecting balls on the entrance or exit to any of the tiles, be it basic, curved, or launch pad. The sensor is easy to make, as the LED and transistor symbols are drawn on the part. We used a stack of five cardboard hexagons glued together, followed by a plywood hexagon with a hole in the centre and a 10 mm, M3 countersunk screw sticking through it. This was then placed on the arm, and could be rotated to the required point.

Figure 4



◀ **Figure 4** The plans for the reflective detector

Top Tip

GraviTrax simulator

You can get a free GraviTrax simulator to run on your mobile device. You can make layouts and see the results with a variety of effects, even from a 'balls eye' view of the run.

You'll Need

- GraviTrax – Starter set **magpi.cc/LusGgA**
- ESSX1140 opto slot **magpi.cc/nQNQNW**
- OPB704 Reflective opto sensor **magpi.cc/fwUrdD**

Top Tip

Track monitor

When gluing the PCB to the track, glue the long edge first, then push the sensor slightly so it is not parallel to the track, and test that it detects the ball before gluing the sides to hold it into position.

Figure 5

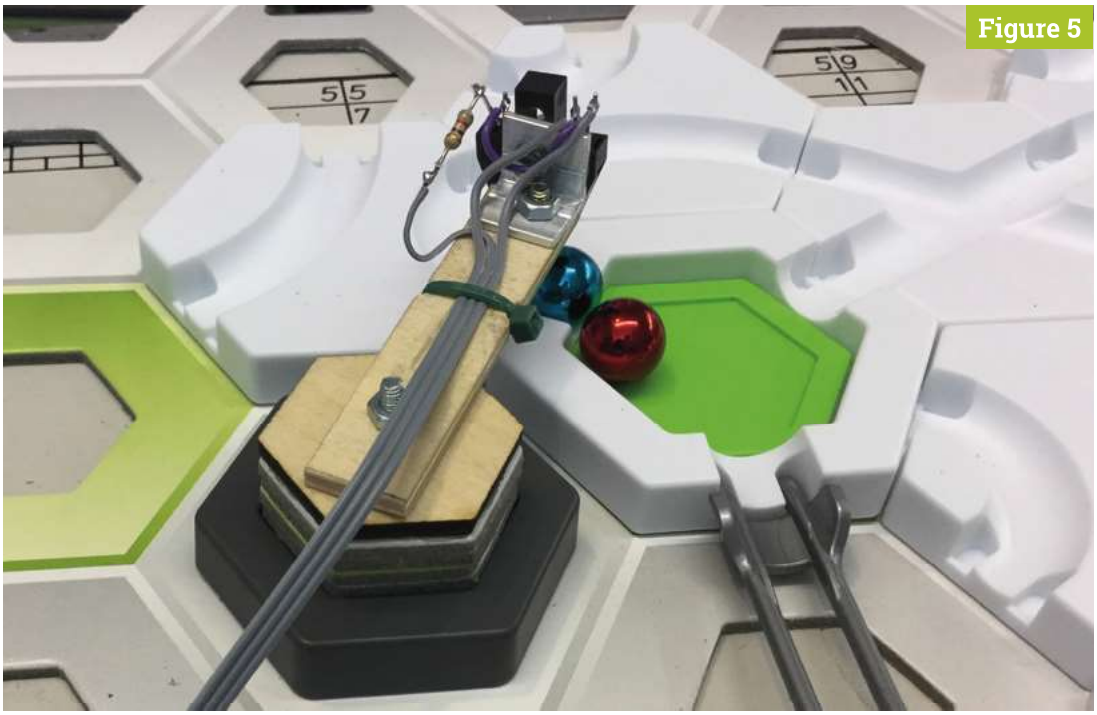


Figure 5 The reflective detector in action

05 Making the reflective detector

Figure 4 shows the measurements of the plywood arm, and the mounting bracket made from a piece of 12×12 mm angle aluminium. The arm should be notched with the corner of a square file; this is so a cable tie can grip the ribbon cable without slipping. The height of the sensor is adjustable due to a long slot in it, allowing it to be slid up and down. You need a gap of about 2 mm between the sensor and the top of a ball to detect it reliably (**Figure 5**). You can mount two arms on a stack to get coverage of another position.

06 A track monitor

The last two sensors have been standalone; the next one modifies a GraviTrax part. The physical layout is shown in **Figure 6**. The TCRT1010 sensor comes with bent leads which neatly wrap round two holes of stripboard. On the strip side, you need to bend the connectors over again and cut them short. Then bend the middle two so they sit on one track and solder them up. You also need a surface-mount 51Ω resistor to make it small, although a 0.25 W through-hole resistor could be used. The assembly needs hot-melt gluing onto the track (**Figure 7**).

Figure 6

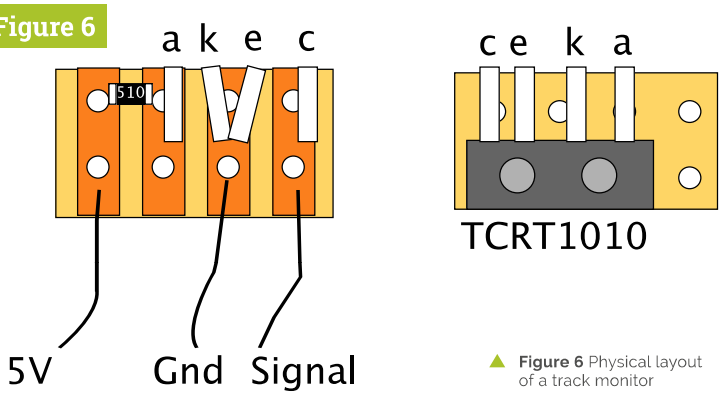


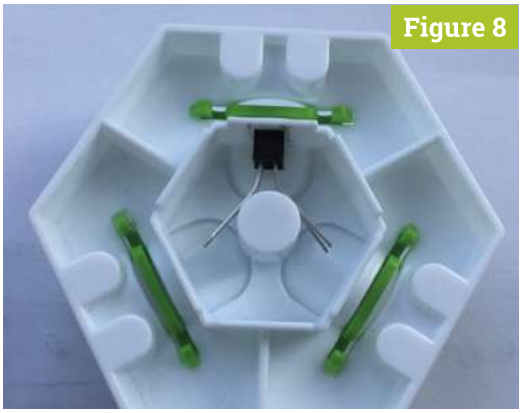
Figure 6 Physical layout of a track monitor

07 Launch pad monitor

Take an OPB706B sensor and wrap it round the centre post in the launch pad tile (**Figure 8**). Push it down and use a pencil to mark the outline on the wall. Then, using a Dremel, and 1 mm router bit, cut to about 1 mm short of the outline you drew, so the part does not go through the hole. Also, cut a slot in the opposite side to let the wires through when attached to a height tile. Paint black the area the sensor is pointing at, and glue a 10×2 mm piece of 1 mm thick styrene to the green plunger with polystyrene glue (**Figure 9**).

08 Switch monitor

This uses ambient light to detect which way the switch is set. Draw in pencil around the switching lever in both positions, showing the area being covered and uncovered. Then drill a 2 mm hole in the middle of this area (**Figure 10**). Next, paint



▲ **Figure 7** The track monitor in action

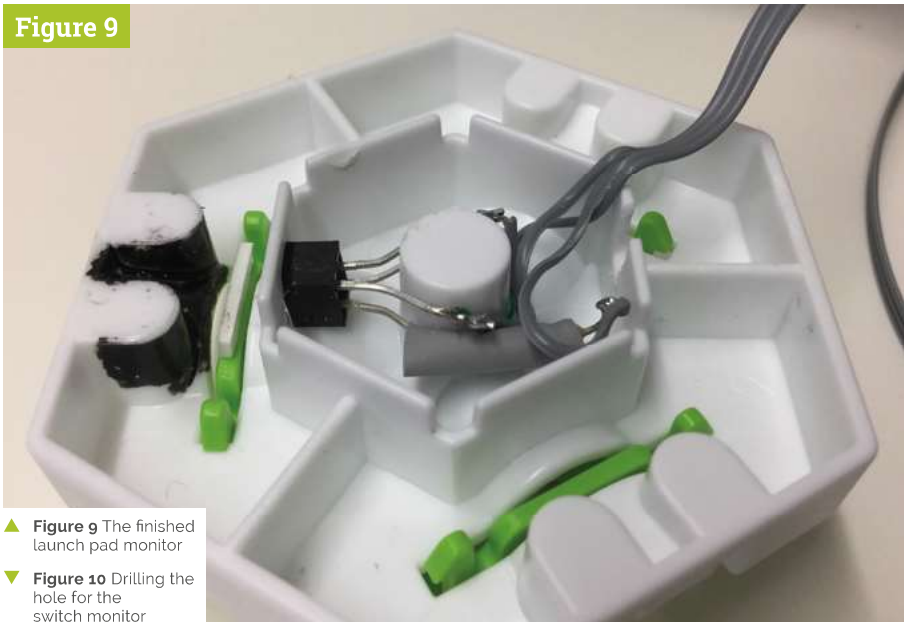
▲ **Figure 8** Marking the hole outline on the launch pad

the underside of the switch tile black, and make sure you paint on the inside of the hole (**Figure 11**). The schematic of this sensor is somewhat different from the others and is shown in **Figure 12**. Finally, **Figure 13** shows the physical layout of the parts. Position the board so you can see the white sensor through the hole and fix with Sugru.

> "The GPIO pins are scanned and when a trigger condition is met"

09 Software

We have written software, shown in the **sound_trigger.py** listing, that monitors these sensors, and triggers sounds, either immediately or after a delay. The software is modular: line 82 determines what GPIO pins you will use and it automatically generates a window size to accommodate the number of pins in this list. Note if you want to use GPIO 14, you should disable SPI before booting with the sensor attached. The sounds' names are in a list at line 88; by simply changing or adding to this list, different sounds can be used. The GPIO pins are scanned and when a trigger condition is met, the event is put in a pending list to be actioned at the correct time.

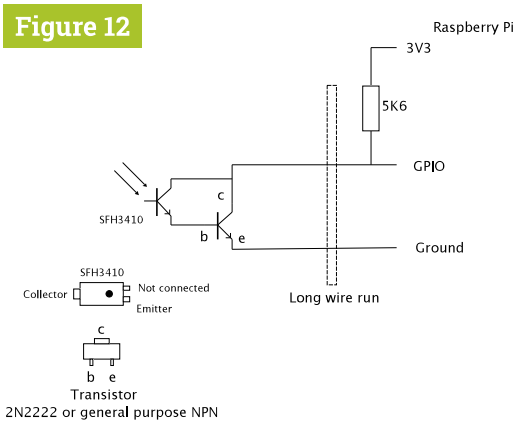


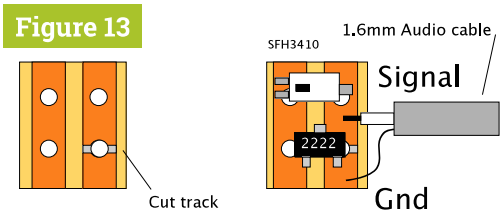
▲ **Figure 9** The finished launch pad monitor

▼ **Figure 10** Drilling the hole for the switch monitor

▲ **Figure 11** Painting parts of the switch tile black

◀ **Figure 12** Schematic of the switch monitor





◀ **Figure 13** Physical layout of the switch monitor

▲ **Figure 14** The software user interface

10 Using the software

The user interface is shown in **Figure 14**. For each line, you can set what the trigger action will be. These states are: disabled, when the signal goes high, when it goes low, or when it goes either high or low. They are changed by clicking on the trigger icon. The delay column, as you might expect, determines a delay between the trigger and the sound, whereas the sound sample played can be changed by the icons on the right. You can change the GPIO pin and note that one pin can trigger different actions; so, for example, you could have the switch tile generating a different sound depending if it is changed to the left or right.

11 Choosing sounds

We found short sounds were generally best, but longer sounds can be useful at the beginning or end of your run. We copied a lot of sounds from the Scratch media library from the path **/usr/share/scratch/Media/Sounds** into our **sounds** directory. Make sure that are all .wav files, because that suffix gets added automatically to the file names. Note that the slot sensor will read as a logic zero with no ball, whereas a reflective sensor will read high in the absence of a ball. There are lots of suitable sounds available online as well.

We have looked at adding optical sensors to detect where a ball is, be it on a track or a tile. Next month we will look at how to add different sorts of LED displays to enhance your GraviTrax layout. In the meantime, if GraviTrax is new to you, then have a play with the different layouts in the accompanying booklet.

sound_trigger.py

› Language: **Python 3**

```
001. #!/usr/bin/env python3
002. # GraviTrax Sound Trigger
003. # By Mike Cook September 2019
004.
005. import time
006. import pygame
007. import os
008. import RPi.GPIO as io
009.
010. pygame.init()
011. pygame.display.set_caption("GraviTrax Sound Trigger")
012. os.environ['SDL_VIDEO_WINDOW_POS'] = 'center'
013. pygame.mixer.quit()
014. pygame.mixer.init(frequency = 22050, size =- 16, channels = 2,
     buffer = 512)
015. pygame.event.set_allowed(None)
016. pygame.event.set_allowed([pygame.KEYDOWN, pygame.QUIT, pygame.
     MOUSEBUTTONDOWN,
017.                          pygame.MOUSEBUTTONUP]
018.                         )
019. textHeight=18
020. font = pygame.font.Font(None, textHeight)
021. backCol = (160, 160, 160) ; lineCol = (128, 128, 0)
022. hiCol = (0, 255, 255)
023.
024. def main():
025.     global screen, lastIn, rows
026.     initIO()
027.     rows = len(inPins)
028.     screen = pygame.display.set_mode([390, 34 + 40*rows],
     0, 32)
029.     init() ; pendPlay = [0]*rows
030.     nowIn = [0]*rows; pendTime = [0.0]*rows
031.     drawScreen()
032.     while True:  # repeat forever
033.         checkForEvent()
034.         for i in range(0, rows):
035.             nowIn[i] = io.input(inPins[inPin[i]])
036.             if lastIn[i] != nowIn[i]:
037.                 lastIn[i] = nowIn[i]
038.                 tmatch = trigNum[i]-1  # match trigger
039.                 if tmatch == 2:
040.                     tmatch = nowIn[i]
041.                 if trigNum[i] != 0 and nowIn[i] == tmatch:
042.                     pendPlay[i] = soundFX[soundNumber[i]]
043.                     pendTime[i] = time.time() + delayTime[i]
044.         for i in range(0, rows):  # check what to play now
045.             if pendTime[i] > 0.0 and time.time()>=pendTime[i]:
046.                 pendPlay[i].play() ; pendTime[i] = 0.0
```

DOWNLOAD THE FULL CODE:
magpi.cc/dhaAam

```
047.
048. def init():
049.     global incRect, decRect, icon, decRect, voiceRect
050.     global inPin, soundNumber, delayTime, triggerRect
051.     global lastIn, trigNum, trigIcon
052.     lastIn = [0]*rows
053.     loadResources()
054.     icon=[pygame.image.load(
"icons/"+str(i)+".png").convert_alpha()
055.                             for i in range(0,2)
056.           ]
057.     incRect = [pygame.Rect((0,0),(15,15))]*rows*3
058.     decRect = [pygame.Rect((0,0),(15,15))]*rows*3
059.     for j in range(0,3):
060.         for i in range(0, rows):
061.             incRect[i+j*rows] = pygame.Rect((76 + j*80,
30 + i*40),(15, 15))
062.             decRect[i+j*rows] = pygame.Rect((76 + j*80,
50 + i*40),(15, 15))
063.     triggerRect = [pygame.Rect((0, 0), (20, 20))]*rows
064.     trigNum = [0]*rows
065.     trigIcon = [pygame.image.load(
"icons/trig"+str(i)+".png").convert_alpha()
066.                                   for i in range(0,4)
067.                 ]
068.     voiceRect = [pygame.Rect((0,0), (15,15))]*rows
069.     for i in range(0, rows):
070.         triggerRect[i] = pygame.Rect((10, 36 + 40*i,20,
20))
071.         voiceRect[i] = pygame.Rect((268, 39 +
i*40),(100, 20))
072.     sounds = rows + len(soundNames)
073.     inPin = [1]*rows ; soundNumber = [0]*sounds
074.     for i in range(0, rows):
075.         inPin[i] = i
076.     for i in range(0, len(soundNames)):
077.         soundNumber[i] = i
078.     delayTime = [0.0]*rows
079.
080. def initIO():
081.     global inPins
082.     inPins = [24, 23, 22, 27, 17, 4, 15, 14]
083.     io.setmode(io.BCM); io.setwarnings(False)
084.     io.setup(inPins, io.IN, pull_up_down = io.PUD_UP)
085.
086. def loadResources():
087.     global soundFX, soundNames
088.     soundNames = ["owl", "Breaking Glass",
"ComputerBeeps1",
089.                   "CymbalCrash", "Fairydust", "Dog1",
"Zoop", "Ya", "Pop"
090.                   ]
091.     soundFX = [pygame.mixer.Sound(
"sounds/"+ soundNames[effect]+".wav")
092.                                   for effect in
range(0,len(soundNames))
093.                ]
094.
095. def drawScreen():
096.     screen.fill(backCol)
097.     for i in range(0,len(incRect)):  # increment /
decrement icons
098.         screen.blit(icon[0], (incRect[i].
left,incRect[i].top))
099.         pygame.draw.rect(screen, lineCol, incRect[i],1)
100.         screen.blit(icon[1], (decRect[i].left,
decRect[i].top))
101.         pygame.draw.rect(screen, lineCol, decRect[i],
1)
102.     for i in range(0,rows):  # draw all triggers
103.         screen.blit(trigIcon[trigNum[i]],
(triggerRect[i].left,
104.                                       triggerRect[i].top)
105.                     )
106.     drawWords("Trigger", 5, 8, (0, 0, 0), backCol)
107.     drawWords("GPIO", 70, 8, (0, 0, 0), backCol)
108.     drawWords("Delay", 138, 8, (0, 0, 0), backCol)
109.     drawWords("Sound", 218, 8, (0, 0, 0), backCol)
110.     updateValues()
111.
112. def updateValues():
113.     for i in range(0,rows):
114.         drawWords(str(inPins[inPin[i]]) + "    ", 48,
 39 + i*40, (0, 0, 0),
115.                   backCol
116.                   )
117.         drawWords("   " + str(round(delayTime[i], 1)) +
"    ", 112, 39 + i*40,
118.                   (0, 0, 0), backCol
119.                   )
120.         pygame.draw.rect(screen, backCol, voiceRect[i],
0)
121.         drawWords(str(soundNames[soundNumber[i]]), 270,
39 + i*40, (0, 0, 0),
122.                   backCol
123.                   )
124.     pygame.display.update()
125.
```

sound_trigger.py (continued)

Language: **Python 3**

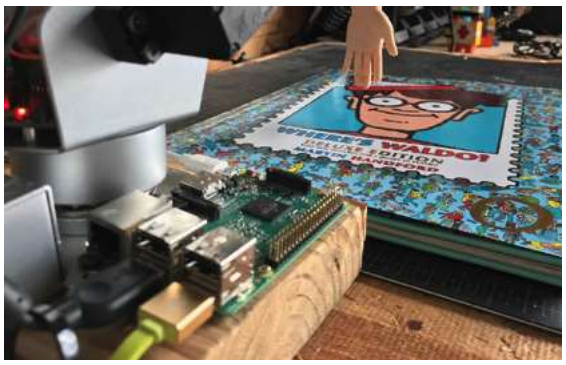
```
def drawWords(words, x, y, col, backCol) :
    textSurface = font.render(words, True, col, backCol)
    textRect = textSurface.get_rect()
    textRect.left = x  # right for align right
    textRect.top = y
    screen.blit(textSurface, textRect)
    return textRect

def handleMouse(pos):  # look at mouse down
    global pramClick, pramInc, trigClick
    #print(pos)
    trigClick = -1
    for i in range(0, rows):
        if triggerRect[i].collidepoint(pos) :
            trigClick = i
            pygame.draw.rect(screen, hiCol, triggerRect[i], 0)
            pygame.display.update()
    pramClick = -1
    pramInc = 0
    for i in range(0, len(incRect)):
      if incRect[i].collidepoint(pos):
         pramClick = i ; pramInc = 1
         pygame.draw.rect(screen, hiCol, incRect[pramClick], 1)
         pygame.display.update()
    for i in range(0, len(decRect)):
        if decRect[i].collidepoint(pos):
            pramClick = i ; pramInc = -1
            pygame.draw.rect(screen, hiCol, decRect[pramClick], 1)
            pygame.display.update()

def handleMouseUp(pos):  # look at mouse up
    global soundNumber, delayTime, inPin
    if trigClick != -1:
     trigNum[trigClick] += 1
     if trigNum[trigClick] > 3:
         trigNum[trigClick] = 0
     pygame.draw.rect(screen, backCol, triggerRect[trigClick], 0)
     screen.blit(trigIcon[trigNum[trigClick]], (triggerRect[trigClick].left,
         triggerRect[trigClick].top))
     updateValues()
    if pramClick != -1:
        if pramClick < rows:  # GPIO Coloumn
            inPin[pramClick] += pramInc
            inPin[pramClick] = constrain(inPin[pramClick], 0, rows-1)
        elif pramClick < rows*2:  # Delay Coloumn
            delayTime[pramClick-rows] += (pramInc / 10)
            delayTime[pramClick-rows] = constrain(delayTime[pramClick - rows],
                                                  0, 5
                                                  )
            if delayTime[pramClick - rows] < 0.01:
                delayTime[pramClick - rows] = 0
        elif pramClick < rows*3:  # Sound coloum
            soundNumber[pramClick - rows*2] += pramInc
            soundNumber[pramClick - rows*2] = constrain(soundNumber[pramClick
                - rows*2], 0, len(soundNames)-1)
        if pramInc !=0:
            if pramInc < 0:
                screen.blit(icon[1], (decRect[pramClick].left,
                    decRect[pramClick].top))
                pygame.draw.rect(screen, lineCol, decRect[pramClick],1)
            else:
                screen.blit(icon[0], (incRect[pramClick].left,
                    incRect[pramClick].top))
                pygame.draw.rect(screen, lineCol, incRect[pramClick], 1)
            updateValues()

def constrain(val, min_val, max_val):
    return min(max_val, max(min_val, val))

def terminate():  # close down the program
    pygame.mixer.quit()
    pygame.quit()  # close pygame
    os._exit(1)

def checkForEvent():  # see if we need to quit
    event = pygame.event.poll()
    if event.type == pygame.QUIT :
        terminate()
    if event.type == pygame.KEYDOWN :
        if event.key == pygame.K_ESCAPE :
            terminate()
    if event.type == pygame.MOUSEBUTTONDOWN :
        handleMouse(pygame.mouse.get_pos())
    if event.type == pygame.MOUSEBUTTONUP :
        handleMouseUp(pygame.mouse.get_pos())

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

10 Best:

AI projects

Bring your Raspberry Pi to artificial life with these machine learning projects

Machine learning and AI are just a normal part of the world now, which in some ways is kind of hard to process. On the plus side, it means we can have computers do really fun, useful (and useless) stuff for us. Here are ten ways to get your Raspberry Pi to learn and do.



▲ There's **Waldo!**

Robot cartoon-hunter

Waldo – or Wally as we call him in the UK – is a very elusive man who likes to travel around the world. The puzzle books asking young folks to find Wally in a busy crowd of people are very popular and can be tricky to solve; that is, unless you're an AI.

▶ **magpi.cc/gApiTp**

▼ Formula **Pi**

Self-driving racers

A lot of Raspberry Pi robots aren't autonomous – the Formula Pi racers are, though: using computer vision and your own bits of code, the aim is to make your robot the fastest and most accurate racer.

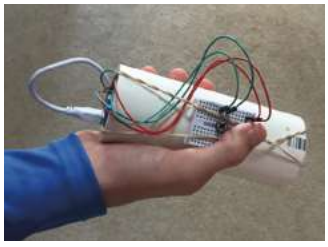
▶ **formulapi.com**

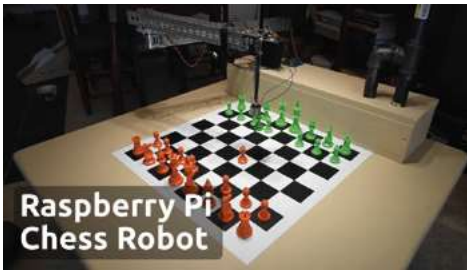
▼ Seeing **wand**

Magical item identifier

This project uses Microsoft's Cognitive Services to look at a picture for identification. When it works, it's pretty magical; however, it doesn't always work. Still, it will then use text-to-speech software to tell you what's in front of you. A future product for blind people, maybe?

▶ **magpi.cc/pfpPwB**





▲ **Raspberry** Turk

Computer chess IRL

The 'Mechanical Turk' was a magic trick where chess players would manipulate mechanical arms to make it look like people were facing a machine that could play chess. The Raspberry Turk is no magic trick – it does it for real.

▶ **raspberryturk.com**

◀ **Cucumber** sorter

Computer-aided vegetable categorisation

One of the promises of AI is that it can help people out with more mundane parts of work. The cucumber sorter allows a farmer to quickly and efficiently categorise his cucumber harvest. We've seen it in action and it is fun.

▶ **magpi.cc/EWpGAp**

▲ UBC Sailbot

Self-driving boat

Using GPS and a series of sensors and motor controllers, the Sailbot is one of a few autonomous sailing-boats that makes use of Raspberry Pi to control itself in races around the world.

▶ **sailbot.org**

◀ Just Keep Swimming

Fish-controlled robot tank

Living in a fish bowl must feel a bit limiting. So Alex Kent decided to allow his goldfish to move with the help of a computer vision project that senses where the fish is swimming, and moves its tank accordingly. Does it notice? Or just forget?

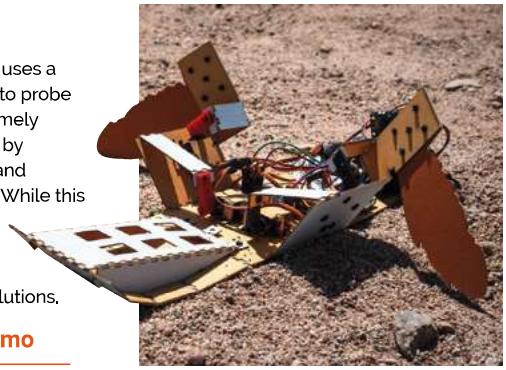
▶ **magpi.cc/ihFKyk**

▼ C-Turtle

Land-mine clearing project

This incredible project uses a low-cost robot design to probe abandoned (and extremely dangerous) minefields by sniffing out the mines and then detonating them. While this does result in each robot's heroic demise, it's much more cost-effective than other solutions.

▶ **magpi.cc/rKHQmo**

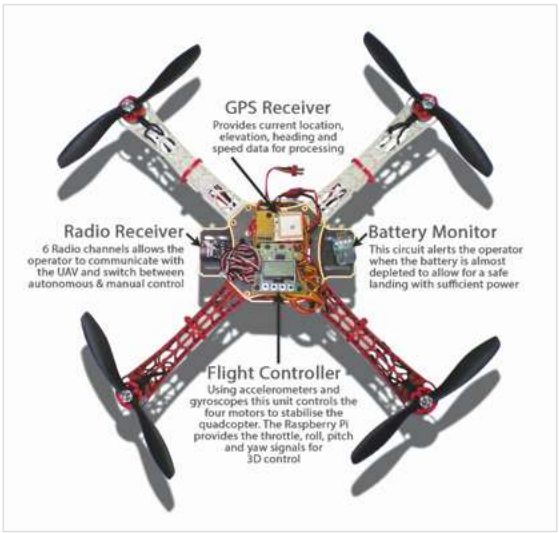


▼ Autonomous Quadcopter

Self-driving drone

This project didn't quite achieve full autonomy for a quadcopter/drone, but it got pretty close. Maybe you can build upon this design and create incredible aerial spectacles with a few drones?

▶ **magpi.cc/ysuieR**

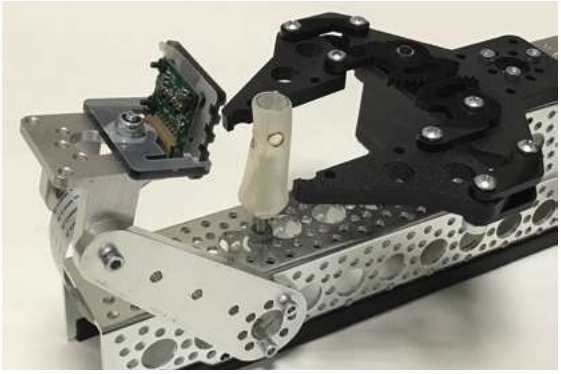


▼ Stent-testing robot

Testing breathable tubes

Stents are little tubes used to keep a patient's airway open. As they are vital, they need to be tested to extremes – this robot is able to control clamps that squish and compress the stent hundreds of thousands of times and monitor if and when it breaks.

▶ **magpi.cc/nivpET**



Learn to code with toys

Play your way to programming prowess. By **Lucy Hattersley**

Turing Tumble

CREATOR

Alyssa and Paul Boswell

Price: £55 / $70

turingtumble.com

Turing Tumble is a vertical marble game, like a Japanese pachinko machine. Blue and red marbles are loaded to the top of the board and roll down the pins, hitting your switches as they go.

The direction of balls is changed via crossovers, ramps, and interceptors. Meanwhile, logic is simulated via bits (arrows that move left or right) and gears (which make the board Turing-complete). An accompanying book has 60 puzzles to solve. This board game has a lot of fans at Raspberry Pi Towers, and you can even pick it up at the Raspberry Pi Store in Cambridge.

Low-cost toys

Low-tech learning that doesn't cost the earth

TOWERS OF HANOI

The Towers of Hanoi puzzle is widely available (or you can easily make your own). It's a great way to think about solving a problem with an algorithm, and is a classic computer program, as shown in this Geeks For Geeks course (**magpi.cc/DXt231**).

magpi.cc/aQesb9

A DECK OF CARDS

You don't need expensive tools to teach coding. A deck of cards can be used to discover conditionals and other programming concepts. Check out this Code.org teaching resource.

magpi.cc/sNMBiq

MATCHBOXES

In the early 1960s Donald Michie, a pioneering British computer scientist, came up with Menace (the Machine Educable Noughts And Crosses Engine). It's a great way to learn ultra-modern AI techniques in a low-cost manner.

magpi.cc/6pw32R

Robo Rally

CREATOR
Richard Garfield

Price:
£40 / $44
magpi.cc/bwKcig

Designed by Richard Garfield of Magic: The Gathering fame and first published in 1994, Robo Rally is a board game where you program robots by lining up cards.

The rules are simple to learn, and it's highly entertaining. The aim is to keep your robot alive in a dangerous factory while trying to ram or shoot other players. Rather than a computer simulating a board game, it's a board game simulating a computer. Each player plays up to five cards per turn, and needs to plan and sequence ahead if they want to win.

Parrot Mambo FLY

CREATOR
Parrot

Price:
£100 / $130
magpi.cc/ARmPhL

Drones are a great toy for learning coding skills, although picking the right one is tricky. Go too high and you'll spend a lot of money with a risk of being 'broken by Boxing Day', but go too low and you'll get a toy without any coding nuances.

Parrot Mambo FLY is our pick of the bunch. It's got an advanced flight controller and sensors to keep it stable when the controls are lost and automatically cuts out the motors on collision.

There's an Android and iPhone app, but pah! Check out the pyparrot API to get coding your drone in Python on a Raspberry Pi (**magpi.cc/xuaKpT**).

For younger ones

Teach them to code from an early age

ROBOT TURTLES

Robot Turtles sneakily teaches little kids to program computers by moving turtles around a board game. Players use 'move' and 'rotate' cards to pick up jewels on the board.
robotturtles.com

BEE-BOT

Bee-Bot is a programmable floor robot controlled via a keypad on its back. It rolls around the floor and can record and play back audio. It moves in 15 cm steps and rotates by 90° turns.
magpi.cc/EGDtQt

CODE-A-PILLAR

This caterpillar robot is reprogrammed by pulling apart and rearranging the eight segments of its body. It's a fun robot that encourages thinking and experimentation in toddlers.
magpi.cc/tiefS6