EVERYDAY ENGINEERING PART 1

SIMON MONK

Simon Monk is the author of the *Raspberry Pi Cookbook* and *Programming Raspberry Pi: Getting Started with Python*, among others.

simonmonk.org
monkmakes.com

BUILD YOUR OWN PARKING SENSORS

You'll Need

- Half-size breadboard
- 7x male-to-female jumper wires
- 2x male-to-male jumper wires
- 4x 470 ohm resistors
- 2x HC-SR04 ultrasonic sensors
- Code: **bit.ly/1KutV7K**

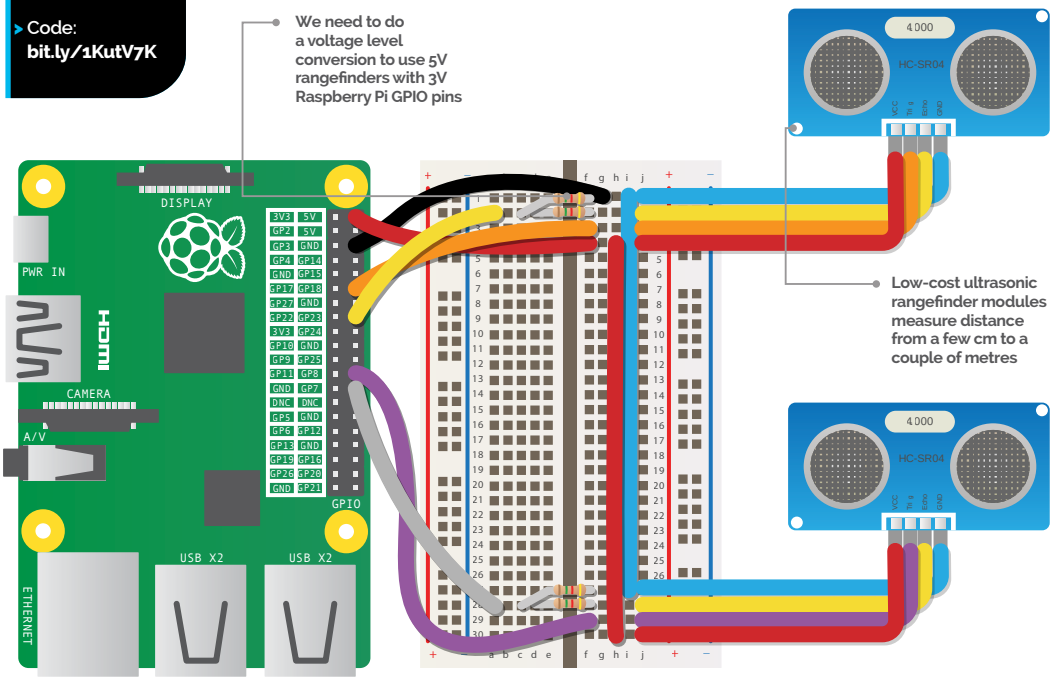
Solve real-world electronic and engineering problems with your Pi and the help of renowned technology hacker and author, **Simon Monk**

Electronics permeates every aspect of modern life and it's easy to take such technology for granted without ever stopping to think just how these things work. Small, low-cost computers like the Raspberry Pi make it possible for hobbyists to put their own take on commercially available products, and also invent new gadgets simply for the fun of it.

In this series, we will be exploring the use of the Raspberry Pi to make all kinds of everyday electronic devices, starting with an ultrasonic parking sensor designed to show you how far the rear corners of your car are from any obstacle.

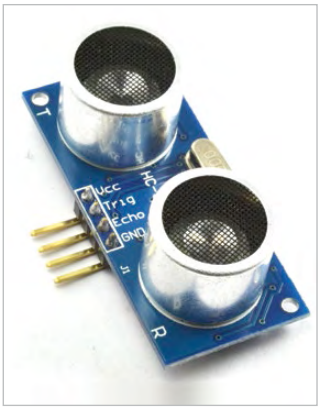
Each of these projects will be constructed using a solderless breadboard and readily-available components, so even if you don't want to develop and install these projects for real, you can prototype them to learn more about engineering and electronic invention.

As you'll see from the list of required components nearby, our first project uses two low-cost ultrasonic rangefinders to measure the distance from the sensor to any obstacle in its path.



While you could attach the rangefinders to your car bumpers, with a sensor at each of the two rear corners of the car and a display positioned so that it is visible from the driver's seat, you could also place the sensors on the wall of your garage (assuming yours is not full of rubbish), so that the display can guide you in and tell you when to stop.

The distance to any obstacle for the right and left sensors is indicated by a rectangle that extends further down the screen as the distance to an obstacle increases. In addition, the actual distance to the obstacle is displayed in cm



Left The HC-SR04 Ultrasonic Rangefinder. Cheap-as-chips sonar

and the rectangle is colour-coded: red if closer than 30cm, green if greater than 100cm, and orange if it's in between.

The ultrasonic rangefinders are of the type that you can buy on eBay, sometimes for less than a pound. These sensors are often used in robot projects to detect obstacles. They use pulses of sound waves to measure the distance to an obstacle over a range of a few cm to several metres. Just search for HC-SR04 and remember to order two.

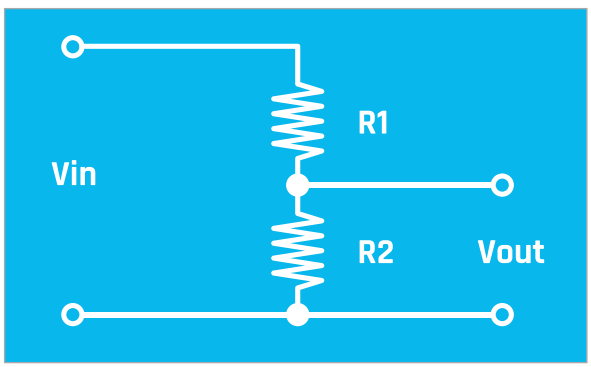
The HDMI display is only needed if you plan to install the project for real in your car or garage; otherwise, you can just use your usual Raspberry Pi monitor. Again, you will find mini HDMI displays at a very reasonable price on eBay. The model we used had a 7-inch display and separate controller board. Look for a display that will operate from 12V if you are going to connect it to your car.

The other parts are probably best bought as an electronics starter kit. The Monk Makes Electronic Starter Kit for Raspberry Pi includes the breadboard and all the parts and wires except the rangefinders. Most starter kits for the Raspberry Pi will include the breadboard, jumper wires and some resistors.

Why use four resistors?

A Raspberry Pi's GPIO pins operate at 3.3V, whereas the rangefinder module's pins operate at 5V. This does not cause a problem when connecting the output of the Raspberry Pi to the input of the rangefinder (for example, a GPIO output on the Raspberry Pi to the Trigger input on the rangefinder) because even though the voltage at the input is a bit low, at 3.3V it will still be high enough to activate the trigger input.

The problem arises when you are going in the opposite direction and the 5V Echo output of the rangefinder needs to connect to a GPIO input on the Raspberry Pi. Putting 5V on a GPIO pin only designed for a maximum of 3.3V could easily damage the pin. Therefore, we use an arrangement of two resistors to reduce this voltage from 5V to 2.5V, where it will still be high enough to register as a high input, but still be well below the maximum of 3.3V.



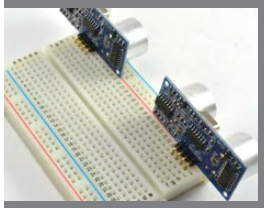
Above 5V in, 3V out – protecting your Raspberry Pi's GPIO pins

More advanced readers may prefer to use different combinations of resistor to set the voltage a bit closer to 3.3V, but the advantage of just halving the voltage is that all four resistors can be of the same value.

Building your parking sensor

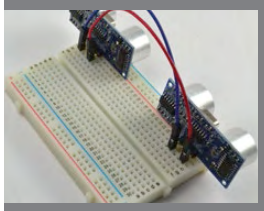
Even if you plan to install the project for real, it's a good idea to start with the rangefinders plugged directly into the breadboard, with the ultrasonic transducers pointing outwards. This will allow you to experiment with the project and make sure that everything is working as it should be, before you commit to some more permanent setup.

PROTOTYPING THE PROJECT



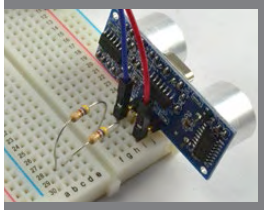
>STEP-01
Fit the rangefinders onto the breadboard

We consructed the prototype build of this project in five steps. First, plug the rangefinders into the breadboard holes at the far ends of the breadboard, as shown in the picture.



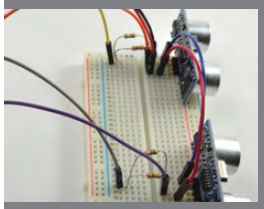
>STEP-02
Join the power connections

Use a male-to-male jumper wire to connect the 5V (labelled Vcc on the rangefinder) pins together, by plugging the jumper wires into the same row as the pins. Do the same thing for the GND connection.



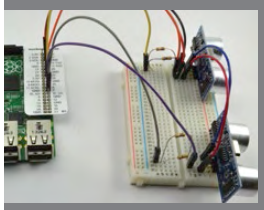
>STEP-03
Add the resistors

Push the leads of the four resistors into the breadboard, as shown in the diagram on the left page. It doesn't matter which way around they go, but be careful that the leads don't touch each other.



>STEP-04
Add the male-to-female jumper wires

Attach the male-to-female jumpers that will connect the breadboard to your Pi. Colour-coding the leads will help you to identify which connection is which when you attach it to your Raspberry Pi.



>STEP-05
Link the breadboard to the Raspberry Pi

Finally, connect all the female ends of the headers to the GPIO pins on your Pi. Working out which can be tricky, but you could use a template like the Raspberry Leaf or the Pi GPIO Reference Board.

Now that the hardware side of the project is done, we need to get the software running. The program is written in Python, using the Pygame library to provide graphics. You can download the program from the internet by typing the following into the command prompt:

```
git clone https://github.com/simonmonk/pi_magazine.git
```

> The Python code for this project is very well commented on GitHub

The program has a graphical user interface, so to run the program, the windowing system must be running. If your Pi is not set to automatically boot into the windowing system, then type the following command to start it up after you have logged in:

```
startx
```

Open an LXTerminal window and type the following commands into it to run the program:

```
cd pi_magazine
sudo python 01_parking_sensor.py
```

After a short delay, the Pygame window will appear.

Try putting your hands in front of each sensor in turn to make sure they are both working okay. If one isn't, check over your wiring carefully.

How the code works

The Python code for this program is very well commented on GitHub (**bit.ly/1KutV7K**), so you'll probably find it handy to have the code up in an editor while we go through it.

The program starts by importing the Python libraries that will be used, and some constants for the GPIO pins. So, if you wanted to swap things around and use different pins, you could just change the numbers to the right of the equals signs. Variables are also used to define the colours that will be used in the user interface by Pygame.

After this, we have some code that initialises the four GPIO pins we want to use. Two of them are set to be outputs, so that they can send out a pulse that causes the rangefinders to send out an ultrasound 'ping'. The other two pins (the 'echo' pins) are set to be inputs, as we need to be able to read them in the program so that we know when the echo has returned, and therefore how long the delay was, so we can calculate distance.

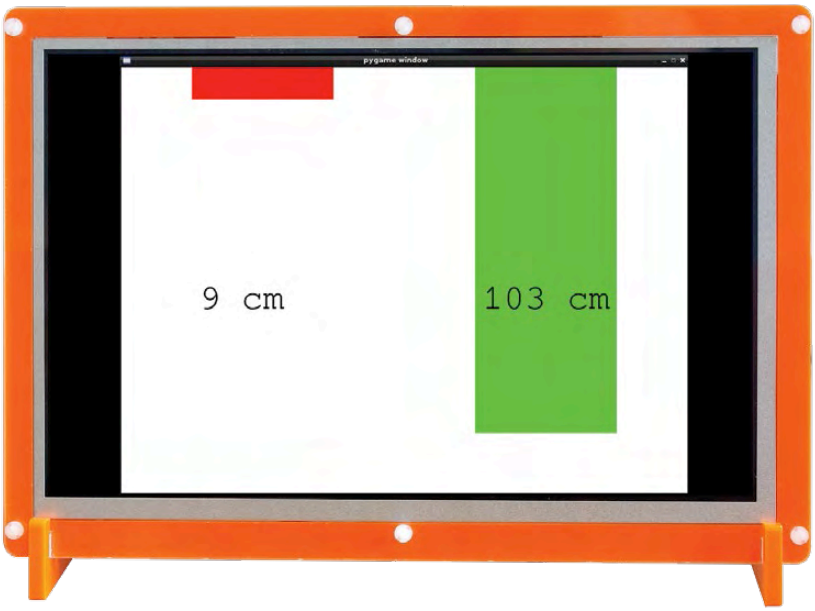
The next three functions contain all the code relating to measuring distance using an ultrasonic rangefinder. The first of these (`send_trigger_pulse`) outputs a pulse of just 0.0001 seconds on the pin supplied as its parameter. This will cause the rangefinder to send out a pulse of ultrasound. The next function (`wait_for_echo`) is responsible for waiting until the echo from that pulse of ultrasound is received, so that the distance can be calculated by the length of time it took for the echo to arrive.

The function `get_distance` puts all this together, first sending a trigger pulse and then timing how long it takes for the echo to arrive. Actually, it's slightly more complicated than that, because the echo cannot be detected until after the pulse has finished sending. If we check too early, we will get a false reading. That is why there are two calls to `wait_for_echo`. The first waits until sending of the pulse is complete and the second actually times the delay until the echo arrives.

Graphical user interface

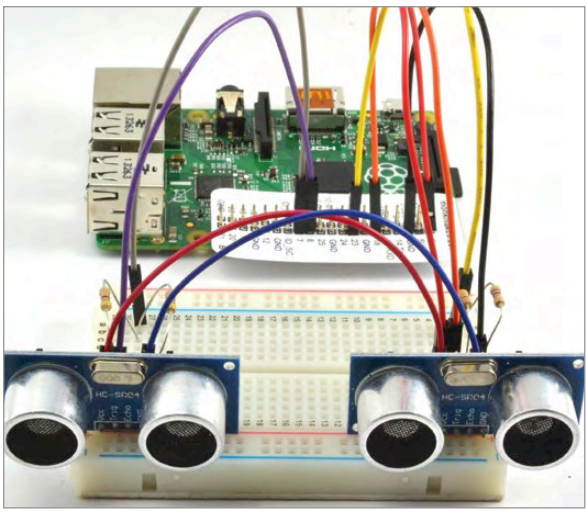
The rest of the code is concerned with the user interface for the project. The function `colour_for_distance` returns a colour to use when drawing the rectangle for that sensor, depending on how large the distance detected is.

The next few lines initialise the Pygame window and define a font to use for the distance readout. While



Below right The complete prototype

Below You could use the affordable HDMIPi screen in your garage



Pygame is designed primarily to make games, it's excellent for any project that uses graphics, like this one. You will likely want to alter the **width** and **height** variables to match the resolution of your display.

The **while** statement starts the main loop. The program will keep looping around the instructions inside, from **while** until the end of the file, until the program window is closed – this kind of loop is known as an infinite loop.

Each time around the loop the Pygame events are checked, and if the Pygame window has been closed (by clicking the little cross in the corner of the window), then the GPIO pins are set to a safe input mode using the **GPIO.cleanup** function and the program exits.

Most of the time the window will not have been closed, so the remainder of the loop will measure the two distances from the rangefinders and then draw rectangles on the screen, using the distance readings to set the height of the rectangle. The height of each rectangle is the distance in cm multiplied by 5 pixels.

Finally, there is a delay of half a second to stop the distance figures updating too fast to read clearly.

Using your parking sensor

Although you can extend the leads to the ultrasonic rangefinder by perhaps a metre or so, any longer than that and you are likely to have problems with the signal. So, if you are installing this project for real in a vehicle, it may be better to site the Raspberry Pi fairly near the sensors and use a longer HDMI lead to connect the Raspberry Pi to the display.

If you are installing this project for real, then you will probably want to make the program start automatically. That way, you don't even need to have a keyboard and mouse attached to the Raspberry Pi. You can find links on how to do this on the Raspberry Pi Forum: **bit.ly/1DaEURv**

The ultrasonic rangefinders are great little devices. You can take the range-finding part of the program for this project and use it in lots of other projects. You could, for instance, use it to just make a distance meter, perhaps displaying the distance in inches or cm. You could also use it to create a theremin-like musical instrument that changes the pitch of the note, depending on the distance of your hand from the rangefinder.

NEXT MONTH

In the next project of this series, we will turn our attention to making a web-controlled door lock that lets you unlock your door remotely.

Language
>PYTHON

DOWNLOAD:
bit.ly/1KutV7K

01_parking_sensor.py

```
import RPi.GPIO as GPIO
import time, sys, pygame

trigger_pin_left = 8
echo_pin_left = 7
trigger_pin_right = 18
echo_pin_right = 23

green = (0,255,0)
orange = (255,255,0)
red = (255,0,0)
white = (255,255,255)
black = (0, 0, 0)

GPIO.setmode(GPIO.BCM)
GPIO.setup(trigger_pin_left, GPIO.OUT)
GPIO.setup(echo_pin_left, GPIO.IN)
GPIO.setup(trigger_pin_right, GPIO.OUT)
GPIO.setup(echo_pin_right, GPIO.IN)

def send_trigger_pulse(pin):
    GPIO.output(pin, True)
    time.sleep(0.0001)
    GPIO.output(pin, False)

def wait_for_echo(pin, value, timeout):
    count = timeout
    while GPIO.input(pin) != value and count > 0:
        count -= 1

def get_distance(trigger_pin, echo_pin):
    send_trigger_pulse(trigger_pin)
    wait_for_echo(echo_pin, True, 10000)
    start = time.time()
    wait_for_echo(echo_pin, False, 10000)
    finish = time.time()
    pulse_len = finish - start
    distance_cm = pulse_len / 0.000058
    return int(distance_cm)

def colour_for_distance(distance):
    if distance < 30:
        return red
    if distance < 100:
        return orange
    else:
        return green

pygame.init()
size = width, height = 800, 600 # the variables alter window size
offset = width / 8

screen = pygame.display.set_mode(size)
myfont = pygame.font.SysFont("monospace", 50)

while True: # the main loop starts here
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            GPIO.cleanup() # set GPIO pins to be inputs
            sys.exit() # quit the program entirely

    left_distance = get_distance(trigger_pin_left, echo_pin_left)
    right_distance = get_distance(trigger_pin_right, echo_pin_right)

    screen.fill(white)
    pygame.draw.rect(screen, colour_for_distance(left_distance),
                     (offset, 0, width / 4, left_distance*5))
    pygame.draw.rect(screen, colour_for_distance(right_distance),
                     (width / 2 + offset, 0, width / 4, right_distance*5))

    left_label = myfont.render(str(left_distance)+" cm", 1, black)
    screen.blit(left_label, (offset + 10, height/2))
    right_label = myfont.render(str(right_distance)+" cm", 1, black)
    screen.blit(right_label, (width/2 + offset + 10, height/2))

    pygame.display.update()
    time.sleep(0.1)
```