https://projects.raspberrypi.org/en/projects/ultrasonic-theremin

Home Projects Code Club CoderDojo Raspberry Jam Sign In

Ultrasonic theremin

Raspberry Pi Electronic components Sonic Pi Python

What you will make

In this resource, you will use an ultrasonic distance sensor to control the notes played by Sonic Pi, and unleash your inner Beach Boy.

What you will learn

By building an ultrasonic theremin, you will learn:

- How to detect distances with an ultrasonic distance sensor
- How to communicate variables between Sonic Pi and Python

What you will need

Hardware

- 330Ω Resistor
- 470Ω Resistor
- Solderless Breadboard
- Ultrasonic Distance Sensor
- 3 x Male to Male Jumper Leads
- 4 x Male to Female Jumper Leads

Software

Software Installation

This project relies on the **latest** version of Sonic Pi and python-osc. To install the software you need, run the following commands in a terminal window:

```
sudo apt update && sudo apt upgrade -y
sudo pip3 install python-osc
```

The Raspberry Pi ultrasonic theremin ›

https://projects.raspberrypi.org/en/projects/ultrasonic-theremin/3

Ultrasonic theremin

Raspberry Pi | Electronic components | Sonic Pi | Python

Setting up the circuitry

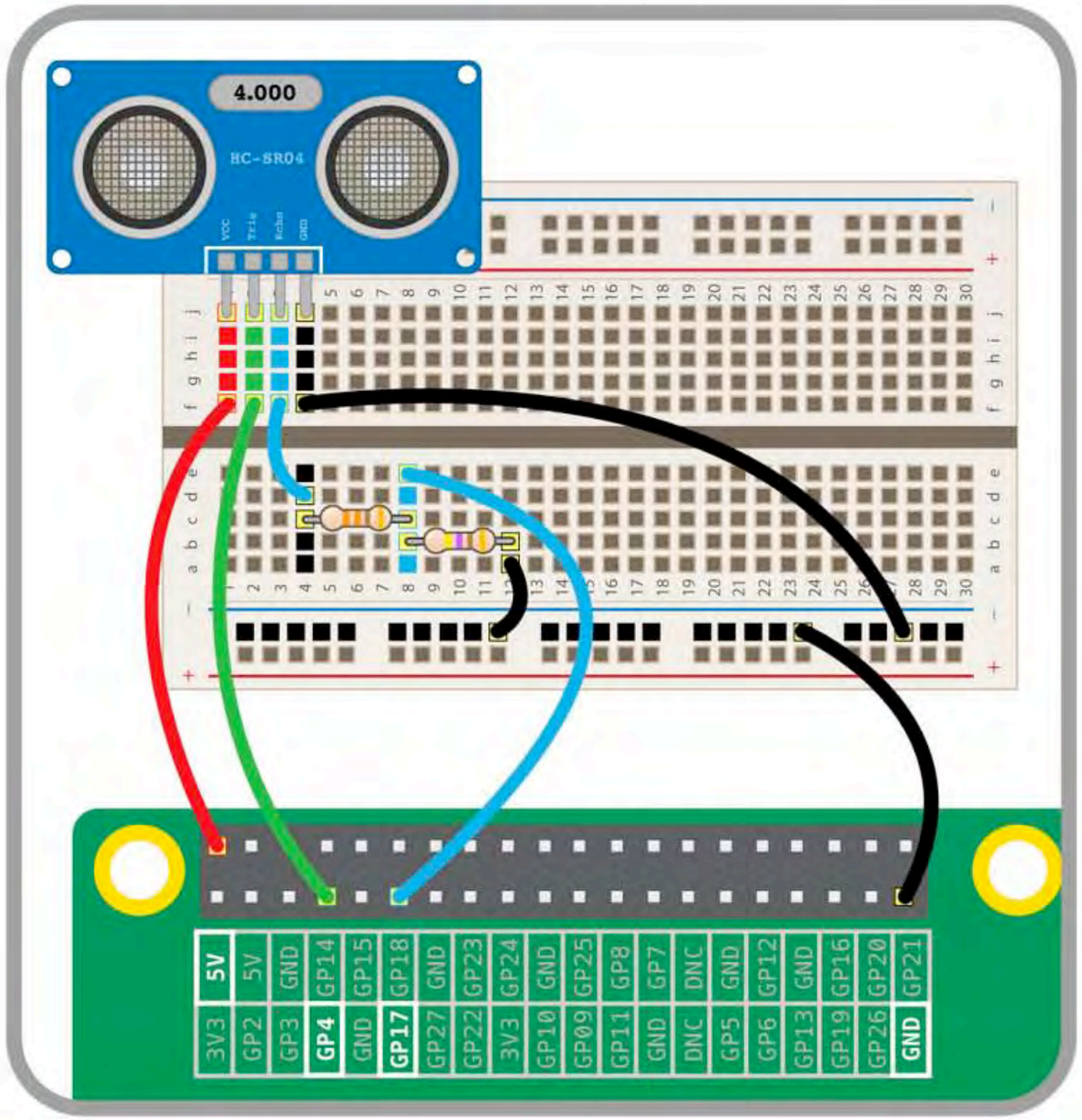
An ultrasonic distance sensor is a device that sends out pulses of ultrasonic sound, and measures the time they take to bounce off nearby objects and be reflected back. They can measure distances fairly accurately, up to about a meter.

An ultrasonic distance sensor has four pins. They are called **Ground** (**Gnd**), **Trigger** (**Trig**), **Echo** (**Echo**) and **Power** (**Vcc**).

To use an ultrasonic distance sensor you need to connect the **Gnd** pin to the ground pin on the Raspberry Pi, the **Trig** pin to a GPIO pin on the Raspberry Pi and the **Vcc** pin to the 5V pin on the Raspberry Pi.

The **Echo** pin is a little more complicated. It needs to be connected through a 330 ohm resistor to a GPIO pin on the Raspberry Pi, and that pin needs to be grounded through a 470 ohm resistor.

The diagram below shows one suggested arrangement for setting this up.



Detecting distance

Detecting distance

Thanks to the abstractions in the GPIO Zero module, you can very easily detect how far away an object is from the distance sensor. If you've wired up the sensor as shown in the diagram, then your echo pin is **17** and your trigger pin is **4**.

- Click on **Menu** > **Programming** > **Python 3 (IDLE)**, to open up a new Python shell.
- In the shell, click on **New** > **New File** to create a new Python file.
- The code to detect distance is below. Type it into your new file, then save and run it.

```
from gpiozero import DistanceSensor
from time import sleep

sensor = DistanceSensor(echo=17, trigger=4)

while True:
    print(sensor.distance)
    sleep(1)
```

The `sensor.distance` is the distance in meters between the object and the sensor. Run your code and move your hand backwards and forwards in front of the sensor. You should see the distance changing, as it is printed out in the shell.

Getting Sonic Pi ready >

Getting Sonic Pi ready

Sonic Pi is going to receive messages from your Python script. This will tell Sonic Pi which note to play.

- Open Sonic Pi by clicking on **Menu** > **Programming** > **Sonic Pi**
- In the buffer that is open, you can begin by writing a `live_loop`. This is a loop that will run forever, but can easily be updated, allowing you to experiment with different sounds. You can also add a line to reduce the time it takes for Sonic Pi and Python to talk to each other.

```
live_loop :listen do
    use_real_time
end
```

- Next you can sync the live loop with the messages that will be coming from Python.

```
live_loop :listen do
    use_real_time
    note = sync "/osc/play_this"
end
```

- The message that comes in will be a list, with the note being the 0th item.

```
live_loop :listen do
    use_real_time
    note = sync "/osc/play_this"
    play note[0]
end
```

- You can set this live loop to play straight away, by clicking on the **Run** button. You won't hear anything yet, as the loop is not receiving any messages.

Sending notes from Python

To finish your program, you need to send note midi values to Sonic Pi from your Python file.

- You'll need to use the OSC library for this part. There are two imports to be added to the top of your file, to allow Python to send messages.

```
from gpiozero import DistanceSensor
from time import sleep

from pythonosc import osc_message_builder
from pythonosc import udp_client

sensor = DistanceSensor(echo=17, trigger=4)

while True:
    print(sensor.distance)
    sleep(1)
```

- Now you need to create a `sender` object that can send the message.

```
from gpiozero import DistanceSensor
from time import sleep

from pythonosc import osc_message_builder
from pythonosc import udp_client

sensor = DistanceSensor(echo=17, trigger=4)
sender = udp_client.SimpleUDPClient('127.0.0.1', 4559)

while True:
    print(sensor.distance)
    sleep(1)
```

- You need to convert the distance into a midi value. These should be integers (whole numbers), and hover around the value **60**, which is middle C. To do this you need to round the distance to an integer, multiply it by 100 and then add a little bit, so that the note is not too low in pitch.

```
from gpiozero import DistanceSensor
from time import sleep

from pythonosc import osc_message_builder
from pythonosc import udp_client

sensor = DistanceSensor(echo=17, trigger=4)
sender = udp_client.SimpleUDPClient('127.0.0.1', 4559)

while True:
    pitch = round(sensor.distance * 100 + 30)
    sleep(1)
```

- To finish off, you need to send the pitch over to Sonic Pi, and reduce the sleep time.

```
from gpiozero import DistanceSensor
from time import sleep

from pythonosc import osc_message_builder
from pythonosc import udp_client

sensor = DistanceSensor(echo=17, trigger=4)
sender = udp_client.SimpleUDPClient('127.0.0.1', 4559)

while True:
    pitch = round(sensor.distance * 100 + 30)
    sender.send_message('/play_this', pitch)
    sleep(0.1)
```

```
while True:
    pitch = round(sensor.distance * 100 + 30)
    sleep(1)
```

- To finish off, you need to send the pitch over to Sonic Pi, and reduce the sleep time.

```
from gpiozero import DistanceSensor
from time import sleep

from pythonosc import osc_message_builder
from pythonosc import udp_client

sensor = DistanceSensor(echo=17, trigger=4)
sender = udp_client.SimpleUDPClient('127.0.0.1', 4559)

while True:
    pitch = round(sensor.distance * 100 + 30)
    sender.send_message('/play_this', pitch)
    sleep(0.1)
```

- Save and run your code and see what happens. If all goes well, you've made your very own theremin.

What next?

What next?

- A real theremin has a secondary control to change the amplitude (volume) of the tone being played. You could do this with a second Ultrasonic distance sensor.
- What happens if you change the timing in the Python file? Can you produce a smoother transition of notes?
- Have a play with different synths, as shown in the Sonic Pi help menu. What effect does changing the synth have on your theremin?

What's next? >