Project 2 – Ultrasonic Theremin

Waving your hands can be a creative process. Feeling the music flow around you is sheer bliss, but what if you could control the music with a wave of your hand.

For this project you will need

A Raspberry Pi

The latest version of Raspbian

A breadboard

An HC-SR04 Ultrasonic Sensor

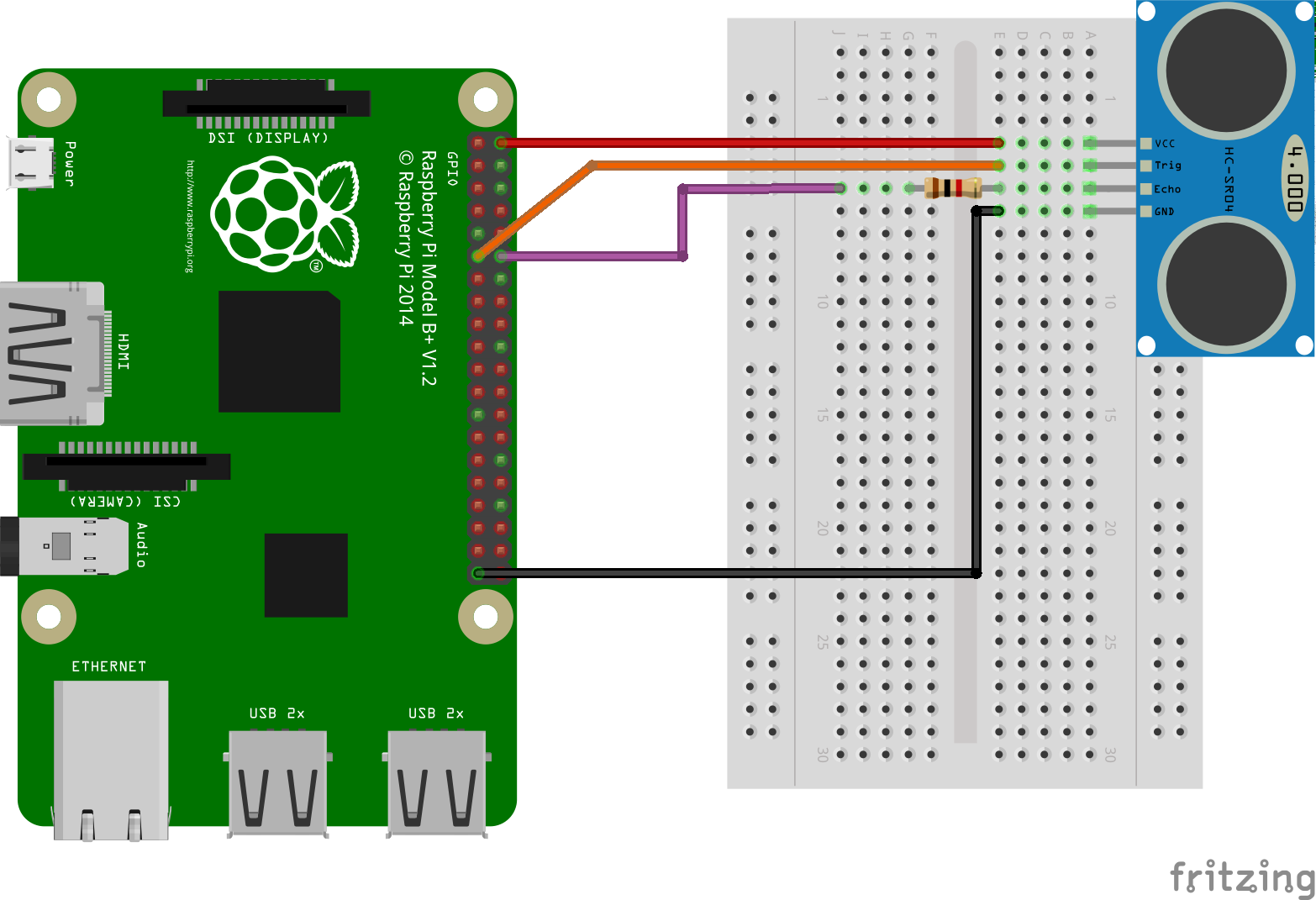
A 1K Ohm resistor (BROWN, BLACK, RED, GOLD)

Male to female wires

A speaker or monitor with audio capabilities.

Building the hardware.

We shall be using an Ultrasonic sensor as a way to control the pitch of our music, and we will need to connect it to our Raspberry Pi as per this diagram.



With the wiring complete, lets attach our keyboard, mouse, HDMI and finally the power to our Raspberry Pi and boot to the desktop.

## Software Install

You will need to have your Raspberry Pi on the Internet to complete this step, but once completed you can safely disconnect from the Internet.

For this project we need to install some extra software that will enable us to use Python Sonic with Sonic Pi. On your Raspberry Pi open a terminal, the icon for which is in the taskbar at the top of the screen and looks like a black monitor. In the terminal type the following and press Enter

*sudo pip3 install python-osc*

After a few minutes the software will be installed and the terminal will be returned to you. You can now close the window and move to the next step.

## Coding the project.

We start by going to the main menu, located in the top left of the screen, and click on

Programming » Sonci Pi. Wait for Sonic Pi to load and in Sonic Pi click on Prefs and select the correct audio output for your setup. For those using external speakers select “headphones”. When done minimise the application. Now go back to the Programming menu and select Python 3.

A new screen will open, in this screen click on File » New Window to open a new editor window. In the new editor window click on File » Save and call your work "ultrasonic.py"

We start our code by importing a series of modules. These will enable us to use the Raspberry Pi GPIO pins. We can also control the pace of our code using “time” and the last import is a link between Python and Sonic Pi.

*import RPi.GPIO as GPIO*

*import time*

*from psonic import \**

Our next two lines setup the GPIO to use the default pin numbering system, and also turn off the majority of information messages, which can sometimes be mistaken for errors.

*GPIO.setmode(GPIO.BCM)*

*GPIO.setwarnings(False)*

Next up we use two variables to store the pins that will be used to trigger the sensor and receive the echo signal.

*pinTrigger = 17*

*pinEcho = 18*

We now setup our trigger and echo pins respectively as an output and an input.

*GPIO.setup(pinTrigger, GPIO.OUT)*

*GPIO.setup(pinEcho, GPIO.IN)*

Now we will create a function that will contain all of the code that will handle triggering the sensor, interpreting the echo that returns, and playing audio.

We need to give our function a name, in this case we shall call it “ultra”

*def ultra():*

The code for the function starts by ensuring that the trigger pin is turned off, we then pause for 0.1 seconds. The trigger pin is activated for a split second before turning it off. We then record the start time of the trigger pulse

*GPIO.output(pinTrigger, False)*

*time.sleep(0.1)*

*GPIO.output(pinTrigger, True)*

*time.sleep(0.00001)*

*GPIO.output(pinTrigger, False)*

*StartTime = time.time()*

Next we use two while loops. The first is used to record the start time of the pulse while no echo signal has been received. But when we receive a signal on the echo pin we will now record the stop time.

*while GPIO.input(pinEcho)==0:*

*StartTime = time.time()*

*while GPIO.input(pinEcho)==1:*

*StopTime = time.time()*

We now do a little maths to work out how long it took the signal to travel from the sensor and back again before we work out the distance by taking the elapsed time and multiplying it by the speed of sounds (cm/s). Because we only need to know the distance to an object and not the total journey distance, we divide the distance by two.

*ElapsedTime = StopTime - StartTime*

*Distance = ElapsedTime \* 34326*

*Distance = Distance / 2*

We will now print the distance, this will appear in the Python shell.

*print("Distance : %.1f" % Distance)*

For the last part of this function we shall create a note based on our distance from the sensor. In this example we divided the distance by two to give us a long range sensor. But if you are short on space you can multiply the distance by 2 or 3 to give a short range reading. Next we use an insutrument, called a synth, in this case it is a modulated pulse. We play the note before briefly pausing. This is the end of the function.

*note = Distance /2*

*use\_synth(MOD\_PULSE)*

*play(note)*

*time.sleep(0.1)*

Our final section of code is an infinite loop that will run the function.

*while True:*

*ultra()*

### Complete Code Listing

*port RPi.GPIO as GPIO*

*import time*

*from psonic import \**

*GPIO.setmode(GPIO.BCM)*

*GPIO.setwarnings(False)*

*pinTrigger = 17*

*pinEcho = 18*

*GPIO.setup(pinTrigger, GPIO.OUT)*

*GPIO.setup(pinEcho, GPIO.IN)*

*def ultra():*

*GPIO.output(pinTrigger, False)*

*time.sleep(0.1)*

*GPIO.output(pinTrigger, True)*

*time.sleep(0.00001)*

*GPIO.output(pinTrigger, False)*

*StartTime = time.time()*

*while GPIO.input(pinEcho)==0:*

*StartTime = time.time()*

*while GPIO.input(pinEcho)==1:*

*StopTime = time.time()*

*ElapsedTime = StopTime - StartTime*

*Distance = ElapsedTime \* 34326*

*Distance = Distance / 2*

*print("Distance : %.1f" % Distance)*

*note = Distance /2*

*use\_synth(MOD\_PULSE)*

*play(note)*

*time.sleep(0.1)*

*while True:*

*ultra()*

## Starting the project

In Python, click on Run >> Run Module to start the code. You should hear a sound. If not ensure that you have selected the correct output by right clicking on the Sound applet, located in the top right of the screen. For speakers attached to the 3.5mm headphone jack, select Analog and ensure the volume is at maximum.

Now wave you hand in front of the sensor to trigger the effect.