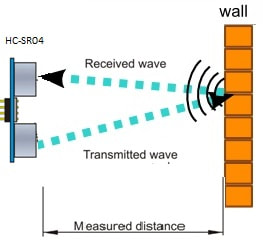
**Interface of HC – SR04 Ultrasonic Range Sensor on the Raspberry Pi**

**Ultrasonic Distance Sensors**

Sound consists of oscillating waves through a medium (such as air) with the pitch being determined by the closeness of those waves to each other, defined as the frequency. Only some of the sound spectrum (the range of sound wave frequencies) is audible to the human ear, defined as the “Acoustic” range. Very low frequency sound below Acoustic is defined as “Infrasound”, with high frequency sounds above, called “Ultrasound”. Ultrasonic sensors are designed to sense object proximity or range using ultrasound reflection, similar to radar, to calculate the time it takes to reflect ultrasound waves between the sensor and a solid object. Ultrasound is mainly used because it’s inaudible to the human ear and is relatively accurate within short distances. You could of course use Acoustic sound for this purpose, but you would have a noisy robot, beeping every few seconds.

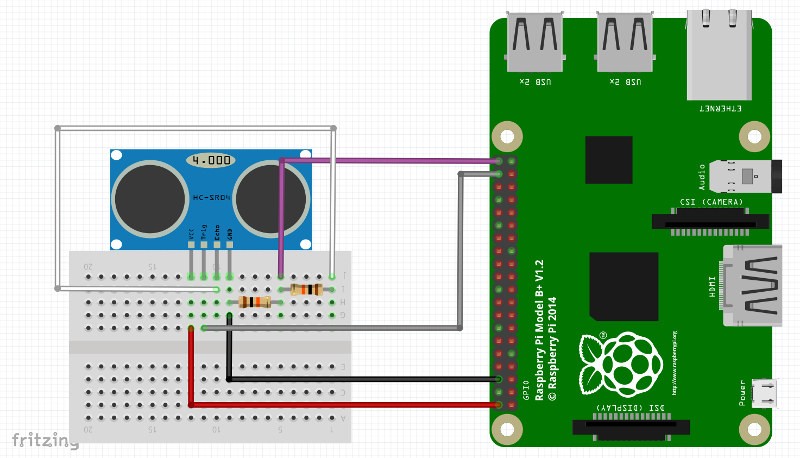
A basic ultrasonic sensor consists of one or more ultrasonic transmitters (basically speakers), a receiver, and a control circuit. The transmitters emit a high frequency ultrasonic sound, which bounce off any nearby solid objects. Some of that ultrasonic noise is reflected and detected by the receiver on the sensor. That return signal is then processed by the control circuit to calculate the time difference between the signal being transmitted and received. This time can subsequently be used, along with some clever math, to calculate the distance between the sensor and the reflecting object.



**Figure1:** Hardware Setup for ultrasonic sensor to measure the distance

The HC-SR04 Ultrasonic sensor we’ll be using in this tutorial for the Raspberry Pi has four pins: ground (GND), Echo Pulse Output (ECHO), Trigger Pulse Input (TRIG), and 5V Supply (Vcc). We power the module using Vcc, ground it using GND, and use our Raspberry Pi to send an input signal to TRIG, which triggers the sensor to send an ultrasonic pulse. The pulse waves bounce off any nearby objects and some are reflected back to the sensor. The sensor detects these return waves and measures the time between the trigger and returned pulse, and then sends a 5V signal on the ECHO pin.ECHO will be “low” (0V) until the sensor is triggered when it receives the echo pulse. Once a return pulse has been located ECHO is set “high” (5V) for the duration of that pulse. Pulse duration is the full time between the sensor outputting an ultrasonic pulse, and the return pulse being detected by the sensor receiver. Our Python script must therefore measure the pulse duration and then calculate distance from this.

**NOTE:** The sensor output signal (ECHO) on the HC-SR04 is rated at 5V. However, the input pin on the Raspberry Pi GPIO is rated at 3.3V. Sending a 5V signal into that unprotected 3.3V input port could damage your GPIO pins, which is something we want to avoid! We’ll need to use a small voltage divider circuit, consisting of two resistors, to lower the sensor output voltage to something our Raspberry Pi can handle.Assemble the Circuit has shown in the Circuit Diagram Below.



**Figure2:** Hardware Setup for ultrasonic sensor to measure the distance

We’ll be using four pins on the Raspberry Pi for this project:

GPIO 5V [Pin 2]  Vcc (5V Power),

GPIO GND [Pin 6]  GND (0V Ground),

GPIO 23 [Pin 16]  TRIG (GPIO Output)

GPIO 24 [Pin 18]  ECHO (GPIO Input)

**Sensing with Python**

Now that we’ve hooked our Ultrasonic Sensor up to our Pi, we need to program a Python script to detect distance. The Ultrasonic sensor output (ECHO) will always output low (0V) unless it’s been triggered in which case it will output 5V (3.3V with our voltage divider!). We therefore need to set one GPIO pin as an output, to trigger the sensor, and one as an input to detect the ECHO voltage change.First, import the Python GPIO library, import our time library (so we make our Pi wait between steps) and set our GPIO pin numbering.

**import RPi.GPIO as GPIO**

**import time**

**GPIO .setmode(GPIO.BCM)**

Next, we need to name our input and output pins, so that we can refer to it later in our Python code. We’ll name our output pin (which triggers the sensor) GPIO 23 [Pin 16] as TRIG, and our input pin (which reads the return signal from the sensor) GPIO 24 [Pin 18] as ECHO.

**TRIG = 23**

**ECHO = 24**

We’ll then print a message to let the user know that distance measurement is in progress. . . .

**print("Distance Measurement In Progress")**

Next, set your two GPIO ports as either inputs or outputs as defined previously.

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

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

Then, ensure that the Trigger pin is set low, and give the sensor a second to settle.

**GPIO.output(TRIG, False)**

**print("Waiting For Sensor To Settle")**

**time.sleep(2)**

The HC-SR04 sensor requires a short 10uS pulse to trigger the module, which will cause the sensor to start the ranging program (8 ultrasound bursts at 40 kHz) in order to obtain an echo response. So, to create our trigger pulse, we set out trigger pin high for 10uS then set it low again.

**GPIO.output(TRIG, True)**

**time.sleep(0.00001)**

**GPIO.output(TRIG, False)**

Now that we’ve sent our pulse signal we need to listen to our input pin, which is connected to ECHO. The sensor sets ECHO to high for the amount of time it takes for the pulse to go and come back, so our code therefore needs to measure the amount of time that the ECHO pin stays high. We use the “while” string to ensure that each signal timestamp is recorded in the correct order. The time.time() function will record the latest timestamp for a given condition. For example, if a pin goes from low to high, and we’re recording the low condition using the time.time() function, the recorded timestamp will be the latest time at which that pin was low.

Our first step must therefore be to record the last low timestamp for ECHO (pulse\_start) e.g. just before the return signal is received and the pin goes high.

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

**pulse\_start = time.time()**

Once a signal is received, the value changes from low (0) to high (1), and the signal will remain high for the duration of the echo pulse. We therefore also need the last high timestamp for ECHO (pulse\_end).

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

**pulse\_end = time.time(**)

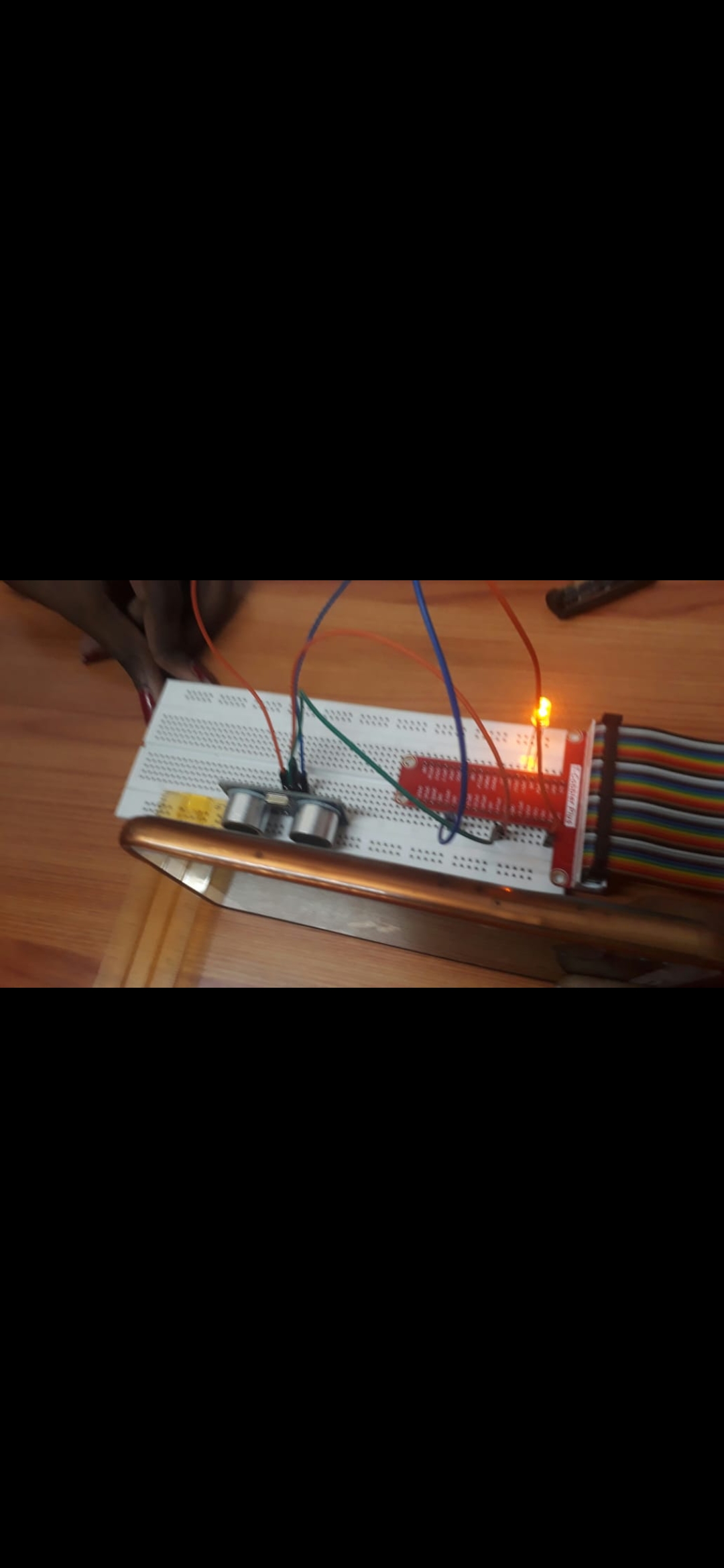
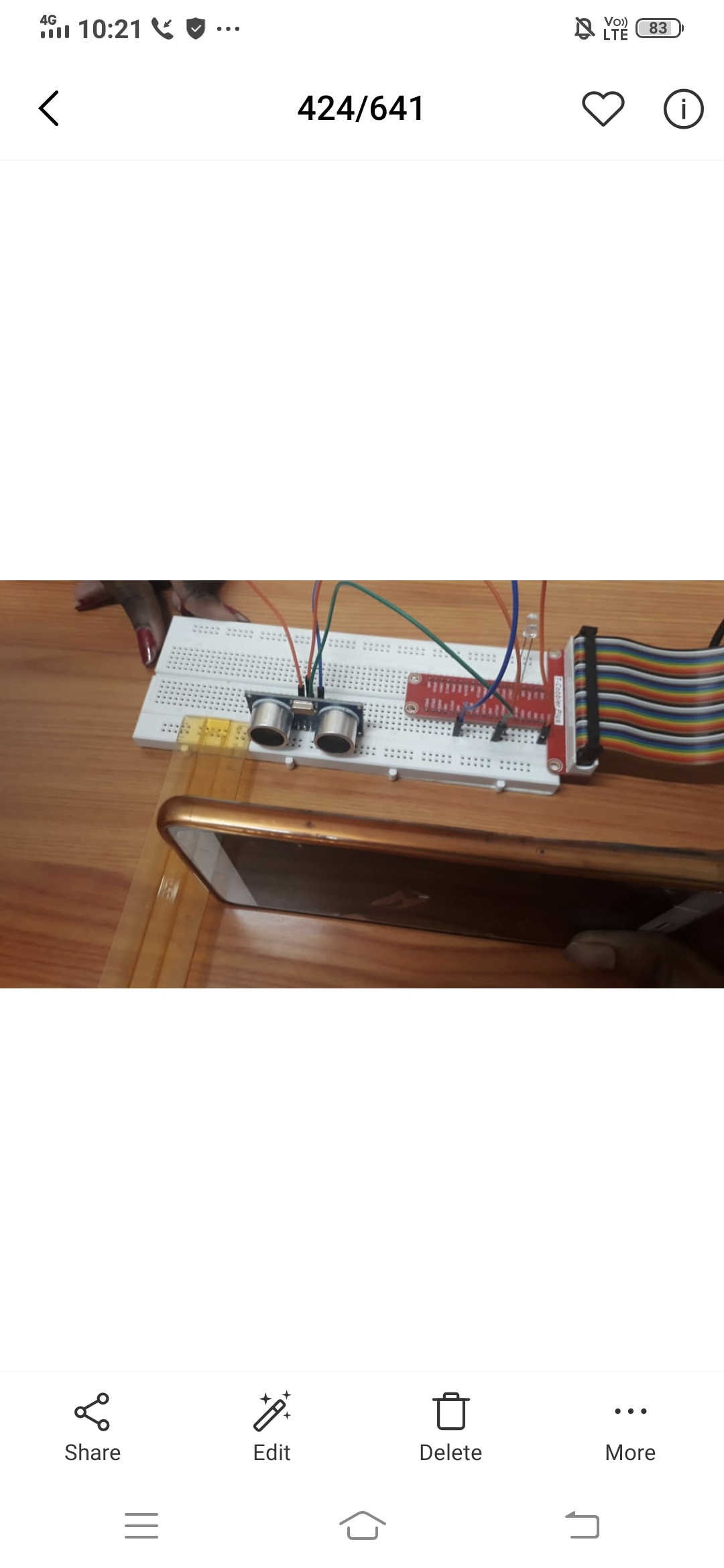
We can now calculate the difference between the two recorded timestamps, and hence the duration of pulse (pulse\_duration).

**pulse\_duration = pulse\_end - pulse\_start**

With the time it takes for the signal to travel to an object and back again, we can calculate the distance using the following formula.

**Speed** = Distances /time

The speed of sound is variable, depending on what medium it’s travelling through, in addition to the temperature of that medium. However, some clever physicists have calculated the speed of sound at sea level so we’ll take our baseline as the 343m/s. If you’re trying to measure distance through water, this is where you’re falling down – make sure you’re using the right speed of sound.

** **

**Figure3:** Hardware Setup for ultrasonic sensor to measure the distance