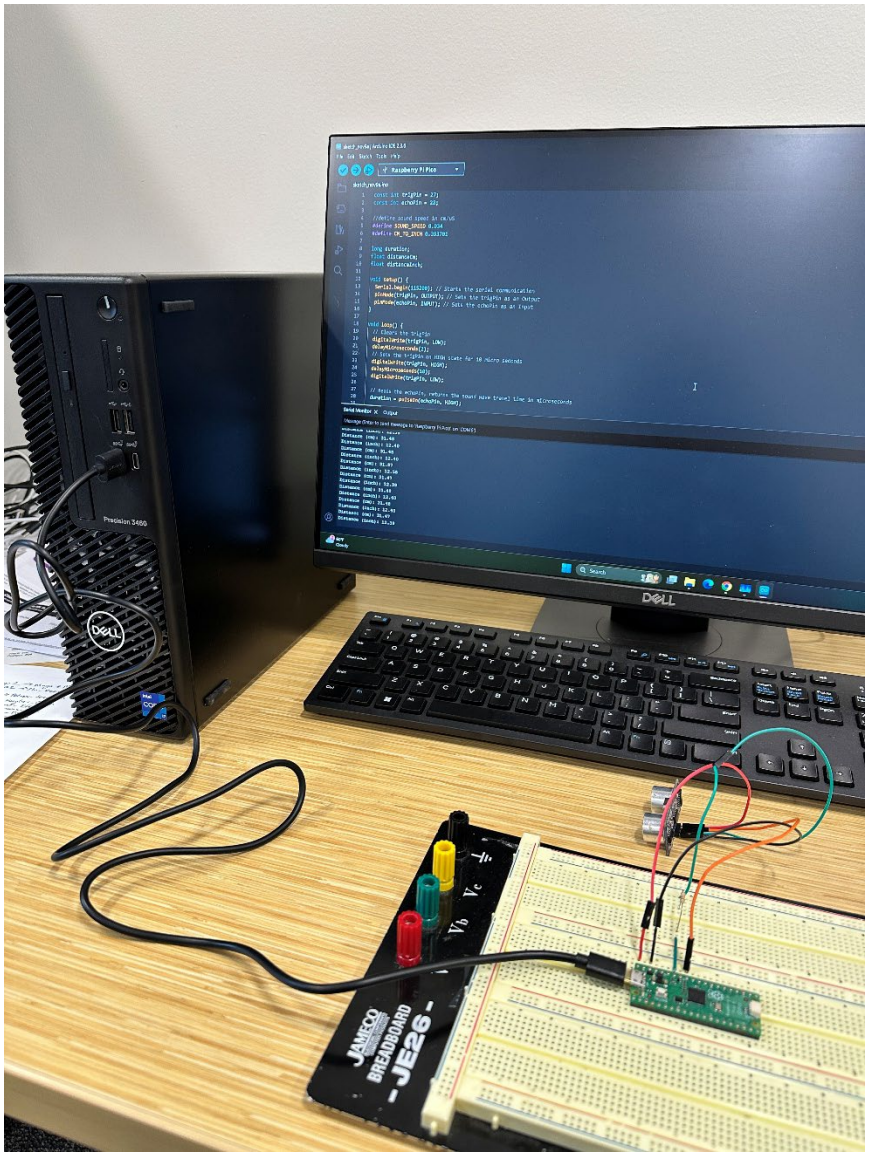
Tutorial: Getting the HC-SR04 Ultrasonic Sensor Working with Your Raspberry Pi Pico

Introduction

Some of you mentioned having trouble with the HC-SR04 and the Pico. I tried it out, and it is actually very straightforward with the correct wiring and code! I've created this custom guide to walk you through it step-by-step, using both the Arduino IDE and the Pico SDK in VS Code. I've tested this myself, and it works perfectly.



This is what your final setup should look like. Notice the resistor between the Echo pin and the Pico.

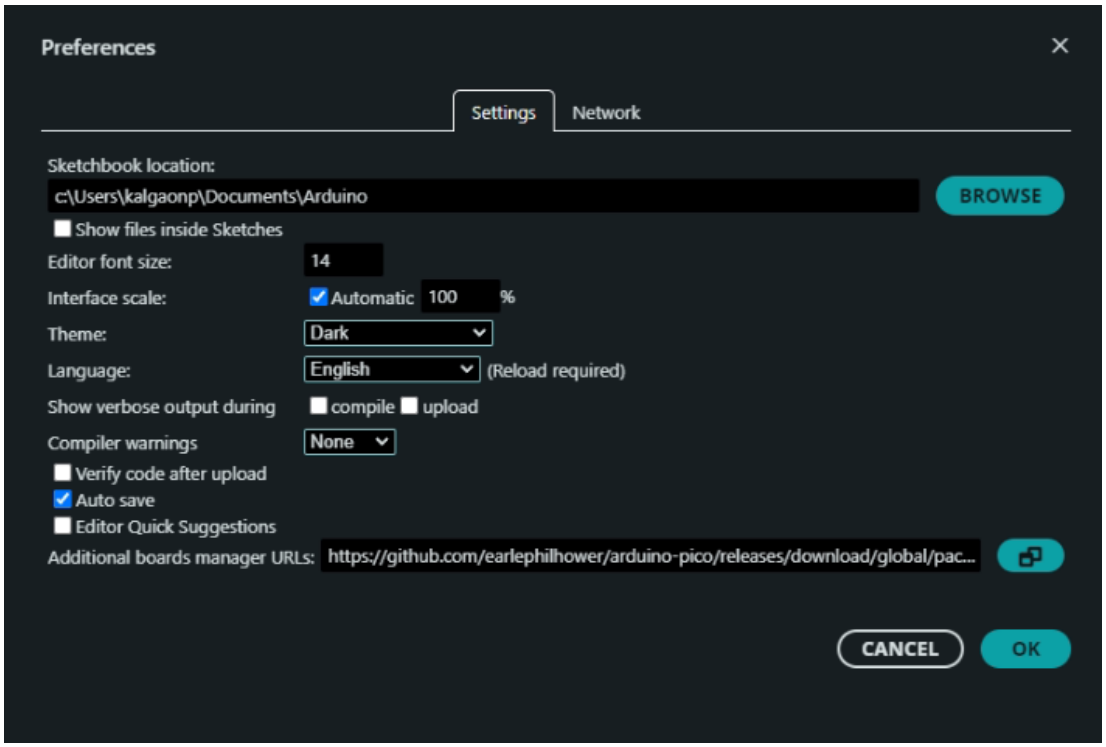
Part 1: Setup with Arduino IDE (The Quickest Way)

This is the fastest way to get started and test the demo code first. Then port it to VS Code.

1. Installing Pico Support in Arduino IDE

If you haven't already, you need to add support for the Raspberry Pi Pico to your Arduino IDE.

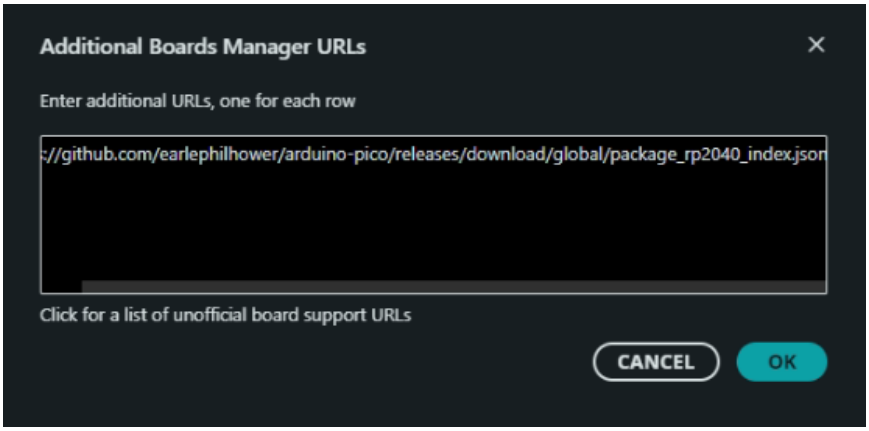
1. Open the Arduino IDE.
2. Go to **File > Preferences**.



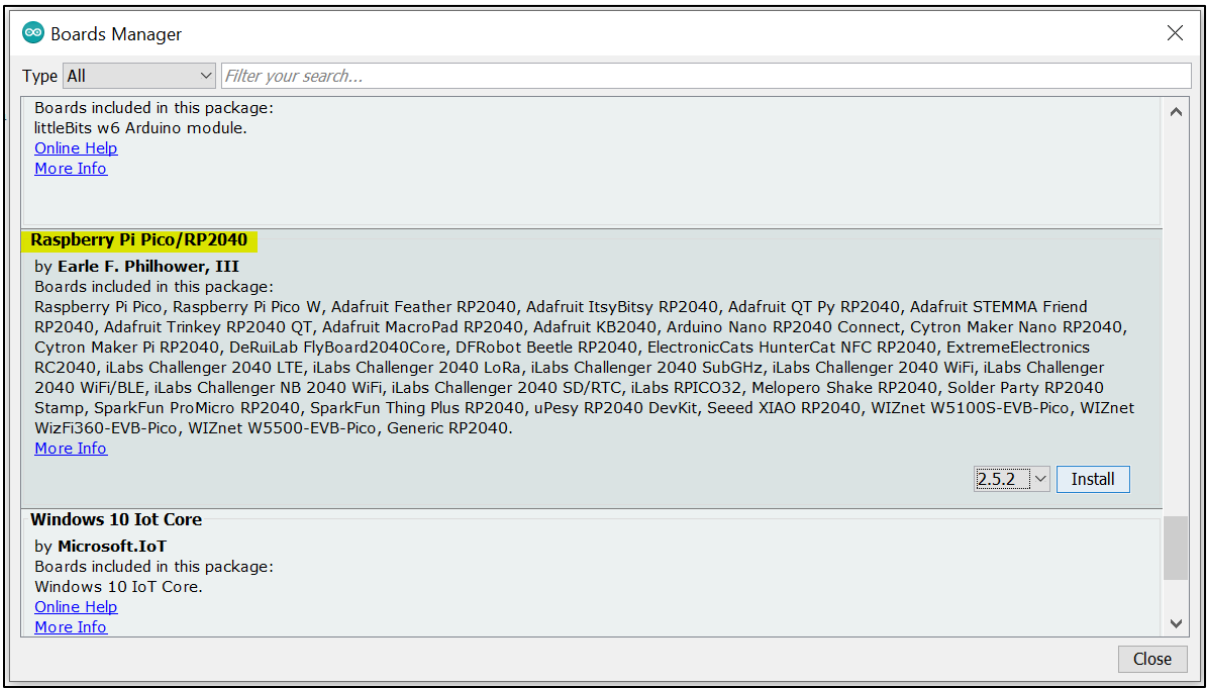
3. In the "*Additional Boards Manager URLs*" field, paste the following URL:

```
https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json
```

(If you already have other URLs in there, you can add this one on a new line.)



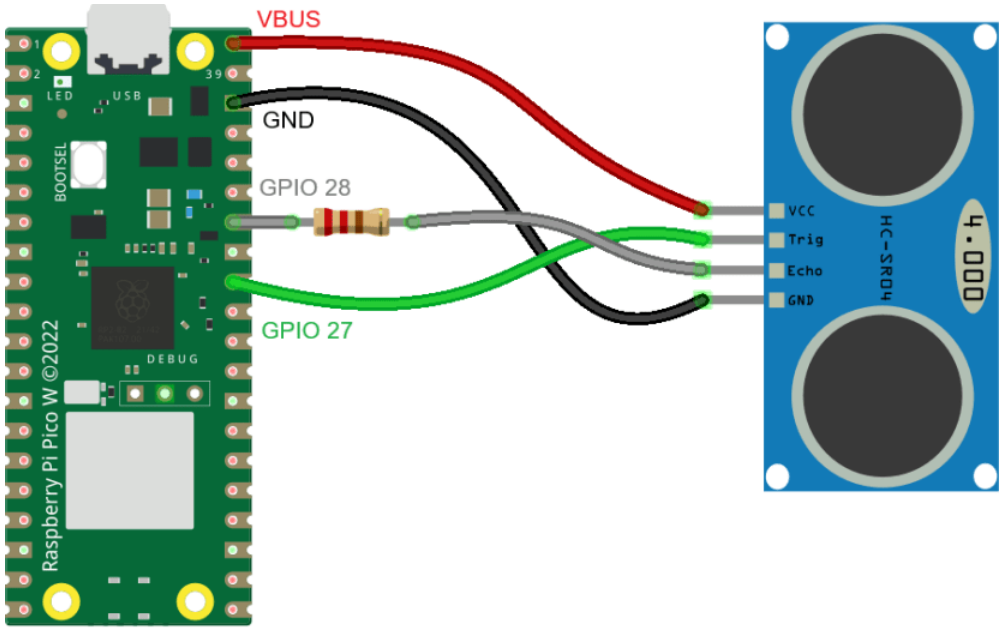
4. Click OK.
5. Now, go to **Tools > Board > Boards Manager**....
6. Search for "**Raspberry Pi Pico/RP2040**" and install the package by **Earle F. Philhower**.



(*Choose the latest version, not ver. 2.5.2 as shown in the screenshot*)

2. Wiring the HC-SR04 to Your Pico

Wire the sensor to your Pico as shown in the diagram and table below. **Pay close attention to the 1kΩ resistor on the Echo pin; this is crucial to protect the Pico's GPIO from the 5V signal.**



Ultrasonic Sensor	RPi Pico
VCC	**VBus** (5V) (Pin 40)
Trig	**GPIO 27** (Pin 32)
Echo	**GPIO 28** in series with a 1000 Ω resistor (Pin 34)
GND	**GND** (e.g., Pin 38)

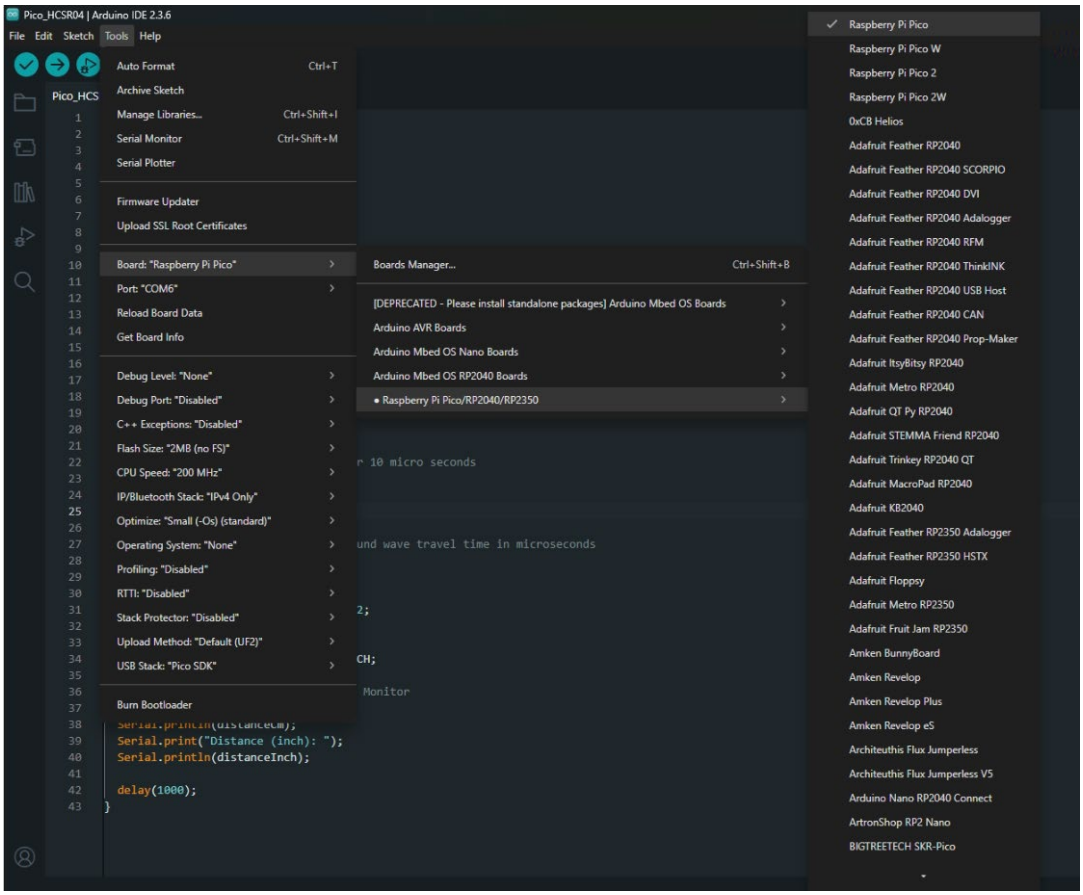
3. The Code and Upload

1. Copy and paste the exact code below into a new sketch in the Arduino IDE.

```
const int trigPin = 27;
const int echoPin = 28;

//define sound speed in cm/uS
#define SOUND_SPEED 0.034
#define CM_TO_INCH 0.393701

long duration;
float distanceCm;
float distanceInch;

void setup() {
  Serial.begin(115200); // Starts the serial communication
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
}

void loop() {
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Reads the echoPin, returns the sound wave travel time in
microseconds
  duration = pulseIn(echoPin, HIGH);

  // Calculate the distance
  distanceCm = duration * SOUND_SPEED/2;

  // Convert to inches
  distanceInch = distanceCm * CM_TO_INCH;

  // Prints the distance in the Serial Monitor
  Serial.print("Distance (cm): ");
  Serial.println(distanceCm);
```

```
    Serial.print("Distance (inch): ");
    Serial.println(distanceInch);

    delay(1000);
}
```

2. **Select your board and port:**
   - Go to **Tools > Board** and select **Raspberry Pi Pico / RP2040 > Raspberry Pi Pico**.
   - Go to **Tools > Port** and select the corresponding serial port.



3. **Upload:**
   - The first time you upload, you need to put the Pico into **Bootloader Mode**. Hold down the **BOOTSEL** button on the Pico, plug it into your computer, and then release the button. The IDE should detect it and upload the code.
   - For subsequent uploads, you can usually just click the upload button normally.

4. Testing

Open the Serial Monitor (**Tools > Serial Monitor**) or by clicking on the symbol shown in the red box below, and set the baud rate to **115200**. You should now see distance readings printing out every second.

Part 2: Porting the Code to VS Code with Pico SDK

Now that you have a working setup, let's port this code to a proper Pico C/C++ project in VS Code. This is the "native" way to program the Pico and is more powerful.

1. Project Structure

Create a new folder for your project with the following files inside, for example:

```
my_hcsr04_project/
├── CMakeLists.txt
├── pico_sdk_import.cmake
└── hcsr04_demo.c
```

2. The Source Code (hcsr04_demo.c)

This code (on the next page) does the exact same thing as the Arduino version, but uses the Pico SDK.

```c
#include <stdio.h>
#include "pico/stdlib.h"
#include "pico/time.h"

#define TRIG_PIN 27
#define ECHO_PIN 28
#define SOUND_SPEED 0.034f // cm per microsecond
#define CM_TO_INCH 0.393701f

int main() {
    stdio_init_all(); // Initialize serial communication

    gpio_init(TRIG_PIN);
    gpio_init(ECHO_PIN);
    gpio_set_dir(TRIG_PIN, GPIO_OUT);
    gpio_set_dir(ECHO_PIN, GPIO_IN);

    while (true) {
        // Send a 10us HIGH pulse to TRIG_PIN
        gpio_put(TRIG_PIN, 0);
        sleep_us(2);
        gpio_put(TRIG_PIN, 1);
        sleep_us(10);
        gpio_put(TRIG_PIN, 0);

        // Wait for the echo pin to go high (start of pulse)
        while (gpio_get(ECHO_PIN) == 0) {
            tight_loop_contents();
        }
        absolute_time_t start_time = get_absolute_time();

        // Wait for the echo pin to go low (end of pulse)
        while (gpio_get(ECHO_PIN) == 1) {
            tight_loop_contents();
        }
        absolute_time_t end_time = get_absolute_time();

        // Calculate pulse duration in microseconds
        int64_t duration_us = absolute_time_diff_us(start_time, end_time);

        // Calculate distance
        float distance_cm = duration_us * SOUND_SPEED / 2.0f;
        float distance_inch = distance_cm * CM_TO_INCH;

        // Print results
        printf("Distance (cm): %.2f\n", distance_cm);
        printf("Distance (inch): %.2f\n\n", distance_inch);

        sleep_ms(1000); // Wait for 1 second
    }
    return 0;
}
```

3. The CMakeLists.txt File

This file tells the build system how to compile your project. Create it with the following content.

```
cmake_minimum_required(VERSION 3.13)

# Initialize the Pico SDK
include(pico_sdk_import.cmake)

project(my_hcsr04_project)

# Initialize the SDK
pico_sdk_init()

# Create the executable
add_executable(hcsr04_demo
    hcsr04_demo.c
)

# Link against the pico_stdlib which includes core functionality
target_link_libraries(hcsr04_demo pico_stdlib)

# Enable USB output, so we can use printf
pico_enable_stdio_usb(hcsr04_demo 1)
pico_enable_stdio_uart(hcsr04_demo 0)

# Create the Pico binary file
pico_add_extra_outputs(hcsr04_demo)
```

4. Building and Uploading

1. Build the project. This will generate a *hcsr04_demo.uf2* file in your build directory.
2. Manually put your Pico into Bootloader Mode (hold BOOTSEL while plugging in).
3. Drag and drop the *hcsr04_demo.uf2* file onto the RPI-RP2 drive that appears.

Open a serial monitor (like TeraTerm using an UART Bridge from Lab 3, or the one in VS Code) to the correct port at 115200 baud, and you will see the same distance readings as before, in the Arduino IDE!

Conclusion

You have now successfully run the HC-SR04 demo using both the Arduino IDE and the native Pico SDK. The key takeaways are the **correct wiring with the voltage-divider resistor** and the logic of triggering the sensor and measuring the echo pulse duration.

Good luck with your projects! 😊

Links for reference:

Physical setup photos:

- https://drive.google.com/file/d/1u8ggcXJ83I7N0cYCO9HdYrEDhN6UyQDp/view?usp=sharing
- https://drive.google.com/file/d/1uuWfkQBvJBguCz3S6ivt6o4srLKgaldG/view?usp=sharing

Video:

- https://drive.google.com/file/d/1QmKfTkoE7Oyf5a5P5sVUFYssd_u46Xmk/view?usp=sharing

GitHub Link:

- https://github.com/priyankkalgaonkar/Pico-HC-SR04-Tutorial.git