LE/ESSE 2220 Algorithmic and Computational Methods

Lab 5: Ultrasonic Distance Measurement and Data Logging

(Fall 2025-2026)

Z. ARJMANDI (ZARARJ@YORKU.CA)

YORK U

Review

Data Logging + Handling

Common time Library Commands

Command	Description	Example
time.time()	Returns current time in seconds since epoch **(Unix time)**	t = time.time()
time.sleep(seconds)	Pauses the program for given seconds	time.sleep(1)
time.strftime(format)	Formats current time as a string (**String From Time**)	time.strftime("%H:%M:%S") → "14:25:03"
time.localtime()	Returns current local time as a struct	time.localtime()
time.ctime()	Returns readable string of current time	time.ctime() → "Thu Oct 9 14:25:03 2025"
time.perf_counter()	High-precision timer for performance measurement	start = time.perf_counter()
time.gmtime()	Returns UTC time as a struct	time.gmtime()

```python
import time

# Get current Unix time
t = time.time()
print("Unix time:", t)
```

```
Unix time: 1739088302.451927
```

YORK U

Data Logging + Handling

Useful txt Commands

Command	Description	Example
open(filename, mode)	Opens a file; returns a file object	f = open("data.txt", "w")
"w"	Write mode – creates or overwrites file	f = open("data.txt", "w")
"a"	Append mode – adds data to the end of the file	f = open("data.txt", "a")
"r"	Read mode – opens existing file for reading	f = open("data.txt", "r")
f.write(string)	Writes a string to the file	f.write("Hello\n")
f.writelines(list)	Writes multiple lines from a list	f.writelines(["A\n", "B\n", "C\n"])
f.read()	Reads the entire file as one string	data = f.read()
f.readline()	Reads one line at a time	line = f.readline()
f.readlines()	Reads all lines into a list	lines = f.readlines()
f.flush()	Forces writing of data to file immediately	f.flush()
f.close()	Closes the file (always do this after writing)	f.close()
with open(...) as f:	Preferred method that auto-closes the file	with open("data.txt", "w") as f:

Data Logging + Handling

Useful csv Library Commands

Command	Description	Example
import csv	Imports the CSV module	import csv
open(filename, mode, newline="")	Opens a CSV file; newline="" prevents blank lines on Windows	f = open("data.csv", "w", newline="")
"w"	Write mode – creates or overwrites the file	open("data.csv", "w", newline="")
"a"	Append mode – adds data to end of file	open("data.csv", "a", newline="")
"r"	Read mode – opens existing CSV file	open("data.csv", "r")
csv.writer(file)	Creates a writer object for writing rows	writer = csv.writer(f)
writer.writerow(list)	Writes one row (as a list)	writer.writerow(["time_s", "distance_cm"])
writer.writerows(list_of_lists)	Writes multiple rows at once	writer.writerows(data)
csv.reader(file)	Creates a reader object for reading rows	reader = csv.reader(f)
next(reader)	Reads or skips the next row (often header)	header = next(reader)
for row in reader:	Iterates through remaining rows	for row in reader: print(row)
list(reader)	Converts all rows to a list of lists	rows = list(reader)
with open(...) as f:	Preferred form – auto-closes file	with open("data.csv", "w", newline="") as f:
delimiter=","	Optional: define a custom separator	csv.reader(f, delimiter=",")
quotechar='"'	Defines the character used for quoting text	csv.writer(f, quotechar='"')
f.close()	Closes the file (not needed if using with)	f.close()

Data Logging + Handling

Example 1 – Writing and Reading a TXT File

```
# Example: Logging temperature readings to a text file

temperatures = [21.5, 22.0, 22.3, 22.1, 21.9]

# Write data to TXT file
with open("temperature_log.txt", "w") as f:
    f.write("reading_no\ttemperature_C\n")
    for i, t in enumerate(temperatures):
        f.write(f"{i+1}\t{t}\n")

print("Data written to temperature_log.txt")

# Read data back from TXT file
with open("temperature_log.txt", "r") as f:
    lines = f.readlines()
    print("File contents:")
    for line in lines:
        print(line.strip())
```

```
reading_no   temperature_C
1            21.5
2            22.0
3            22.3
4            22.1
5            21.9
```

YORK U

Data Logging + Handling

Example 2 – Writing and Reading a CSV File

```python
# Example: Logging temperature readings to a CSV file
import csv

temperatures = [21.5, 22.0, 22.3, 22.1, 21.9]

# Write data to CSV file
with open("temperature_log.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(["reading_no", "temperature_C"])
    for i, t in enumerate(temperatures):
        writer.writerow([i+1, t])

print("Data written to temperature_log.csv")

# Read data back from CSV file
with open("temperature_log.csv", "r") as f:
    reader = csv.reader(f)
    header = next(reader)
    print("Header:", header)
    for row in reader:
        print(row)
```

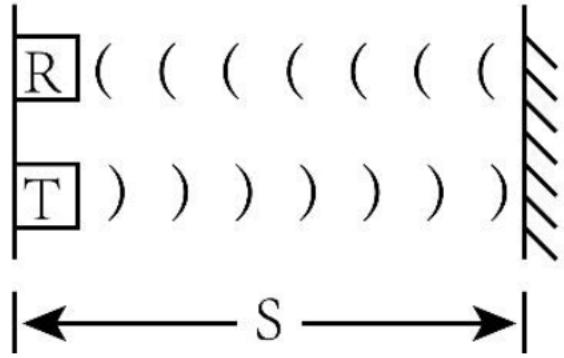
```
Header: ['reading_no', 'temperature_C']
['1', '21.5']
['2', '22.0']
['3', '22.3']
['4', '22.1']
['5', '21.9']
```

YORK U

Lab 5

Lab 5: Ultrasonic Distance Measurement and Data Logging

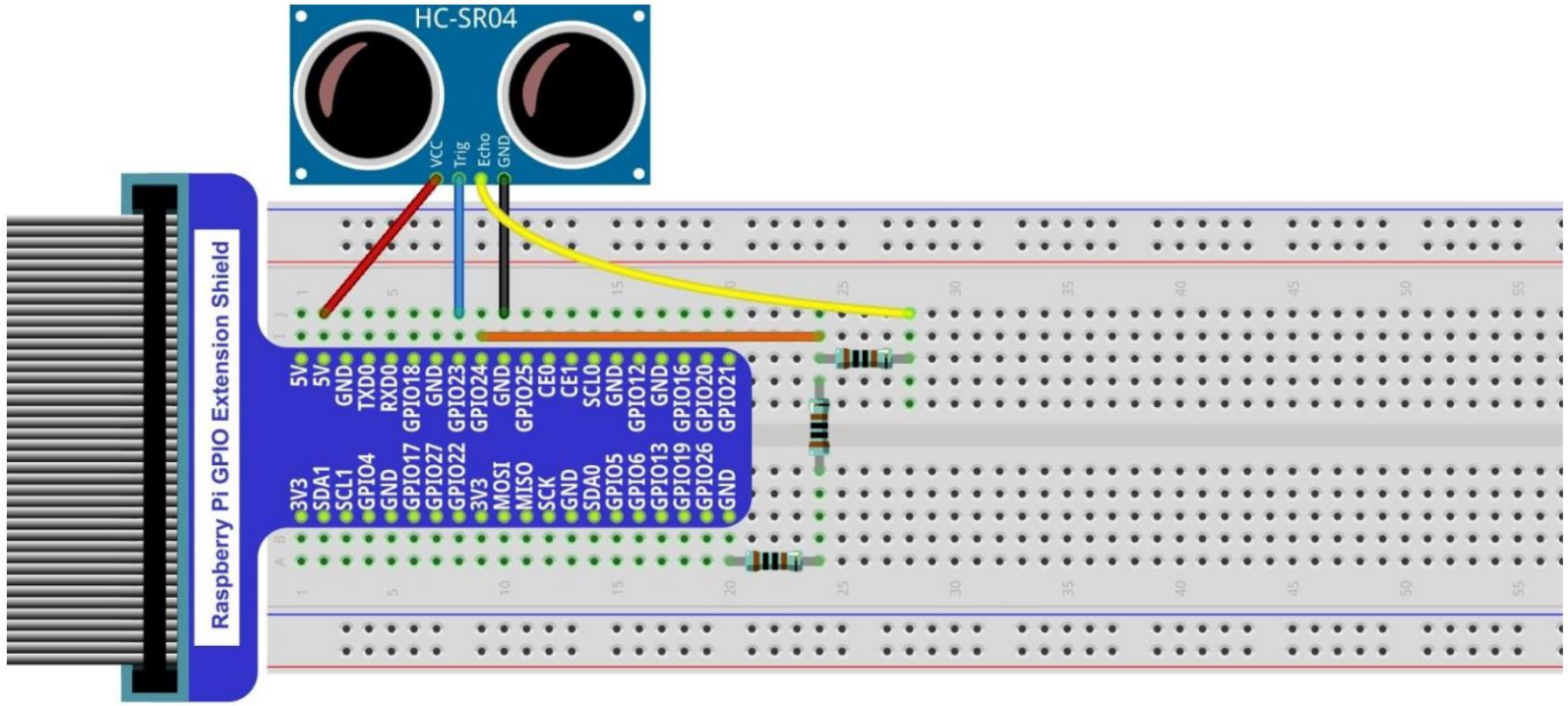
Ultrasonic Ranging



YORK U

Lab 5: Ultrasonic Distance Measurement and Data Logging

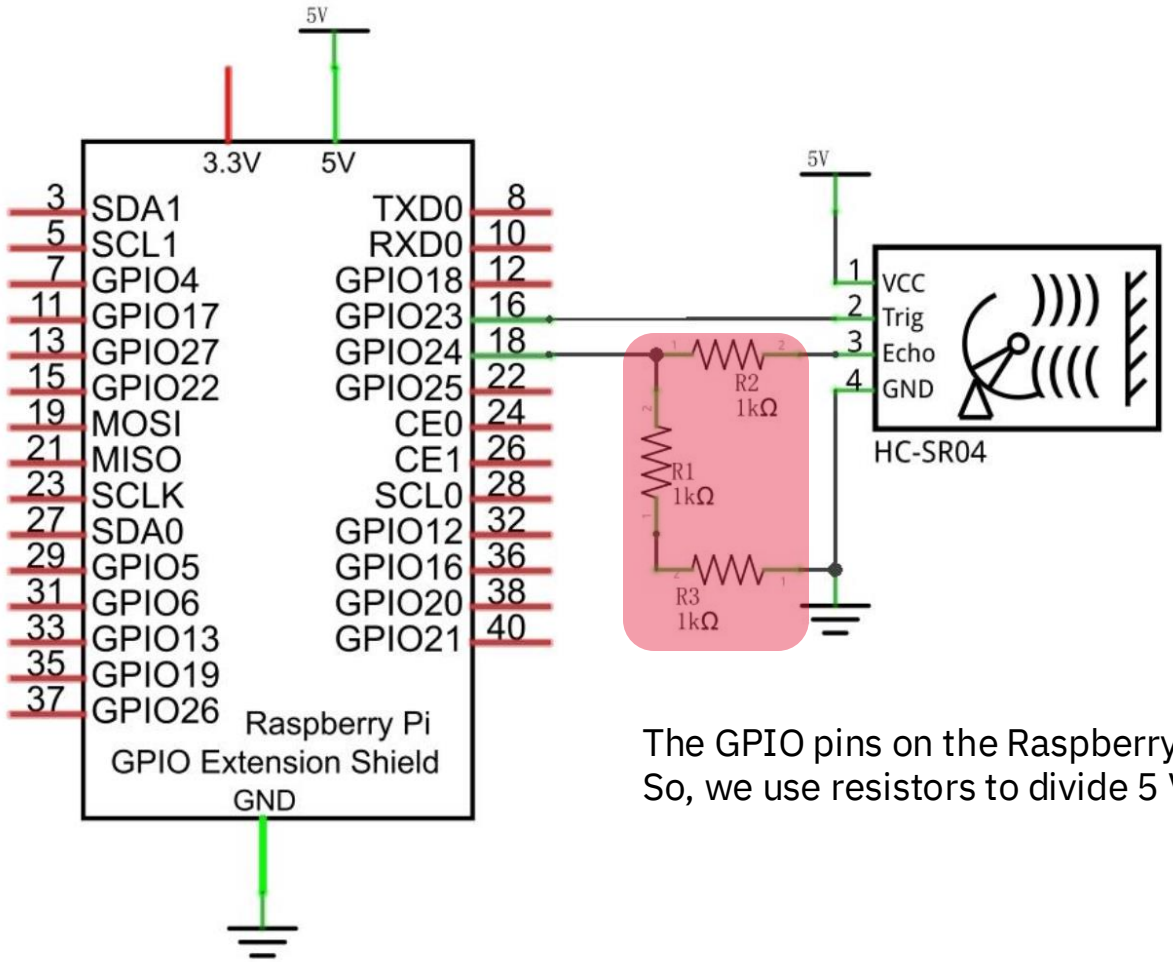
- Build the Circuit



YORK U

Lab 5: Ultrasonic Distance Measurement and Data Logging

- For more details on this circuit, please refer to Tutorial Chapter 21 (available on E-Class).

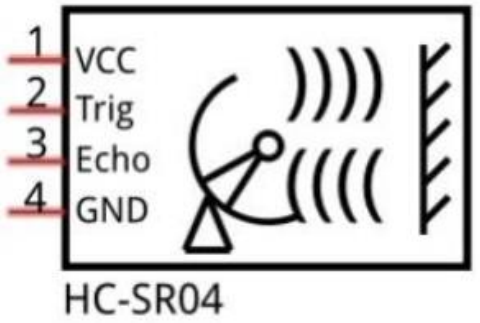


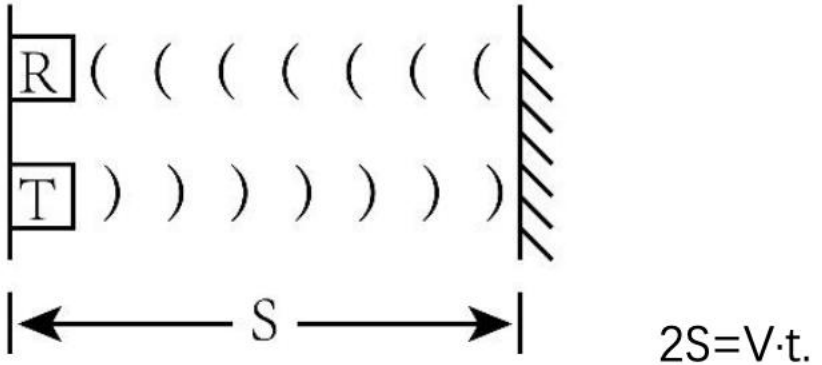
The GPIO pins on the Raspberry Pi only tolerate 3.3 V max , more can damage the Pi.
So, we use resistors to divide 5 V down to about 3.3 V before it reaches the Pi's input pin.

YORK U

What Happens Inside the Ultrasonic Module

- Pin description:



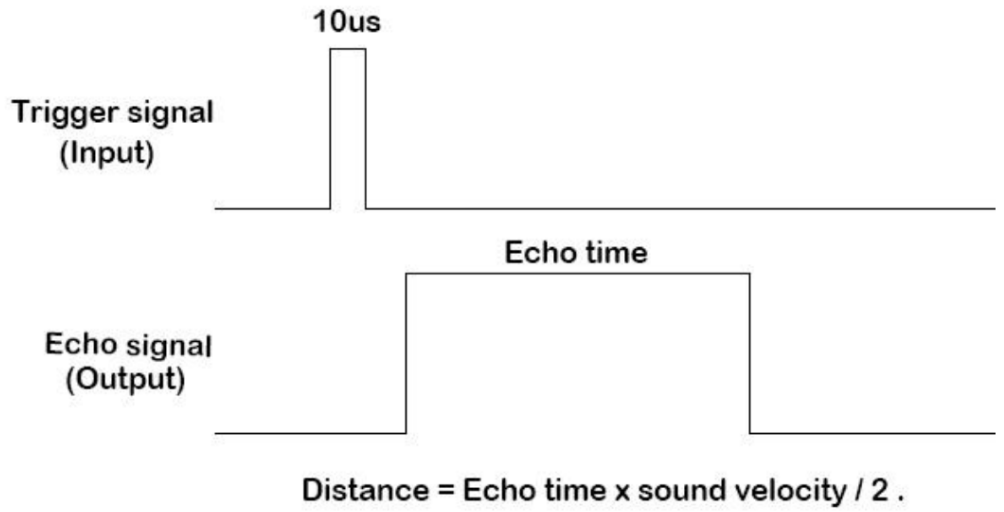


Pin	Type	Function
VCC	Power	Connects to **5V** (or 3.3V on some boards). Powers the ultrasonic module.
Trig (Trigger)	Input	This is where you send a short **10 µs HIGH pulse** from your Raspberry Pi. It tells the sensor to send an ultrasonic burst.
Echo	Output	This pin goes **HIGH** for the duration that the ultrasonic wave travels to the obstacle and back. Your code measures this pulse width to calculate distance.
GND	Ground	Connects to **GND** of your Raspberry Pi. Completes the circuit.

What Happens Inside the Ultrasonic Module

- One **transmitter** — it sends ultrasonic pulses (~40 kHz).
- One **receiver** — it listens for the reflected echo.

Step	Event	Echo Pin
Send 10 µs Trigger	Module emits sound	Goes HIGH
Waiting for echo	Listening for reflection	Stays HIGH
Echo detected	Wave reflected & received	Goes LOW
MCU measures HIGH duration	→ Time of flight	



Distance = Echo time x sound velocity / 2 .

YORK U

Lab 5: Steps

1- Test the existing code

- Run UltrasonicRanging.py.
- Confirm the sensor prints distance values correctly in the terminal.

2- Record data in a text file (TXT)

- Modify the code to save each measurement with time and distance.
- Record data for about 60 seconds while moving an object slowly toward and away from the sensor.
- Save as distance_log.txt.
  - The first column is elapsed time in seconds since the start of recording.
  - The second column is the measured distance in centimeters.
- **Specify in your report** the sampling interval (time between readings) and total duration of the recording.

```
time_s    distance_cm
0.000     50.32
0.100     49.95
0.200     49.80
...
```

3- Record data in a CSV file and visualize

- Use Python's csv module to save the same data in distance_log.csv.
- Open the CSV file in Excel.
- Insert a Scatter or Line chart with Time (s) on the X-axis and Distance (cm) on the Y-axis.
- Label the chart as **Distance vs Time**.

YORK U

Report Format (short, personal, verifiable)

Setup

- Attach a **clear photo of your circuit** showing the ultrasonic sensor (HC-SR04 or similar) connected to the Raspberry Pi (with resistors on the Echo pin if used).
- Copy and paste your **final code** (main program)
- Include **meaningful comments** explaining each main part of your code.

Observations

- Record the **distance readings** displayed in the terminal while your program runs.
- Open your **distance_log.txt** file, describe what kind of data is stored there and how it's formatted.
- Open your **distance_log.csv** file in Excel, insert a **Distance vs Time** plot.
- Include a **screenshot of a portion of the collected data** (both TXT and CSV file).
- **Copy and paste your graph** (Distance vs Time) into the report.
- Clearly **state the sampling interval** (time between readings) and **total recording duration** used.
- Compare the data from the TXT and CSV files, are they identical or slightly different? Why?
- How consistent are the readings when the object is still? What happens when the object moves faster or slower?

Analysis

- How does the Echo pin indicate that an ultrasonic pulse has returned?
- Why do we divide the total time by 2 when calculating distance?
- What are two possible causes of inaccurate or noisy distance readings?
- How could you modify your program to reduce these errors (hardware or software changes)?
- If you increase the sampling rate (shorter delay), what effect might it have on data quality or reliability?
- How could you extend this setup to measure speed or detect motion?

Rubric

- **Circuit Setup – 1 pts**

- **Code – 4 pts**
  - Program runs correctly without errors – 1 pts
  - Includes meaningful comments explaining each major section (setup, trigger, echo, data logging) – 1 pt
  - Properly logs data in both TXT and CSV files with correct formatting – 2 pt

- **Observations – 3 pts**
  - Includes terminal readings, TXT file structure, and Excel Distance vs Time plot.
  - Answers all questions clearly.

- **Analysis – 2 pts**
  - Answers all analysis questions clearly.