

Tutorial 14: Processing & Arduino

CSC 413 - Designing Creativity Support Tools
Submitted by: Brandon Rogers and Zhenhui Zhu

June 19th, 2025

Table of Contents

| | |
|---|----|
| Introduction | 3 |
| What You'll Learn | 3 |
| Sensor Interaction | 3 |
| What is Processing?..... | 4 |
| Tutorial Project | 4 |
| Materials | 4 |
| Procedures | 5 |
| Step 1 – Set up the hardware | 5 |
| Step 2 – Upload Arduino Code | 6 |
| Step 3 – Open and Run Processing Code | 7 |
| Code Breakdown | 8 |
| Arduino Code | 8 |
| Processing Code Breakdown | 9 |
| References | 12 |
| Appendix A: Arduino Sketch Code | 13 |
| Appendix B: Processing Sketch Code | 13 |
| Appendix C: Installing Processing (non-snap version) on Linux | 13 |
| Contribution Document | 14 |

Introduction

This tutorial covers the creation of an interactive visual system that responds to distance, using an Arduino and Processing. Using the HC-SR04 ultrasonic sensor instead of a conventional light sensor, we create a touchless experience where the user's hand can move closer to and further from the sensor, with dynamic visuals that respond to and encourage user interaction.

The project is a mix of simple electronics and creative code. Arduino deals with the measurement of the distance (receiving and interpreting ultrasonic echo signals), with Processing transforming it into dynamic animated graphics. This provides an opportunity to other users for physical computing, data physicalization, and tangible interaction in single project.

What You'll Learn

- How to use an ultrasonic sensor (HC-SR04) to detect distance.
- How to transmit sensor data from Arduino to a computer using serial communication.
- How to visualize live sensor input using Processing.

Sensor Interaction

The HC-SR04 ultrasonic sensor emits a high-frequency sound wave through its transmitter and listens for the echo on its receiver. By measuring the time it takes for the echo to return, the Arduino calculates the distance to the object using the speed of sound:

$$\text{distance} = \text{time} \times \text{speed of sound} \div 2$$

Based on this timing and the speed of sound, the sensor calculates the distance to nearby objects—up to about 4 meters—with reliable accuracy. This sensor works well in

projects where contactless motion detection is useful, such as gesture controls, proximity-based lighting, or sound-reactive installations.

What is Processing?

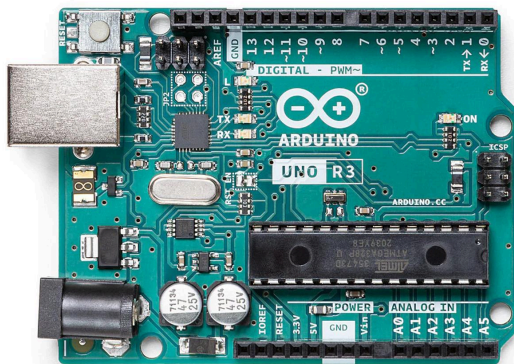
Processing is a flexible software sketchbook and a language for learning how to code within the context of the visual arts. It's built on Java and is designed to make visual and interactive programming accessible to beginners, artists, and designers.

In this tutorial, Processing plays a key role in turning physical sensor input (from the ultrasonic sensor) into real-time, animated visuals. As the Arduino sends data through the serial port, Processing reads that data and generates graphics that respond to distance. This type of system is commonly used in data physicalization, interaction design, and new media art.

Arduino & Processing Tutorial Project

Materials:

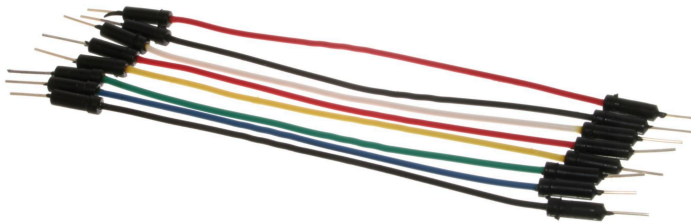
- 1 x Arduino UNO R3 board



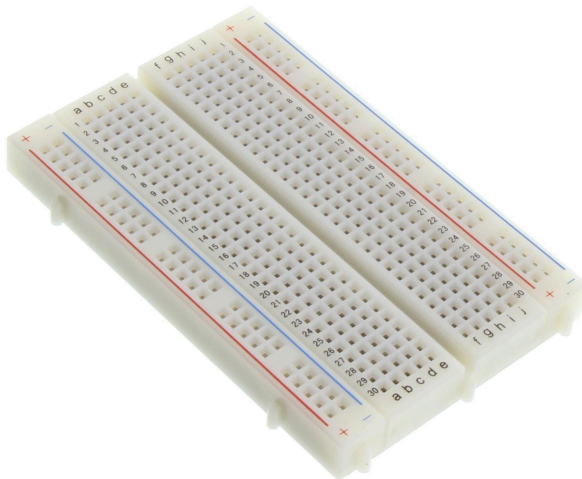
- 1 x HC-SR04 Ultrasonic Sensor



- Jumper wires



- 1 x Breadboard

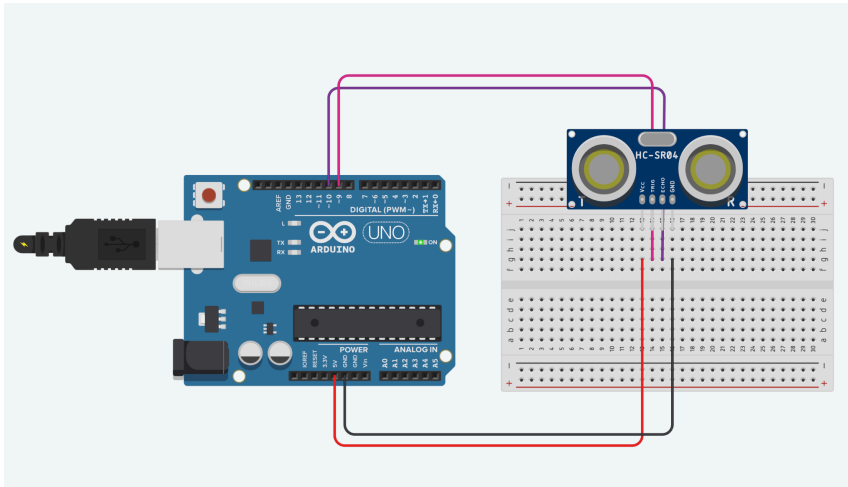


- Computer with Arduino IDE and Processing installed

Procedures:

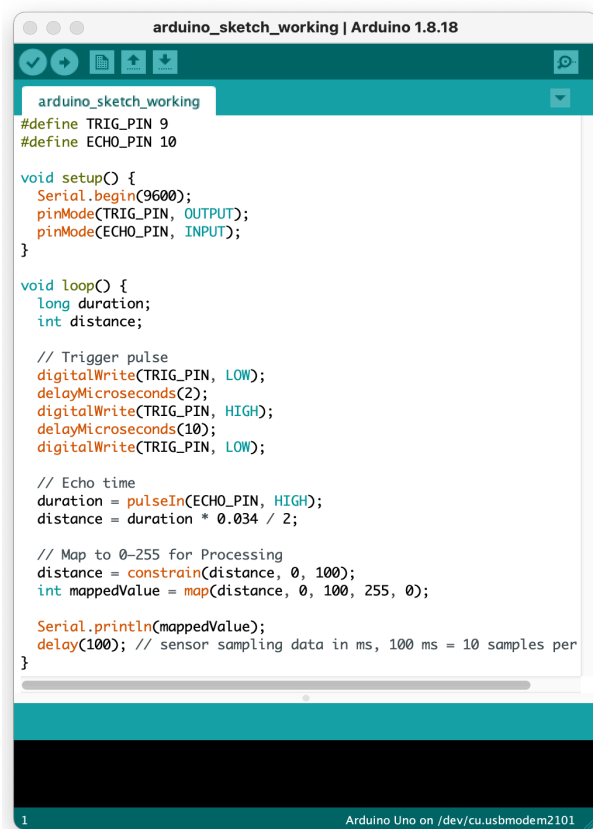
Step 1 – Set up the hardware

1. Plug the ultrasonic sensor into your breadboard
2. Connect the ultrasonic sensor to the Arduino:
 - VCC → 5V
 - GND → GND
 - Trig → Pin 9
 - Echo → Pin 10
3. Confirm all connections are secure.



Step 2 – Upload Arduino Code

1. Open the Arduino IDE.
2. Copy and paste the code from Appendix A.



```
arduino_sketch_working | Arduino 1.8.18

arduino_sketch_working

#define TRIG_PIN 9
#define ECHO_PIN 10

void setup() {
  Serial.begin(9600);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
}

void loop() {
  long duration;
  int distance;

  // Trigger pulse
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  // Echo time
  duration = pulseIn(ECHO_PIN, HIGH);
  distance = duration * 0.034 / 2;

  // Map to 0-255 for Processing
  distance = constrain(distance, 0, 100);
  int mappedValue = map(distance, 0, 100, 255, 0);

  Serial.println(mappedValue);
  delay(100); // sensor sampling data in ms, 100 ms = 10 samples per
}

1 Arduino Uno on /dev/cu.usbmodem2101
```

3. Connect your Arduino via USB.
4. Select the correct port and board type.
5. Upload the code.
6. Open the Serial Monitor to confirm it prints distance values



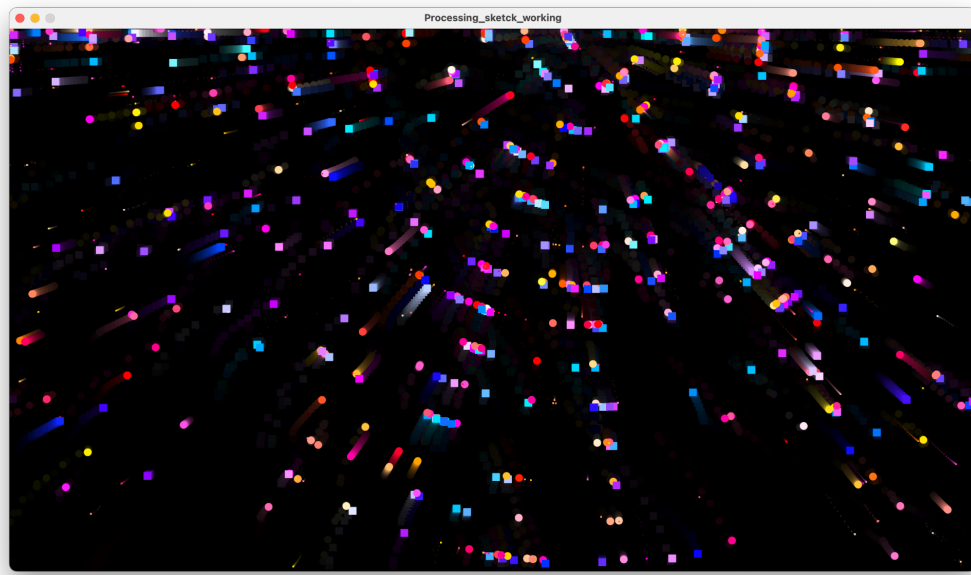
```
/dev/cu.usbmodem2101

220
182
115
166
161
153
148
115
110
110
87
131
115
102

Autoscroll Show timestamp Newline 9600 baud Clear output
```

Step 3 – Open and Run Processing Code

1. Launch Processing.
2. Open the provided sketch (see Appendix B).
3. Run the sketch.
4. Check that the visuals respond.



Code Breakdown

Arduino Code [Tutorial14_Arduino.ino]

```
#define TRIG_PIN 9  
#define ECHO_PIN 10
```

Defines the digital pins used for the ultrasonic sensor.

```
void setup() {  
  Serial.begin(9600);  
  pinMode(TRIG_PIN, OUTPUT);  
  pinMode(ECHO_PIN, INPUT);  
}
```

Initializes serial communication and sets up the pin modes.


```

void loop() {
  long duration;
  int distance;

  // Trigger pulse
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  // Echo time
  duration = pulseIn(ECHO_PIN, HIGH);
  distance = duration * 0.034 / 2;

  // Map to 0-255 for Processing
  distance = constrain(distance, 0, 100);
  int mappedValue = map(distance, 0, 100, 255, 0);

  Serial.println(mappedValue);
  delay(100); // sensor sampling data in ms, 100 ms = 10 samples per second
}

```

Sends a 10 microsecond pulse from the trigger pin to start measuring and reads the echo pin duration, converts it into distance, constrains the value, and prints it via serial.

Processing Code [Tutorial14_Processing.pde]

```

int n = 1000;

float[] m = new float[n];
float[] x = new float[n];
float[] y = new float[n];
float[] vx = new float[n];
float[] vy = new float[n];
float[] redchannel = new float[n];
float[] bluechannel = new float[n];
float[] greenchannel = new float[n];
float[] shape = new float[n];

```

Initializes 1000 particles. Each has mass, position, velocity, RGB color, and shape type.

```
Serial myPort;  
String val = "";  
int sensorVal = 0;
```

Sets up variables for serial communication and sensor value.

```
void setup() {  
  size(1280, 720); // Safe windowed mode (instead of fullScreen)  
  fill(0, 10);  
  reset();  
}
```

Sets canvas size, applies transparent black fill for trails and calls reset() to initialize particles.

```
// Print available ports  
printArray(Serial.list());  
String portName = Serial.list()[0]; // Adjust port index as needed  
myPort = new Serial(this, portName, 9600);
```

Lists serial ports and opens the first at 9600 baud.

```
void draw() {  
  if (myPort.available() > 0) {  
    val = myPort.readStringUntil('\n');  
    if (val != null) {  
      try {  
        sensorVal = Integer.parseInt(val.trim());  
      } catch (Exception e) {  
        println("Invalid input: " + val);  
      }  
    }  
  }  
}
```

Reads a line from the serial port and converts it into an integer sensor value.

```
noStroke();  
fill(0, 30);  
rect(0, 0, width, height);
```

Background update. Applies semi-transparent overlay for motion trails.

```
float sensorMappedY = map(sensorVal, 0, 255, 0, height);
```

Maps sensor value to vertical screen position.

```
for (int i = 0; i < n; i++) {  
  float dx = mouseX - x[i];           // horizontal control from mouse  
  float dy = sensorMappedY - y[i];     // vertical control from ultrasonic  
  
  float d = sqrt(dx * dx + dy * dy);  
  if (d < 1) d = 1;  
  
  float f = cos(d * 0.06) * m[i] / d * 2;  
  
  vx[i] = vx[i] * 0.4 - f * dx;  
  vy[i] = vy[i] * 0.2 - f * dy;  
}
```

Calculates attractive force toward the mouse and sensor position and updates velocity and position.

```
for (int i = 0; i < n; i++) {  
  x[i] += vx[i];  
  y[i] += vy[i];  
  
  if (x[i] < 0) x[i] = width;  
  else if (x[i] > width) x[i] = 0;  
  
  if (y[i] < 0) y[i] = height;  
  else if (y[i] > height) y[i] = 0;  
  
  if (shape[i] > 2) fill(bluechannel[i], greenchannel[i], 255);  
  else fill(255, bluechannel[i], redchannel[i]);  
  
  if (shape[i] > 2) rect(x[i], y[i], 10, 10);  
  else if (shape[i] > 1 && shape[i] <= 2) rect(x[i], y[i], 2, 2);  
  else ellipse(x[i], y[i], 10, 10);  
}
```

Then draws each particle using its color and shape.

```
void reset() {  
    for (int i = 0; i < n; i++) {  
        m[i] = randomGaussian() * 8;  
        x[i] = random(width);  
        y[i] = random(height);  
        bluechannel[i] = random(255);  
        redchannel[i] = random(255);  
        greenchannel[i] = random(255);  
        shape[i] = random(0, 3);  
    }  
}
```

Randomizes all particle attributes: position, velocity, color, and shape.

References

Processing. (n.d.). *Releases · processing/processing4*. GitHub.

<https://github.com/processing/processing4/releases>

Reference. (n.d.). Processing. <https://processing.org/reference>

Visualization with Arduino and Processing. (n.d.). <https://www.arduino.cc/education/visualization-with-arduino-and-processing/>

Appendix A: Arduino Sketch Code

[Tutorial14_Arduino.ino](#)

Appendix B: Processing Sketch Code

[Tutorial14_Processing.pde](#)

Appendix C: Installing Processing (non-snap version) on Linux

Download the latest **portable Linux x64** release from the official Processing GitHub releases page: <https://github.com/processing/processing4/releases>

Look for the file named:

`processing-4.4.4-linux-x64-portable.zip`

Save the ZIP file to your Downloads folder or another location.

Open a terminal and navigate to the folder where you saved the file.

Extract the contents of the ZIP archive:

`unzip processing-4.4.4-linux-x64-portable.zip`

Navigate into the extracted directory:

`cd Processing/bin`

Run the Processing IDE:

`./Processing`

Contribution Document

| Group Member | Contribution |
|----------------|---|
| | |
| Brandon Rogers | Arduino and Processing code |
| | Wired Arduino for code testing |
| | Appendix Sections A,B,C |
| | References Section |
| | Minor document formatting |
| | |
| Zhenhui Zhu | Code testing, verified for cross platform functionality |
| | Introduction and Processing Overview sections, including background explanation of the tools used |
| | Tutorial Project and Code Breakdown sections |
| | Helped format and edit the final document to ensure clarity and cohesion across all sections |
| | |