

# Radar Visualization Pipeline Report

Aiysha Mei Frutiger, Sandro Barbazza, Senanur Ates,  
University of Basel  
Computer Architecture

January 16, 2026

## Abstract

Abstract of project

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Planning</b>	<b>2</b>
<b>3</b>	<b>Hardware</b>	<b>2</b>
<b>4</b>	<b>Servo Sweep Control</b>	<b>2</b>
<b>5</b>	<b>System Overview: From Echo to Visualization</b>	<b>2</b>
<b>6</b>	<b>Visualization</b>	<b>3</b>
6.1	Implementation path . . . . .	3
6.2	Serial parsing and “bucket” state . . . . .	4
<b>7</b>	<b>Evaluation</b>	<b>4</b>
<b>8</b>	<b>Conclusion</b>	<b>4</b>
<b>9</b>	<b>References</b>	<b>4</b>
<b>10</b>	<b>Declaration of Authorship</b>	<b>4</b>

# 1 Introduction

We built a compact ultrasonic “radar-style” scanner that turns echo timing into a live 2D visualization. An ultrasonic pulse is emitted and the return echo duration encodes time-of-flight, which the microcontroller converts to distance. By sweeping the sensor with a servo, each measurement becomes an (*angle, distance*) pair that can be rendered in polar form. The project is primarily an educational end-to-end system that makes the hardware-software stack observable: sensing, embedded processing, USB serial transport, OS I/O abstraction, and real-time visualization.

## Contributions.

- End-to-end ultrasonic scan pipeline from time-of-flight echo to 2D visualization.
- Arduino firmware for servo sweep, echo-to-distance conversion, and newline-framed serial protocol.
- Processing visualization with robust line parsing and bucket-based refresh behaviour.

## 2 Planning

## 3 Hardware

## 4 Servo Sweep Control

## 5 System Overview: From Echo to Visualization

Figure 1 summarizes the end-to-end pipeline. The ultrasonic sensor returns an *echo pulse width*, i.e., a time duration proportional to the time-of-flight. The Arduino controls the servo sweep via PWM, triggers the sensor, measures the echo duration (e.g., with `pulseIn`), converts it to a distance in centimeters, and transmits one newline-terminated ASCII record per measurement (`angle,cm`). Over USB the payload is a byte stream which the operating system exposes as a serial device (e.g., a COM port on Windows). The Processing program opens this port, frames the incoming bytes into lines, parses angle and distance, and updates the visualization state used for rendering.



Figure 1: End-to-end pipeline from sensing on the Arduino to visualization on the host.

## 6 Visualization

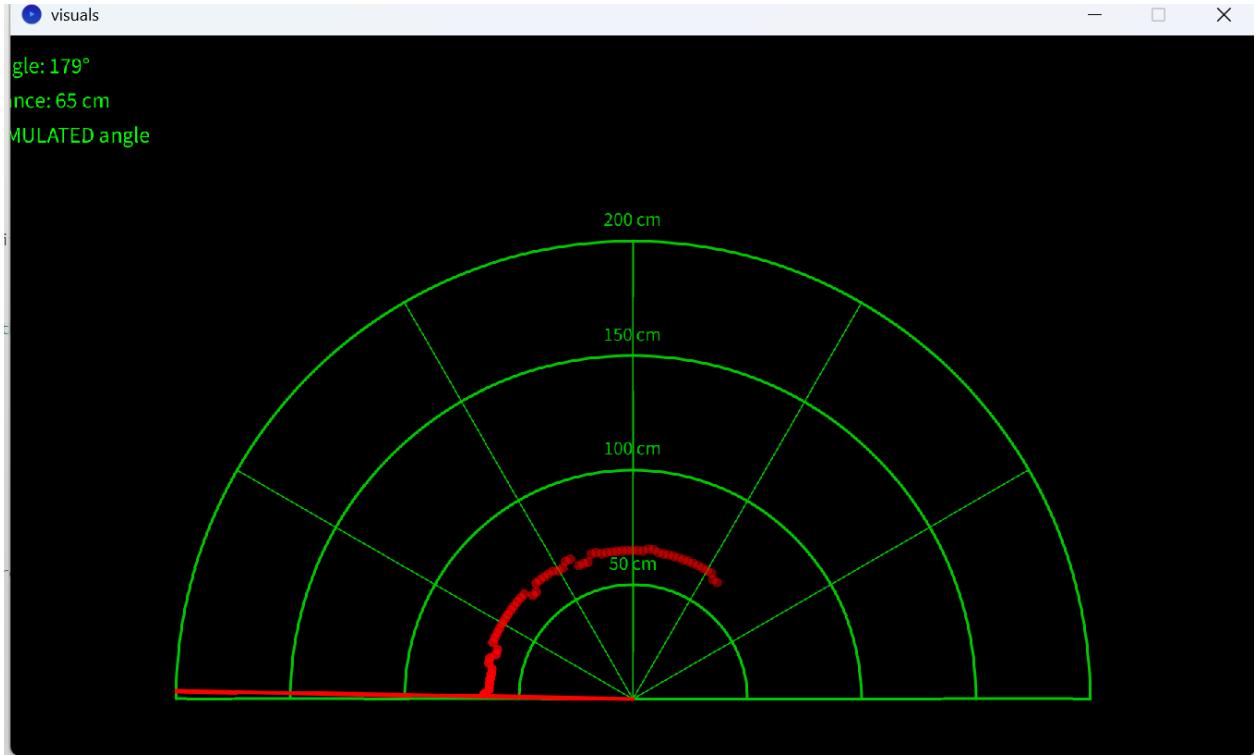


Figure 2: Final Processing-based radar visualization (grid, sweep line, and detections).

The goal of the visualization is to make streamed sensor measurements immediately interpretable as a radar-style scan. The display renders a half-circle field of view ( $0^\circ$ – $180^\circ$ ) with a moving sweep line representing the current scan direction. Each incoming measurement (*angle, distance*) is mapped to polar coordinates: the angle determines the direction of the sweep and the distance determines the radius of a “blip” point. A static background grid (concentric arcs and radial dividers) provides scale and orientation (Figure 2).

### 6.1 Implementation path

We initially prototyped the UI in C using `raylib`. While rendering worked well, the end-to-end path was unstable due to low-level serial I/O handling on Windows (port selection, timing after reset, partial lines, and port contention). To reduce OS-specific complexity and iterate faster, we switched to Processing, whose serial library directly supports line-based framing and event-driven reads.



Figure 3: C/`raylib` prototype in simulated sweep mode (no robust sensor-driven end-to-end path).

## 6.2 Serial parsing and “bucket” state

Processing treats the serial connection as a continuous byte stream but frames it into complete records using newline termination. Each line is trimmed, split by comma, and parsed into numeric values (*angle* and *distance*). Invalid readings are encoded as `-1` and interpreted as “no detection”.

To prevent unbounded accumulation of points and to mimic a radar refresh, the scan area is discretized into small angle slices (“buckets”) of fixed width (e.g., `ANGLE_BUCKET_DEG`). Each bucket stores the most recent distance for that slice and is overwritten when the sweep returns. If a `-1` reading arrives for a slice, the corresponding bucket is cleared. This keeps memory bounded and creates the intuitive behavior that detections persist briefly and disappear naturally on rescan.

## 7 Evaluation

## 8 Conclusion

We built an end-to-end ultrasonic “radar-style” system that turns echo time-of-flight into a live 2D visualization. The project made the full stack tangible: the Arduino converts timing signals into serial records, USB carries them as bytes, the OS exposes a serial port, and Processing parses the stream into a radar-style display.

A key lesson was robustness across layers. We handled practical serial-port issues (exclusive access, startup noise) and, since the servo was unavailable early on, designed the visualization to run in simulated sweep mode and switch automatically to real-angle mode when angle data is present.

**Self-assessment and outlook.** We deliberately kept the project relatively simple in order to fully understand each layer of the pipeline. With more time, we could increase complexity by adding an on-device LCD display (instead of a PC-based UI) and integrating basic controls such as a power/on-off button and a calibration routine. Overall, we achieved the original goal of a working, explainable end-to-end system.

**Division of labor.** All group members contributed to assembling the hardware setup. Sandro implemented the main Arduino sensing and communication code and built the stand/chassis. Senanur developed the Arduino servo sweep control. Aiysha implemented the Processing visualization, including parsing and bucket-based rendering.

## 9 References

- Processing (project homepage + downloads): <https://processing.org/> <https://processing.org/download/>
- Processing Serial library reference (Serial class): <https://processing.org/reference/libraries/serial/Serial.html>
- Processing Serial library reference (bufferUntil): [https://processing.org/reference/libraries/serial/Serial\\_bufferUntil\\_.html](https://processing.org/reference/libraries/serial/Serial_bufferUntil_.html)
- Processing Serial library reference (readStringUntil — common alternative used with newline framing): [https://processing.org/reference/libraries/serial/Serial\\_readStringUntil\\_.html](https://processing.org/reference/libraries/serial/Serial_readStringUntil_.html)
- Processing Serial library reference (serialEvent): [https://processing.org/reference/libraries/serial/serialEvent\\_.html](https://processing.org/reference/libraries/serial/serialEvent_.html)
- Project source code repository: <https://github.com/ateschsen/Radar.git>

## 10 Declaration of Authorship

ChatGPT was used solely to improve the clarity and coherence of the report’s language. All ideas, analyses, and interpretations presented reflect the group’s own work and research.