

180° Spatial Profiler and Obstacle Proximity Detection System Spatial Target Tracking and Real-Time Polar Radar Visualization Using an Ultrasonic Sensor, Servo Motor, Arduino, and NI LabVIEW

Amar Hamdan¹ Neeraj Jyothidas² Dr. Saravanakumar R.³

Dept. Of Electronics And Instrumentation, School Of Electrical Engineering
Vellore Institute Of Technology, Vellore, India

Abstract— This report documents the design, implementation and replication procedure of the 180° Spatial Profiler, a low-cost mechanically scanned ultrasonic SONAR built for a Virtual Instrumentation course. An HC-SR04 ultrasonic ranger mounted on an SG90 servo is swept from 0° to 180° and back in 1° steps under the control of an Arduino Uno. At each step the firmware triggers the sensor, times the echo with `pulseIn()`, converts the pulse width to integer centimetres using 58.3 $\mu\text{s}/\text{cm}$, and sends an ASCII record "angle,distance" at 115200 baud over USB. A National Instruments LabVIEW virtual instrument reads the stream through VISA, converts the polar readings to Cartesian coordinates, and presents a polar radar display and a distance-versus-angle graph. The report gives the underlying theory, the complete verified firmware with a line-by-line explanation, a wiring and assembly guide, a reference LabVIEW implementation, validation protocols, a troubleshooting guide and the known limitations. The authors report that the assembled system works; however, no quantitative measurement dataset has yet been collected, so no accuracy, range or resolution is claimed. Analytical estimates, such as a 15° beam footprint of about 26 cm at 1 m and a nominal 10.86 s per 181-step pass, are labelled as such, and the results tables are provided blank.

Index Terms— Ultrasonic ranging, SONAR, Arduino Uno, HC-SR04, SG90 servo, LabVIEW, virtual instrumentation, serial communication, polar mapping, time of flight.

I. INTRODUCTION

Motivation

Knowing how far away the nearest obstacle is, and in which direction, is one of the oldest problems in mobile sensing. Sonar-based mapping was developed for mobile robots in the 1980s [1], [2], and ultrasonic ranging is still found in parking aids, level gauges, hobby robots and teaching laboratories [3]. A 40 kHz transducer pair is inexpensive, needs no optics and works in the dark. A single ranger, however, only answers "how far in front of me?". To obtain a plan view of the surroundings the sensor must be pointed in many directions. Scanning lidars do this well but cost far more than a course project can justify; an ultrasonic ranger on a hobby servo is a low-cost alternative that produces a coarse polar picture of the scene.

1. Educational Relevance

The instrument touches every layer of a measurement chain: a transducer, a timing problem, an actuator that must be scheduled, a serial link, text parsing, a coordinate transformation and a graphical display. It therefore combines embedded programming, instrumentation concepts (resolution, accuracy, systematic error, repeatability) and virtual instrumentation in one small project, and it exposes students to the limits of a cheap sensor rather than hiding them.

2. Objectives, Scope And Contribution

The objectives are to (i) sweep an HC-SR04 across a half plane from 0° to 180° in 1° steps using an SG90 servo, (ii) measure the echo time with an Arduino Uno, (iii)

stream angle and distance to a PC as compact ASCII records over USB, and

(iv) display the data in LabVIEW as a polar radar view and a distance-versus-angle graph with an obstacle-proximity indication. The scope is a single-sensor, two-dimensional scan with an Arduino Uno, an HC-SR04 and an SG90. The authors report that the built system is working.

The contributions of this report are: the verified firmware with a detailed explanation, an exact description of the serial protocol, hardware and wiring documentation, a reference LabVIEW implementation, a step-by-step replication procedure, validation and troubleshooting guides, and an honest statement of the limits of the design. Practical limitations are the wide ultrasonic beam, the open-loop servo angle, specular reflection, temperature sensitivity and the slow refresh of a mechanical scan (Section XIII).

3. Evidence Conventions

Every quantitative statement carries one of four evidence levels, named where a number appears: implemented (present in the authors' firmware or hardware), analytical (derived from equations and datasheet values), proposed (not implemented) and not yet measured. Only data collected with the protocol of Section X may be presented as experimental results. Figure captions state the image type: original diagrams drawn for this report, author-supplied photographs or screenshots (placeholders in this draft), or measured-result graphs (none yet). No externally sourced images are embedded in this draft.

II. LITERATURE REVIEW AND TECHNICAL BACKGROUND

1. Ultrasonic Time-Of-Flight Ranging

In pulse-echo ranging a transducer emits a short ultrasonic burst and the range follows from the round-trip time and the speed of sound [3]. The speed of sound in air rises with temperature and, to a smaller extent, with humidity; Cramer gives accurate expressions for its dependence on temperature, pressure, humidity and CO₂ concentration [4]. The HC-SR04 module used here implements pulse-echo ranging internally: a 10 μ s trigger pulse makes it emit a 40 kHz burst, and it returns a pulse on its echo pin whose width equals the round-trip time [5].

2. Servo-Scanned Sonar, Obstacle Detection And Mapping

Elfes showed that sonar readings could be turned into an occupancy grid, storing a probability of occupancy per cell rather than a binary value, because a single return does not say where inside the beam the reflector lies [1]. Moravec and Elfes combined many wide-angle readings into high-resolution maps [2]. Borenstein and Koren used ultrasonic readings in a histogram grid for fast obstacle avoidance and discussed the practical difficulties of wide beams, crosstalk and missed returns [6]. Thrun et al. treat the beam through probabilistic sensor models [7]. The present instrument is simpler: one sensor rotates about a fixed axis, there is no map fusion, and the display shows the most recent range for each commanded angle.

3. Beam Width, Angular Resolution, Reflections And Missed Detections

Kuc and Siegel developed a physically based model of acoustic reflection from planes, corners and edges, and showed why smooth surfaces at oblique incidence may return no echo toward the sensor [8]. This specular behaviour matters for any servo-swept ranger because a wall seen at a grazing angle can fail to appear in the scan. The survey by Kleeman and Kuc collects these effects (wide beam, specularity, crosstalk and multiple reflections) together with processing techniques [3]. Because the HC-SR04 datasheet specifies a measuring angle of about 15° [5], the angular resolution of the scan is limited by the beam, not by the 1° servo step (Section VI).

4. Labview Virtual Instrumentation And Serial Acquisition

A LabVIEW virtual instrument (VI) consists of a front panel (controls and indicators) and a graphical block diagram [9]. Serial instruments are accessed through the NI-VISA API, which provides functions to configure a serial port, read and write bytes and close the session, and which can end a read on a termination character [10]. These facilities are what the reference implementation in Section VIII relies on.

5. Positioning Of This Work

Hobbyist versions of the “servo plus HC-SR04” idea commonly use a Processing or Python sketch to draw a sweep line (not surveyed systematically here). This report instead uses LabVIEW as the front end and concentrates on documentation, reproducibility, the limits that follow from the transducer, and a validation protocol, without asserting performance that has not been measured.

III. OVERALL SYSTEM ARCHITECTURE

The measurement chain, from the scene to the screen, is:

- **Obstacle** reflects ultrasound back toward the sensor.
- **Ultrasonic transmission and reflection:** the HC-SR04 emits a 40 kHz burst when triggered and listens for the echo.
- **Echo timing:** the module raises its ECHO pin for the round-trip time; the Arduino measures this width.
- **Distance calculation:** the Arduino divides the width in μs by 58.3 to obtain centimetres.
- **Angle-distance packet:** the Arduino prints “angle,distance” for the servo’s commanded angle.
- **USB communication:** the packet reaches the PC as a COM port stream at 115200 baud.
- **LabVIEW parsing:** VISA Read and string parsing recover two integers.
- **Coordinate transformation and visualisation:** polar values become (x, y) for the radar view, and the distance-versus-angle graph is updated.

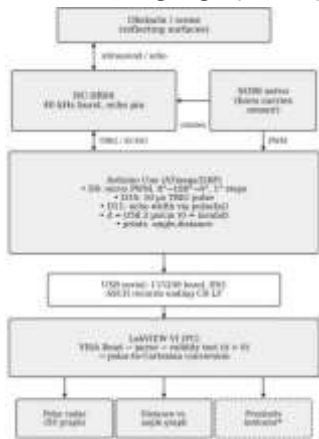


Fig. 1. Overall system block diagram (original diagram).
*The proximity indicator is described in the project

source material; its implementation status is to be confirmed (Section VIII-G).

Fig. 1 shows the blocks. The **HC-SR04** converts a trigger pulse into an acoustic measurement and an echo pulse. The **SG90 servo** turns the sensor to the commanded direction; the sensor is mounted on the servo horn, so the servo angle is the direction of the acoustic axis. The **Arduino Uno** is the only controller: it generates the servo command on D9, the trigger on D10, measures the echo on D11, computes the distance and prints it. The **USB serial link** carries power and data. The **LabVIEW VI** parses each record, rejects invalid readings and draws the displays.

There are five signal paths: the acoustic path (sensor to target and back), the electrical trigger and echo lines (D10, D11), the servo PWM line (D9), the USB serial stream (Arduino to PC) and the data flow inside the VI. Data flow is one-way: the firmware contains no serial-input code, so it cannot be started, stopped or commanded from the PC. It begins sweeping at power-up and streams continuously; any “Start/Stop” control in the VI can therefore only start or stop reading and displaying, not the sweep itself. Because each record carries its own angle, the VI does not need to synchronise with the start of a sweep.

1. Hardware Components And Selection

Table I summarises the manufacturer figures quoted for the three main parts. These are datasheet values [5], [11], [12]; clone modules vary, so they must be checked against the authors’ actual parts. None of them is a measurement made in this project.

Table I Key Datasheet Figures (Not Measured)
Sg90 Control 50 Hz Pwm, Pulse-Width Positioning

Part	Quantity	Datasheet value
HC-SR04	Supply / current	5 V DC / about 15 mA
	Frequency	40 kHz
	Range / resolution	2–400 cm / 0.3 cm
	Measuring angle	15°
	Trigger	10 μs TTL pulse
SG90	Control	50 Hz PWM, pulse-width positioning
	Rotation	about 180°
	Speed	about 0.1 s per 60° (4.8 V)
	Stall torque	about 1.8 kg·cm (4.8 V)
Uno	MCU / clock	ATmega328P / 16 MHz
	Logic level	5 V
	PC link	USB, appears as a COM port



Fig. 2. Author-supplied photographs of the actual components (placeholders):

(a) Arduino Uno, (b) HC-SR04, (c) SG90, (d) breadboard and wiring.

3.

Arduino Uno

Purpose: sequences the scan, drives the servo, triggers and times the sensor, and reports each reading. **Operation:** the ATmega328P runs at 16 MHz [12]; the Servo library generates the servo pulses [13], and `pulseIn()` measures the echo width [14]. A USB-serial interface presents the board as a COM port. **Why chosen:** its 5 V logic matches the HC-SR04 echo level, so no level shifter is needed, and its COM port is well supported by VISA on Windows. **Limitations:** the firmware is single-threaded and blocks during measurement; the USB port supplies limited current; opening the serial port resets the board.

4. Hc-Sr04 Ultrasonic Sensor

Purpose: measures range to the nearest reflecting surface inside its beam. **Operation:** a 10 μ s HIGH pulse on TRIG starts a measurement; the module transmits a 40 kHz burst and then holds ECHO HIGH for the round-trip time [5]. **Why chosen:** it is cheap, widely

documented and 5 V compatible. **Limitations:** the roughly 15° beam smears bearing (Section VI), there is a 2 cm minimum range, soft or angled surfaces may return no echo, and the result depends on air temperature.

5. Sg90 Servo Motor

Purpose: points the sensor. **Operation:** a 50 Hz pulse train positions the horn in proportion to pulse width [11]; the servo's internal feedback closes its own loop, but the Arduino receives no position feedback. **Why chosen:** light, cheap and sufficient for a sensor weighing a few grams. **Limitations:** deadband, backlash and wiring tension mean the true angle may differ from the commanded angle by an unknown amount; start-up and reversal current peaks can disturb a shared supply.

6. Usb Cable, Breadboard, Jumper Wires And Mounting

The **USB cable** must be a data cable (some are charge-only); it powers the Uno and carries the serial stream. The **breadboard and jumpers** allow quick assembly and share the

+5 V and ground rails; loose or long wires cause intermittent readings and should be strain-relieved so they do not pull on the rotating sensor. The **mechanical mounting** fixes the servo body to a stable base and the sensor to the horn, with the sensor axis over the servo axis so the scan is symmetric about the centre line. The authors should record here how the sensor was fixed to the horn and what base was used: [AUTHORS TO COMPLETE].

7. Power Supply And Additional Components

The Uno and sensor are powered through USB. The servo draws much more current during start and reversal than the sensor, and a dip on a shared rail can reset the Uno or corrupt timing. A common remedy is an electrolytic capacitor (about 470 μ F) across the servo supply or a separate 5 V supply with a common ground; this is proposed practice, not confirmed for the actual build. [AUTHORS TO STATE: servo powered from the Uno 5V pin or from an external supply; capacitor fitted or not; any other parts used.]

8. Hardware Wiring And Assembly

A. Pin Connections

Table II lists the connections. The signal pins D9, D10 and D11 are implemented in the firmware (constants SERVO_PIN, TRIG_PIN and ECHO_PIN) and are therefore verified against the code. The power and ground wiring follows standard

practice for these parts and must be confirmed against the

TABLE II
9. Pin And Power Connections

Signal	Uno pin	Connected to	Status
Trigger	D10	HC-SR04 TRIG	From firmware
Echo	D11	HC-SR04 ECHO	From firmware
Servo signal	D9	SG90 signal (orange)	From firmware
Sensor supply	5V	HC-SR04 VCC	To confirm
Sensor ground	GND	HC-SR04 GND	To confirm
Servo supply	5V or external 5 V	SG90 V+ (red)	To confirm
Servo ground	GND (common)	SG90 GND (brown)	To confirm
Serial / power	USB	PC COM port	115200 baud, 8N1

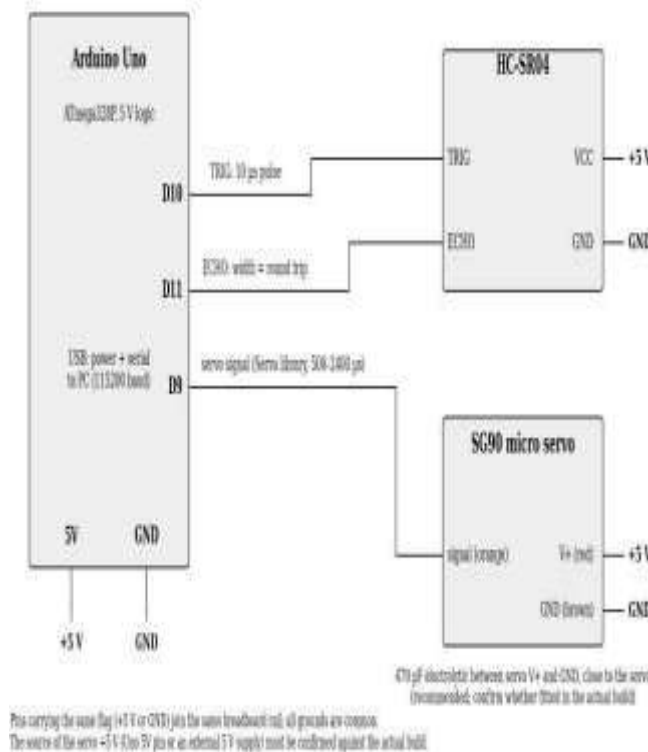


Fig. 3. Wiring diagram (original diagram). Signal connections follow the firmware constants; power connections and the servo supply source are to be confirmed against the built unit.

1 Mechanical Mounting and Alignment

Fix the servo body to a rigid base so it cannot rotate when the horn moves. Fix the sensor to the horn with its two transducers facing outward and with the sensor axis over the servo's rotation axis; otherwise the scan origin is offset and the polar view is distorted at short range. Choose the zero direction deliberately. The firmware moves the servo to 0° at start-up, so attach the horn while the servo rests at a known angle (for example, run a temporary centring sketch that only calls attach(9, 500, 2400) and write(90), which is an assembly aid and not part of the project firmware), so that 90° corresponds to straight ahead. Leave enough slack in the sensor wires so they do not pull on the horn through the whole 180° travel.

2. Power Supply and Common Ground

All grounds (Uno, sensor, servo and any external supply) must be joined. If the servo is fed from an external 5 V supply, connect only its +5 V to the servo and leave the Uno 5V pin for the sensor. A capacitor of about 470 µF across the servo supply, placed near the servo, reduces supply dips (proposed practice). Symptoms of an inadequate supply are listed in Table XV.

Wiring Checks and Safe Start-Up

- With USB unplugged, compare every wire against the connection table and the diagram; check polarity of the servo and sensor connectors.
- Check there is no short between the +5 V and GND rails (continuity test).
- Clear the sweep area of loose objects; the servo starts moving immediately after the 1 s start-up delay and then sweeps continuously.
- Plug in USB, watch for the servo moving to 0° and sweeping, and ensure that neither the sensor nor its wires snag at the end stops.
- If the servo buzzes at 0° or 180°, unplug at once: the pulse range 500–2400 µs may be driving it into its mechanical stop.

80 cm ($\alpha \approx 5.7^\circ$) would therefore span roughly 21°. Real beam sensitivity falls off toward the edge, so the true spread will differ and must be measured.

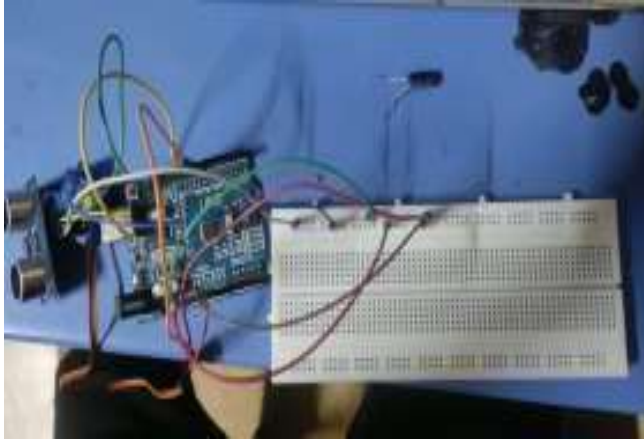


Fig. 4. Author-supplied photograph of the complete assembled scanning unit (placeholder).

IV. THEORY AND MATHEMATICAL MODEL

1 Time-of-Flight Ranging

The sensor measures the time Δt between emitting a burst and receiving its echo. The sound travels to the target and back, so the one-way distance is

$$d = c\Delta t/2 \quad (1)$$

where d is the distance, c the speed of sound and Δt the round-trip echo time; the factor of two removes the return trip. At $c =$

$m/s = 0.03432 \text{ cm}/\mu s$, one centimetre of range costs $2/0.03432 \approx 58.3 \mu s$ of round trip, so for Δt in μs and d in cm

$$d_{cm} \approx \Delta t_{\mu s}/58.3 \quad (2)$$

This is the "divide by 58" rule, and the firmware constant `US_PER_CM`. As an analytical check, a target at 2 cm gives an echo width of about 117 μs and one at 400 cm about 23.3 ms.

2. Temperature Dependence and Uncertainty Sources

The speed of sound in dry air is approximately [4]

$$c \approx 331.3 + 0.606 T \quad (3)$$

with c in m/s and T in $^{\circ}C$. Because the firmware fixes the constant at 58.3 $\mu s/cm$ (equivalent to $c \approx 343.2 \text{ m/s}$, i.e. $20^{\circ}C$), a different air temperature produces a proportional range error. With $c_0 = 343.2 \text{ m/s}$ the fractional error of the reported range is

$$\varepsilon = c_0/c(T) - 1 \approx -0.0018 (T - 20) \quad (4)$$

that is, about -0.18% per $^{\circ}C$ above $20^{\circ}C$ (analytical). For a 100 cm target this is about +1.7 cm at $10^{\circ}C$, -1.8 cm at $30^{\circ}C$ and -3.5 cm at $40^{\circ}C$. Other uncertainty sources are the 1 cm rounding in the firmware ($\pm 0.5 \text{ cm}$), the sensor's own resolution (0.3 cm, datasheet), echo-edge jitter, humidity (smaller effect), target tilt and surface, beam spread and servo angle error. None of these has been quantified by measurement in this project.

3. Beam Width and Angular Resolution

For a measuring angle β the beam footprint at range d is

$$w = 2d \cdot \tan(\beta/2) \quad (5)$$

With $\beta = 15^{\circ}$ (datasheet) w is about 26 cm at 1 m and 53 cm at 2 m (analytical; Fig. 5). Two objects closer together than w at that range cannot be separated by the scan however small the servo step is. One degree of servo rotation moves the axis by only 1.75 cm at 1 m, so successive beams overlap almost completely: the 1° step is the angular step, not the angular resolution. For an idealised sharp-edged beam, a small object of angular width α appears over about $\alpha + \beta$; an 8 cm object at

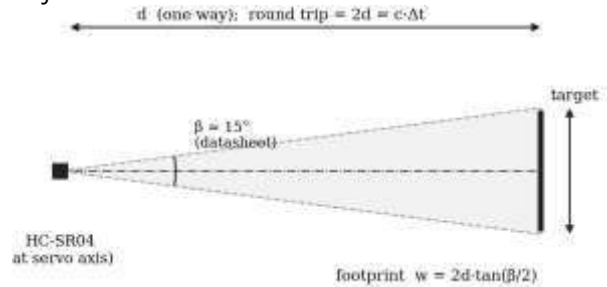


Fig. 5. Time-of-flight and beam-footprint geometry (original diagram, not to scale).

4. Polar Coordinates and Transformation

Each record is a polar measurement (r, θ) with r the distance in cm and θ the commanded servo angle in degrees. For a Cartesian display,

$$x = r \cos \theta, y = r \sin \theta \quad (6)$$

with θ converted to radians $(\theta \cdot \pi/180)$ before the trigonometric functions are applied. The origin is the servo axis; θ is measured from the $+x$ axis counter-clockwise, so 0° lies on the right of the display, 90° straight up (ahead of the sensor) and 180° on the left (Fig. 6). Which physical direction corresponds to 0° depends on how the horn was fitted and must be

verified (Stage 9, Section IX). Because only a half plane is scanned, $y \geq 0$ and the display needs x from -400 to $+400$ cm and y from 0 to 400 cm. Readings with $r = 0$ are invalid and must not be converted, or they would plot at the origin.

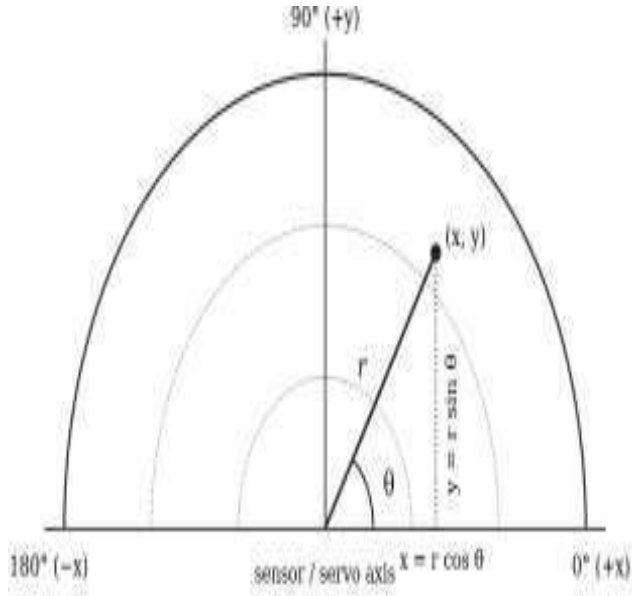


Fig. 6. Angle convention and polar-to-Cartesian transformation used for the radar display (original diagram).

5. Resolution, Accuracy and Repeatability

Following standard measurement-systems terminology [15], resolution is the smallest change the instrument can display (here 1 cm, because the firmware prints integer centimetres), accuracy is closeness to the true value, and repeatability is the scatter of repeated readings of an unchanged target. High resolution does not imply high accuracy: an uncompensated temperature error can exceed several resolution steps. Accuracy and repeatability of this instrument are not yet measured.

6. Arduino Firmware And Programming Guide

A. Installing, Compiling and Uploading

- Install the Arduino IDE from the official Arduino website. The Servo library is included with the IDE.
- Connect the Uno with a USB data cable.
- Choose Tools → Board → Arduino Uno.
- Choose Tools → Port and select the new COM port (if unsure, unplug and re-plug the board and see which entry appears).

- Paste the firmware of Listing 1 into a new sketch and save it.
- Click Verify; expect a successful compilation message.
- Click Upload; the servo should move to 0° and then begin sweeping.
- Open Tools → Serial Monitor, select 115200 baud, and check for lines such as "45,102". Garbled text means the baud rate is wrong. Close the Serial Monitor before starting LabVIEW, because only one program can hold the COM port.

Typical faults: no port listed (charge-only cable or missing driver), upload errors (wrong board or port selected, or another program holding the port).

A. Firmware Listing

```
#include <Servo.h>

// pins
const int TRIG_PIN = 10; // sensor TRIG -> D10
const int ECHO_PIN = 11; // sensor ECHO -> D11
const int SERVO_PIN = 9; // servo orange wire -> D9

// scan settings
const long BAUD_RATE = 115200;
const int STEP_DEG = 1; // angle step
const unsigned long SETTLE_MS = 15; // wait for servo to reach angle
const unsigned long STEP_PERIOD_MS = 60; // total time per step
const unsigned long ECHO_TIMEOUT_US = 30000UL; // max echo time at 400 cm
const float US_PER_CM = 58.3; // round-trip microseconds per cm
const int MIN_CM = 2;
const int MAX_CM = 400;

Servo scanner;

// Returns distance in cm, or 0 if there is no valid echo.
int measureCm() {
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);

    unsigned long t = pulseIn(ECHO_PIN, HIGH, ECHO_TIMEOUT_US);
    if (t == 0) return 0;
    float d = t / US_PER_CM;
    if (d < MIN_CM || d > MAX_CM) return 0;
    return (int)(d + 0.5f);
}

void scanStep(int angle) {
    unsigned long start = millis();
    scanner.write(angle);
    delay(SETTLE_MS);
    int d = measureCm();
    Serial.print(angle);
    Serial.print(',');
    Serial.println(d);
    while (millis() - start < STEP_PERIOD_MS) delay(1);
}
```

```
void scanStep(int angle) {
    unsigned long start = millis();
    scanner.write(angle);
    delay(SETTLE_MS);
    int d = measureCm();
    Serial.print(angle);
    Serial.print(",");
    Serial.println(d);
    while (millis() - start < STEP_PERIOD_MS) delay(1);
}

void setup() {
    Serial.begin(BAUD_RATE);
    pinMode(TRIG_PIN, OUTPUT);
    pinMode(ECHO_PIN, INPUT);
    digitalWrite(TRIG_PIN, LOW);
    scanner.attach(SERVO_PIN, 500, 2400);
    scanner.write(0);
    delay(1000);
}

void loop() {
    for (int a = 0; a <= 180; a += STEP_DEG) scanStep(a);
    for (int a = 180; a >= 0; a -= STEP_DEG) scanStep(a);
}
```

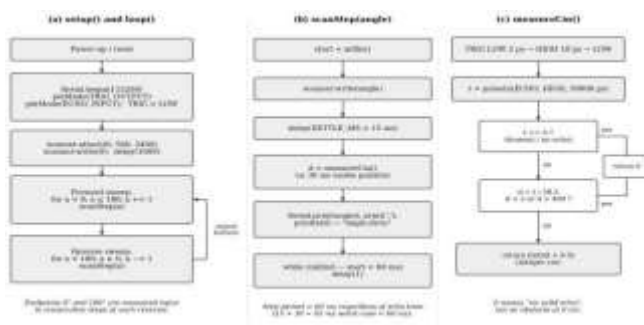


Fig. 7. Firmware flowchart (original diagram): (a) setup() and loop(), (b) scanStep(), (c) measureCm().

A. Line-by-Line Explanation

B. Library and pins. #include <Servo.h> makes the Servo class available, which generates the servo pulse train and provides attach() and write(). D10 is the trigger output, D11 the echo input and D9 the servo signal. On an Uno the Servo library uses Timer1, which disables analogWrite() on D9 and D10; this firmware does not use analogWrite(), so there is no conflict [13].

Table Iii Firmware Constants

Constant	Value	Meaning
TRIG_PIN	10	Trigger output
ECHO_PIN	11	Echo input
SERVO_PIN	9	Servo signal
BAUD_RATE	115200	Serial speed (bit/s)
STEP_DEG	1	Angle increment
SETTLE_MS	15 ms	Wait after servo command
STEP_PERIOD_MS	60 ms	Time allotted per step
ECHO_TIMEOUT_US	30000 μs	Max wait for echo
US_PER_CM	58.3	Round-trip μs per cm
MIN_CM, MAX_CM	2, 400	Accepted range (cm)

A. Scan parameters. BAUD_RATE (115200) sets the serial speed. STEP_DEG (1) is the angular increment. SETTLE_MS

(15) is a fixed wait after the servo command. STEP_PERIOD_MS (60) is the time allotted to one step.

ECHO_TIMEOUT_US (30000UL) is the longest the firmware waits for an echo; it exceeds the 23.3 ms needed for 400 cm. US_PER_CM (58.3) converts echo width to range. MIN_CM and MAX_CM (2 and 400) bound the accepted range. These three times play different roles: the settling delay is an

unconditional pause for the servo, the echo timeout is an upper bound on how long the measurement may block, and the step period is the total budget per angle that fixes the scan rate.

A. measureCm(). The function drives TRIG LOW for 2 μs to give a clean edge, HIGH for 10 μs (the pulse the sensor requires) and LOW again. pulseIn(ECHO_PIN, HIGH, 30000) waits for ECHO to rise, times the HIGH interval in μs and returns 0 if no complete pulse occurs within the timeout [14]. A return of 0 gives "no echo". Otherwise $d = t / 58.3$, which is (2): the pulse width is the round-trip time, and the constant 58.3 μs/cm already contains the factor of two. Values below 2 cm or above 400 cm are rejected as 0, and (int)(d + 0.5f) rounds to the nearest whole centimetre, so the printed range has ±0.5 cm quantisation.

B. scanStep(angle). It records the start time, commands the servo with write(angle) (which returns at once while the servo moves), waits 15 ms, takes one measurement, prints the angle, a comma and the distance with println(), and then idles in a delay(1) loop until 60 ms have elapsed since the start. The 60 ms therefore includes settling, measuring and printing.

C. setup(). It opens the serial port at 115200 baud, sets TRIG as output and ECHO as input, holds TRIG LOW, attaches the servo to D9 with pulse limits of 500 μs (0°) and 2400 μs (180°), moves to 0° and waits 1 s for the servo to arrive.

D. loop(). The first for-loop commands 0° to 180° and the second 180° back to 0°, both inclusive. Consequently the 180° position is measured in two consecutive steps at the reversal, and the 0° position likewise at the start of each new cycle. Each cycle is 362 steps. This behaviour belongs to the original firmware and is preserved.

E. Serial Protocol

The Arduino sends one ASCII line per measurement in the form angle,distance. The examples below are illustrative, not experimental results:

```
0,45
1,44
2,44
3,0
```

The angle is an integer in degrees (0–180) and the distance an integer in centimetres (0 or 2–400). A comma separates the fields. Serial.println() ends each record with carriage return and line feed (CR LF, 0x0D 0x0A); LabVIEW should use LF as the termination character, so a trailing CR remains in the string and is harmless to a numeric parse. A distance of **0 means an invalid reading** (timeout or out-of-range echo), not an obstacle at 0 cm, and LabVIEW must exclude it from target plotting. The angle is the commanded angle; there is no timestamp, no validity flag and no direction field (direction can be inferred from successive angles). A record is 5 to 9 bytes ("180,400" plus CR LF is 9), which takes at most 0.78 ms at 115200 bit/s 8N1 (analytical).

TABLE IV

Parameter	Value
Baud rate	115200
Data bits / parity / stop	8 / none / 1
Flow control	none
Delimiter	comma
Firmware terminator	CR LF (println)
LabVIEW termination char	LF (0x0A), enabled

A. Timing and Scan-Rate Analysis

The nominal scan time follows from the step period [13] as $T_{\text{scan}} = N \cdot T_{\text{step}}$ (7)

With $T_{\text{step}} = 60$ ms the step rate is at most 16.7 steps/s. One pass of $N = 181$ positions takes 181×60 ms = 10.86 s, and a forward-plus-reverse cycle of 362 steps takes 21.72 s, excluding the 1 s start-up delay. Table V lists these analytical figures; they are derived from the configured constants and are not measured scan durations.

Table V Nominal Timing (Analytical)

Quantity	Value
Step period	60 ms (≤ 16.7 steps/s)
Positions per pass	181
Time per pass	10.86 s
Steps / time per cycle	362 / 21.72 s
Worst-case work in a step	$15 + 30 = 45$ ms
Echo width at 400 cm	23.3 ms
Serial time per record	≤ 0.78 ms

A failed measurement waits for the whole 30 ms timeout, so settling plus timeout occupy 45 ms of the 60 ms; a successful 1 m echo (about 5.8 ms) finishes much earlier and the remainder is padding. Since 45 ms is below 60 ms, the period stays close to 60 ms in both cases, although millis() resolution and the 1 ms polling delay may add up to about a millisecond per step. Real durations must be measured (Section X). The 15 ms settling time is ample for a 1° move at the servo's rated speed (about 1.7 ms), but the real settling behaviour under load and supply sag is unknown; if the servo is still moving when the ping fires, the printed angle is slightly off. The two endpoint repetitions add 0.12 s per cycle and produce duplicate records at 0° and 180° , which are harmless when the VI writes into an array indexed by angle.

A. Firmware Limitations

- pulseIn() blocks the processor until the echo ends or times out; nothing else can run meanwhile.
- Only one measurement is taken per angle, so an outlier or a missed echo shows directly on the display.
- Invalid readings are coded as 0, with no separate validity flag, timestamp or direction field.
- Distances are rounded to whole centimetres.
- The reported angle is the commanded servo angle, not a measured one; true angular accuracy and settling time are not established by the firmware.
- There is no temperature compensation, median filtering, optical homing, PID loop or serial command input.

The practical consequences are that scan timing is fixed, the display inherits the 1 cm quantisation and any systematic temperature error, and angular error cannot be detected in software. Optional improvements (not implemented here) include a temperature sensor for compensation, several pings with a median per angle, a validity flag in the record and an angle encoder or limit switch for homing.

V. LABVIEW VIRTUAL INSTRUMENT IMPLEMENTATION

Evidence status. The screenshots of the as-built front panel and block diagram had not been supplied when this draft was prepared. This section therefore describes a reference implementation that is consistent with the project source material (VISA serial driver, comma-

delimited parsing, a polar radar display, a distance-versus-angle graph, a Start/Stop control and an object-detection indicator) and with the firmware's serial format. No block is asserted to exist in the authors' VI until it has been checked against the screenshots; the authors must reconcile each item before submission (Appendix A).

1 Front Panel

The reference front panel has: a **VISA resource name** control (COM-port selector); a **Stop** button; a numeric **alarm threshold** (cm); an **Object Detected** LED; a **radar XY graph**; a **distance-versus-angle graph**; and optional numeric indicators for the current angle and distance. The radar graph shows detected points in the half plane, and the second graph shows the latest distance for each angle from 0° to 180°. The Start/Stop control and alarm indicator are mentioned in the project presentation; the COM-port selector is a standard addition. Whether each is present in the as-built VI is to be confirmed.

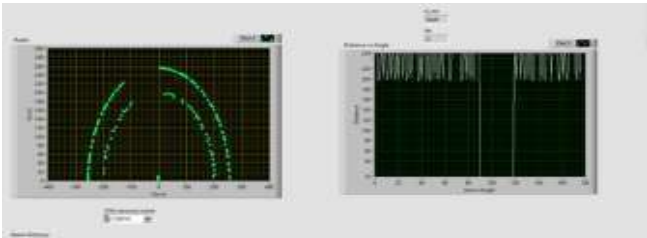


Fig. 8. Author-supplied screenshot of the as-built LabVIEW front panel (placeholder).

2. Block Diagram

Fig. 10 shows the reference data flow and Table VI explains each block: what it does, how it connects and how a beginner configures it. A beginner should build the VI in stages and test each stage with an indicator or probe before wiring the next.

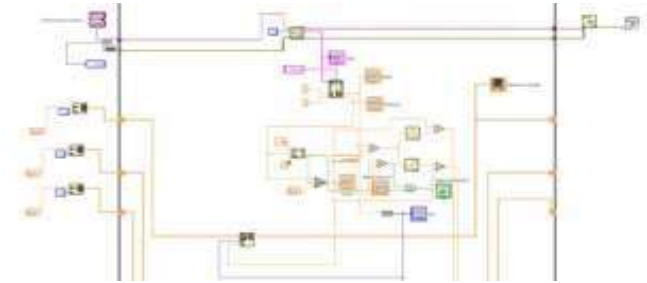


Fig. 9. Author-supplied screenshot of the as-built LabVIEW block diagram (placeholder).



Fig. 10. Reference VI data flow (original diagram; to be reconciled with the as-built block diagram).

TABLE VI REFERENCE BLOCK-DIAGRAM FUNCTION	
Block	Function, connections, configuration
VISA Configure Serial Port	Placed before the loop. Inputs: VISA resource name (from the control), baud rate 115200, data bits 8, parity none, stop bits 1, flow control none, termination character 10 (LF) with <i>Enable Termination Char</i> TRUE, timeout about 10000 ms. Outputs the resource and error wires to every later VISA node.
Wait + VISA Flush I/O Buffer	Opening the port resets an Uno, so wait about 2 s and discard the receive buffer to avoid a partial first record.
While Loop	Contains the acquisition. The Stop button is wired to the conditional terminal (stop if true); wire the error wire to it as well to stop on errors.
VISA Read	Byte count larger than the longest record (for example 32). Returns when LF arrives or the timeout expires; output is the read string.
Scan From String	Format string "%d,%d"; outputs angle and distance as integers. Leaves the trailing CR unused. (Alternative: Search/Split String at the comma and Decimal String To Number.)
Case Structure	Boolean selector: no parse/read error AND distance > 0 AND $0 \leq \text{angle} \leq 180$. True case stores the distance; False case stores NaN.
Shift register + Initialize Array	A 181-element DBL array initialised to NaN, carried in a shift register. Replace Array Subset writes the value at index = angle, so the array always holds the latest value per angle.
For Loop (auto-indexing)	Runs over the array. For index i : $x = d \cdot \cos(i \cdot \pi / 180)$, $y = d \cdot \sin(i \cdot \pi / 180)$. NaN propagates and is not plotted.
Bundle / Build Array	Bundle x and y into an XY cluster for the points plot; Build Array of clusters adds the sweep cursor, a two-point line from (0, 0) to (400 cos θ , 400 sin θ).
XY Graph, Waveform Graph	XY graph shows the radar view; the 1-D distance array wired to a Waveform Graph ($x_0 = 0$, $\Delta x = 1$) gives distance versus angle.
Alarm logic	Compare ($d < \text{threshold}$) AND valid, wired to the LED; optional hysteresis uses a Boolean shift register. See Section VIII-G.
VISA Close, Simple Error Handler	After the loop: release the port and report any error. Skipping VISA Close can leave the COM port locked.

6. Serial Communication

The LabVIEW settings must match the firmware (Table VII). Select the Uno's COM port in the resource control (it appears as COMx or ASRLx::INSTR), set 115200 baud, 8 data bits, no parity, 1 stop bit and no flow control, and enable the LF termination character. VISA Read then returns exactly one record per call. A VISA timeout error (typically code-1073807339) means no complete record arrived within the timeout: check the port, baud

rate and that no other program holds the port. After a re-plug the COM number may change.

Table VII Labview And Firmware Serial Settings

Setting	LabVIEW	Firmware
Port	Uno COM port	USB serial
Baud rate	115200	115200
Data bits	8	8
Parity	None	None
Stop bits	1	1
Flow control	None	None
Terminator	LF, enabled	CR LF
Timeout	≈10 s (adjustable)	—

7. Data Processing

Each record is parsed into two integers. A record is valid if parsing succeeded, the distance is greater than zero and the angle lies in 0–180. The distance is stored in the array element whose index equals the angle, which is how the software associates range with bearing. For an invalid record the reference design stores NaN so that stale data from an earlier sweep do not persist and the graphs show a gap. An optional three-point median across neighbouring angles could suppress isolated outliers; it is proposed and not part of the firmware.

8. Radar Display

The radar is an XY graph with x from –400 to +400 cm and y from 0 to 400 cm, autoscaling off and an equal scale on both axes (plot area twice as wide as high). Points are drawn without connecting lines. The coordinate transformation is (6), with degrees converted to radians. The sweep cursor shows the current direction. Range rings (additional plots of precomputed arcs) are optional. The data buffer is the fixed 181-element array, overwritten sweep after sweep; the graph is refreshed every loop iteration (about 16.7 Hz at the nominal rate).

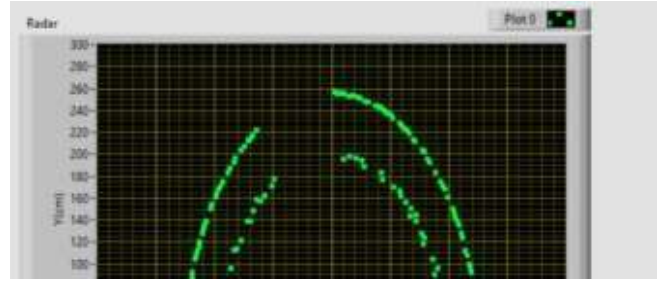


Fig. 11. Author-supplied screenshot of the running radar display with a known target (placeholder).

9. Distance-Versus-Angle Graph

The horizontal axis is the servo angle (0–180°) and the vertical axis the distance in cm (0–400). A dip marks a nearer object; a gap (NaN) marks no valid echo. The dip is wider than the object because of the beam width (Section VI). Each new record updates one point, so the curve is refreshed progressively along each sweep.

10. Proximity Alarm

The project presentation lists an “object detection alarm indicator”, and an earlier paper draft described a threshold alarm with 5 cm hysteresis. The firmware has no alarm: any alarm runs on the PC. Its implementation in the as-built VI is to be confirmed, and hysteresis must not be claimed unless verified. The reference logic is: set the LED when a valid d is below the threshold d_{on} ; with the optional hysteresis, clear it only when d exceeds $d_{on} + 5$ cm. Hysteresis prevents flicker when a target sits at the boundary.

VI. STEP-BY-STEP REPLICATION PROCEDURE

Stage 1: Gather and inspect the components. Do: Collect an Arduino Uno, USB data cable, HC-SR04, SG90 with horn, breadboard, jumper wires, a base, optionally a 470 μ F capacitor or external 5 V supply, and a PC with the Arduino IDE, LabVIEW and NI-VISA. Expect: All parts are undamaged and pins are straight. If it fails: Replace damaged parts; test the cable on another device.

Stage 2: Assemble the mechanical scanning unit. Do: Fix the servo to the base, then fit the horn at a known angle and fix the sensor to it with its axis over the servo axis (Section V). Expect: The sensor turns by hand-free command without binding or wire tension. If it fails: Re-seat the horn, add wire slack, tighten the base.

Stage 3: Complete and verify the wiring. Do: Wire as in Table II and Fig. 3, then run the checks of Section V-D. Expect: No shorts, correct polarity, all grounds common. If it fails: Re-check each connection against the table.

Stage 4: Upload the firmware. Do: Follow Section VII-A with Listing 1. Expect: The IDE reports a successful upload and the servo sweeps. If it fails: Check board, port, cable; close other serial programs.

Stage 5: Verify servo movement and distance independently. Do: Open the Serial Monitor at 115200 baud and hold a flat object about 20 cm from the sensor. Expect: Smooth motion, about 16–17 lines per second of “angle,distance”, and distances near 20 cm when the object is in the beam, 0 when nothing is in range. If it fails: See Table XV (servo, readings, serial).

Stage 6: Configure the LabVIEW environment. Do: Install LabVIEW with NI-VISA, close the Serial Monitor and confirm that the Uno’s port appears in the VISA resource list. Expect: The port is listed. If it fails: Install the USB-serial driver; re-plug; select the correct resource.

Stage 7: Build or open the virtual instrument. Do: Create the front panel and block diagram of Section VIII, or open the

authors’ VI. Expect: The VI runs with no broken arrow. If it fails: Fix type or dimension mismatches; read the error list.

Stage 8: Test data acquisition and parsing. Do: Temporarily show the raw string and the two parsed numbers in indicators. Expect: Angle cycles 0 to 180 and back; distances track a moving hand; invalid readings show 0. If it fails: Check baud rate, termination character, timeout and the format string.

Stage 9: Verify radar coordinates and graph updates. Do: Place a flat panel about 50 cm away on the right of the sensor, then on the left, then straight ahead. Expect: The point appears on the matching side and at the matching angle; the distance graph shows a dip at that angle. If it fails: Check degrees-to-radians conversion; if mirrored, correct the horn orientation or document an explicit angle mapping.

Stage 10: Run the complete integrated system. Do: Run several sweeps, then press Stop. Expect: Stable

display; the loop ends and the port closes. If it fails: Check the stop wiring and VISA Close; see Table XV.

Stage 11: Capture photographs and screenshots. Do: Take the placeholders listed in this report (components, assembly, front panel, block diagram, running radar, set-up). Expect: All placeholders are replaced by author-supplied images. If it fails: Retake sharper or better-lit images.

Stage 12: Perform calibration and validation. Do: Run the protocol of Section X and record the room temperature. Expect: Completed data tables. If it fails: Repeat suspicious runs; do not alter data to fit expectations.

Stage 13: Document the final demonstration. Do: Photograph both authors and the faculty guide with the running system, and record the final firmware and VI versions. Expect: Demonstration record. If it fails: Repeat the demonstration.

cm × 20 cm), an 8 cm diameter cylinder (cardboard tube) and a soft target (folded cloth). Use a steel tape for distances, a printed protractor sheet under the servo axis for angles, a thermometer for room temperature and a stopwatch. This is a proposed protocol; no results exist yet.

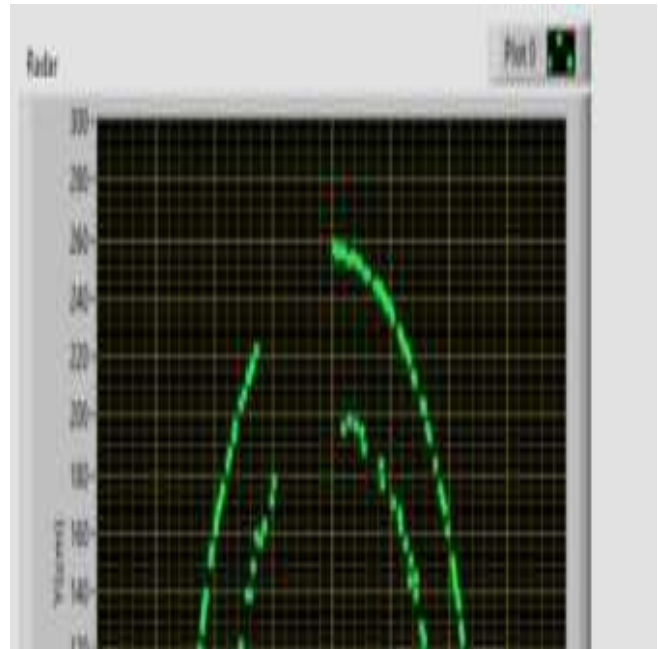
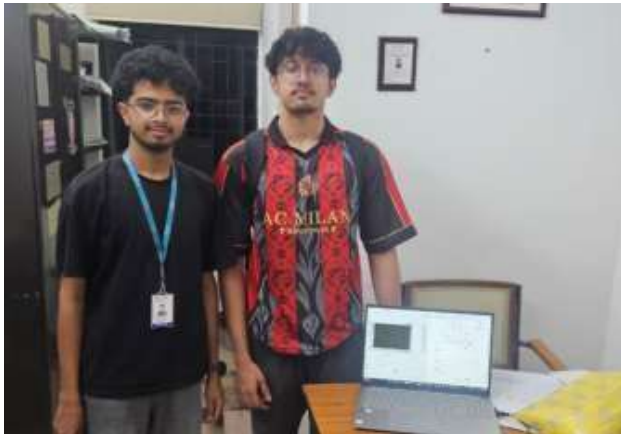


Fig. 13. Author-supplied photograph of the experimental set-up and test targets (placeholder).

1. Experiments

- Distance accuracy: flat panel on the 90° axis at 10, 20, 50, 100, 200 and 300 cm; ten readings at each distance.
- Angular positioning: cylinder at 100 cm placed at 30°, 60°, 90°, 120° and 150°; record the bearing as the centre of the contiguous run of minimum-distance readings and the width of that run.
- Repeatability: repeat one fixed target many times, within a sweep and across sweeps, and report the standard deviation.
- Target shape and orientation: panel tilted by 0°, 15°, 30°, 45° and 60°; cylinder; soft cloth.
- Multiple obstacles: two cylinders at 100 cm, narrowing the gap until they merge on the display; also near-and-far pairs.
- Scan duration: time ten passes with a stopwatch and compare with the nominal 10.86 s.
- Missing echoes and false detections: count zero readings per sweep, and record any ghost points, in empty and cluttered scenes.



2. Metrics

With d_m the measured distance and d_r the reference distance,

$$e_{abs} = |d_m - d_r| \quad (8)$$

$$e_{\%} = |d_m - d_r|/d_r \times 100 \quad (9)$$

Record the room temperature with every run, because the systematic error of (4) is otherwise confounded with other errors. The following tables are blank templates; no value in them has been measured.

Fig. 12. Author-supplied photograph of the authors presenting the project to the faculty guide (placeholder; to be added if available).

VII. EXPERIMENTAL SETUP AND VALIDATION

1. Setup

Place the scanner at the edge of a table so that the 0°–180° arc covers the room, with at least 2 m of clear space beyond the nearest target. Suggested targets are a flat cardboard panel (20

Table Viii Distance Accuracy (Blank)

d_r (cm)	Mean d_m (cm) e_{abs} (cm)	$e_{\%}$ (%)	T (°C)
10			
20			
50			
100			
200			
300			

TABLE IX ANGULAR POSITIONING (BLANK)

Ref. angle (°)	Displayed angle (°)	Error (°)	Run width (°)
30			
60			
90			
120			
150			

TABLE X REPEATABILITY (BLANK)

Distance (cm)	N	Mean (cm)	Std. dev. (cm)
---------------	---	-----------	----------------

TABLE XI
SCAN DURATION (BLANK; NOMINAL 10.86 S PER PASS)

Pass	Direction	Time (s)	Diff. (s)
1			
2			
3			

TABLE XII
TARGETS, ORIENTATION AND MISSED ECHOES (BLANK)

Test	Setup	Detected?	Notes
Panel tilt	0°/15°/30°/45°/60°		
Cylinder	100 cm		
Soft cloth	100 cm		
Two targets	gap =		
Empty scene	zero count/sweep		

VIII. RESULTS AND DISCUSSION

Verified system behaviour. The authors report that the assembled system works: the Arduino sweeps the servo, streams “angle,distance” records and LabVIEW displays them.No quantitative measurement dataset has been supplied, so this report states no measured accuracy, range, repeatability, angular error or scan duration, and presents no experimental graph. Table XIII lists analytical figures only; they are expectations derived from datasheet values and equations, not observations. Table XIV lists what is established and what remains to be measured. TABLE XIII

Table Xiii

Analytical Estimates (Not Measurements)

Quantity	Value	Basis
Beam footprint at 1 m / 2 m	26 cm / 53 cm	Eq. (5), $\beta = 15^\circ$
Lateral step per 1° at 1 m	1.75 cm	Geometry
Temperature error	$\approx 0.18\%$	Eq. (4)
Distance rounding	≈ 0.5 cm	Firmware
Time per pass / cycle	10.86 s / 21.72 s	60 ms $\times$ steps
Echo width, 2 cm / 400 cm	117 μ s / 23.3 ms	Eq. (1)

Table Xiv Evidence Status

Item	Status
Firmware, pins, serial format	Implemented (Listing 1)
Hardware assembled and operating	Reported by authors
LabVIEW radar and graph	Reported; screenshots pending
Proximity alarm / hysteresis	To be confirmed
Distance accuracy, repeatability	Not yet measured
Angular accuracy, beam spread	Not yet measured
Scan duration	Not yet measured
Target-type and missed-echo behaviour	Not yet measured

[INSERT EXPERIMENTAL GRAPH: MEASURED vs REFERENCE DISTANCE, ONLY AFTER REAL DATA ARE COLLECTED]

Fig. 14. Measured-Result Graph (Placeholder). Only Graphs Generated From Real Measurements May Be Used Here

Required before final submission: the data of Tables VIII–XII; screenshots of the front panel, block diagram and running display; the confirmed power wiring; and a statement of whether the alarm exists in the as-built VI. Results must then be discussed against the analytical expectations, stating where they agree or differ and the likely XII.Troubleshooting Guide

Table Xv Troubleshooting By Layer

Symptom	Likely cause	Check / fix
HARDWARE		
Servo does not move	No power; signal not on D9; wires swapped	Check orange to D9, red to +5 V, brown to GND; check a common ground; try a separate 5 V supply
Servo jitters, buzzes or Uno resets	Supply sag; servo pushing on an end stop	Add 470 μ F at the servo or an external 5 V supply with a common GND; stop at once if it buzzes at an end stop
All readings 0 or wrong	TRIG/ECHO swapped; sensor unpowered; soft or angled target; target out of 2–400 cm	Verify D10/D11 and VCC; use a flat target about 20 cm away, square to the sensor
Constant distance	Nearby fixed object or wire in the beam	Clear the beam and wires; note the 2 cm minimum range
90° is not straight ahead	Horn fitted ~~off-centre~~	Re-fit the horn with the servo at a known angle
FIRMWARE		
Garbage in Serial Monitor	Baud-rate mismatch	Set 115200
Upload fails	Wrong board or port; port busy	Select Arduino Uno and the correct port; close other programs
Noisy distance	One ping per angle; soft or tilted targets	Expected behaviour; change target; an optional median is proposed
Sweep time differs from 10.86 s	Nominal estimate, real timing differs	Measure it (Table XI); check the constants
COMMUNICATION		
No COM port	Charge-only cable; missing driver	Use a data cable; install the USB-serial driver
VISA timeout or port busy	Serial Monitor open; wrong resource; wrong baud	Close other programs; reselect the port; set 115200
First data missing or delayed	Uno resets when the port opens	Wait about 2 s, then flush the buffer

Symptom	Likely cause	Check / fix
LABVIEW		
Parse errors or zeros	Termination or format mismatch; partial first record	Format “%d,%d”; LF termination enabled; discard the first read
Blank or delayed graph	Graph not wired; axis ranges wrong; all values NaN	Use probes; set axes –400...400 and 0...400; check validity logic
Radar mirrored or upside down	Angle convention; degrees used as radians	Convert $\theta \rightarrow \pi/180$; correct the horn orientation or document a mapping
False or missing targets	Beam spread; specular reflection; multipath	Reposition the target; see Section XIII
Old points persist	Array not overwritten for invalid data	Write NaN for invalid records
Alarm LED flickers	No hysteresis	Add hysteresis (proposed)

Limitations and Future Improvements

Beam width and angular discrimination. The 15° beam limits lateral resolution to about 26 cm at 1 m (analytical) and smears the apparent bearing and width of small objects; the 1° step does not improve this.

Servo positioning. The reported angle is the commanded angle; backlash, load and wiring may cause an unknown deviation, and there is no encoder.

Reflections. Flat surfaces at oblique incidence may return no echo, soft materials absorb ultrasound, thin objects can be missed, and multipath and crosstalk are possible near corners [8]. **Target orientation.** The same object can give different ranges or none depending on its angle. **Temperature.** The fixed $\mu\text{s}/\text{cm}$ gives about 0.18 % range error per °C (analytical). **Scan latency.** One pass takes about 10.9 s nominally, so the display is a snapshot and moving objects may be missed.

Future work (proposed, not implemented): temperature measurement and speed-of-sound compensation; several pings with a median per angle; centroid-based bearing estimation using the centre of the minimum-distance run; calibration against the validation data; an occupancy-grid display [1]; an angle encoder or limit switch for homing; a narrower-beam transducer or a different ranging sensor, which would improve angular resolution more than any servo change; and a microcontroller with a hardware pulse-capture peripheral.

IX. CONCLUSION

This report documented a 180° ultrasonic spatial profiler built from an HC-SR04, an SG90 servo, an Arduino Uno and a LabVIEW front end. The firmware sweeps 0° to 180° and back in 1° steps, takes one `pulseIn()` measurement per angle, converts it to integer centimetres with 58.3 $\mu\text{s}/\text{cm}$ and sends “angle,distance” records at 115200 baud; LabVIEW parses them and shows a polar radar and a distance-versus-angle graph. The system is reported to work, and the report supplies the theory, the verified firmware, a wiring and replication guide, a validation protocol and a troubleshooting guide. Its educational value lies in exposing every layer of a measurement chain. The principal limits are the wide beam, the open-loop servo angle, specular reflection, the uncompensated temperature dependence and the slow scan. Because no measurements have been collected yet, no accuracy

or range is claimed; the analytical estimates must be tested with the protocol of Section X before any such claim is made.

School of Electrical Engineering; the earlier paper listed Control and Automation).

- Servo power source (Uno 5V pin or external), capacitor fitted or not, mounting method and base.
- As-built LabVIEW front-panel and block-diagram screenshots; confirm every block of Table VI, the COM selector, Start/Stop control, alarm and any hysteresis.
- All author-supplied photographs listed as placeholders.
- Reference details marked “to be added” (datasheet and Arduino documentation URLs, access dates, NI-VISA edition, Doebelin edition and year); re-verify all DOIs online.

1. Validation Tasks

- Fill Tables VIII–XII from real measurements and add a measured-result graph.
- Measure actual scan time and compare with 10.86 s.
- Check zero-angle direction and servo end-stop behaviour.
- Decide whether the official IEEE Word template is required; this draft imitates the IEEE conference format (two columns, Times, numbered references) and is not the official template.

2. Source-Material Discrepancies Resolved

- The presentation status slide lists firmware and VI as “yet to be done”; this report follows the authors’ statement that the project works.
- The earlier paper contains a bench-measurement table with no supplied data and states elsewhere that the bench campaign is future work; it was not reused.
- The earlier paper’s hysteresis alarm, three-point median and temperature correction are not in the firmware and are treated as proposed or to-be-confirmed.
- The earlier paper attributes 4 μs timing to `micros()`; the firmware uses `pulseIn()`, whose resolution was not characterised, and its output rounding to 1 cm dominates quantisation.

- The earlier paper gives a maximum record of 8 bytes; "180,400" plus CR LF is 9 bytes. Serial.println() sends CR LF, not LF alone.
- The presentation gives 58.2 $\mu\text{s}/\text{cm}$ and "9600 / 115200 bps";
- the firmware uses 58.3 and 115200.
- A "Start/Stop" control cannot command the Arduino, which has no serial input code.

REFERENCES

1. A. Elfes, "Sonar-based real-world mapping and navigation," IEEE J. Robot. Autom., vol. 3, no. 3, pp. 249–265, Jun. 1987, doi: 10.1109/JRA.1987.1087096.
2. H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in Proc. IEEE Int. Conf. Robot. Autom., St. Louis, MO, USA, 1985, pp. 116–121.
3. L. Kleeman and R. Kuc, "Sonar sensing," in Springer Handbook of Robotics, B. Siciliano and O. Khatib, Eds. Berlin, Germany: Springer, 2008, pp. 491–519.
4. O. Cramer, "The variation of the specific heat ratio and the speed of sound in air with temperature, pressure, humidity, and CO₂ concentration," J. Acoust. Soc. Am., vol. 93, no. 5, pp. 2510–2516, May 1993, doi: 10.1121/1.405827.
5. Elec Freaks, "Ultrasonic ranging module HC-SR04," product data sheet. [URL and access date to be added.]
6. J. Borenstein and Y. Koren, "The vector field histogram—Fast obstacle avoidance for mobile robots," IEEE Trans. Robot. Autom., vol. 7, no. 3, pp. 278–288, Jun. 1991, doi: 10.1109/70.88137.
7. S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics. Cambridge, MA, USA: MIT Press, 2005.
8. R. Kuc and M. W. Siegel, "Physically based simulation model for acoustic sensor robot navigation," IEEE Trans. Pattern Anal. Mach. Intell., vol. PAMI-9, no. 6, pp. 766–778, Nov. 1987, doi: 10.1109/TPAMI.1987.4767983.
9. J. Travis and J. Kring, LabVIEW for Everyone, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2006.
10. National Instruments, NI-VISA User Manual. Austin, TX, USA. [Edition, URL and access date to be added.]
11. TowerPro, "SG90 9 g micro servo," product data sheet. [URL and access date to be added.]
12. Microchip Technology, ATmega48A/PA/88A/PA/168A/PA/328/P Data Sheet, DS40002061, Chandler, AZ, USA.
13. Arduino, "Servo library reference," Arduino documentation. [URL and access date to be added.]
14. Arduino, "pulseIn()," Arduino language reference. [URL and access date to be added.]
15. E. O. Doebelin, Measurement Systems: Application and Design, 6th ed. New York, NY, USA: McGraw-Hill Education. [Co-author and year to be verified.]