AI-Enabled Arduino RADAR: YOLO-Based Object Detection

Maseeha Banu
Baladev DR, Animesh M, Manukrishna, Samyak
Department of AIML, Malnad College of Engineering, Hassan, India
Email: mb@mcehassan.ac.in

Abstract

An Arduino-based ultrasonic scanner can show the direction and approximate range of a nearby reflecting object, but it cannot identify the object from distance data alone. This paper describes a prototype that extends this type of scanner with webcam-based recognition using a YOLO model. An Arduino UNO rotates an HC-SR04 sensor on a servo motor through a nominal 0°–180° sweep, measures the echo duration, and sends angle–distance readings to a host computer over a serial connection. At the same time, a laptop webcam provides video frames to a Python application. OpenCV handles image capture, and the YOLO detector returns object labels, confidence scores, and bounding boxes. Processing presents the ultrasonic readings in a live radar-style view. A separate detection-records interface stores the object label, confidence, ultrasonic distance, servo angle, and threat level in CSV format. The alert logic uses configurable distance limits and detection conditions to assign Low, Medium, or High levels. Since no measured test dataset was provided, the paper does not claim numerical accuracy or response-time results. It explains the implementation, identifies the conditions needed to associate camera detections with ultrasonic returns, and sets out a procedure for evaluating the prototype. The device is an ultrasonic scanning and computer-vision system; it is not a Doppler radar or LiDAR system.

Index Terms

Arduino UNO, HC-SR04, ultrasonic ranging, servo scanning, YOLO, object detection, OpenCV, computer vision, Processing, serial communication, CSV logging.

I. INTRODUCTION

A servo-mounted ultrasonic sensor provides a simple way to inspect the space in front of an embedded device. As the servo changes the sensor's direction, the controller records the distance returned at each angular position. Plotting these readings on a radar-style display makes the position of nearby reflecting surfaces easier to understand. The limitation is that the sensor reports range, not object identity. A return at a particular angle could come from a person, a wall, furniture, or another surface, and the ultrasonic measurement alone cannot distinguish between them.

A webcam can provide a second kind of information: visual evidence about what is in the scene. Object-detection models estimate both the class and image location of visible objects. YOLO is a single-stage detection approach that predicts bounding boxes and class scores from an input image [1]. Later versions developed the model further, including changes to the training approach and predictions at multiple image scales [2], [3]. In this project, YOLO supplies the object label while the HC-SR04 remains responsible for distance measurement. The achievable detection rate depends on the selected model and the host computer.

The prototype brings these functions together using an Arduino UNO, HC-SR04 sensor, servo motor, laptop webcam, Python, OpenCV, a YOLO detector, Processing, and CSV storage. The Arduino records the sensor angle and measured range. The webcam and detector provide visual object labels and confidence scores. Processing displays the ultrasonic scan, while the records interface keeps a history of detection events. A configurable rule set assigns each valid observation to a Low, Medium, or High alert level.

The contribution of this work is the integration of range sensing, visual recognition, live visualization, alert classification, and event logging in one working prototype. It does not introduce a new object-detection model or a new ultrasonic ranging method. A central design consideration is that the camera and ultrasonic sensor produce separate observations. A camera label can be linked to a particular range return only after the two sensors have been calibrated and their observations aligned in time.

II. RELATED WORK AND LITERATURE SURVEY

A. YOLO-Based Object Detection: Redmon et al. introduced YOLO as a unified object detector that predicts bounding boxes and class probabilities from a complete image in a single network evaluation [1]. This single-stage formulation reduced the need for a multi-step region-proposal pipeline. YOLO9000 extended the approach to a larger set of object categories and explored joint detection and classification training [2]. YOLOv3 introduced predictions at multiple scales, improving the representation of objects with different spatial sizes [3]. Bochkovskiy et al. presented YOLOv4 and studied architectural and training choices intended to balance detection accuracy and inference speed [4]. These works establish the algorithmic background for real-time object recognition. They do not imply that every YOLO implementation has the same speed or accuracy; those properties depend on model size, input resolution, training data, runtime, and hardware.

B. Ultrasonic Ranging and Embedded Control: The HC-SR04 is a time-of- flight ranging module. It emits an ultrasonic burst after a trigger pulse and

provides an ECHO pulse whose duration represents the round-trip propagation time [5]. An Arduino-class microcontroller can generate the trigger signal, measure the echo pulse, and control a servo to obtain directional samples. This arrangement is widely used in educational obstacle-detection prototypes because the control and measurement tasks can be implemented with a small embedded board. The resulting system should be described accurately as an ultrasonic scanner rather than an electromagnetic radar.

C. Vision and Interactive Visualization: OpenCV provides tools for camera capture and image processing [6]. In the proposed system, OpenCV supports frame acquisition and image preparation, while the YOLO implementation performs object detection. Processing provides a programming environment for interactive visual output and is used here to draw the angular sweep and distance returns [7]. These software components allow the host computer to coordinate sensing, inference, and display.

D. Integration Gap: The reviewed foundations address object detection, ultrasonic ranging, embedded control, and visualization as separate technical capabilities. The present prototype combines them into one monitoring workflow and adds CSV-based event records and a configurable alert classification. It does not claim that a new peer-reviewed Arduino radar algorithm is introduced. Instead, its focus is the practical integration of established components and the handling of asynchronous sensor streams.

III. PROBLEM STATEMENT

A basic Arduino ultrasonic scanner can indicate the approximate distance of an obstacle at a servo angle, but its output is limited to range and direction. It cannot identify the semantic category of the visible object. A raw serial stream is also difficult to interpret continuously, and without event records it is inconvenient to review earlier detections or compare system behaviour. In addition, distance-only monitoring does not provide a configurable multi-level alert policy.

This project addresses these limitations by adding webcam-based object recognition, a live radar-style display, three-level alert classification, and a searchable record of detection events to an Arduino ultrasonic scanner. The HC-SR04 provides the distance value, while YOLO identifies objects visible to the webcam. The system records the label, confidence, distance, servo angle, and alert level. These fields describe the two sensing streams; they do not, by themselves, prove that a particular camera detection produced a particular ultrasonic return. Such a match requires

 ©IJSRED: All Rights are Reserved

camera-to-sensor calibration and timestamp alignment.

IV. OBJECTIVES OF THE PROPOSED SYSTEM

The objectives are: 1) to develop a servo-controlled ultrasonic scanning mechanism over a nominal 0°–180° range; 2) to measure obstacle distance using the HC-SR04; 3) to identify visible objects using a YOLO-based detector; 4) to integrate Python-based image processing with Arduino sensor readings; 5) to display scan measurements through Processing; 6) to implement configurable Low, Medium, and High threat levels; 7) to maintain detection records containing object label, confidence, distance, angle, and threat level; and 8) to store these records in CSV format for subsequent inspection and analysis.

V. PROPOSED SYSTEM ARCHITECTURE

The system is organized around four tasks: ultrasonic measurement, visual recognition, display and record management, and alert evaluation. For each scan position, the Arduino sets the servo angle, triggers the HC-SR04, measures the ECHO pulse, calculates the range, and sends the angle and distance to the computer. Separately, the webcam provides frames to Python. OpenCV acquires the frames, and the YOLO model returns class labels, confidence scores, and bounding boxes. The host application uses these outputs to update the Processing display and the detection-records interface. The alert logic evaluates valid range readings together with any configured detection rules. A buzzer is included only where it is connected and controlled by the actual implementation.

Fig. 1. System block diagram (placeholder): HC-SR04 and servo connected to Arduino UNO; Arduino serial output and webcam input flow to the host computer for Python/OpenCV/YOLO processing, Processing visualization, event records, and alert logic.

OpenCV and YOLO; host to Processing radar visualization and detection-records interface; alert logic to software notification and optional buzzer.]

The two data streams do not update at the same rate. The servo produces a range sample only after reaching each scan position, whereas the webcam can capture frames continuously and the detector processes them at its own rate. Timestamps are therefore important when comparing observations. Unless a calibrated association procedure is available, the display should show the camera result and the ultrasonic return as separate pieces of information.

VI. HARDWARE AND SOFTWARE REQUIREMENTS

TABLE I. HARDWARE COMPONENTS AND FUNCTIONS Arduino UNO — Controls servo positioning, ultrasonic triggering, echo timing, and serial transmission. HC-SR04 ultrasonic sensor — Measures distance from ultrasonic echo travel time. Servo motor — Rotates the sensor assembly through the configured angular sweep. Laptop webcam — Captures frames for visual object recognition. Buzzer (if implemented) — Provides an audible indication for selected alert states. Laptop/host computer — Runs Python, the YOLO model, Processing, and the record interface.

TABLE II. SOFTWARE COMPONENTS AND FUNCTIONS Arduino IDE — Firmware development and upload to Arduino UNO. Python — Coordinates camera inference, serial parsing, alert logic, and record creation. YOLO model — Detects object categories and returns confidence and bounding boxes. OpenCV — Captures and prepares webcam frames. Processing IDE — Renders the radar-style visualization. Visual Studio Code — Supports code development, integration, and debugging. Serial communication — Transfers angle and distance measurements from Arduino to the host. CSV storage — Maintains structured detection records.

For the paper to be reproducible, the final implementation details should identify the YOLO version and weight file, inference package, host-computer specifications, webcam resolution, serial baud rate, and alert thresholds. Values not confirmed from the project configuration are deliberately not supplied here.

VII. METHODOLOGY

A. Initialization: The Arduino configures the servo and HC-SR04 pins and initializes serial communication. The host application loads the YOLO weights, opens the webcam and serial port, and initializes the Processing connection and record file. Camera or serial connection failures should be reported rather than silently treated as valid input.

B. Angular Scanning: The servo moves the ultrasonic sensor through successive angular positions between the configured endpoints. At each position, the mechanism allows for settling before a measurement is taken. The scan can proceed in one direction and return in the opposite direction. Angular step size determines the number of samples and affects total scan time.

C. Ultrasonic Acquisition: The Arduino sends a short trigger pulse and measures the duration of the ECHO signal. A timeout is required so that a missing echo does not block the scan indefinitely. A timeout or invalid reading is marked unavailable; it must not be replaced with an invented distance.

D. Visual Recognition: Python captures a webcam frame using OpenCV and passes it to the YOLO inference interface. The detector returns bounding boxes, class labels, and confidence values. Detections below the configured confidence threshold are discarded or treated as low-confidence observations.

E. Integration and Display: The host parses the serial record and updates the radar display with angle and distance. Independently, it updates the camera panel and record interface with visual detections. If calibrated sensor association has been implemented, an association module may match image-space detections to sensor scan sectors using geometry and timestamps. If not, the label and range must remain separate fields without implying a verified object-to-range match.

F. Alert and Logging: The alert module evaluates the current valid measurement against configured thresholds and optional detection rules. It assigns Low, Medium, or High and updates the software interface. If a buzzer is implemented, it is activated according to the configured alert policy. The record manager stores timestamp, object label, confidence, ultrasonic distance, servo angle, and threat level in CSV format.

VIII. YOLO-BASED OBJECT DETECTION

YOLO is a family of single-stage object detectors. At inference time, an input image is processed by a neural network that predicts candidate object locations and class scores. Post-processing, commonly including confidence filtering and non-maximum suppression, removes weak or overlapping candidate boxes. The precise network structure and output format vary across YOLO versions; therefore, the implementation should identify the actual model and runtime rather than treating all versions as identical [1]–[4].

For a detection i, the output can be represented as

$D_i = (c_i, p_i, x_i, y_i, w_i, h_i)$, (1) where c_i is the predicted class label, p_i is the confidence score, (x_i, y_i) is the bounding-box centre, and w_i and h_i are the box width and height in image coordinates. A detection is accepted when $p_i \geq \tau$, where τ is the configured confidence threshold. Confidence is a model score and must not be interpreted directly as the probability that an object is dangerous.

OpenCV is used to initialize the webcam, retrieve frames, and perform image operations required by the detector interface. The YOLO model performs the semantic object detection. The detected label and confidence can be displayed and logged. A monocular webcam does not directly provide metric distance. Consequently, the system uses the HC-SR04 measurement for the distance field; bounding-box size must not be converted into distance unless a separate calibrated depth method is developed and validated.

IX. ULTRASONIC DISTANCE MEASUREMENT AND SERVO SCANNING

The HC-SR04 estimates range from the round-trip travel time of an ultrasonic pulse. A trigger pulse initiates an ultrasonic burst, and the module raises its ECHO output for a duration corresponding to the sound travel to the reflecting object and back [5]. Let t_e be the measured ECHO pulse duration and v_s be the speed of sound. The one-way distance d is

$$d = (v_s \times t_e) / 2. \quad (2)$$

The division by two accounts for the outgoing and returning paths. If v_s is expressed in centimetres per second and t_e in seconds, d is obtained in centimetres. The speed of sound depends on environmental conditions, particularly temperature. If temperature is not measured, the implementation should state the assumed nominal speed and recognize the resulting environmental error.

At each servo position, the controller records an angular coordinate θ and a distance d. The scan can be represented as

$S = \{(\theta_k, d_k, t_k) \mid k = 1, 2, \ldots, N\}$, (3) where θ_k is the servo angle, d_k is the measured range, t_k is the sample timestamp, and N is the

 ©IJSRED: All Rights are Reserved

number of samples in a sweep. The radar interface can map a valid measurement to a two-dimensional point using

$x_k = d_k \cos(\theta_k)$, $y_k = d_k \sin(\theta_k)$, (4) after converting θ_k to radians and selecting a consistent display coordinate convention. This is a visualization transform; it does not increase sensor accuracy.

The commanded servo angle is an estimate of the sensor's bearing. Mechanical backlash, mounting alignment, servo repeatability, and the finite acoustic beam width can cause deviations between the commanded angle and the effective direction of the measured echo. The scan should therefore be interpreted as a sequence of directional sectors rather than perfectly isolated points.

X. PYTHON AND ARDUINO INTEGRATION

The Arduino handles the operations that directly control the hardware: moving the servo, triggering the ultrasonic sensor, timing the echo, and transmitting the result. The host computer runs Python to read the serial data, acquire webcam frames, execute YOLO inference, evaluate alert conditions, update the interfaces, and write CSV records. This arrangement keeps neural-network inference on the computer rather than the Arduino UNO, whose processing and memory resources are not suited to running the selected desktop vision model.

A serial message should use a defined, unambiguous format. For example, an angle–distance record may contain an angle and a range separated by a delimiter and terminated by a line ending. The exact delimiter, units, baud rate, and invalid-reading convention must match the firmware and host parser. The Python application should validate the number and type of fields, reject malformed records, and handle serial disconnection and port contention. The Arduino serial monitor should not remain open when another application owns the same port.

The camera and ultrasonic subsystems run independently, so their outputs should be timestamped before any attempt is made to combine them. Two readings that occur close together in time may still refer to different objects. A reliable match also needs a calibrated relationship between the camera image and the sensor's angular field, as well as a defined time window and matching rule. If these are not available, the application should present the camera classification and ultrasonic range separately.

XI. REAL-TIME RADAR VISUALIZATION USING PROCESSING

Processing receives the angle and distance stream from the host and renders a radar-style display. A typical interface contains an angular sweep line, range arcs, obstacle markers, numeric distance, and current alert state. The servo angle determines the sweep direction, while the HC-SR04 distance determines the radial position of a valid return. Camera object labels and confidence values can be presented in a separate panel or attached to a radar point only when a calibrated association has been implemented.

The display must continue drawing while new serial records arrive. To achieve this, incoming data should be buffered and parsed without holding up the rendering loop. A complete, valid record updates the displayed angle and range. Invalid packets, missing echoes, old readings, and a lost connection should be made visible to the operator. Camera labels should remain visually distinct from ultrasonic range markers unless a calibrated association has been implemented.

XII. THREAT-LEVEL CLASSIFICATION

The threat module maps configured sensor and detection conditions into three categories: Low, Medium, and High. Let d denote the latest valid HC-SR04 distance. Let T_H and T_M denote the configured distance boundaries, with $T_H < T_M$. A generic distance policy is

$A(d) = \text{High}, 0 < d \le T_H \text{ Medium}, T_H < d \le T_M \text{ Low}, d > T_M.$ (5)

The values of T_H and T_M are project configuration parameters and must be taken from the actual firmware or host application. They are not supplied in this paper and should not be guessed. If the sensor reading is invalid or unavailable, the system should use an explicit unavailable state rather than classifying it as Low.

The project may also include configured detection conditions, such as raising an alert when a selected object class is detected above a confidence threshold. If such a rule is implemented, it should be stated precisely. YOLO confidence is not a physical danger probability, and an object class alone does not establish a threat. A buzzer, if implemented, can be activated for selected levels; software notifications can display the corresponding text or visual indicator. Hysteresis or a persistence interval may be used to prevent rapid level changes around a threshold, but these features should only be claimed if present in the implementation.

XIII. DETECTION RECORD MANAGEMENT AND CSV STORAGE

The detection-records interface maintains a structured history of observations. A record can contain: (i) timestamp, (ii) object label, (iii) YOLO confidence score, (iv) HC-SR04 distance, (v) servo angle, and (vi) threat level. These fields provide a compact description of the visual and ultrasonic information available at the time of the event. Where camera-to-sensor association has not been validated, the record should include an association status or use separate visual and range event records so that the data are not misinterpreted.

CSV is a plain-text tabular format that can be inspected using spreadsheet software or processed using Python libraries. Each detection event is appended as a row with a consistent header. Numeric fields should use consistent units and decimal conventions. Missing values should be represented consistently, for example as an empty field or a defined null marker. The timestamp format should be consistent and preferably include date and time. The application should handle file-creation errors, avoid overwriting earlier records unintentionally, and flush buffered records at suitable intervals.

The CSV history supports later analysis of object-class frequency, distance distributions, angle coverage, alert-level counts, and event timing. It does not by itself guarantee that a detection is correct; the validity of each record depends on the underlying detector, sensor measurement, and any association logic.

XIV. EXPERIMENTAL SETUP AND IMPLEMENTATION

The prototype uses an Arduino UNO connected to an HC-SR04 ultrasonic sensor and a servo motor. The sensor is mounted so that its sensing direction rotates with the servo. The laptop webcam supplies the visual stream to the host computer. Python executes the YOLO detector and coordinates serial data handling, record management, and alert evaluation. Processing renders the radar view. The buzzer is included in the evaluation only if it is physically connected and activated by the implemented alert logic.

For reproducibility, the final paper should report the actual Arduino pin mapping, servo model, scan step, servo settling delay, ultrasonic timeout, serial baud rate, message format, webcam resolution and frame rate, YOLO variant and weight identifier, confidence threshold, inference device, Processing version, CSV schema, and threat thresholds. These are implementation-specific values and are intentionally left as placeholders where they were not provided.

The test plan should include targets at known distances and multiple angular positions. A reference measuring instrument should be used to obtain comparison distances. Visual tests should include the object categories supported by the selected model, varied backgrounds, different illumination, partial occlusion, and multiple objects. Integration tests should include serial disconnection, missing echo, webcam failure, invalid packets, and CSV file access errors. All test conditions and repetitions should be recorded.

XV. RESULTS AND PERFORMANCE ANALYSIS

The project materials available for this paper did not include a measured test dataset. It would therefore be misleading to report numerical accuracy, detection range, frame rate, latency, or percentage improvement. Table III is provided as a results sheet to be completed after the prototype has been tested under documented conditions.

TABLE III. RESULTS TEMPLATE Object detection precision — TP/(TP+FP) on labelled test frames — [Insert result] Object detection recall — TP/(TP+FN) on labelled test frames — [Insert result] mAP — Evaluate using a stated IoU threshold and labelled dataset — [Insert result] Mean absolute distance error — Mean |d_measured − d_reference| — [Insert result] cm Servo angular deviation — Difference between commanded and measured bearing — [Insert result]° Radar update rate — Display updates per second — [Insert result] Hz Inference response time — Frame acquisition to displayed

 ©IJSRED: All Rights are Reserved

detection — [Insert result] ms Alert response time — Threshold condition to notification/buzzer — [Insert result] ms CSV logging success — Correctly stored events / attempted events — [Insert result] %

For N range observations, mean absolute error is

MAE = (1/N) Σ(i=1 to N) |d_i – r_i|, (6) where d_i is the measured distance and r_i is the reference distance. If the reference distance is non-zero, percentage error can be calculated as

E_i = (|d_i – r_i| / r_i) × 100%. (7) For object detection, precision and recall are

Precision = TP/(TP+FP), Recall = TP/(TP+FN), (8) where TP, FP, and FN denote true positives, false positives, and false negatives under a stated class-matching and intersection-over-union criterion. The dataset, class definitions, confidence threshold, and IoU criterion must be reported.

The discussion should consider the limitations of each subsystem. Ultrasonic error can arise from target orientation, surface characteristics, environmental conditions, and acoustic reflections. YOLO detections can be affected by lighting, occlusion, motion blur, and classes absent from the training data. The overall response time depends on servo scan duration, sensor timeout, webcam frame rate, inference time, serial transfer, and display update scheduling. CSV testing should verify both data completeness and correct field alignment. These measurements are required before claims about system performance can be made.

XVI. ADVANTAGES AND APPLICATIONS

The prototype brings together two kinds of information: the HC-SR04 reports the distance of a reflecting surface, and YOLO identifies visible object categories. The servo allows the sensor to inspect different directions instead of measuring only along a fixed axis. Processing provides a more readable view of the scan, while the CSV file preserves event details for later review. The alert rules provide a consistent way to label proximity conditions.

Potential applications include educational demonstrations, embedded-systems laboratories, introductory robotics experiments, controlled indoor obstacle monitoring, and prototype studies of asynchronous sensor integration. The system is intended as an academic demonstrator and should not be treated as a certified safety, security, or industrial sensing product.

XVII. LIMITATIONS

The HC-SR04 measures ultrasonic time of flight; it does not measure Doppler shift or object velocity and is not an electromagnetic radar sensor. Its readings depend on target geometry, surface properties, acoustic reflections, and environmental conditions. Servo scanning is sequential rather than simultaneous, and mechanical backlash and mounting alignment affect the angular estimate.

The webcam and ultrasonic sensor have different fields of view and coordinate systems. Without calibration, a camera label cannot be assigned confidently to a particular ultrasonic return. A conventional monocular webcam also does not directly provide metric depth. YOLO performance depends on the selected model, training classes, image resolution, lighting, occlusion, and host-computer resources. The Arduino UNO does not run the YOLO model; inference is performed by the host computer. The system also depends on a working USB/serial connection, webcam, and host application.

XVIII. FUTURE SCOPE

Future work can introduce a measured camera-to-sensor calibration and timestamp-based association algorithm so that a visual detection can be linked to an ultrasonic scan sector with quantified confidence. Repeated ultrasonic samples and filtering can be evaluated to reduce measurement fluctuations. Servo calibration can improve angular repeatability. The alert policy can be extended with documented hysteresis, persistence, and user-configurable class rules. A systematic labelled dataset and reference distance measurements should be collected to quantify detection quality, ranging error, latency, and integration reliability. Any additional hardware or algorithms should be treated as future work until they are implemented and evaluated.

XIX. CONCLUSION

The paper described a prototype that adds YOLO-based visual recognition and event management to an Arduino ultrasonic scanner. The Arduino UNO and servo-mounted HC-SR04 provide angle and distance readings, while a webcam, Python, and OpenCV supply image frames to the YOLO detector. Processing displays the ultrasonic scan, and the record interface stores labels, confidence scores, range, angle, and alert level in CSV format. The system therefore combines visual classification with directional range monitoring without treating the two measurements as interchangeable. Linking a camera detection to a particular ultrasonic return will require calibration and time alignment. Because no measured dataset was available, the paper sets out evaluation methods rather than claiming unverified performance. The prototype provides a practical basis for further work on sensor association and quantitative testing.

REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[2] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 7263–7271, doi: 10.1109/CVPR.2017.690.

[3] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” arXiv:1804.02767, 2018, doi: 10.48550/arXiv.1804.02767.

[4] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” arXiv:2004.10934, 2020, doi: 10.48550/arXiv.2004.10934.

[5] “HC-SR04 Ultrasonic Ranging Module,” SparkFun Electronics, technical datasheet. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf. [Accessed: Oct. 31, 2026].

[6] OpenCV, “OpenCV Documentation.” [Online]. Available: https://docs.opencv.org/. [Accessed: Oct. 31, 2026].

[7] Processing Foundation, “Processing Reference.” [Online]. Available: https://processing.org/reference/. [Accessed: Oct. 31, 2026].

[8] Arduino, “Arduino UNO Rev3.” [Online]. Available: https://docs.arduino.cc/hardware/uno-rev3/. [Accessed: Oct. 31, 2026].

 ©IJSRED: All Rights are Reserved