

An Autonomous Object-Following Robot Using Raspberry Pi 4B with Real-Time Image Processing

Sarthak Devikar¹, Onkar Bandichhode², Omkar Jadhav³, Dr. Sunil Mahajan⁴

^{1,2,3} Student, Department of Electronics, MAEER's MIT Arts, Commerce and Science College, Alandi (D), Pune, Maharashtra, India

⁴ Associate Professor, Department of Electronics, MAEER's MIT Arts, Commerce and Science College, Alandi (D), Pune, Maharashtra, India

Abstract—The Object Following Robot using Raspberry Pi 4B is an autonomous robotic system designed to detect, track, and follow a specific object with minimal human intervention. The project addresses limitations of conventional robotic systems, such as dependence on basic sensors, difficulty in distinguishing objects, sensitivity to lighting conditions, and limited real-time processing capabilities. The proposed system integrates a Raspberry Pi 4B, Raspberry Pi Camera Module, ultrasonic sensor (HC-SR04), L298N motor driver, DC motors, and a compact robot chassis. Python programming and image-processing techniques are used to analyze real-time camera input and identify the target object based on color, shape, or movement. The Raspberry Pi acts as the central processing unit and generates appropriate control signals for the motor driver, enabling the robot to move forward, backward, left, or right according to the target's position. The ultrasonic sensor continuously measures distance and supports obstacle detection and collision avoidance. The developed prototype demonstrates autonomous object tracking, real-time movement control, compact construction, low-cost implementation, and practical integration of hardware and software. The system is particularly suitable for indoor environments, educational demonstrations, small-scale automation, and robotics research. Although the robot has limitations related to lighting, object recognition accuracy, processing capacity, battery life, and response delay, it provides a strong foundation for future enhancements involving artificial intelligence, machine learning, advanced sensors, wireless connectivity, and intelligent navigation.

Index Terms—Autonomous robot, embedded systems, image processing, object tracking, obstacle avoidance, OpenCV, Raspberry Pi 4B, ultrasonic sensor

I. INTRODUCTION

A. General Background

Robotics and automation have moved from industrial settings into homes, hospitals, warehouses, and classrooms. Among autonomous systems, object-following robots form an important class: they detect a target, estimate its position relative to themselves, and move to keep it in view at a safe distance without human control. Such robots combine computer vision, embedded systems, sensing, and control engineering, and they find applications in automated delivery, warehouse material handling, personal assistance, hospital logistics, and surveillance.

An object-following robot works on a continuous sense-process-act cycle. Sensors and a camera capture data about the environment; a processor interprets that data to locate the target; and actuators, usually motors, move the robot so that the target remains at a desired position and distance. With advances in embedded hardware, open-source image-processing libraries, and machine learning, it is now possible to build low-cost robots that would earlier have required specialized industrial hardware.

B. Problem Statement

Many existing low-cost followers rely on infrared (IR) or ultrasonic sensors alone, which perform poorly in cluttered scenes and cannot identify a specific target. Vision-based followers distinguish targets better, but they are sensitive to illumination and demand more computation than 8-bit microcontrollers such as the Arduino can provide. Industrial tracking platforms are bulky and expensive

for small-scale indoor use. There is therefore a need for a compact, affordable, and easily reproducible platform that combines vision-based target detection with distance sensing.

C. Proposed Solution and Contributions

This paper presents the design, implementation, and evaluation approach of an object-following robot built around the Raspberry Pi 4B. The Pi processes video from its camera module using Python and OpenCV to locate a target by color, shape, or motion, and drives two DC motors through an L298N dual H-bridge driver. An HC-SR04 ultrasonic sensor provides distance measurement for maintaining a safe following distance and for collision avoidance. The main contributions are:

- An integrated vision-plus-ranging architecture on a single low-cost platform.
- A simple and transparent control logic that maps target position and distance to wheel motion.
- A documented setup and implementation procedure suitable for students and researchers.
- An analysis of limitations and a roadmap for improvement.

D. Organization of the Paper

Section II reviews related work. Section III identifies the research gap. Section IV states the objectives, and Section V gives the system analysis and requirements. Section VI presents the theoretical background. Section VII describes the proposed methodology and system design with diagrams. Section VIII covers hardware and software implementation. Section IX explains testing and calibration, and Section X discusses results. Sections XI to XV cover feasibility, advantages and limitations, applications, safety considerations, conclusion, and future scope.

II. LITERATURE REVIEW

Object-following robots have been implemented with several sensing strategies. They can be grouped into sensor-based, vision-based, and hybrid approaches.

A. Sensor-Based Following

Sensor-based designs use IR reflectance or ultrasonic time-of-flight to keep a target within range. Typical implementations place two or three sensors at the

front of the robot and steer toward the side that reports the nearest object. They are inexpensive and easy to program on microcontrollers, but they provide only coarse range or presence information. They cannot tell the intended target from any other object, and they are easily disturbed by nearby clutter, sunlight (for IR), and soft or angled surfaces that absorb or deflect echoes (for ultrasonic).

B. Vision-Based Following

Vision-based designs use a camera and classical image-processing operations such as color-space conversion, thresholding, morphological filtering, and contour analysis, most commonly through the OpenCV library [1]. Color-based tracking in the HSV space is popular because hue is relatively insensitive to brightness changes compared with RGB, although it still degrades under strong shadows or colored ambient light. Shape-based methods use contour approximation or the Hough transform to find circles or polygons, and motion-based methods use frame differencing or background subtraction.

C. Single-Board Computers in Robotics

Single-board computers such as the Raspberry Pi [2]–[5] have made real-time vision practical on small mobile robots. They offer a full Linux environment, a camera serial interface, GPIO, wireless connectivity, and enough CPU power for moderate-resolution frame processing. Compared with Arduino-class controllers, they can run OpenCV directly and support richer software such as web dashboards and remote access.

D. Deep-Learning and Hybrid Approaches

More recent work uses deep-learning detectors such as YOLO [6] and MobileNet-based models for robust recognition of people and objects. These generally require hardware acceleration, for example a Coral TPU or a Jetson module, to run in real time on embedded platforms. Hybrid designs combine a camera for bearing estimation with a ranging sensor for distance, which reduces the dependence on image-only depth cues. The present work follows this hybrid direction using inexpensive components.

III. RESEARCH GAP

Table I summarizes the gaps found in existing systems and the approach adopted in this work to address each of them.

TABLE I RESEARCH GAP ANALYSIS

Focus area	Gap in existing systems	Limitation	Proposed approach
Sensing	Many robots rely only on IR or ultrasonic sensors	Fail in cluttered scenes; cannot identify the target	Camera-based detection combined with HC-SR04 ranging
Lighting	Color-based vision degrades in poor or uneven light	Unstable or false tracking	HSV thresholding with calibrated ranges
Object differentiation	Difficulty distinguishing multiple moving objects	Wrong target may be tracked	Largest-contour selection within a calibrated color range
Processing	Arduino-class controllers have limited compute	Advanced image processing is not feasible in real time	Raspberry Pi 4B quad-core Cortex-A72 running OpenCV
Cost and size	Industrial trackers are bulky and expensive	Impractical for small indoor use	Compact chassis; prototype cost about INR 6,000–9,000
Responsiveness	Slow processing or delayed sensor response	Fast-moving targets are lost	Frame-by-frame position feedback driving motor commands

IV. OBJECTIVES AND SCOPE

A. Objectives

- To design a vision-based robotic system that identifies and follows an object using a camera connected to the Raspberry Pi.
- To implement Python and OpenCV image-processing algorithms for real-time detection and tracking based on color, shape, or movement.
- To integrate the Raspberry Pi 4B, camera module, HC-SR04 ultrasonic sensor, L298N motor driver, and DC motors into one functional platform.
- To move the robot automatically toward the target while maintaining a safe, predefined distance.
- To use the ultrasonic sensor for obstacle detection and collision avoidance.
- To develop Python control logic for forward, backward, left, and right motion based on target position.
- To build a compact, portable, battery-powered prototype.
- To demonstrate that low-cost embedded hardware can perform robotic functions usually found in more expensive systems.
- To provide a foundation for human-following, surveillance, and machine-learning based tracking robots.

B. Scope

The system is intended for indoor use on smooth surfaces, with a single colored target in the camera's field of view. Applications include automation and material handling, home assistance, hospital logistics, security and surveillance, and education. Outdoor operation, multi-target tracking, and deep-learning recognition are outside the present scope and are listed as future work.

V. SYSTEM ANALYSIS AND REQUIREMENTS

A. Existing System

Existing followers suffer from sensor dependency, sensitivity to lighting, poor object differentiation, limited processing on microcontrollers, size and cost constraints, and slow response to fast-moving targets. These limitations motivate the proposed design.

B. Fact-Finding Techniques

- Observation of how robots follow people or objects, including movement, sensing, and behavior.
- Internet research on Raspberry Pi, OpenCV, ultrasonic sensing, and motor drivers.
- Document analysis of datasheets (HC-SR04, L298N) and Raspberry Pi documentation.
- Experimentation, such as testing distance measurement and motor control separately before integration.
- Discussion with the project guide and subject experts.

C. Functional Requirements

- The system shall capture live video from the camera.
- The system shall detect a target of a configured color and compute its position.
- The system shall measure the distance to the target or obstacle.
- The system shall drive the motors forward, backward, left, right, or stop based on the control rules.
- The system shall stop when the target is lost or an obstacle is closer than the safe distance.

D. Non-Functional Requirements

- Real-time response with low perceptible delay.
- Portability: battery powered and compact.
- Low cost and use of readily available parts.
- Modularity so that components can be replaced or upgraded.
- Safety: the robot must stop on loss of target or on a close obstacle.

E. Software Requirements

The software environment used for development and operation is listed in Table II.

TABLE II SOFTWARE SPECIFICATION

Item	Specification
Operating system	Raspberry Pi OS (Debian-based Linux)
Language	Python 3
Libraries	OpenCV, PiCamera/Picamera2, RPi.GPIO, NumPy, time, os
Tools	Raspberry Pi Imager, Python IDLE, terminal, SSH/VNC or Raspberry Pi Connect
Host OS for setup	Windows / Linux / macOS

F. Hardware Requirements

The hardware components, their specifications, and their functions are given in Table III.

TABLE III HARDWARE SPECIFICATION

Component	Specification	Function
Raspberry Pi 4B	Quad-core Cortex-A72 (64-bit), 1.5 GHz, 4 GB RAM, 40-pin GPIO	Central processing unit
Pi Camera Module	8 MP/12 MP, CSI interface	Captures live video
Ultrasonic sensor	HC-SR04	Distance and obstacle measurement
Motor driver	L298N dual H-bridge, 5–35 V input	Direction and PWM speed control
DC geared motors	BO geared motors with wheels (2)	Propulsion and turning
Chassis and caster	2-wheel acrylic base plus caster	Mechanical support and balance
Power source	Rechargeable Li-ion pack / power bank; 5 V, 3 A for the Pi	Portable power
Storage	16/32 GB microSD	OS, code, and OpenCV
Wiring	Jumper wires, breadboard	Interconnection

VI. THEORETICAL BACKGROUND

A. Raspberry Pi 4B

The Raspberry Pi 4 Model B [7] is a single-board computer with a 1.5 GHz quad-core ARM Cortex-A72 CPU, 2/4/8 GB LPDDR4 RAM, dual micro-HDMI outputs, two USB 3.0 and two USB 2.0 ports, Gigabit Ethernet, dual-band Wi-Fi, Bluetooth 5.0, a CSI camera connector, and a 40-pin GPIO header. It is powered through USB-C at 5 V and 3 A. Its GPIO pins use 3.3 V logic and can supply only a few milliamperes each, so external loads such as motors must be switched through a driver.

B. HSV Color Space and Thresholding

A camera frame is delivered in BGR format. In the HSV representation, hue (H) encodes the color type, saturation (S) encodes color purity, and value (V) encodes brightness. Because the color identity is isolated in a single channel, a target can be selected by keeping pixels whose H , S , and V lie within lower and upper bounds. In OpenCV, H ranges from 0 to 179. The resulting binary mask marks target pixels as white.

C. Morphological Operations and Contours

Erosion removes small isolated white regions (noise), and dilation restores the size of the remaining regions and fills small holes. Contour extraction then finds the boundaries of connected white regions. The largest contour by area is taken to be the target. Its centroid is computed from image moments as

$$c_x = M_{10}/M_{00}, \quad c_y = M_{01}/M_{00} \quad (1)$$

where M_{ij} are the image moments of the contour, and its area or enclosing-circle radius serves as a rough proximity cue.

D. H-Bridge Motor Driver and PWM

An H-bridge uses four switches arranged around a motor so that current can be sent through it in either direction. The L298N [8] contains two such bridges, allowing two DC motors to be controlled independently. For each motor, two input pins select forward, backward, or stop, and an enable pin accepts a pulse-width-modulation (PWM) signal. PWM varies the duty cycle of a fixed-frequency square wave, so that the average voltage across the motor, and thus its speed, changes with the duty cycle.

E. Differential Drive

The robot uses two independently driven fixed wheels and a passive caster. When both wheels turn forward at equal speed, the robot moves straight. When the left wheel is slower than the right wheel, the robot curves left, and vice versa. Turning in place is achieved by driving the wheels in opposite directions.

F. Ultrasonic Ranging

The HC-SR04 [9] emits a burst of 40 kHz ultrasonic waves after receiving a 10 μ s trigger pulse. It then raises its echo pin for a time equal to the round-trip time t of the sound (Fig. 1). The distance is

$$d = (t \times v)/2 \quad (2)$$

where v is the speed of sound, about 343 m/s at 20 °C. The module is specified for roughly 2 cm to 400 cm range. Its echo pin outputs 5 V, so a voltage divider is needed to protect the Pi's 3.3 V inputs.

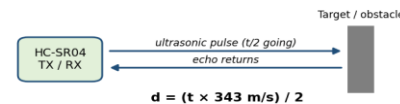


Fig. 1. Principle of ultrasonic distance measurement

VII. PROPOSED METHODOLOGY AND SYSTEM DESIGN

A. System Overview

The system follows a sense–process–act architecture. The camera module and the ultrasonic sensor form the sensing layer. The Raspberry Pi 4B forms the processing layer, where video frames are analyzed and control decisions are made. The L298N motor driver, the two DC motors, and the wheels form the actuation layer. A rechargeable battery supplies power. Fig. 2 shows the block diagram.

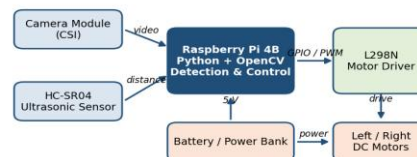


Fig. 2. Block diagram of the object-following robot

B. Software Architecture

The software is organized in layers, as shown in Fig. 3. The application script contains the control logic. It calls OpenCV for vision, the camera library for capture, and RPi.GPIO for hardware pins, all running on Raspberry Pi OS.

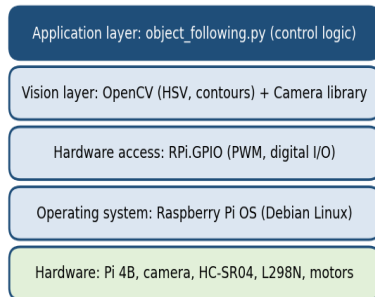


Fig. 3. Software and hardware layers

C. Image Processing Pipeline

Each frame passes through the pipeline shown in Fig. 4. The frame is resized to a moderate resolution (for example 640×480 or smaller) to keep the frame rate acceptable, blurred to reduce noise, converted to HSV, thresholded, cleaned with morphological operations, and reduced to the centroid of the largest contour. All of these steps are available as standard OpenCV functions [10].



Fig. 4. Image-processing pipeline for target detection

D. Control Strategy

The horizontal position of the target decides steering, and the ultrasonic distance decides whether to move forward or stop. The frame is divided into left, center, and right zones (Fig. 5). Table IV lists the decision rules.

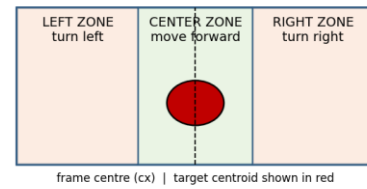


Fig. 5. Zone-based steering decision

TABLE IV DECISION RULES OF THE CONTROLLER

Condition	Robot action	Motor state
Target in center zone and distance > safe distance	Move forward	Both motors forward
Target in left zone	Turn left	Left slower/reverse, right forward
Target in right zone	Turn right	Right slower/reverse, left forward
Distance < safe distance (target too close or obstacle)	Stop or move backward	Both motors stopped/reversed
Target not found	Stop and scan again	Both motors stopped

To avoid jitter, a dead-band is used around the frame center, so small errors do not cause turning. Speed may be scaled with the horizontal error using PWM, so that large errors give sharper turns and small errors give gentle corrections.

E. Flow of Operation

On start-up the system initializes the camera, sensor, and GPIO pins. It then loops: capture a frame and detect the object; if the object is found, compute the error and read the distance, then drive the motors; if it is not found, stop and scan again (Fig. 6). The loop ends when the user stops the program, at which point the motors are stopped and GPIO is cleaned up.

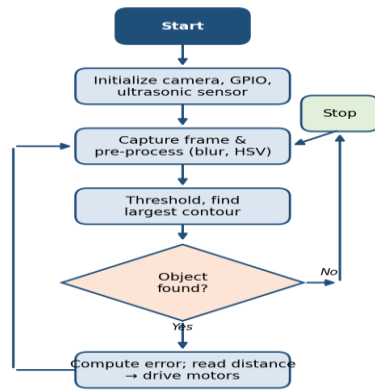


Fig. 6. Flowchart of the object-following algorithm

F. UML Diagrams

Fig. 7 shows the use case diagram, in which the user starts and stops the robot and the system detects, follows, and avoids obstacles. Fig. 8 gives the class diagram of the main software entities. Fig. 9 shows the sequence of interactions during one control cycle.

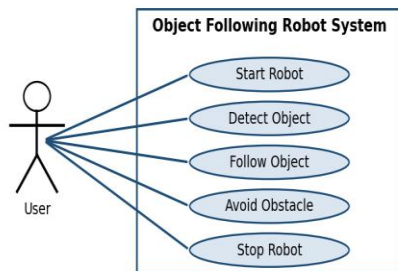


Fig. 7. Use case diagram

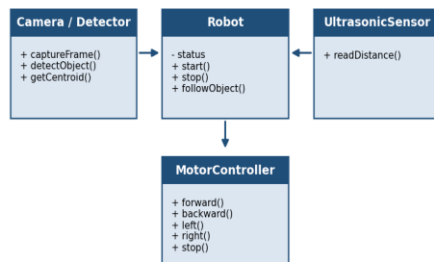


Fig. 8. Class diagram

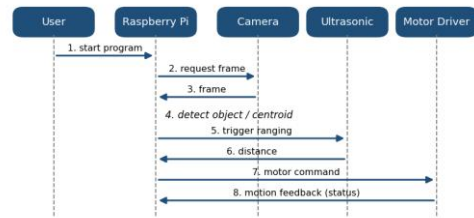


Fig. 9. Sequence diagram of one control cycle

G. Data Dictionary

The main variables used in the control program are described in Table V.

TABLE V DATA DICTIONARY

Variable	Type	Description
frame	ndarray	Current BGR image from the camera
hsv_lower / hsv_upper	tuple	Lower and upper HSV thresholds of the target color
cx, cy	int	Centroid coordinates of the largest contour
area	float	Contour area used as proximity cue
error_x	int	cx minus frame center; sign gives turn direction
distance_cm	float	Distance measured by HC-SR04
SAFE_DIST	float	Minimum allowed distance before the robot stops
duty_left / duty_right	int (0–100)	PWM duty cycles for left and right motors

H. Proportional Steering Equations

To obtain smooth turning, the horizontal error is converted into a differential wheel speed. Let W be the frame width, c_x the target centroid, and $e = c_x - W/2$ the horizontal error in pixels. The normalized error, which lies between -1 and $+1$, is

$$e_n = e/(W/2) \quad (3)$$

With a base duty cycle D_b and a proportional gain K_p , the left and right duty cycles are

$$D_{\text{left}} = D_b + K_p \times e_n \times D_b \quad (4)$$

$$D_{\text{right}} = D_b - K_p \times e_n \times D_b \quad (5)$$

and each is clipped to the range 0–100 %. When the target is to the right ($e_n > 0$), the left wheel speeds up and the right wheel slows down, turning the robot right. When $|e|$ is smaller than the dead-band, e_n is set to zero so that the robot drives straight. The forward speed itself may also be reduced as the measured distance approaches `SAFE_DIST`, so that the robot decelerates smoothly instead of stopping abruptly.

I. Suggested Operating Parameters

Table VI lists typical starting values for the main tuning parameters.

TABLE VI SUGGESTED STARTING PARAMETERS (TO BE TUNED EXPERIMENTALLY)

Parameter	Typical starting value	Note
Frame resolution	320×240 to 640×480	Lower resolution gives higher frame rate
Gaussian blur kernel	11×11	Reduces sensor noise
Minimum contour area	300–500 px	Ignores small false detections
Dead-band	≈ 10 % of frame width	Prevents oscillation
SAFE_DIST	20–30 cm	Depends on braking distance
Base duty cycle D_b	40–60 %	Tune for smooth motion
Gain K_p	0.3–0.8	Higher values give sharper turns

VIII. IMPLEMENTATION

A. Raspberry Pi Setup

- Download Raspberry Pi Imager from the official website and install it on a computer.
- Insert the microSD card, choose Raspberry Pi OS, select the card as storage, and write the image.
- Insert the card into the Pi and connect keyboard, mouse, monitor (micro-HDMI), and the 5 V/3 A USB-C supply.
- On first boot, set country, language, time zone, keyboard, user account, and Wi-Fi, then update the system.

- Enable the camera interface, install OpenCV and required Python packages, and enable remote access (SSH/VNC or Raspberry Pi Connect).
- After development, remove the monitor and peripherals so the robot runs standalone.

B. Hardware Assembly and Wiring

The two geared motors and wheels are mounted on the chassis, with a caster wheel at the rear for balance. The L298N is wired to the motors, and its input pins are connected to four Pi GPIO pins (two per motor), with the enable pins driven by PWM-capable GPIO for speed control. The camera is attached through the CSI ribbon cable, and the HC-SR04 trigger and echo pins go to two further GPIO pins. All grounds (battery, driver, Pi) must be joined to a common ground. Fig. 10 shows the connection scheme.

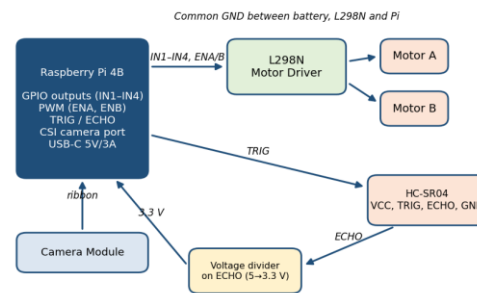


Fig. 10. Connection diagram of the robot

C. Power Distribution

The Raspberry Pi needs a stable 5 V supply at up to 3 A, whereas the L298N and motors use the battery voltage directly. A regulator (or a separate power bank for the Pi) is therefore used, as shown in Fig. 11. Powering the Pi and the motors from separate sources reduces brown-outs caused by motor current surges.



Fig. 11. Power distribution

D. Motor Control Test

Before integrating vision, the motor-control program (*motor_control.py*) was run from the terminal to verify the driver and wiring. The console output confirmed that the robot executed the commands Moving Forward, Left Forward, Right Forward, Moving Backward, Left Backward, Right Backward, and Stopping in sequence, showing that each motor responded correctly in both directions. The full tracking program is run with *python3 object_following.py*.

E. Reference Code Structure

The listing below shows the structure of the tracking program. Pin numbers and HSV values are configuration constants that must be set to match the actual wiring and target.

```
import cv2, time, numpy as np
import RPi.GPIO as GPIO
# --- configuration (edit for your setup) ---
IN1, IN2, IN3, IN4, ENA, ENB = ...
TRIG, ECHO = ... # BCM pins of HC-SR04
HSV_LOWER = np.array([h1, s1, v1])
HSV_UPPER = np.array([h2, s2, v2])
SAFE_DIST = 25 # cm
DEADBAND = 40 # pixels around frame centre

# IN1-IN4, ENA, ENB: BCM pins of the L298N

def read_distance():
    # send 10 us pulse on TRIG, time the ECHO pulse
    # return (pulse_width * 34300) / 2 # cm

def detect(frame):
    blur = cv2.GaussianBlur(frame, (11, 11), 0)
    hsv = cv2.cvtColor(blur,
cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, HSV_LOWER,
HSV_UPPER)
    mask = cv2.dilate(cv2.erode(mask, None, 2),
None, 2)
    cnts, _ = cv2.findContours(mask,
cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    if not cnts: return None
    m = cv2.moments(max(cnts,
key=cv2.contourArea))
    return int(m['m10']/m['m00']) # cx
```

while running:

```
cx = detect(capture())
if cx is None or read_distance() < SAFE_DIST:
    stop()
elif cx < W/2 - DEADBAND: turn_left()
elif cx > W/2 + DEADBAND: turn_right()
else: forward()
GPIO.cleanup()
```

F. Mechanical and Camera Design Considerations

The chassis should distribute weight evenly, with the battery and Raspberry Pi mounted low and near the drive axle so that the robot does not tip during sudden stops. The caster wheel is placed at the rear so that the drive wheels carry most of the load and the robot can pivot smoothly. The camera is mounted at the front, tilted slightly downward, at a height that keeps the target in view at the intended following distance. A rigid mount is important, since vibration blurs frames and disturbs contour detection. The ultrasonic sensor is mounted next to the camera and pointed along the same axis, so that the bearing given by the camera and the distance given by the sensor refer to the same object.

Cable management also affects reliability. The CSI ribbon cable must not be bent sharply or pulled by motor vibration, jumper connections should be secured or soldered for the final prototype, and motor wires should be kept away from the sensor and camera cables to reduce electrical noise.

G. Common Problems and Remedies

Table VII lists the problems most likely to be met during commissioning, with their usual causes and remedies.

TABLE VII TROUBLESHOOTING GUIDE

Symptom	Likely cause	Remedy
Pi reboots when motors start	Voltage drop from motor current surge	Use a separate supply for the Pi; add a capacitor across the motor supply
Robot drifts on a straight path	Motor speed mismatch	Trim PWM duty cycles per wheel
Robot oscillates left and right	Dead-band too small or turning too aggressive	Widen dead-band; scale PWM with error

Symptom	Likely cause	Remedy
Target not detected	HSV range wrong or poor lighting	Re-sample HSV values; improve lighting
Wrong object followed	Background has similar color	Choose a distinct target color; limit minimum contour area
Erratic distance readings	Soft or angled surface; noise on ECHO line	Average several readings; check voltage divider and grounding
Low frame rate	Resolution too high	Reduce resolution or skip frames

IX. TESTING AND CALIBRATION

A. Component-Level Testing

- Camera: verify live preview and frame capture.
- Ultrasonic sensor: compare readings against a tape measure at known distances.
- Motor driver: verify direction and speed control with the motor test program.
- Power: check that the Pi does not reset when the motors start.
-

B. Calibration

- Choose a target with a saturated, distinctive color that differs from the background.
- Sample the target's HSV values under the intended lighting and set lower and upper bounds with some margin.
- Adjust the dead-band and SAFE_DIST so that the robot neither oscillates nor stops too late.
- Tune PWM duty cycles so that both wheels give equal speed on a straight path; compensate for motor mismatch if the robot drifts.
- Re-check thresholds whenever lighting changes.

C. System-Level Test Plan

The system-level tests and their expected outcomes are summarized in Table VIII.

TABLE VIII SYSTEM-LEVEL TEST PLAN

Test	Procedure	Expected outcome
Static target	Place target ahead in center	Robot moves forward and stops at SAFE_DIST
Left/right offset	Place target to left or right	Robot turns toward target
Moving target	Walk slowly with target	Robot follows and keeps distance
Target loss	Remove target from view	Robot stops and scans
Obstacle	Place object in path	Robot stops before collision
Lighting change	Repeat under dim light	Note degradation in detection

X. RESULTS AND DISCUSSION

The prototype demonstrated the intended behavior of detecting a colored target, turning toward it, advancing while the target was ahead, and stopping when the target was lost or too close. The motor-control test in Section VIII-D validated forward, backward, and turning motion.

A. Cost Analysis

The approximate cost of each component is given in Table IX.

TABLE IX ESTIMATED COST OF THE PROTOTYPE

Component	Approximate cost (INR)
Raspberry Pi 4B	4,000 – 6,000
Pi Camera Module	800 – 1,200
Motor driver	150 – 250
DC motors with wheels	300 – 500
Robot chassis	300 – 600
Caster wheel	50 – 100
Battery / power bank	400 – 700
Jumper wires	100 – 200
Total	6,000 – 9,000

The estimated cost is far below that of commercial tracking platforms, and the software stack is free and open source.

B. Performance Measurements

Quantitative performance should be reported from actual experiments. Table X is provided for this purpose.

TABLE X PERFORMANCE MEASUREMENTS

Parameter	Test condition	Measured value
Detection success rate	Bright indoor light	—
Detection success rate	Dim light	—
Frame rate (fps)	640×480	—
Tracking latency (ms)	Detection to motor response	—
Distance error (cm)	Ultrasonic vs. tape measure	—
Maximum target speed followed	Walking pace	—

C. Discussion

The results show that a Raspberry Pi 4B is sufficient for real-time color-based tracking at modest resolution, and that combining vision with ultrasonic ranging gives more robust following than either alone: the camera provides direction, and the ultrasonic sensor provides distance and a safety stop. The main sources of error are lighting variation, similar-colored background objects, motor mismatch causing drift, and latency from sequential capture, processing, and actuation.

D. Comparison with Other Approaches

Table XI compares the proposed system qualitatively with two common alternatives.

TABLE XI QUALITATIVE COMPARISON OF APPROACHES

Feature	IR/ultrasonic-only robot	Arduino + camera module	Proposed Pi 4B system
Target identification	None	Limited	Color/shape/contour based

Feature	IR/ultrasonic-only robot	Arduino + camera module	Proposed Pi 4B system
Image processing	No	Very limited	OpenCV in real time
Distance sensing	Yes	Optional	Yes (HC-SR04)
Expandability	Low	Low to medium	High (Wi-Fi, ML, extra sensors)
Relative cost	Lowest	Low	Moderate
Suitable for learning CV	No	Partly	Yes

The comparison is qualitative and reflects the design characteristics of each approach rather than measured benchmarks. A quantitative comparison would require running the same task on each platform, which can be done as future work.

XI. FEASIBILITY STUDY

- *Technical*: all components are readily available, compatible, and modular; the Pi has enough power for OpenCV at modest resolution.
- *Economic*: the total cost of about INR 6,000–9,000 is low, and no software licenses are needed.
- *Operational*: after power-on, the robot works with minimal human intervention and is easy to demonstrate.
- *Time*: assembly takes a few days and software setup about a week, with the whole project achievable in 4–6 weeks.
- *Social*: the project supports STEM education and can be extended to assistive and warehouse applications.
- *Environmental*: it uses low-voltage DC power, produces no emissions, and uses reusable parts.

XII. ADVANTAGES AND LIMITATIONS

A. Advantages

- *Autonomous operation*: detects and follows the target without manual intervention.
- *Real-time capability*: the quad-core Pi handles image processing and motor control together.
- Compact, portable, and low power, suitable for classrooms, labs, and homes.
- Low cost and easy to source; open-source software with wide community support.
- *Flexible and modular*: detection color, distance thresholds, and motion logic can be changed with small code edits, and new sensors or wireless modules can be added.
- Strong educational value in embedded systems, image processing, PWM motor control, and system integration.

B. Limitations

- *Recognition accuracy*: color/shape/contour methods can confuse similar-colored objects and struggle with busy backgrounds.
- *Lighting sensitivity*: accuracy drops in low light or strong shadows.
- Limited compute for deep-learning models without accelerators such as Coral TPU or Jetson.
- *Battery life*: motors and camera drain the battery quickly, and a battery management system is needed for long runs.
- *Field of view*: the camera's field of view is roughly 62°, so fast targets can leave the frame.
- *Terrain*: the small chassis works best on smooth indoor floors.
- *Latency*: capture, processing, and actuation happen sequentially, adding a small delay.
- *Calibration*: motor speeds, camera focus, and detection thresholds need careful tuning.

XIII. APPLICATIONS

- Industrial and warehouse material handling.
- Home assistance, carrying items and following a person.
- Hospital delivery of medicines and supplies.
- Security and surveillance, following suspicious objects.
- Educational demonstrations in robotics, AI, and embedded systems.

- Foundation for autonomous vehicles and intelligent mobile robots.

XIV. SAFETY AND ETHICAL CONSIDERATIONS

- *Collision safety*: the robot must stop when the target is lost or when an obstacle is closer than the safe distance, and the motors must be stopped on any program error.
- *Electrical safety*: use correct polarity, fuses or protection where possible, and never exceed the L298N and Pi voltage ratings.
- *Privacy*: a camera-equipped robot can record people, so recordings should not be stored or shared without consent, especially in surveillance uses.
- *Responsible use*: following people should be done only with their knowledge and permission.

XV. CONCLUSION AND FUTURE SCOPE

A. Conclusion

This paper presented an autonomous object-following robot based on the Raspberry Pi 4B. By combining camera-based detection with ultrasonic ranging and an L298N-driven differential drive, the prototype detects, tracks, and follows a target in real time using inexpensive, widely available components. The project integrates hardware interfacing, embedded Python programming, and image processing into a working platform and serves as a practical learning and research base. While lighting dependence, moderate recognition accuracy, limited processing power, and battery life constrain performance, these can be addressed through the enhancements below.

B. Future Scope

- Deep-learning detection (YOLO, TensorFlow Lite, MobileNet) for accurate multi-object and person recognition, with accelerators if needed.
- Sensor fusion with IR, LiDAR, or IMU sensors for better obstacle avoidance and navigation.
- Wi-Fi/Bluetooth/IoT integration for remote control, live monitoring, and cloud analytics.
- Voice and gesture control for natural human-robot interaction.
- Improved power management, such as a BMS, efficient regulators, or solar charging, and PID-

controlled or encoder-based motors for smoother motion.

- Learning-based behavior and route optimization; extension to delivery, warehouse, hospital, and surveillance applications.

ACKNOWLEDGMENT

The authors thank the project guide and the Department of Electronics for their guidance and support.

REFERENCES

- [1] G. Bradski, "The OpenCV library," Dr. Dobb's J. Softw. Tools, 2000.
- [2] E. Upton and G. Halfacree, Raspberry Pi User Guide, 4th ed. Wiley, 2016.
- [3] S. Monk, Programming the Raspberry Pi: Getting Started with Python, 3rd ed. McGraw-Hill Education, 2021.
- [4] D. Molloy, Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux. Wiley, 2016.
- [5] R. Grimmett, Raspberry Pi Robotic Projects. Packt Publishing, 2015.
- [6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2016, pp. 779–788.
- [7] Raspberry Pi Foundation. [Online]. Available: <https://www.raspberrypi.com/documentation>
- [8] STMicroelectronics, L298 Dual Full-Bridge Driver, Datasheet.
- [9] Elec Freaks, HC-SR04 Ultrasonic Ranging Module, Datasheet.
- [10] OpenCV documentation. [Online]. Available: <https://docs.opencv.org>