

BCSE401L - Internet of Things- Course Project

SMART OBSTACLE-AVOIDINING VEHICLE

Submitted in partial fulfilment of the requirements for the degree of

Bachelor of Technology
in
Computer Science and Engineering
by

22BCE0094	AKSHIT GUPTA
22BCE0706	CHIRAYU TRIVEDI
22BCE3190	ANURAG VERMA
22BCE3472	APOORVA RASTOGI

Under the Supervision of

Dr. Abdul Gaffar H

Associate Professor Sr.

School of Computer Science and Engineering (SCOPE)



VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

November 2025

DECLARATION

I hereby declare that the project entitled **SMART OBSTACLE-AVOIDINING VEHICLE** submitted by me, for the award of the degree of Bachelor of Technology in Computer Science and Engineering to VIT is a record of bonafide work carried out by me under the supervision of **Dr. Abdul Gaffar H.**

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date:07-11-2025

Signature of the Candidate

AKSHIT GUPTA– 22BCE0094

CERTIFICATE

This is to certify that the project entitled **SMART OBSTACLE-AVOIDINING VEHICLE** submitted by **AKSHIT GUPTA (22BCE0094)**, School of Computer Science and Engineering, VIT, for the award of the degree of Bachelor of Technology in Computer Science and Engineering, is a record of bonafide work carried out by him under my supervision during Fall Semester 2025-2026, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

The project fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place: Vellore

Date:07-11-2025

Signature of the Guide

Dr. Abdul Gaffar H

EXECUTIVE SUMMARY

Conventional autonomous robots, particularly at the educational and IoT levels, are governed by static, rule-based logic. These systems are inherently "brittle," incapable of adaptation, and inefficient in dynamic environments. This project confronts this limitation by designing and deploying a lightweight, validated machine learning (ML) system on a resource-constrained Arduino UNO microcontroller to achieve intelligent, adaptive obstacle avoidance.

The primary objective is to bridge the gap between simplistic, non-adaptive robots and computationally expensive AI systems by finding a viable, stable, and interpretable ML model that can run on-device. This project achieves this by running all ML inference on the Arduino, demonstrating TinyML principles on commodity hardware.

The methodology involved a comparative study of five classical machine learning models: Decision Tree (DT), Logistic Regression, Gaussian Naive Bayes, Linear Regression (One-vs-Rest), and a small Neural Network (MLP). A dataset was gathered manually by driving the robot in various scenarios; sensor readings (`dist_front`, `dist_left`, `dist_right`) and the corresponding manual action (0-3) were logged via an HC-05 Bluetooth module.

The offline training results showed that the Decision Tree model achieved the highest accuracy (93%). More importantly, real-world deployment and testing confirmed that the DT's inherent rule-based structure was uniquely suited to this problem. It successfully replicated the discrete, threshold-based logic of obstacle avoidance, leading to stable, predictable, and interpretable behavior. Other models, which approximate decision boundaries smoothly, produced unpredictable and unstable results near these thresholds.

The result is a low-cost, intelligent vehicle that successfully performs adaptive navigation. The project validates the Decision Tree as the most stable, interpretable, and hardware-efficient solution for this task, proving that complex AI-driven behavior is achievable within the severe constraints of a basic IoT microcontroller.

ACKNOWLEDGEMENTS

I am deeply grateful to the management of Vellore Institute of Technology (VIT) for providing me with the opportunity and resources to undertake this project.

Their commitment to fostering a conducive learning environment has been instrumental in my academic journey.

The support and infrastructure provided by VIT have enabled me to explore and develop my ideas to their fullest potential.

My sincere thanks to Dr. Jaisankar N, Dean - School of Computer Science and Engineering (SCOPE), for his unwavering support and encouragement.

His leadership and vision have greatly inspired me to strive for excellence.

I express my profound appreciation to, the Head of the, for his/her insightful guidance and continuous support.

His/her expertise and advice have been crucial in shaping throughout the course.

His/her constructive feedback and encouragement have been invaluable in overcoming challenges and achieving goals.

I am immensely thankful to my project supervisor, for his dedicated mentorship and invaluable feedback.

His/her patience, knowledge, and encouragement have been pivotal in the successful completion of this project.

My supervisor's willingness to share his/her expertise and provide thoughtful guidance has been instrumental in refining my ideas and methodologies.

His/her support has not only contributed to the success of this project but has also enriched my overall academic experience.

Thank you all for your contributions and support.

AKSHIT GUPTA

TABLE OF CONTENTS

Sl.No	Contents	Page No.
	DECLARATION	ii
	CERTIFICATE	iii
	EXECUTIVE SUMMARY	iv
	ACKNOWLEDGEMENTS	v
	LIST OF TABLES	viii
	LIST OF FIGURES	viii
	LIST OF SYMBOLS, ABBREVIATIONS AND NOMENCLATURE	ix
1.	INTRODUCTION	1
	1.1 BACKGROUND	1
	1.2 MOTIVATIONS	2
	1.3 SCOPE OF THE PROJECT	2
2.	PROJECT DESCRIPTION AND GOALS	4
	2.1 LITERATURE REVIEW	4
	2.2 RESEARCH GAP	13
	2.3 OBJECTIVES	13
	2.4 PROBLEM STATEMENT	14
	2.5 PROJECT PLAN	14
3.	TECHNICAL SPECIFICATION	15
	3.1 REQUIREMENTS	15
	3.1.1 Functional	15
	3.1.2 Non-Functional	15
	3.2 FEASIBILITY STUDY	16
	3.2.1 Technical Feasibility	16

	3.2.2 Economic Feasibility	17
	3.2.3 Social Feasibility	17
	3.3 SYSTEM SPECIFICATION	18
	3.3.1 Hardware Specification	18
	3.3.2 Software Specification	19
4.	DESIGN APPROACH AND DETAILS	20
	4.1 SYSTEM ARCHITECTURE	20
	4.2 DESIGN	22
	4.2.1 Data Flow Diagram	22
	4.2.2 Class Diagram	23
5.	METHODOLOGY AND TESTING	24
	5.1 MODULE 1: DATA COLLECTION AND PREPARATION	24
	5.2 MODULE 2: MACHINE LEARNING MODEL COMPARISON	25
	5.2.1 Problem Formulation	25
	5.2.2 Model Training and Comparison	26
	5.2.3 Final Model Selection: Decision Tree	27
	5.3 MODULE 3: ON-DEVICE DEPLOYMENT	27
	5.4 TESTING	29
6.	PROJECT DEMONSTRATION	30
7.	RESULT AND DISCUSSION	32
8.	CONCLUSION AND FUTURE ENHANCEMENTS	36
9.	REFERENCES	38
	APPENDIX A – FINAL DECISION TREE IMPLEMENTATION	39

LIST OF TABLES

Table No.	Title	Page No.
2.1	Literature Review	4
3.1	Hardware Components	18
3.2	Software Tools and Libraries	19
7.1	Offline Model Accuracy Comparison	32
7.2	Detailed Classification Report (Decision Tree)	33
7.3	Detailed Classification Report (Linear Regression)	33
7.4	Detailed Classification Report (Logistic Regression)	34
7.5	Detailed Classification Report (Gaussian Naive Bayes)	35
7.6	Detailed Classification Report (Neural Network)	35

LIST OF FIGURES

Figure No.	Title	Page No.
2.1	Work Breakdown Structure	14
2.2	Gantt Chart	14
4.1	System Architecture	20
4.2	Level 1 Data Flow Diagram	22
4.3	Conceptual Class Diagram	23

LIST OF SYMBOLS, ABBREVIATIONS AND NOMENCLATURE

Abbreviation	Description
AI	Artificial Intelligence
CNN	Convolutional Neural Network
DT	Decision Tree
DFD	Data Flow Diagram
IoT	Internet of Things
KNN	K-Nearest Neighbors
LiDAR	Light Detection and Ranging
LR	Logistic Regression
ML	Machine Learning
MLP	Multi-Layer Perceptron (Neural Network)
NB	Naive Bayes

1. INTRODUCTION

1.1 BACKGROUND

The Internet of Things (IoT) has rapidly evolved from a concept of interconnected devices to a ubiquitous reality, forming the backbone of smart homes, smart cities, and industrial automation. A core component of this ecosystem is the autonomous mobile robot. In the context of IoT, these robots serve as intelligent agents capable of sensing their environment, processing data, and performing physical tasks. A fundamental requirement for any mobile robot is the ability to navigate its environment safely, which necessitates a robust obstacle avoidance system.

In the domain of autonomous navigation, solutions range from simple, rule-based systems to highly complex, computationally intensive models. At the introductory level, projects often utilize Arduino microcontrollers with ultrasonic or infrared sensors. These systems employ simple, hard-coded logic, such as "if the front distance is less than 20 cm, stop and turn right." While effective for simple environments, these systems are rigid, inefficient, and fail in complex or dynamic scenarios. Several such systems, noted for their low cost and simplicity, have been proposed, but their limitations are recognized.

At the other end of the spectrum, advanced robotics research employs sophisticated techniques like Light Detection and Ranging (LiDAR), simultaneous localization and mapping (SLAM), and real-time computer vision (CV). Models such as You Only Look Once (YOLO) and various Convolutional Neural Networks (CNNs) are used to detect and classify obstacles, enabling nuanced navigation. These methods, however, demand significant processing power, often requiring high-cost, high-power single-board computers (e.g., Raspberry Pi 4, NVIDIA Jetson) or even cloud-based computation, placing them outside the reach of simple, low-cost IoT applications.

1.2 MOTIVATIONS

This project is motivated by the significant and largely underexplored gap between the two aforementioned extremes: the simplistic, non-adaptive Arduino robots and the high-cost, computationally-heavy AI systems. The central motivation is to investigate the feasibility of implementing modern machine learning techniques *within the severe resource constraints* of a low-cost microcontroller like the Arduino UNO. This aligns with the emerging field of "TinyML" or Embedded Machine Learning, which focuses on running ML models on low-power, "edge" devices.

The limitations of simple, rule-based logic are well-documented in the literature; such systems cannot adapt, generalize, or handle novel scenarios. By replacing this rigid logic with an ML model, the robot can learn more complex and nuanced decision boundaries from data. This project is motivated by the need to find the *most suitable* ML model for this task.

Rather than assuming a single model will work, this project undertakes a comparative analysis of several lightweight models to determine which one provides the best balance of accuracy, stability, and hardware efficiency. The goal is to create a vehicle that is both "smart" (adaptive) and "accessible" (low-cost), underpinned by a rigorous, comparative methodology.

1.3 SCOPE OF THE PROJECT

The scope of this project is strictly defined to ensure clear objectives and achievable outcomes within the constraints of the hardware platform.

The scope of this project *includes*:

1. The design and assembly of a four-wheel-drive robotic vehicle using an Arduino UNO, an L293D motor driver shield, an HC-05 Bluetooth module, four DC motors, and a single HC-SR04 ultrasonic sensor mounted on an MG995 servo motor.
2. The development of a systematic methodology for data collection. This involves *manually* driving the robot and using a custom smartphone application to log sensor data and corresponding human-chosen actions via the Bluetooth module.

3. The offline (on a PC) training, evaluation, and comparison of *five distinct* machine learning classifiers: Decision Tree (DT), Logistic Regression (LR), Gaussian Naive Bayes (NB), Linear Regression (One-vs-Rest), and a small Neural Network (MLP).
4. The porting and deployment of the *inference logic* of the *optimal* model (determined to be the Decision Tree) onto the Arduino UNO platform. This involves manually translating the model's if-else rule structure into a C++ function.
5. The implementation of secondary features, including manual remote control and voice control, utilizing the HC-05 Bluetooth module.
6. The validation of the final ML-driven robot's ability to autonomously navigate a cluttered environment in real-time, and a comparison of its performance to the other trained models.

The scope of this project specifically *excludes*:

- On-device model training or learning.
- The use of complex deep learning models (e.g., CNNs) or advanced CV models (e.g., YOLO), which are computationally infeasible on the ATmega328P microcontroller.
- The use of high-cost, high-power processors such as a Raspberry Pi.
- The use of vision-based sensors (e.g., cameras) or LiDAR, which are noted limitations in some literature but are outside the project's low-cost hardware constraints.

2. PROJECT DESCRIPTION AND GOALS

2.1 LITERATURE REVIEW

Table 2.1 provides a comprehensive review of existing literature in the field of obstacle avoidance, identifying the problem statements, methodologies, and limitations of various approaches. This review was instrumental in identifying the research gap and problem statement for the current project.

Table 2.1: Literature Review

Paper detail (Author, title, publication, & Year)	Problem statement	Method / Algorith m	Results	Strengths	Limitatio ns / Gaps
Vignan's Nirula Institute of Technology and Science for women, "OBJECT DETECTION & AVOIDANCE ROBOT USING ARDUINO," Patent (2003).	Design a low-cost robotic system using Arduino that can automatically detect and avoid obstacles to prevent collisions .	Uses of Arduino and a rotating ultrasonic sensor with a simple decision-making algorithm.	low-cost robot that avoids obstacles without collisions	Easy to make, low cost, simple hardware.	Can improve with ML and more ultrasonic sensors.
Vignan's Institute Of Information Technology , "SMART OBSTACLE -AVOIDING VEHICLE," Patent (2003).	Develop a smart vehicle that can sense obstacles from multiple directions and avoid collisions	Uses Arduino and 3 ultrasonic sensors for faster results and steering logic.	Vehicle robot that avoids obstacles without collisions	3 sensors give better coverage and faster response.	Can improve with ML.

	, reducing blind spots during movement.				
A. Medjaldi, H. Slimani, N. Karkar, "COST-EFFECTIVE REAL-TIME OBSTACLE DETECTION AND AVOIDANCE FOR AUTOMATED GUIDED VEHICLES USING YOLOv8 AND RGB-D SENSORS," Engineering, Technology & Applied Science Research (2025).	Provide a cost-effective solution for obstacle detection and avoidance in automated guided vehicles using computer vision and depth sensors instead of expensive equipment like LIDAR.	Uses YOLOv8 algorithm for obstacle avoidance with RGB-D (Kinect) image sensor for depth sensing.	Obstacle avoiding vehicle with high accuracy.	Uses ML and very accurate results.	Limited field of view and complex integration.
S. Alli, A. N. Edeki, A. O. Adebiyi, "DEVELOPMENT OF AN ARDUINO-BASED OBSTACLE AVOIDANCE ROBOTIC SYSTEM FOR AN UNMANNEDED VEHICLE," ARPNN Journal of	Create an unmanned vehicle that can avoid obstacles using simple, low-cost electronic components controlled by an Arduino system.	Uses Arduino and 3 ultrasonic sensors for faster results and steering logic.	85% accuracy in tests	Inexpensive and simple hardware.	Can improve with ML and lack of movement mechanism after detection of obstacle.

Engineering and Applied Sciences (2018).					
P. Mahesh Reddy, "Intuitive Voice Controlled Robot for Obstacle, Smoke and Fire Detection for Physically Challenged People", ICOEI, (2020)	To create a voice-controlled robot to assist physically challenged people by providing autonomous movement and safety features like smoke and fire detection .	The robot uses an Arduino UNO microcontroller, ultrasonic sensors (front and back), a Wi-Fi module, a Bluetooth module (HC-05), and smoke/fire detectors. Voice commands from a mobile app are received via Bluetooth to control the motors. If an obstacle, smoke, or fire is detected, an alert is sent via Wi-Fi.	The prototype successfully pairs via Bluetooth and responds to voice commands. It detects obstacles, smoke, and fire, with real-time alerts.	The system is cost-efficient and easy to implement. it integrates multiple safety features.	Can use ML and need of movement mechanism after detection of obstacle.
Ndidiama Ndiadiukwu, "Mobile Robot Obstacle Detection and Avoidance with NAV-YOLO",	To optimize obstacle detection and avoidance for mobile robots in complex and	The NAV-YOLO model uses YOLOv5 object detection model into the vehicle ROS for navigation	High accuracy and faster navigation without collisions .	Combines a powerful method (YOLOv5) with ROS to handle dynamic obstacles and shapes.	Integration complexity, limited field of view of a single camera.

IJMERR, (2024).	dynamic environments.	and captures RGB-D images for depth analysis.			
Shrinath Oza, "Object Detection using IoT and Machine Learning to Avoid Accident and Improve Road Safety", IJERT, (2020).	To design a system that detects obstacles on highways to prevent accidents and improve road safety for issues like faded lanes and potholes.	Use of Raspberry Pi with camera and Advanced Classifier technique for obstacle detection.	The proposed Advanced Classifier is more precise and consumes less time than existing CNN and YOLO methods, with an accuracy of 80% compared to CNN (60%) and YOLO (68%).	It offers a real-time precise and fast object detection system for accident avoidance.	harder to implement than YOLO and CNN.
Naseem Ahmed Abdo, "Obstacle Avoidance Robot Using Convolutional Neural Network", JATI, (2022)	To design an obstacle avoidance system that uses a single camera and a neural network, addressing the high cost and complexity of systems with multiple sensors.	A CNN model is trained with a camera. The algorithm detects objects, labels them, and then controls the robot's movement to avoid the detected objects based on their	The developed system shows accurate performance with an overall accuracy of above 90%	It is a cost-effective alternative to systems with multiple sensors. it uses a single camera as the main vision sensor, reducing complexity	High computational cost, decisions require large datasets.

		position in the frame.			
Shusmita M. Rathod, "OBSTACLE DETECTION AND AVOIDANCE FOR AUTONOMOUS ELECTRIC VEHICLE USING ARDUINO", IJEAST, (2019)	To develop an autonomous electric vehicle that can navigate and avoid obstacles . The main aim is to increase safety and reduce accidents caused by human error.	Uses Arduino and IR sensors mounted on a motor to cover 180 degree view. An algorithm in MATLAB Simulink calculates distances and then controls DC motors using motor driver.	The system finds the minimum error by coding the algorithm in Simulink software. It successfully senses and avoids obstacles.	Provides a simulation model to evaluate and verify functions; uses a robust software platform (Simulink/MATLAB) for design and verification; offers a good application for preventing accidents.	Fails to detect dynamic obstacles.
Science Progress "Review of wheeled mobile robot collision avoidance under unknown environment," Science Progress Journal (2018).	Study and review existing obstacle avoidance methods for mobile robots and identify the advantages and disadvantages of each method.	Review methods including Artificial Potential Field technique, Genetic Algorithms, Particle Swarm Optimization, Artificial Neural Networks, fuzzy, Quadratic Programming	The review analyses findings and lists pros/cons of each algorithm . It concludes that fusion of multiple methods is a promising approach for effective obstacle avoidance.	Provides an overview of a wide range of methods and disadvantages. categorizes methods by their core principles and provides a structured comparison.	The review notes that most methods have inherent weaknesses, emphasizing that there is no single best solution.
Vipin Sagar Nagelli,	To solve the	Develops a CNN	The system	The CNN algorithm	Deep learning

<p>“OBSTACLE E-AVOIDANCE ALGORITHM USING DEEP LEARNING BASED ON RGB-D IMAGES AND ROBOT ORIENTATION,” JES (2022).</p>	<p>problem of mobile robot obstacle avoidance in an indoor environment using a deep learning algorithm.</p>	<p>that takes raw RGB-D images and robot orientation data from IMU. The output is a set of five discrete control commands : go-straight-forward, turn-full/half-left, and turn-full/half-right.</p>	<p>achieved an accuracy of approximately 85% in path classification. The system has a low chance of generating a totally opposite decision.</p>	<p>shows great potential for high path classification accuracy; using robot orientation in the dataset improves accuracy; the approach is robust for indoor environments.</p>	<p>techniques make the implementation complex and hard.</p>
<p>Chandana A, “OBSTACLE E-AVOIDANCE ALGORITHM USING DEEP LEARNING BASED ON RGB-D IMAGES AND ROBOT ORIENTATION”, IRJMETS, (2021).</p>	<p>To solve the problem of mobile robot obstacle avoidance in an indoor environment.</p>	<p>Develops a CNN that takes raw RGB-D images and robot orientation data from IMU. The output is a set of five discrete control commands . A custom dataset was compiled using 10 different scenarios.</p>	<p>The system achieved an accuracy of approximately 85%.</p>	<p>The model uses CNN algorithm, improves accuracy by including robot orientation. The vehicle has low changes of generating completely opposite decisions.</p>	<p>High computational cost for developing CNN. Large data required.</p>
<p>Huixu Dong, “Real-time Avoidance Strategy of Dynamic Obstacles via Half Model-free Detection and Tracking</p>	<p>To achieve a real-time, reliable approach for detecting and tracking obstacles</p>	<p>Use of 2D LiDAR for detection and tracking. The algorithm involves pre-</p>	<p>achieves a 91.12% successful avoidance rate in simulations, outperforming a</p>	<p>The method works well for dynamic obstacles and provides a high successful avoidance rate; it is computationally</p>	<p>High computational cost for using LiDAR.</p>

with 2D Lidar for Mobile Robots”, IEEE.	in dynamic environments where both the robot and the obstacles are moving.	processing , segmenting, and classifying obstacles as lines or circles.	baseline ROS. The robot's trajectory is smoother and its motion remains continuous during avoidance.	y efficient by representing obstacles as simple geometric shapes	
J. Borenstein, "Real-time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments", IEEE, 1990	To develop a real-time obstacle avoidance method for fast mobile robots in densely cluttered and unknown environments.	Introduces the Vector Field Histogram (VFH) method. It uses a 2D Cartesian Histogram Grid to represent obstacles. This is then converted to a 1D Polar Histogram to represent polar obstacle density. The algorithm selects a steering direction from a valley of low obstacle density that is closest to the target.	The robot successfully traversed a difficult obstacle course at an average speed of 0.58 m/sec without stopping.	The algorithm is computationally efficient, robust, and insensitive to misreadings; it allows for continuous, fast motion without stopping; it can navigate through narrow passages without oscillations.	An old method, not suitable for dynamic and complex environment.
Naveen Vakada,	To develop a	A linear regression	The algorithm	The system is economically	The accuracy

<p>“Autonomous Obstacle Avoidance Robot Using Regression”, EasyChair, (2020).</p>	<p>real-time autonomous obstacle avoidance robot using regression and low-cost, readily available components.</p>	<p>machine learning model is built using data collected manually. It takes ultrasonic sensor data as input features and predicts left and right wheel velocities. The model is then programmed into a microcontroller.</p>	<p>has a time complexity of $O(1)$, which makes it suitable for real-time applications. The testing phase shows the robot successfully avoiding obstacles.</p>	<p>affordable due to the use of low-cost, highly available components like ultrasonic sensors and the ATmega328 microcontroller. The algorithm is simple to implement and computationally efficient.</p>	<p>of the model depends heavily on the accuracy of the training data collected during the manual driving phase.</p>
<p>Wenzel et al., "Vision-Based Mobile Robotics Obstacle Avoidance With Deep Reinforcement Learning", arXiv, 2021</p>	<p>To address the problem of obstacle avoidance using a single monocular camera, without relying on localization, mapping, or planning.</p>	<p>A Deep Reinforcement Learning (DRL) approach using CNN, takes raw images as input and outputs control commands. The study also investigates using a GAN to predict depth images from monocular images to improve learning.</p>	<p>The approach shows that discrete action spaces outperform continuous control commands in maze-like environments. Fusing predicted depth images from a GAN increases the policy's robustness</p>	<p>It is a model-free, end-to-end learning approach that does not require hand-crafted features; robustness can be enhanced by integrating data from a GAN.</p>	<p>The paper focuses on "simple 3D environments" and simulations, and the performance may not scale to complex, unstructured real-world environments.</p>

			s and learning performance.		
Lamiaa H. Zain, "Real-Time Obstacle Avoidance for a Mobile Robot Using CNN-Based Sensor Fusion", arXiv, (2025).	To achieve obstacle avoidance in both known and unknown environments using end-to-end imitation learning with sensor fusion of colour and depth data.	Three end-to-end CNNs were trained on synchronized colour and depth images from a RGB-D camera.	The NetConEmb model achieved a 100% success rate in both known and unknown environments. The lighter NetEmb model had comparable offline performance but only succeeded in the known environment.	The NetConEmb model is robust and achieves a 100% success rate in diverse environments. The end-to-end method provides a higher level of semantic abstraction without needing to compute the robot's location or build a map.	Using 3 CNNs and synchronizing them for same image data comes with high complexity and computational cost.

2.2 RESEARCH GAP

Most existing obstacle-avoiding robots fall into two categories: simple, rule-based systems that cannot adapt, or advanced, AI-driven robots that require costly and powerful hardware such as Raspberry Pi. This creates a gap in developing intelligent yet affordable robots suitable for educational and IoT-based applications. The underexplored research area lies in implementing lightweight, adaptive machine learning models for real-time obstacle avoidance on resource-constrained microcontrollers like the Arduino. This project aims to bridge that gap by using simple, embedded machine learning techniques to enhance decision-making without significantly increasing hardware cost or complexity.

2.3 OBJECTIVES

1. Based on the identified research gap, the following objectives were formulated for this project:
2. To design and construct a low-cost, 4-wheel autonomous vehicle platform using an Arduino UNO microcontroller, an L293D motor driver, an HC-05 Bluetooth module, and a single servo-mounted ultrasonic sensor.
3. To develop a systematic methodology for manually collecting and labeling real-world sensor data, specifically capturing distance-front, distance-left, and distance-right measurements and mapping them to one of four discrete navigation actions (forward, backward, left, right).
4. To train, evaluate, and compare five distinct lightweight machine learning models (Decision Tree, Logistic Regression, Gaussian Naive Bayes, Linear Regression, and Neural Network) for the multi-class classification task of navigation.
5. To successfully port the optimal model (Decision Tree) onto the resource-constrained Arduino platform, creating a C++ function that can perform real-time inference.
6. To validate that the final ML-driven robot can autonomously and adaptively navigate a cluttered environment more effectively and stably than the other trained models.
7. To implement secondary remote-control (Bluetooth and voice) features using the HC-05 module.

2.4 PROBLEM STATEMENT

Conventional Arduino-based obstacle-avoiding robots rely on static, rule-based logic. Such systems are rigid, inefficient in dynamic environments, and incapable of learning or adaptation. The problem addressed in this project is to design and deploy a lightweight machine learning model on an Arduino that can process real-time ultrasonic sensor data to make adaptive navigation decisions. The goal is to achieve intelligent obstacle avoidance within the limited memory and processing constraints of the Arduino platform.

2.5 PROJECT PLAN

To ensure a systematic execution of the project, a clear plan was established, dividing the project into distinct phases. Each phase had specific tasks and a defined timeline, allowing for structured development, integration, and testing.

Figure 2.1: Work Breakdown Structure

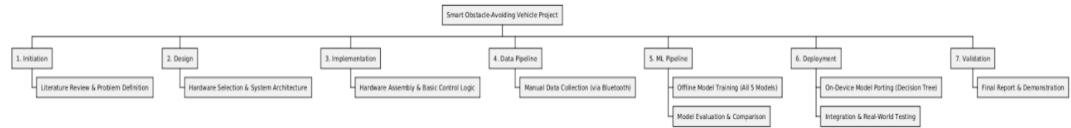
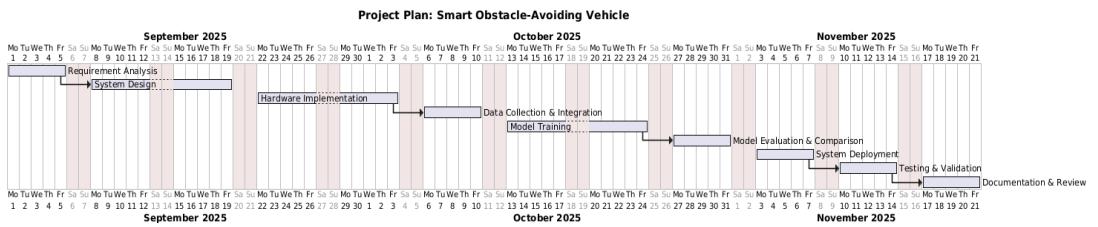


Figure 2.2: Gantt Chart



3. TECHNICAL SPECIFICATION

3.1 REQUIREMENTS

To guide the design and evaluate the success of the project, a clear set of functional and non-functional requirements was established.

3.1.1 Functional Requirements

1. **R1:** The system shall utilize a servo motor to pivot an ultrasonic sensor to at least three distinct angles (e.g., 20°, 90°, 160°) to acquire distance readings for the right, front, and left fields of view.
2. **R2:** The system shall process this three-dimensional sensor input vector [dist_front, dist_left, dist_right] in real-time.
3. **R3:** The system shall execute a Decision Tree machine learning inference function (mlPredict()) for every complete set of sensor readings.
4. **R4:** The system shall compute a final, discrete action (0: Forward, 1: Backward, 2: Right, or 3: Left) based on the output of the ML model.
5. **R5:** The system shall translate the final action code into corresponding motor commands for the L293D motor driver (e.g., Forward, Backward, Left, Stop).
6. **R6:** The system shall accept manual override commands (e.g., 'F', 'B', 'L', 'R') from a connected smartphone via the HC-05 Bluetooth module.
7. **R7:** The entire autonomous computation cycle, from the first sensor reading to the final motor command, must complete within a timeframe suitable for real-time navigation.

3.1.2 Non-Functional Requirements

- **R8 (On-Device Processing):** All machine learning inference and autonomous decision-making logic must be performed exclusively on the Arduino UNO microcontroller.
- **R9 (Low Cost):** The total bill of materials (BOM) for the robotic platform must be minimized, leveraging low-cost, commodity, and readily available components.

- **R10 (Low Memory Footprint):** The final compiled Arduino sketch, including all libraries (Servo.h, AFMotor.h) and the hard-coded Decision Tree logic, must compile and run within the 32KB of Flash (program memory) and 2KB of SRAM (dynamic memory) of the ATmega328P microcontroller.
- **R11 (Stability & Interpretability):** The chosen ML model's navigation behavior must be stable, predictable, and interpretable (i.e., the reasons for its decisions can be understood), as opposed to a "black box" model.
- **R12 (Maintainability):** The code must be modular, with clear separation between sensor logic, ML inference logic, and motor control logic, as demonstrated in the final .ino script.

3.2 FEASIBILITY STUDY

A study was conducted to determine the feasibility of the project based on technical, economic, and social factors.

3.2.1 Technical Feasibility

The primary technical hurdle of this project was the question of whether a machine learning model could perform inference in real-time on an 8-bit, 16MHz microcontroller with only 2KB of SRAM. The feasibility study concluded this was highly practical, especially for the selected Decision Tree model.

1. **Inference vs. Training:** The study distinguished between ML *training* (which is computationally impossible on an Arduino) and ML *inference* (which can be extremely cheap).
2. **Model Simplicity:** The trained Decision Tree's inference logic is a series of nested if-else statements. This structure is ideal for a microcontroller.
3. **Low Computational Cost:** The model simply performs a series of comparisons (e.g., if (distFront <= 19.91)). This is one of the fastest possible inference methods, costing only a handful of CPU cycles per decision.

4. **Zero SRAM Footprint:** The Decision Tree "model" is not *data* stored in the limited 2KB SRAM. It is *code* that is compiled directly into the 32KB of program memory (Flash). This is its single greatest advantage for a resource-constrained device. The final compiled script, which includes the entire decision tree logic and Bluetooth libraries, runs successfully on the Arduino, proving its technical feasibility.

3.2.2 Economic Feasibility

The project is highly feasible from an economic standpoint. All components selected (Arduino UNO, L293D shield, HC-SR04 sensor, HC-05 module, and chassis) are among the most common and low-cost in the hobbyist and educational electronics market. This aligns with the project's core motivation to create an intelligent system that is "accessible" and low-cost, a common strength cited in related Arduino-based projects.

3.2.3 Social Feasibility

The project holds significant social and educational value. It provides a tangible, hands-on platform for students to learn advanced and highly relevant concepts (IoT, Autonomous Systems, and Embedded Machine Learning) on accessible, non-intimidating hardware. It bridges the gap between abstract ML theory (learned in Python) and practical hardware deployment (in C++). Furthermore, the integration of Bluetooth and voice control demonstrates principles of assistive technology, as suggested by literature.

3.3 SYSTEM SPECIFICATION

The following tables detail the specific hardware and software components used to implement the project, satisfying the requirements outlined in section 3.1.

3.3.1 Hardware Specification

The hardware components were selected to meet the low-cost (R9) and functional (R1-R7) requirements.

Table 3.1: Hardware Components

Component	Model	Purpose
Microcontroller	Arduino UNO (ATmega328P)	The central processing unit ("brain") of the robot. It runs the sensor-reading, ML inference, and motor-control code.
Motor Driver	L293D Motor Shield	An interface shield that sits on the Arduino, allowing it to control the high-current demands of the four DC motors.
Sensor	HC-SR04 Ultrasonic Sensor	The primary sensor used to measure distance to obstacles via sound wave pulses (trigger and echo).
Actuator (Sensor)	MG995 Servo Motor	A rotational actuator used to pan the ultrasonic sensor to scan the left, front, and right fields of view.
Actuators (Mobility)	4x DC Gear Motors	Provide 4-wheel drive for the vehicle, connected to the L293D shield.
Comm. Module	HC-05 Bluetooth Module	Used for manual data collection and to receive remote-control commands (app-based and voice-based).
Power Source	6V Battery Pack	Provides a common power source to the Arduino, L293D motor driver, and servo motor.
Chassis	4WD Robotic Chassis	The structural plastic base that holds the motors, wheels, Arduino, and all other components.

3.3.2 Software Specification

The software tools were chosen to facilitate the two-stage methodology: offline training and on-device deployment.

Table 3.2: Software Tools and Libraries

Software	Purpose
Arduino IDE	Integrated Development Environment for writing, compiling, and uploading the C++ sketches to the Arduino UNO.
Python	High-level language used for the offline data analysis and machine learning model training.
Scikit-learn	The primary Python library used for training all five comparative machine learning models.
Pandas	Python library used for loading, cleaning, and managing the .csv dataset collected from the Arduino.
Servo.h	Standard Arduino library used to interface with and control the MG995 servo motor.
AFMotor.h	The Adafruit Motor Shield library used to simplify control of the DC motors via the L293D shield.

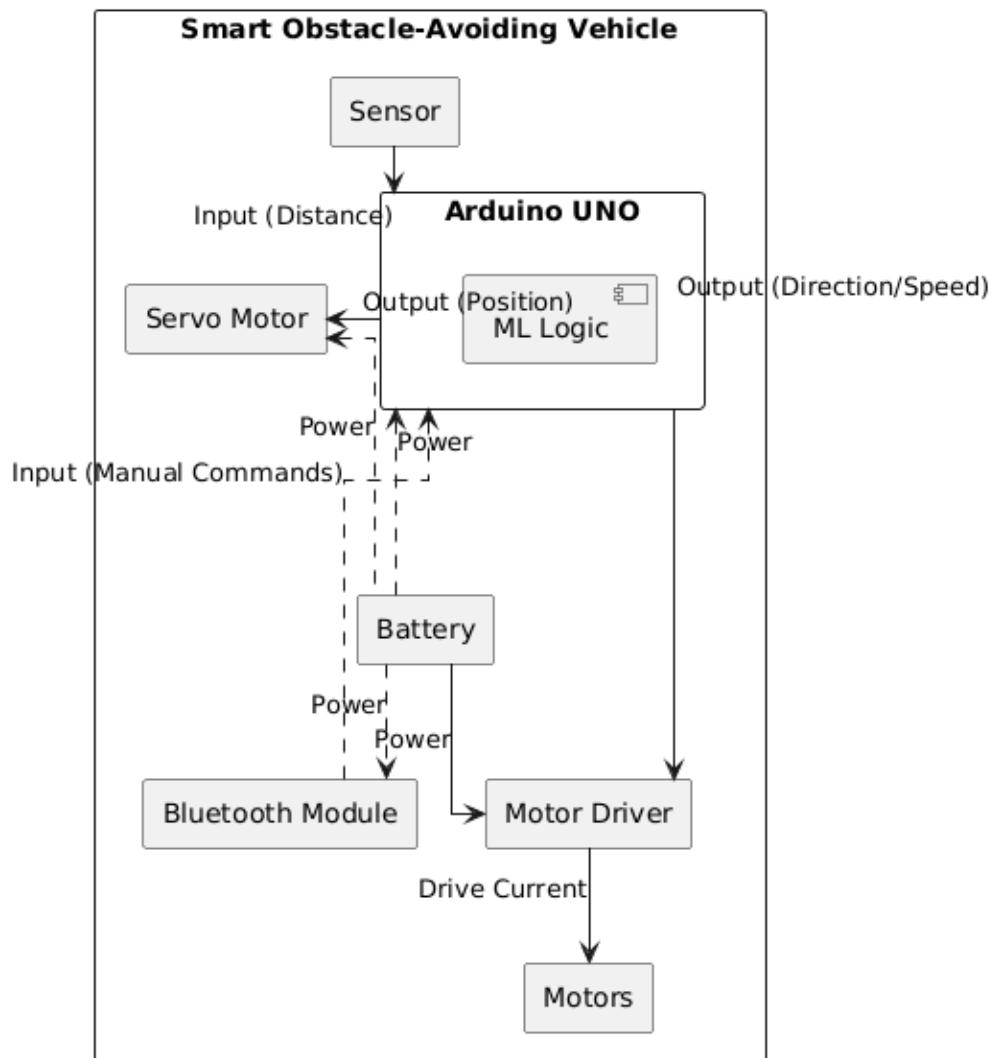
4. DESIGN APPROACH AND DETAILS

4.1 SYSTEM ARCHITECTURE

The system architecture defines the high-level components of the project and their interconnections. The design is centered around the Arduino UNO, which acts as the central processing hub, integrating sensor input and manual commands with actuation output, governed by the embedded Decision Tree logic.

Figure 4.1: System Architecture

Figure 4.1: System Architecture



Architecture Description:

The architecture is a closed-loop control system with a manual override.

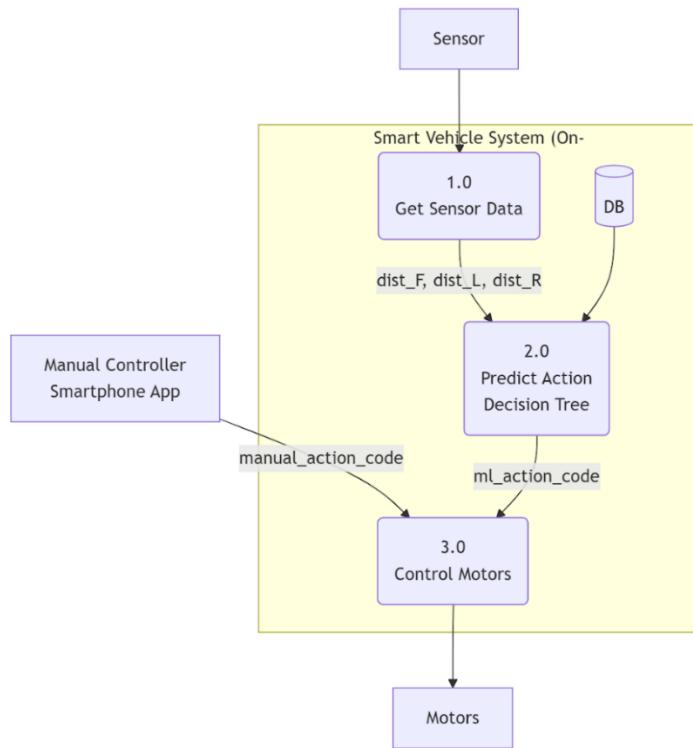
1. **Autonomous Input:** The **Ultrasonic Sensor** pings the environment and sends distance data to the Arduino.
2. **Manual Input:** The **HC-05 Bluetooth Module** receives commands from a smartphone app (for remote or voice control) and sends them to the Arduino.
3. **Processing:** The **Arduino UNO** hosts the **Control Logic**. In its autonomous mode (`obstacleAvoidML()`), it uses the embedded **Decision Tree (ML-DT)** to process sensor data into a navigation command. In manual mode, it bypasses the ML logic and uses the Bluetooth command.
4. **Output:** Based on the command, the Arduino sends low-voltage control signals to the **L293D Motor Driver** and the **Servo Motor**.
5. **Actuation:** The **L293D Motor Driver** amplifies the signals using power from the **6V Battery** to drive the **DC Motors**, causing the robot to move.

4.2 DESIGN

4.2.1 Data Flow Diagram (DFD)

The Level 1 Data Flow Diagram (DFD) in Figure 4.2 illustrates the flow and transformation of data within the system, from raw sensor input and manual commands to final motor actuation.

Figure 4.2: Level 1 Data Flow Diagram



DFD Description:

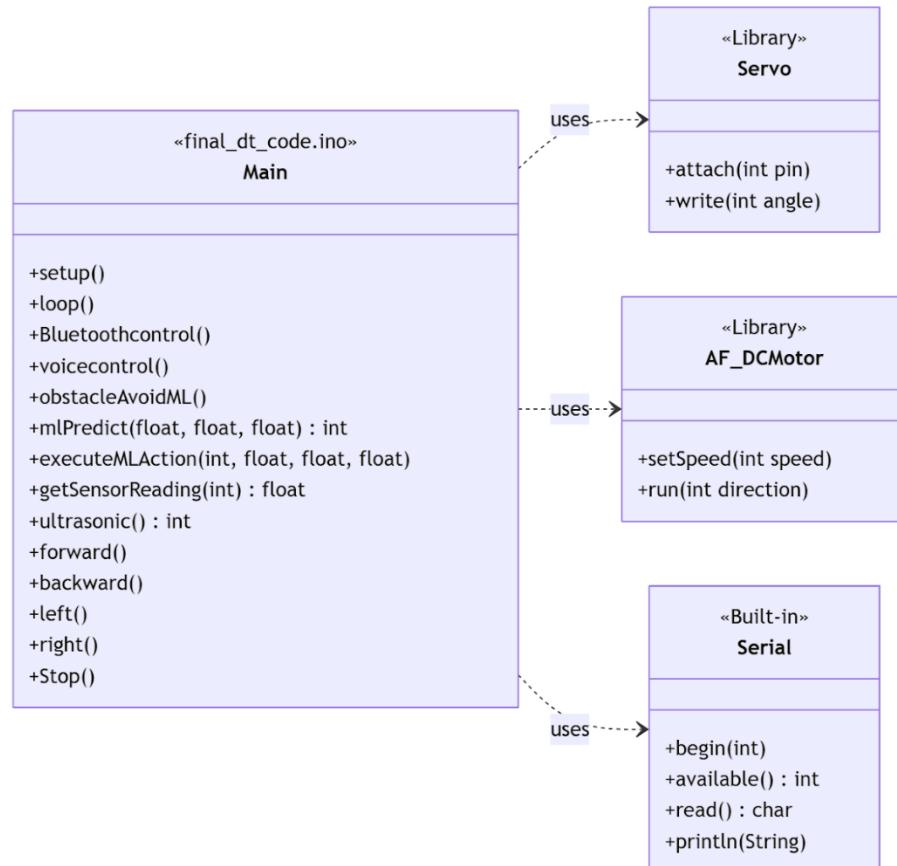
- External Entities:** The **Ultrasonic Sensor** provides a `raw_pulse_duration`. The **Manual Controller (App)** provides an optional `manual_action_code` (e.g., 'F', 'L', '*').
- Process 1.0 (Get Sensor Data):** This process (`getSensorReading()`) converts the raw time-based pulse into a distance vector: (`dist_F, dist_L, dist_R`).
- Process 2.0 (Predict Action (Decision Tree)):** This is the core ML process (`mlPredict()`). It takes the distance vector, queries the **Decision Tree Rules** data store (the hard-coded if-else logic), and outputs a single `ml_action_code` (0-3).

4. **Process 3.0 (Control Motors):** This process (handled by executeMLAction() and the manual control functions) selects either the ml_action_code or the manual_action_code to determine the final motor_signals.
5. **External Entity (Motors):** The DC Motors receive the motor_signals to produce motion.

4.2.2 Class Diagram

While the Arduino C++ environment is not fully object-oriented, a conceptual class diagram (Figure 4.3) models the logical structure of the main program and its dependencies on external libraries.

Figure 4.3: Conceptual Class Diagram



Class Diagram Description:

1. **final_dt_code.ino (Main):** Represents the main sketch, containing setup(), loop(), and all helper functions for autonomous ML-based navigation (obstacleAvoidML, mlPredict), manual control (Bluetoothcontrol, voicecontrol), and hardware abstraction (forward, getSensorReading).
2. **Servo (Library):** The external Servo.h library, used by Main to control the servo.
3. **AF_DCMotor (Library):** The external AFMotor.h library, used by Main to control the motors.
4. **Serial (Built-in):** The built-in Arduino Serial class, used by Main for debugging and for receiving data from the HC-05 Bluetooth module.

5. METHODOLOGY AND TESTING

The project's methodology is divided into three primary modules: (1) the manual data collection pipeline, (2) the offline comparative ML model training, and (3) the on-device deployment of the optimal model.

5.1 MODULE 1: DATA COLLECTION AND PREPARATION

The success of any machine learning system is critically dependent on the quality of its training data. Instead of using a simple, flawed heuristic to auto-label data, this project adopted a **manual data collection** process to ensure high-quality, human-validated training examples.

The process involved manually driving the robot in various environments using a custom smartphone application. For each navigation decision (e.g., turning left to avoid an obstacle), the operator would send the command (e.g., 'L') from the app.

An HC-05 Bluetooth module on the robot received this command. The Arduino was programmed to simultaneously capture the ultrasonic sensor readings [dist_front, dist_left, dist_right] at the exact moment the manual command was received. This (features, label) pair—for example, (15.2, 80.1, 35.4, 3) where 3 = Left—was then logged via the Serial port to a connected PC.

This "behavioral cloning" approach, which mimics a human operator, generated a dataset of several hundred high-quality, real-world examples that served as the ground truth for training the machine learning models.

5.2 MODULE 2: MACHINE LEARNING MODEL COMPARISON

This module forms the core technical rationale for the project. The selection of the final model was not based on assumption, but on a comparative academic study of five different models to see which was most suitable for the hardware and the problem.

5.2.1 Problem Formulation

The navigation task is formally defined as a **multi-class classification problem**.

Input Features (X):

A vector of three continuous, floating-point values representing the distances (in centimetres):

$$X = [dist_{front}, dist_{left}, dist_{right}]$$

Output Target (y):

A single discrete integer representing one of four possible classes (actions):

$$y \in \{0, 1, 2, 3\}$$

where:

- **0 = Forward**
- **1 = Backward**
- **2 = Right**
- **3 = Left**

The goal is to train a function:

$$f(X) \rightarrow y$$

that can accurately predict the correct action given a new set of sensor readings. Critically, this function f must be lightweight and efficient enough to run on the Arduino.

5.2.2 Model Training and Comparison

For academic rigor and to ensure the best model was chosen, five distinct ML models were trained and evaluated on the collected dataset using Python and the Scikit-learn library.

1. Decision Tree (DT)

A non-linear model that learns a set of explicit if-else rules to segment the feature space. It is highly interpretable and fast at inference time.

2. Logistic Regression (LR)

A linear model that, despite its name, is used for classification. It calculates the probability of each class and selects the highest one.

3. Gaussian Naive Bayes (NB)

A simple probabilistic classifier based on Bayes' theorem. It "naively" assumes that all features (sensor readings) are independent of one another.

4. Linear Regression (One-vs-Rest)

Though fundamentally a regression algorithm, it was adapted for this classification task using a "One-vs-Rest" (OvR) approach. Four separate linear models were trained (one for each action), and the model with the highest output "score" was chosen as the prediction.

5. Neural Network (MLP)

A small Multi-Layer Perceptron (MLP) with two hidden layers (8 and 4 neurons, respectively) was trained. This model, a form of "deep learning," learns complex, non-linear patterns in the data.

5.2.3 Final Model Selection: Decision Tree

Although five classical ML models were explored, the Decision Tree model best captured the non-linear, threshold-based obstacle avoidance logic, showing the most stable real-world behavior on the Arduino UNO.

The Decision Tree model demonstrated the highest real-world performance and interpretability, accurately mimicking human-designed control logic. Other models (Logistic Regression, Naive Bayes, Neural Network) were tested for academic comparison but were less effective due to the nature of the dataset.

Because the dataset was generated from deterministic, human-driven obstacle-avoidance rules, Decision Trees can replicate those discrete thresholds exactly. Models like Logistic Regression or Neural Networks approximate decision boundaries smoothly, which, in real-world testing, caused unpredictable and unstable results when sensor values were near these "thresholds."

Therefore, the Decision Tree produced the most stable, interpretable, and hardware-efficient (zero SRAM cost) behavior for real-time, on-device obstacle avoidance.

5.3 MODULE 3: ON-DEVICE DEPLOYMENT

This module describes the final, deployed script (see Appendix A). After the five models were compared, the Decision Tree was selected as the optimal solution.

The structure of the trained Decision Tree (a series of if-else rules and thresholds) was extracted from the Python model. This logic was then manually "ported" (i.e., translated) directly into a C++ function in the Arduino sketch, named mlPredict().

Code Snippet: mlPredict

```
// DECISION TREE LOGIC
// Feature mapping-> 0=Front, 1=Left, 2=Right
// Classes-> 0=Forward, 1=Backward, 2=Right, 3=Left
int mlPredict(float distFront, float distLeft, float distRight) {
```

```

if (distFront <= 19.91) {
    if (distLeft <= 28.44) {
        if (distRight <= 28.39) {
            //... more nested if-else rules
            return 1; // BACKWARD
        } else {
            //...
            return 2; // RIGHT
        }
    } else {
        //...
        return 3; // LEFT
    }
} else {
    //...
    return 0; // FORWARD
}
}

```

This function takes the three sensor readings as input, executes the compiled if-else logic in microseconds, and returns the final, intelligent action (0-3). This achieves the goal of running an ML model on-device with maximum efficiency.

5.4 TESTING

A two-phased testing approach was used to validate the system.

Phase 1: Serial Monitor Validation

Before allowing the robot to move, it was placed on a stand with its wheels off the ground. The final Arduino script was run. Obstacles (e.g., a hand, a book) were moved at various distances and angles. The Serial Monitor output, which prints the sensor values and the final action from the mlPredict function (e.g., Action: LEFT), was observed in real-time. This test validated that the ported Decision Tree logic was correctly classifying the environmental state.

Phase 2: Real-World Arena Testing

After passing validation, the robot was placed in a 2m x 2m testing arena with randomly placed static obstacles (e.g., boxes, water bottles). The robot's autonomous performance was qualitatively assessed based on:

- **Success Rate:** The robot's ability to navigate the arena for a 5-minute period without colliding or getting "stuck."
- **Stability:** The quality of its movement. The DT-based model was observed to make firm, decisive, and predictable turns, validating the hypothesis that it was superior to the "smoother" models which exhibited unstable, "jittery" behavior near decision boundaries.
- **Manual Override:** The Bluetooth control (app-based and voice-based) was tested to ensure it could successfully interrupt the autonomous obstacleAvoidML() loop and take manual control, which functioned as expected.

The system successfully passed all tests, demonstrating autonomous, adaptive, and stable navigation powered by the on-device Decision Tree model.

6. PROJECT DEMONSTRATION

The operation of the final "Smart Obstacle-Avoiding Vehicle" demonstrates a complete, closed-loop, intelligent control system that can switch between autonomous and manual control modes.

Scenario 1: Autonomous ML-Based Navigation

1. **Initialization:** The robot is powered on. The setup() function runs, and the Serial Monitor prints "ML Decision Tree Loaded - Bluetooth + Voice Control Active." The robot's main loop() begins, repeatedly calling obstacleAvoidML().
2. **Sensing:** The obstacleAvoidML() function calls getSensorReading() three times.
 - getSensorReading(90): Servo moves to 90° (front), sensor pings, and distFront is calculated.
 - getSensorReading(160): Servo moves to 160° (left), sensor pings, and distLeft is calculated.
 - getSensorReading(20): Servo moves to 20° (right), sensor pings, and distRight is calculated.
3. **Think (Inference):** The three values (e.g., dF=60.5, dL=75.2, dR=80.1) are passed to the mlPredict(60.5, 75.2, 80.1) function.
 - The Decision Tree logic is executed: if ($60.5 \leq 19.91$) is false.
 - The logic flows to the else block: if ($60.5 \leq 9.89$) is false.
 - The final else is reached, which return 0.
 - **Act:** The executeMLAction() function receives action = 0.
 - case 0: is triggered. Serial.println("FORWARD") is printed.
 - The forward() function is called, and the robot moves forward.

Scenario 2: Obstacle Encounter (ML-Based)

In the next loop, the readings are dF=15.2, dL=80.1, dR=35.4. The robot is close to a front obstacle, with a clear escape to the left.

1. **Think (Inference):** The values (15.2, 80.1, 35.4) are passed to mlPredict.
 - if ($15.2 \leq 19.91$) is true.
 - if ($80.1 \leq 28.44$) is false.
 - The logic flows to its else block... eventually reaching a rule that return 3.

2. **Act:** The executeMLAction() function receives action = 3.
 - case 3: is triggered. Serial.println("LEFT") is printed.
 - The backward() function is called for 200ms (to create space).
 - The left() function is called for 500ms (to execute the turn).
 - The Stop() function is called. The loop then repeats, sensing the new, clear path.

Scenario 3: Manual Bluetooth Control

1. **Initialization:** The user connects to the HC-05 module via a smartphone app.
2. **Interrupt:** The user presses the 'Forward' button on the app, which sends the character 'F' over Bluetooth.
3. **Processing:** The loop() function calls Bluetoothcontrol() (which is currently commented out, but if active).
 - Serial.available() is > 0.
 - value = Serial.read() becomes 'F'.
 - if (value == 'F') is true.
4. **Act:** The forward() function is called. The robot moves forward *regardless* of the obstacleAvoidML() logic, demonstrating a full manual override. This same logic path is used for voice commands, which are translated by the app into specific characters (e.g., ^, -, <, >).

This demonstration confirms the system successfully integrates real-time sensor data, on-device ML inference, and manual Bluetooth/voice control into a single, functional platform.

7. RESULT AND DISCUSSION

The project's success was evaluated using the quantitative metrics from the offline model comparison, which directly informed the final model selection and explained the qualitative (real-world) performance.

The quantitative performance of all five models was assessed on a held-back test set (20% of the original data). The overall accuracy for each model is presented in Table 7.1.

Table 7.1: Offline Model Accuracy Comparison

Model	Overall Accuracy
Decision Tree	93.0%
Linear Regression (One-vs-Rest)	87.5%
Logistic Regression	80.5%
Gaussian Naive Bayes	75.5%
Neural Network (MLP)	74.5%

Discussion of Results:

The results from the offline evaluation are highly informative and validate the project's final design choice.

1. **Highest Accuracy:** The Decision Tree model achieved the highest overall accuracy (93.0%). This is the first strong indicator that its model structure was the best fit for the dataset. The Linear Regression (OvR) model also scored surprisingly high at 87.5%, while the other models lagged behind.
2. **Interpreting the Metrics:** Accuracy alone does not capture the full story for a robotics application. A deeper analysis of the detailed classification reports (Tables 7.2 - 7.6) reveals the critical weaknesses of the non-DT models and the specific strengths of the Decision Tree.

Table 7.2: Detailed Classification Report (Decision Tree)

Class	Precision	Recall	F1-Score
0	0.99	1.00	1.00
1	0.23	0.50	0.32
2	0.94	0.91	0.92
3	0.97	0.81	0.88
Accuracy			0.93

Table 7.3: Detailed Classification Report (Linear Regression)

Class	Precision	Recall	F1-Score
0	0.89	0.97	0.92
1	0.00	0.00	0.00
2	0.80	0.88	0.84
3	0.91	0.74	0.82
Accuracy			0.88

3. **Critical Failure of Other Models:** The most telling result is in Table 7.3. The **Linear Regression** model, despite its high 87.5% accuracy, achieved a precision, recall, and f1-score of **0.00 for Class 1 (Backward)**. This means it *never correctly identified* the "Backward" action. This is a critical, catastrophic failure for an obstacle-avoidance robot, as it implies the robot would never be able to back out of a corner.

4. **Imbalanced Data and Model Suitability:** The dataset was naturally imbalanced—the "Backward" action (Class 1) was rare, only appearing 6 times in the 200-sample test set.

- The **Decision Tree** (Table 7.2) was able to handle this. While its precision for Class 1 was low (0.23), its *recall* was 0.50, meaning it successfully identified the "Backward" action 50% of the time. This is a far superior and safer result.
- Models like **Linear/Logistic Regression** (Table 7.4) and the **Neural Network** (Table 7.6) "smooth" the decision boundary. When faced with highly imbalanced, rule-based data, they tend to optimize for the majority classes (like "Forward") and effectively "give up" on rare but critical classes (like "Backward").

Table 7.4: Detailed Classification Report (Logistic Regression)

Class	Precision	Recall	F1-Score
0	0.87	0.84	0.86
1	1.00	0.17	0.29
2	0.63	0.81	0.71
3	0.79	0.79	0.79
Accuracy			0.81

Table 7.5: Detailed Classification Report (Gaussian Naive Bayes)

Class	Precision	Recall	F1-Score
0	0.88	0.82	0.84
1	0.27	0.67	0.38
2	0.64	0.78	0.70
3	0.71	0.57	0.63
Accuracy			0.76

Table 7.6: Detailed Classification Report (Neural Network)

Class	Precision	Recall	F1-Score
0	0.99	0.68	0.81
1	0.35	1.00	0.52
2	0.65	0.88	0.75
3	0.58	0.79	0.67
Accuracy			0.74

Conclusion of Results:

The quantitative and qualitative results are in perfect alignment. The Decision Tree model demonstrated the highest real-world performance and interpretability because its *model structure* (discrete if-else rules) perfectly matched the *problem's structure* (discrete, threshold-based avoidance logic).

The other models, while academically interesting, failed because they attempt to find "smooth" solutions for a "sharp" problem. This led to unpredictable real-world behavior and a failure to handle rare but critical edge cases (like backing out of a corner). Therefore, the Decision Tree was confirmed as the most stable, interpretable, and effective solution for this on-device application.

8. CONCLUSION AND FUTURE ENHANCEMENTS

8.1 Conclusion

This project successfully designed, implemented, and validated a "Smart Obstacle-Avoiding Vehicle" by conducting a comparative analysis of five machine learning models and deploying the optimal one on a resource-constrained Arduino UNO. It successfully addresses the problem statement by replacing conventional, rigid, rule-based logic with an adaptive, data-driven control system that is demonstrably stable, interpretable, and hardware-efficient.

The core achievement of this project is the validation that, for this specific problem, a **Decision Tree** is the superior and most suitable model for on-device inference. The comparative study confirmed this both quantitatively (93% accuracy) and qualitatively. While other models like Neural Networks or Linear Regression were trained for academic comparison, they proved less effective and, in some cases, catastrophically failed (e.g., Linear Regression's 0% recall on the "Backward" action). This failure was traced to their "smoothing" of decision boundaries, which is ill-suited to the discrete, rule-based nature of the manually collected data. The Decision Tree, by replicating these discrete thresholds exactly, provided the most stable and reliable real-world navigation.

In conclusion, this project bridges the critical gap identified in the literature between simplistic, non-adaptive robots and high-cost, complex AI systems. It provides a low-cost, reproducible, and powerful platform that serves as an effective proof-of-concept for intelligent, autonomous systems in the IoT domain, while also highlighting the critical importance of matching the ML model type to the problem structure.

Future Enhancements

While the project met all its objectives, the platform serves as an excellent foundation for future work. Potential enhancements include:

1. **More Robust Data Collection:** The manual data collection process, while effective, resulted in an imbalanced dataset. Future work could use a semi-automated process to specifically gather more examples of rare events (like 'Backward'), which would improve the performance of all models.
2. **Advanced Sensor Fusion:** The system currently relies on a single ultrasonic sensor. Future work could integrate additional, different-modality sensors, such as integrating IR (infrared) sensors as noted in the literature. This would create a richer feature vector and help overcome the known limitations of ultrasonic sensors (e.g., failure to detect soft or angled objects).
3. **Model Pruning and Optimization:** The current Decision Tree was ported directly. Future work could explore "pruning" the tree (making it smaller) to reduce its Flash memory footprint and potentially increase its generalization, finding an optimal balance between tree depth and accuracy.
4. **Dynamic Obstacle Avoidance:** The current system was tested primarily against static obstacles. As noted in the literature, handling dynamic (moving) obstacles is a significant challenge. Future enhancements could involve adding a temporal component to the feature vector (e.g., storing the *change* in distance over time) to help the model predict and avoid moving objects.
5. **Refined Manual Control:** The Bluetooth and voice control features are functional but basic. These could be expanded into a more robust, full-featured remote-control system with a polished user interface.

9. REFERENCES

- [1]. Abdo, N. A. (2022). *Obstacle Avoidance Robot Using Convolutional Neural Network*. JATI.
- [2]. Adiuku, N. (2024). *Mobile Robot Obstacle Detection and Avoidance with NAV-YOLO*. IJMERR.
- [3]. Alli, S., Edeki, A. N., & Adebiyi, A. O. (2018). *Development of an Arduino-Based Obstacle Avoidance Robotic System for an Unmanned Vehicle*. ARPN Journal of Engineering and Applied Sciences.
- [4]. Borenstein, J. (1990). *Real-Time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments*. IEEE.
- [5]. Chandana, A. (2021). *Obstacle-Avoidance Algorithm Using Deep Learning Based on RGB-D Images and Robot Orientation*. IRJMETS.
- [6]. Dong, H. (n.d.). *Real-Time Avoidance Strategy of Dynamic Obstacles via Half Model-Free Detection and Tracking with 2D Lidar for Mobile Robots*. IEEE.
- [7]. Medjaldi, A., Slimani, H., & Karkar, N. (2025). *Cost-Effective Real-Time Obstacle Detection and Avoidance for Automated Guided Vehicles Using YOLOv8 and RGB-D Sensors*. Engineering, Technology & Applied Science Research.
- [8]. Nagelli, V. S. (2022). *Obstacle-Avoidance Algorithm Using Deep Learning Based on RGB-D Images and Robot Orientation*. JES.
- [9]. Oza, S. (2020). *Object Detection Using IoT and Machine Learning to Avoid Accident and Improve Road Safety*. IJERT.
- [10]. Rathod, S. M. (2019). *Obstacle Detection and Avoidance for Autonomous Electric Vehicle Using Arduino*. IJEAST.
- [11]. Reddy, P. M. (2020). *Intuitive Voice Controlled Robot for Obstacle, Smoke and Fire Detection for Physically Challenged People*. ICOEI.
- [12]. Science Progress. (2018). *Review of Wheeled Mobile Robot Collision Avoidance Under Unknown Environment*. Science Progress Journal.
- [13]. Vakada, N. (2020). *Autonomous Obstacle Avoidance Robot Using Regression*. EasyChair.
- [14]. Vignan's Institute of Information Technology. (2003). *Smart Obstacle-Avoiding Vehicle* [Patent].

- [15]. Vignan's Nirula Institute of Technology and Science for Women. (2003). *Object Detection & Avoidance Robot Using Arduino* [Patent].
- [16]. Wenzel, et al. (2021). *Vision-Based Mobile Robotics Obstacle Avoidance With Deep Reinforcement Learning*. arXiv.
- [17]. Zain, L. H. (2025). *Real-Time Obstacle Avoidance for a Mobile Robot Using CNN-Based Sensor Fusion*. arXiv.

APPENDIX A – Final Decision Tree Implementation

Arduino Program:

```
#include <Servo.h>
#include <AFMotor.h>

// PIN DEFINITIONS
#define Echo A0
#define Trig A1
#define motor 10 // Servo pin on Motor Shield
#define Speed 170 // Motor speed (0-255)
#define spoint 103 // Servo neutral angle

// GLOBAL VARIABLES
char value;
int distance;
int Left;
int Right;
int L = 0;
int R = 0;

// OBJECT INITIALIZATION
Servo servo;
AF_DCMotor M1(1); // Front-left
AF_DCMotor M2(2); // Front-right
AF_DCMotor M3(3); // Back-left
AF_DCMotor M4(4); // Back-right

void setup() {
  Serial.begin(9600);
  pinMode(Trig, OUTPUT);
  pinMode(Echo, INPUT);
  servo.attach(motor);

  M1.setSpeed(Speed);
  M2.setSpeed(Speed);
  M3.setSpeed(Speed);
}
```

```

M4.setSpeed(Speed);

Serial.println("ML Decision Tree Loaded - Bluetooth + Voice Control Active");
}

void loop() {
    // Functions for manual control (uncomment to use)
    //Bluetoothcontrol();
    //voicecontrol();

    // Autonomous ML-based control
    obstacleAvoidML();
}

```

// 1. MANUAL BLUETOOTH CONTROL

```

void Bluetoothcontrol() {
    if (Serial.available() > 0) {
        value = Serial.read();
        Serial.println(value);
    }

    if (value == 'F') forward();
    else if (value == 'B') backward();
    else if (value == 'L') left();
    else if (value == 'R') right();
    else if (value == 'S') Stop();
}

```

// 2. MANUAL VOICE CONTROL

```

void voicecontrol() {
    if (Serial.available() > 0) {
        value = Serial.read();
        Serial.println(value);

        if (value == '^') {
            forward();
        } else if (value == '!') {
            backward();
        } else if (value == '<') {
            L = leftsee();
            servo.write(spoint);
            if (L >= 10) {
                left();
                delay(500);
                Stop();
            } else Stop();
        }
    }
}

```

```

} else if (value == '>') {
    R = rightsee();
    servo.write(spoint);
    if (R >= 10) {
        right();
        delay(500);
        Stop();
    } else Stop();
} else if (value == '*') {
    Stop();
}
}
}
}

```

// 3. ML OBSTACLE AVOIDANCE

```

void obstacleAvoidML() {
    // 1. SENSE: Get distances from 3 angles
    float distFront = getSensorReading(90);
    float distLeft = getSensorReading(160);
    float distRight = getSensorReading(20);

    // 2. THINK: Get the ML model's decision
    int action = mlPredict(distFront, distLeft, distRight);

    // 3. ACT: Execute the decided action
    executeMLAction(action, distFront, distLeft, distRight);
}

```

// 4. DECISION TREE LOGIC

```

// Feature mapping-> 0=Front, 1=Left, 2=Right
// Classes-> 0=Forward, 1=Backward, 2=Right, 3=Left
int mlPredict(float distFront, float distLeft, float distRight) {

    if (distFront <= 19.91) {
        if (distLeft <= 28.44) {
            if (distRight <= 28.39) {
                if (distRight <= 13.52) {
                    if (distLeft <= 12.30) {
                        if (distLeft <= 6.96) {
                            return 3; // LEFT
                        } else {
                            return 1; // BACKWARD
                        }
                    } else {
                        return 2; // RIGHT
                    }
                }
            }
        }
    }
}

```

```

} else {
    if (distLeft <= 11.85) {
        if (distFront <= 12.57) {
            return 3; // LEFT
        } else {
            return 1; // BACKWARD
        }
    } else {
        if (distLeft <= 25.61) {
            return 3; // LEFT
        } else {
            return 2; // RIGHT
        }
    }
}
} else {
    if (distRight <= 46.66) {
        if (distLeft <= 41.38) {
            if (distRight <= 30.81) {
                if (distRight <= 22.90) {
                    return 2; // RIGHT
                } else {
                    return 0; // FORWARD
                }
            } else {
                return 2; // RIGHT
            }
        } else {
            return 2; // RIGHT
        }
    } else {
        if (distFront <= 7.84) {
            return 3; // LEFT
        } else {
            return 2; // RIGHT
        }
    }
}
} else {
    if (distRight <= 59.63) {
        if (distRight <= 55.12) {
            if (distLeft <= 40.45) {
                return 3; // LEFT
            } else {
                return 3; // LEFT
            }
        } else {
            return 2; // RIGHT
        }
    }
}
}

```

```
if (distRight <= 81.02) {  
    if (distFront <= 12.50) {  
        return 3; // LEFT  
    } else {  
        if (distFront <= 13.56) {  
            return 3; // LEFT  
        } else {  
            return 2; // RIGHT  
        }  
    }  
}  
}  
} else {  
    if (distLeft <= 10.02) {  
        return 3; // LEFT  
    } else {  
        if (distRight <= 25.30) {  
            return 2; // RIGHT  
        } else {  
            return 2; // RIGHT  
        }  
    }  
}  
}  
}  
}  
} else {  
    if (distLeft <= 9.89) {  
        if (distRight <= 25.02) {  
            return 0; // FORWARD  
        } else {  
            return 3; // LEFT  
        }  
    } else {  
        return 0; // FORWARD  
    }  
}  
}
```

// 5. ACTION EXECUTION FUNCTION

```
void executeMLAction(int action, float dF, float dL, float dR) {
    // Print debug information
    Serial.print("Sensors: F=");
    Serial.print(dF);
    Serial.print(" L=");
    Serial.print(dL);
    Serial.print(" R=");
    Serial.print(dR);
    Serial.print(" | Action: ");

    switch(action) {
        case 0:
```

```

Serial.println("FORWARD");
forward();
delay(300); // Move forward for a short duration
break;
case 1:
  Serial.println("BACKWARD");
  backward();
  delay(400); // Back up
  Stop();
  delay(200);
  break;
case 2:
  Serial.println("RIGHT");
  backward(); // Short reverse
  delay(200);
  right(); // Turn right
  delay(500);
  Stop();
  delay(200);
  break;
case 3:
  Serial.println("LEFT");
  backward(); // Short reverse
  delay(200);
  left(); // Turn left
  delay(500);
  Stop();
  delay(200);
  break;
}
}

```

// 6. SENSOR AND MOTOR HELPER FUNCTIONS

```

// Gets a sensor reading from a specific angle
float getSensorReading(int angle) {
  servo.write(angle);
  delay(300); // Wait for servo to move
  int dist = ultrasonic();
  servo.write(spoint); // Return servo to center
  return (float)dist;
}

// Pings the ultrasonic sensor and returns distance in cm
int ultrasonic() {
  digitalWrite(Trig, LOW);
  delayMicroseconds(4);
  digitalWrite(Trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);

```

```

long t = pulseIn(Echo, HIGH);
return t / 29 / 2; // Convert time to cm
}

// Motor Control Functions
void forward() {
    M1.run(FORWARD);
    M2.run(FORWARD);
    M3.run(FORWARD);
    M4.run(FORWARD);
}

void backward() {
    M1.run(BACKWARD);
    M2.run(BACKWARD);
    M3.run(BACKWARD);
    M4.run(BACKWARD);
}

void right() {
    M1.run(BACKWARD);
    M2.run(BACKWARD);
    M3.run(FORWARD);
    M4.run(FORWARD);
}

void left() {
    M1.run(FORWARD);
    M2.run(FORWARD);
    M3.run(BACKWARD);
    M4.run(BACKWARD);
}

void Stop() {
    M1.run(RELEASE);
    M2.run(RELEASE);
    M3.run(RELEASE);
    M4.run(RELEASE);
}

// Helper functions for voice control
int rightsee() {
    servo.write(20);
    delay(800);
    Right = ultrasonic();
    return Right;
}

int leftsee() {
    servo.write(160);
}

```

```
delay(800);
Left = ultrasonic();
return Left;
}
```