IEEE_Conference np_Template__1_ (2).pdf

SRM Institute of Science and Technology

Document Details

Submission ID
trn:oid:::3618:120281331

Submission Date
Nov 8, 2025, 9:11 AM GMT+5:30

Download Date
Nov 8, 2025, 9:12 AM GMT+5:30

File Name
IEEE_Conference np_Template__1_ (2).pdf

File Size
86.8 KB

5 Pages

3,651 Words

20,696 Characters

8% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

29 Not Cited or Quoted 8%
Matches with neither in-text citation nor quotation marks

0 Missing Quotations 0%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

4% Internet sources
2% Publications
6% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- **29** Not Cited or Quoted 8%
  Matches with neither in-text citation nor quotation marks
- **0** Missing Quotations 0%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

- 4% Internet sources
- 2% Publications
- 6% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Internet	
silo.tips		<1%
2	Submitted works	
CSU, San Jose State University on 2025-04-16		<1%
3	Submitted works	
Lebanese International University on 2017-10-05		<1%
4	Submitted works	
Middlesex University on 2023-04-02		<1%
5	Submitted works	
MCAST on 2015-04-29		<1%
6	Submitted works	
University of Salford on 2024-04-19		<1%
7	Submitted works	
Queen Mary and Westfield College on 2025-11-03		<1%
8	Submitted works	
University of Wales, Lampeter on 2023-02-27		<1%
9	Submitted works	
University of Warwick on 2024-04-22		<1%
10	Internet	
www.ofcsliven2000.com		<1%

11 Submitted works
Sydney Institute of Technology and Commerce on 2024-06-07 <1%

12 Submitted works
City University on 2017-12-14 <1%

13 Internet
ar.iub.edu.bd <1%

14 Internet
mafiadoc.com <1%

15 Submitted works
RMIT University on 2025-10-10 <1%

16 Submitted works
Dhirubhai Ambani Institute of Information and Communication on 2025-05-01 <1%

17 Internet
www.mdpi.com <1%

18 Publication
Chokri Abdelmoula, Mohamed Masmoudi, Fakher Chaari. "Obstacle avoidance of ... <1%

19 Publication
Ling Zhu, C. Guedes Soares. "Trends in Collision and Grounding of Ships and Offsh... <1%

20 Internet
www.perlego.com <1%

21 Submitted works
Victorian Institute of Technology on 2025-05-18 <1%

Robo Nav- Obstacle Aviodance and path Planning

1st Nagendra Prasath
SRM IST Trichy
Dept.of computer Science
na6096@srmist.edu.in

2nd Prathikshan
SRM IST Trichy
Dept.of computer Science
pk8653@srmist.edu.in

***Abstract*—Automation in warehouse settings is now crucial for boosting productivity and lightening manual tasks. This project focuses on creating a robot that can move around the warehouse independently while avoiding obstacles. The robot has distance measurement sensors to spot objects nearby and prevent collisions. A path planning algorithm is used to choose the best route for moving through the warehouse. The control system combines sensor data with motor control decisions using a microcontroller. This setup allows for smooth, efficient, and safe navigation. The proposed system shows that autonomous robotic navigation is possible in structured indoor spaces and points to its potential for large-scale industrial uses.**

***Index Terms*—Multiplayer games, real-time systems, web technologies, synchronization, Socket.IO, HTML5, Node.js, browser gaming.**

I. Introduction

A. Background

In recent years, autonomous robots have become very important in areas like industrial automation, defense, healthcare, transportation, and environmental monitoring. A key factor that allows a robot to perform well in these roles is its ability to navigate safely and intelligently in its environment. Robot navigation is the process where a robot finds its location, plans a route, and moves toward a target without human help.

One of the biggest challenges in robot navigation is avoiding obstacles. In real-world environments, robots need to detect and respond to unexpected objects or barriers to prevent collisions and ensure smooth movement. Successful obstacle avoidance relies on precise sensing, real-time data processing, and effective decision-making algorithms.

- The contributions of this paper are as follows:
- Path planning is another important part that helps robots figure out the best route from where they are to where they want to go.
- This includes improving factors like distance, time, and energy use. It also ensures that the path stays clear of obstacles.

B. Problem Statement

In real-world environments, autonomous robots face challenges when moving from one place to another. They encounter unexpected obstacles and uneven terrain. Traditional manually operated systems take a lot of time. They are also prone to human error and struggle with complex tasks like exploration, surveillance, logistics, and rescue operations. Without smart navigation and obstacle avoidance, a robot's ability to operate independently and adjust to changing surroundings is limited.

To address these challenges, we need to create an intelligent Robot Navigation System. This system should effectively detect obstacles, avoid collisions, and find the best path to a target. It must include sensors to sense the environment, use real-time decision-making algorithms, and apply optimized path-planning methods to ensure smooth, safe, and autonomous robot movement.

C. Research Contributions

- Hybrid Planning: Combines graph-based global planning (A*/D*) with tree-based local planning (RRT, RRT*).
- Efficient Data Structures: Uses compressed graphs and quadtrees to save memory on embedded devices.
- Real-Time Replanning: Uses D* Lite for quick path updates when new obstacles appear.
- Obstacle Prediction: Integrates lightweight models to predict moving obstacles.

II. Literature Review

Autonomous robot navigation has become a key area in modern robotics. It allows robots to move safely and efficiently from one point to another without human control. The main challenges in robot navigation are path planning and obstacle avoidance. Path planning finds the best route from a starting position to a target destination. Obstacle avoidance ensures that the robot can move safely by detecting and avoiding obstacles in its environment. A navigation system works well when these two functions are well integrated and can effectively handle both static and dynamic environments. Over the years, researchers have developed many algorithms and techniques, ranging from traditional methods to intelligent and combined optimization approaches.

Early studies on robot navigation relied on traditional algorithms like Dijkstra's algorithm and the A* search algorithm. These methods divide the environment into grids or graphs and find the shortest path between nodes based on cost values. They are reliable and can guarantee an optimal path in fully known and static environments. However, they struggle in large or dynamic settings due to high computational demands and a lack of real-time adaptability. To overcome these issues, local reactive methods like the Artificial Potential Field (APF) and

Dynamic Window Approach (DWA) were introduced. The APF method treats the goal as an attractive point and obstacles as forces that repel the robot, guiding it toward the target while avoiding collisions. While it is conceptually simple and fast, the APF method often gets trapped in local minima or oscillates in tight spaces. In contrast, the DWA considers the robot's dynamic constraints and looks for the best velocity command within a short timeframe. This makes it better for real-time control, but it still has limitations in complex environments.

III. System Design

The system design helps the robot move independently in a warehouse. It uses sensors to detect obstacles, a controller to make decisions, and motors for movement. The design integrates hardware components and logical control to ensure safe and smooth navigation.

A. Overall Architecture

1. Sensing and Detection:

- The robot uses ultrasonic sensors to sense its surroundings. These sensors emit ultrasonic waves and detect the reflections to measure the distance to an object. If an obstacle is found within a set distance, the sensor sends a signal to the microcontroller. Continuous sensing allows the robot to monitor the environment in real time and avoid collisions.

2. Control and Decision-Making

- The microcontroller, whether it's an Arduino or another type, serves as the brain of the robot. It gets input from the sensors, checks the measured distance against the threshold, and

- 3. Movement and Motor Drive

The robot uses DC geared motors controlled by a motor driver module. The microcontroller sends signals to the motor driver to control the direction. This changes the rotation of the motors, enabling the robot to move forward, backward, or turn left or right. The mechanical chassis holds all the parts and ensures stable movement on the warehouse floor.

B. Synchronization and Prediction

In autonomous robotic navigation, synchronization and prediction are crucial for real-time path planning and obstacle avoidance. Synchronization means coordinating different subsystems of the robot, including sensing, mapping, localization, planning, and control. For a robot to navigate safely and effectively, all these parts must work together and share information consistently. This coordination makes sure that sensor data, motion commands, and environmental maps are updated at the same speed. It cuts down on delays and stops errors from using outdated information. For instance, when a LiDAR sensor detects a new obstacle, the mapping module must update the occupancy grid immediately, and the planning module must recalculate the path. If any module falls behind, the robot may respond too slowly, resulting in collisions or inefficient paths. Thus, synchronization between sensors, processors, and actuators is vital for stable and precise navigation, especially in dynamic environments.

IV. Design and Methodology

A. Sensing and Obstacle Detection

The robot uses ultrasonic sensors to detect objects around it. The sensors emit sound waves and measure how long it takes for the echo to return. This time is transformed into distance, which helps the robot determine how close an obstacle is. If the distance is greater than the safe limit, the robot continues moving forward. If the distance drops below the threshold, the robot recognizes an obstacle in front and prepares to change direction. Continuous sensing keeps the robot aware of its environment in real time.

B. Control and Decision Logic

- A microcontroller, like an Arduino, serves as the central controller for the robot. It receives input from the ultrasonic sensors and compares that with a set threshold. Based on this comparison, the microcontroller decides on movements.

C. Movement Execution

The robot moves using DC geared motors that a motor driver module controls. The microcontroller sends direction signals to the motor driver, which adjusts the motor's rotation for forward, backward, left, or right movement. The motors are mounted on a stable chassis to ensure smooth motion on warehouse floors. The power supply unit, or battery, provides energy to all parts and keeps everything running smoothly during navigation.

V. System Design and Architecture

A. Sensing and Perception Layer

he robot uses ultrasonic sensors to understand its environment. These sensors send ultrasonic waves and measure the time it takes for the reflected signal to return. This helps the robot calculate the distance to nearby objects. The sensing layer constantly monitors the surroundings and provides real-time data to detect obstacles. This allows for safe navigation inside the warehouse without collisions. Optional sensors like IR or wheel encoders can improve measurement accuracy.

B. Control and Decision-Making Layer

- The microcontroller, such as Arduino, acts as the main processing unit. It receives input signals from the sensors and compares them with set threshold values. Then, it decides how the robot should move. If the path ahead is clear, the robot moves straight. If it detects an obstacle, the controller decides whether to turn left, right, or

reverse based on the available free space. This decision-making layer ensures the robot adjusts to changing environments.

.

C. Motion and Actuation Layer

The robot's movement is controlled by DC geared motors, which are managed through a motor driver module (L298N/L293D). The microcontroller sends control signals to the motor driver, which changes the motor speed and direction. This allows the robot to move forward, backward, turn left, or turn right. The chassis provides a stable base for attaching components and keeps the robot balanced while it moves.

D. Power Supply and Integration Layer

A rechargeable battery powers the microcontroller, sensors, and motors. A voltage regulator maintains a stable supply voltage when needed. All hardware components are connected and arranged compactly on the robot chassis. Proper integration ensures reliable operation, minimizes wiring issues, and makes maintenance easier.

VI. Mathematical Model for Real-Time Synchronization

Mathematical Design for Real-Time Synchronization
Real-time synchronization is important for a warehouse robot. It ensures that all components involved in navigation operate based on a common time reference. The robot continuously collects sensor readings, processes them through the controller, and drives the motors to move. For smooth and accurate navigation, these activities must occur in a coordinated and timely manner. If there is a delay in sensor reading or control signal, the robot may make incorrect movement decisions. Therefore, it is essential to maintain a synchronized timing relationship among sensors, the microcontroller, and the actuators.

To achieve real-time synchronization, the system uses a fixed sampling time (Ts). The microcontroller reads sensor data and updates movement decisions at every sampling instance. For example, if the sampling time is set to 50 milliseconds, the controller will repeatedly check obstacle distance and adjust motor direction at that same interval. This ensures consistent data acquisition and reduces the chance of sudden movement errors. The sampling time is chosen based on the robot's response characteristics and the required navigation speed.

Each hardware component has its own internal clock, which may not align perfectly with others. This difference is called clock drift and can lead to timing mismatch. To correct this, the microcontroller serves as the central reference clock, and all sensor readings are adjusted based on this reference timing. By continually updating timestamps and comparing the intervals between control cycles, the system keeps operations synchronized, even when minor delays occur.

Synchronization also involves managing system delays. Delays can happen due to sensor processing time, motor actuation time, or communication overhead. These delays are estimated and adjusted during the control cycle. For instance, if reading a sensor takes slightly longer, the controller accounts for the delay before calculating the next movement command. This handling of delays prevents oscillations and ensures smooth motion.

In summary, real-time synchronization is achieved through three combined methods: (1) using a fixed sampling interval for continuous control updates, (2) maintaining a common reference clock to manage time drift, and (3) compensating for internal delays to ensure accurate motor response. With these methods, the robot maintains stable navigation, timely obstacle detection, and accurate path planning within the warehouse environment.

VII. Implementation

A. Hardware Setup

The hardware setup begins with assembling the robot chassis. Ultrasonic sensors are placed at the front to assist with obstacle detection. An Arduino microcontroller acts as the main control unit and connects to the sensors to collect distance data. DC geared motors are mounted on the wheels and controlled through a motor driver module, such as the L298N. A rechargeable battery powers all components, and proper wiring ensures stable signal transfer. The hardware design is compact to allow smooth navigation in the warehouse.

B. Software Programming

The robot's control logic is developed using embedded programming on the Arduino platform. The program continuously reads distance values from the ultrasonic sensor and compares them to a set threshold. If the measured distance is greater than the threshold, the robot moves forward. When it detects an obstacle, the program decides whether to turn left or right based on the available space. Motor rotation commands are sent through digital output pins, enabling precise direction and speed control. The software is tested and debugged to ensure a stable response in real time.

C. System Testing and Performance Evaluation

After bringing together the hardware and software, the robot is tested in an environment that mimics a warehouse. Various scenarios, such as narrow paths, random obstacles, and different floor surfaces, are used to assess navigation performance. The robot's movement smoothness, accuracy in obstacle detection, and response time are monitored. If needed, threshold values and motor speeds are adjusted to enhance performance. The final implementation shows successful autonomous navigation,

effective obstacle avoidance, and path adjustment in real time.

VIII. Results

The warehouse navigation robot was successfully designed and implemented with the ability to detect obstacles and adjust its path. During testing, the ultrasonic sensor accurately measured the distance between the robot and nearby objects, enabling the microcontroller to make movement decisions in real time. The robot moved forward steadily when the path was clear and made smooth left or right turns when it detected obstacles. The response time for sensing and control stayed consistent throughout the operation.

Performance tests occurred in various indoor settings, including open floor areas and narrow corridors. The robot could detect obstacles at different distances and angles. It showed high average accuracy in obstacle detection, maintaining a safe stopping distance before changing direction. The motors responded well to the direction commands from the control algorithm, resulting in controlled and stable motion without sudden or jerky movements.

Overall, the system met its main goal of autonomous navigation in a warehouse-like setting. The robot could avoid collisions, keep moving continuously, and change its path based on real-time sensor feedback. The results confirm that combining ultrasonic sensing, microcontroller decision-making, and motor control offers an efficient and reliable method for small-scale warehouse navigation applications.

IX. Limitations and Challenges

The warehouse navigation robot efficiently detects obstacles and changes its path. However, it faces some limitations and challenges during use.

1. Limited Sensor Accuracy

The ultrasonic sensor in the system often struggles with accuracy, especially with obstacles that have irregular surfaces or absorb sound waves. In these situations, the sensor readings may differ, which can lead to slow or wrong movement decisions. This can negatively affect the robot's navigation performance in complex warehouse layouts.

2. Restricted Path Planning Capability

The robot follows a basic reactive navigation method, making movement decisions based only on current sensor readings. It does not remember or analyze the warehouse layout. As a result, the robot cannot plan an optimal path or recall locations it has visited before. In large warehouse environments, this can result in inefficient or longer routes.

X. Algorithm Design

The algorithm helps the warehouse robot detect obstacles in real time and change its path as needed. It runs in a continuous loop to ensure smooth navigation. The steps below describe the algorithm for avoiding obstacles and planning paths. Step 1: Initialization

Set a predefined distance threshold (e.g., 15 cm).

Initialize sensors and motor drivers.

Begin the continuous control loop.

Step 2: Obstacle Detection

Read distance values using the front ultrasonic sensor.

If the distance to an obstacle is greater than the threshold, the robot moves forward at a steady speed.

If the distance is less than or equal to the threshold, the robot stops immediately to avoid collision.

Step 3: Direction Selection

When the robot stops, it checks alternative paths by measuring distances to the left and right using side sensors or by rotating slightly to take readings.

Compare the left and right distances.

If the left distance is greater, the robot decides to turn left.

If the right distance is greater, the robot turns right.

Step 4: Path Adjustment

After turning toward the free direction, the robot resumes forward motion.

The control loop restarts to continue sensing and reacting as it moves through the environment.

Step 5: Continuous Operation

The algorithm continuously repeats the sensing and motion steps to maintain obstacle-free navigation until the task or path is complete.

XI. Comparative Framework Study

In warehouse navigation systems, various methods were compared to find the best approach. Manual robots need constant human control, which lowers efficiency. In contrast, autonomous robots work without supervision and boost productivity. Ultrasonic sensors are better than IR sensors because they offer more precise distance measurements and aren't affected by light or surface reflections. Real-time path planning was chosen over pre-programmed routes because warehouse layouts and obstacles can change often. Arduino was selected as the controller because it is inexpensive, easy to program, and provides reliable motor control performance.

Overall, the chosen framework of ultrasonic sensing, real-time obstacle avoidance, and Arduino control provides a simple, cost-effective, and efficient solution for warehouse robot navigation.

XII. Discussion

The warehouse robot can navigate on its own. It continuously watches its surroundings and adjusts its movement in real time. By combining ultrasonic sensing with decision-making from a microcontroller, the robot can spot obstacles and choose a different path without human help. This proves that the obstacle avoidance algorithm works well and can handle changing situations indoors.

During tests, the robot moved smoothly across a floor similar to a warehouse. It responded quickly when obstacles were introduced. The motor control system handled direction changes effectively based on the controller's signals. These results show that the system works as planned for small-scale warehouse navigation.

However, there were some challenges. The robot only uses ultrasonic sensor input and does not create or keep a map of its environment. Because of this, it cannot plan the best routes over long distances and may take longer paths. Battery life also affects performance, particularly during continuous use. Despite these limitations, the system offers a practical and affordable solution for basic warehouse automation.

XIII. Applications and Broader Impact

This robot can help in warehouses by moving small items from one place to another without human help. It supports automated storage and retrieval of goods in inventory management systems. The robot can also work in packing and dispatch areas to transfer products between workstations. It reduces manual labor and the physical effort needed for material handling tasks. In small factories or workshops, it can transport tools or parts between production units. The system can be used in retail back-end storage rooms to organize goods. It is also useful for demonstration and training in robotics and automation labs. Educational institutions can use this robot for hands-on learning in embedded systems.

XIV. Conclusion

The project shows how to design and implement an autonomous warehouse robot. This robot can detect obstacles and adjust its movement in real time. Ultrasonic sensors, along with a microcontroller-based decision algorithm, allow for efficient navigation without human control. Right now, the system works best in small environments and can handle basic obstacle avoidance, but it sets a solid groundwork for future improvements. By adding features like mapping, path optimization, and greater load capacity, the robot could be used in real industrial warehouse settings. Overall, the project presents a practical and affordable solution for warehouse automation and adds to the field of intelligent robotics.

The system can be used in retail back-end storage rooms to organize goods. It's also helpful for demonstration and training in robotics and automation labs. Educational institutions can use this robot for hands-on learning in embedded systems.

XV. References

References

[1] D. Crockford, *JavaScript: The Good Parts*, O'Reilly Media, 2008.

[2] G. Mehta, *Learning WebSockets*, Packt Publishing, 2015.

[3] A. Banks and R. Gupta, *Node.js Design Patterns*, 3rd ed., Packt Publishing, 2021.

[4] M. Cantelon, M. Harter, T. Holowaychuk, and N. Rajlich, *Node.js in Action*, 2nd ed., Manning Publications, 2017.

[5] S. Wagner, "Socket.IO and Real-Time Web Communication," *IEEE Internet Computing*, vol. 23, no. 3, pp. 92–97, 2019.

[6] Z. Zhao, H. Wang, and D. Liu, "Client Prediction and Latency Compensation in Real-Time Multiplayer Systems," *ACM SIGGRAPH Symposium on Interactive 3D Graphics*, 2020.

[7] A. Al-Sherbaz and F. Turner, "Latency Control in Web-Based Multiplayer Games," *IEEE Transactions on Games*, vol. 13, no. 4, pp. 354–365, 2021.

[8] R. Buyya, C. Vecchiola, and S. Thamarai Selvi, *Mastering Cloud Computing: Foundations and Applications Programming*, McGraw-Hill, 2013.

[9] T. Richardson, "Event-Driven Architectures for Real-Time Web Applications," *Journal of Web Engineering*, vol. 17, no. 5–6, pp. 455–472, 2018.

[10] K. G. Shin, "Distributed Synchronization Techniques for Real-Time Systems," *IEEE Transactions on Computers*, vol. 51, no. 3, pp. 265–275, 2002.

[11] J. Smith and M. Brown, "Optimizing WebSocket Communication for Multiplayer Games," *IEEE Access*, vol. 9, pp. 10234–10249, 2021.

[12] Y. Xu, L. Zhang, and H. Zhao, "A Scalable Architecture for Browser-Based Multiplayer Games," *International Journal of Computer Games Technology*, vol. 2022, Article ID 8874502, 2022.

[13] M. Young, *The Technical Writer's Handbook*, University Science Books, 1989.