Computers and Electrical Engineering 124 (2025) 110375

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Hybrid robot navigation: Integrating monocular depth estimation and visual odometry for efficient navigation on low-resource hardware

Ankit Vashisht [a], Geeta Chhabra Gandhi [b], Sumit Kalra [c], Dinesh Kumar Saini [b,*]

[a] Indian Institute of Technology Delhi (IITD), India
[b] Manipal University Jaipur, Jaipur, India
[c] Indian Institute of Technology Jodhpur (IITJ), India

ARTICLE INFO

Keywords:
Simultaneous localization and mapping (SLAM)
Monocular depth estimation (MDE)
Visual odometry (VO)
Performance optimization (PO)
Sustainable computation (SC)

ABSTRACT

Robotic navigation is a complex task requiring accurate localization, environmental perception, path planning, and control of actuators. Traditional navigation systems rely on pre-built maps or map building techniques such as simultaneous localization and mapping (SLAM). However, these approaches unnecessarily map the entire environment, including all objects and obstacles, making them computationally intensive and slow, particularly on resource-constrained devices. While mapless navigation methods address some of these issues they are often too impulse-based, lacking reliance on planning. Recent advances in deep learning have provided solutions to many navigation paradigms. In particular, Monocular Depth Estimation (MDE) enables the use of a single camera for depth estimation, offering a cost-effective alternative to selective mapping. While these approaches effectively address navigation challenges, they still face issues related to scalability and computational efficiency. This paper proposes a novel hybrid approach to robot navigation that combines map-building techniques from classical visual odometry (VO) with maples techniques that uses deep learning-based MDE. The system employs an object detection model to identify target locations and estimate travel distances, while the MiDaS MDE model provides relative depth to detect the nearest obstacle and navigable gaps after image segmentation removes floor and ceiling areas, enhancing the robot's perception of free spaces. Wheel odometry (WO) and VO determine the robot's position and its metric distance from detected nearest obstacle. An instantaneous Grid map is then formed with robot's position, navigable gap, nearest obstacle and the goal location. Path planning is conducted using a modified A-star (A*) algorithm, followed by path execution with a Proportional Integral Derivative (PID) controller. The system's performance is evaluated at both the modular level and the final system level using various metrics, such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and inference time for depth estimation, and navigation success rate across different robot speeds for final navigation performance. Additionally, a Friedman statistical test is conducted to validate the results. Experimental results show that the proposed approach reduces memory and computational demands, enabling real-world navigation on low-resource hardware. To our knowledge, this is the first integration of MDE-based mapless navigation with VO-based map-building, presenting a novel direction for research.

* Corresponding author.
E-mail addresses: Ankit.vashisht@ee.iitd.ac.in (A. Vashisht), geeta.gandhi@jaipur.manipal.edu (G.C. Gandhi), sumitk@iitj.ac.in (S. Kalra), dineshkumar.saini@jaipur.manipal.edu (D.K. Saini).

https://doi.org/10.1016/j.compeleceng.2025.110375
Received 5 November 2024; Received in revised form 10 April 2025; Accepted 15 April 2025
Available online 21 April 2025

1. Introduction

Robotic navigation is a multifaceted task that involves localizing a robot's position, accurately perceiving its surroundings, planning trajectories, and controlling actuators. Navigation systems are generally categorized into three groups: map-based, map-building, and map-less systems [1]. Map-based navigation systems rely on geometric models or topological maps of the environment to navigate [2], while map-building systems can create maps during runtime and localize the robot relative to the created map [3]. Map-less systems on the other hand, operate without an environmental map and generate motion commands based on just their sensor data.

Over the past decades, classical geometry-based visual odometry has shown impressive progress, demonstrating superior performance in both feature-based methods [4] and direct methods [5]. These advancements have pushed classical visual odometry toward real-world applications [6]. A classic pipeline for visual odometry typically includes camera calibration, feature detection, feature matching (or tracking), outlier rejection (e.g., RANSAC), motion estimation, scale estimation, and local optimization (Bundle Adjustment), forming a well-recognized framework. However, the robustness of these methods often fails to meet the high-reliability requirements in challenging environments, such as those with changing illumination and dynamic conditions [7]. Additionally, it is also often not resource-friendly, particularly in terms of computational and memory demands required by backend bundle adjustment.

Many state-of-the-art visual SLAM (VSLAM) systems, such as ORB-SLAM, achieve real-time performance on powerful PCs or laptops but may be hindered by the sequencing or timing of the pipeline [8]. Some VSLAM systems struggle with real-time processing under computational or time constraints [9], and others that prioritize efficiency may suffer performance losses [10]. These issues are exacerbated when attempting to replicate the same performance on less powerful devices [11]. While Bundle Adjustment (BA) remains the preferred back-end solution for high accuracy and robustness, its computational expense, with its large number of states and iterative optimization process, presents a significant bottleneck [12].

Map-less systems, which operate without an environmental map, generate motion commands based on sensor data. This approach eliminates the need for storing and maintaining maps, resulting in lower computational complexity, memory requirement and faster response times. Map-less systems are particularly useful in situations where building and maintaining a map is impractical, such as when a robot operates for a short period or navigates an unknown, dynamic, and complex environment [13]. However, mapless systems are often overly simplistic and impulse-based, lacking robust planning capabilities. This limitation becomes problematic in dynamic and complex environments where robots must navigate longer paths or handle intricate scenarios.

In Recent times, advancements in machine learning, particularly deep learning, have inspired researchers to explore data-driven approaches as alternative solutions. Unlike conventional model-based approaches that rely on explicit algorithms, learning-based methods use deep neural networks to extract features and build implicit models. These networks, trained on large datasets, can generate poses and describe scenes even in challenging environments with high dynamics and poor lighting conditions [14]. Deep learning-based localization and mapping methods generally exhibit better robustness and accuracy compared to their traditional counterparts.

Related approaches in SLAM can be categorized into four main types: incremental motion estimation (visual odometry), global re-localization, mapping, and loop-closing and SLAM back-ends [14]. Deep learning serves as a universal approximator for certain functions of SLAM or individual localization/mapping algorithms. For instance, visual odometry can be achieved through end-to-end deep neural networks that approximate the function from images to pose [15]. These learned models can inherently incorporate resilience to featureless areas, dynamic lighting conditions, and motion blur, which are typically challenging to model.

Monocular Depth Estimation has emerged as a significant advancement within this domain, enabling the use of a single camera to estimate depth. By leveraging deep learning, MDE and other such Deep Neural Networks provide an effective solution to the scale-ambiguity problem often encountered in monocular SLAM, offering depth estimates with an absolute scale [16]. This capability not only enhances SLAM performance but also reduces dependency on expensive sensors like LiDAR. Deep learning is also applied to solve the association problem in SLAM, such as relocalization, semantic mapping, and loop-closure detection [17]. Additionally, deep learning can automatically discover features relevant to specific tasks, such as features suitable for bundle adjustment in SLAM [18].

Despite these advancements, deep learning techniques face challenges, including reliance on large datasets with accurate labels, lack of interpretability, and higher computational costs [14]. The real-world deployment of deep learning models in robotic navigation systems must consider computational and energy constraints on resource-limited devices, such as miniature robots or AR/VR devices [14]. Scalability remains an open problem, as deep learning models are often evaluated in limited scenarios and may not generalize well to different environments [14]. Moreover, achieving a balance between model accuracy and efficiency is crucial for practical applications on resource-constrained platforms [14].

This paper introduces a novel hybrid navigation system that represents a significant innovation by seamlessly integrating complementary techniques from map-based and mapless methods. Unlike traditional approaches, this work prioritizes the nearest obstacle and navigable gaps using a computationally efficient pipeline, making it ideal for low-resource hardware. The hybrid system incorporates classical visual odometry with deep learning-based monocular depth estimation, image segmentation, and object detection to achieve both robustness and efficiency. Key innovations include:

A unique combination of MDE-based mapless navigation with VO-based map building, balancing real-time perception and long-term planning.

1. A modified A* algorithm tailored to prioritize free space and enhance path planning efficiency.
2. A focus on minimizing computational and memory requirements, enabling deployment on devices with limited resources.

Designed specifically for low-resource platforms, the system employs an object detection model to identify target locations and estimate travel distances, while the MiDaS MDE TensorFlow Lite model generates depth information to locate obstacles and gaps after segmenting out floor and ceiling areas. VO determines the robot's position relative to obstacles, forming an instantaneous grid map of navigable space. Path planning utilizes a modified A* algorithm, prioritizing nodes with maximum free space, and path execution employs a Proportional Integral Derivative (PID) controller. The instantaneous map includes only the location of the nearest obstacle as a graph node, gap node, goal node, and the robot's pose node, significantly reducing memory requirements. The combination of visual odometry and the MiDaS V2.0 model ensures both methods are fast, requiring less computational power and time.

By integrating these complementary techniques, the proposed hybrid system addresses the limitations of existing methods, ensuring robust navigation in dynamic and unknown environments for low-resource devices. This work introduces a scalable and efficient solution for robotic navigation on low-resource hardware, offering significant contributions to both map-based and mapless navigation methodologies.

Each component in this pipeline serves a distinct and essential function, making ablation infeasible. Visual Odometry (VO) provides metric depth estimation, which is necessary for understanding the absolute distances of obstacles. Monocular Depth Estimation (MDE), on the other hand, detects the nearest obstacle but lacks absolute depth without VO. Object detection is required for target identification, and the modified A-Star algorithm ensures effective path planning. Since these components are interdependent, disabling any one of them would render the system incomplete rather than providing meaningful performance insights. Instead of isolating components, we demonstrate the effectiveness of the complete pipeline through experimental validation, showcasing its robustness in dynamic environments.

By integrating these complementary techniques, the proposed hybrid system addresses the limitations of existing methods, ensuring robust navigation in dynamic and unknown environments for low-resource devices. This work introduces a scalable and efficient solution for robotic navigation on low-resource hardware, offering significant contributions to both map-based and map-less navigation methodologies.

2. Related work

In the field of robotics, SLAM (Simultaneous Localization and Mapping) has been a cornerstone for enabling autonomous

Table 1
Related work on map building and mapless navigation.

Category	Study	Key contributions	Challenges/Limitations
Map Building	Abrate et al. [19]	Investigated EKF-SLAM using an infrared-only sensor on a mobile mini robot.	Sparse and noisy sensor data affected SLAM accuracy and reliability.
Hardware Modification	Bonato et al. [20]	Implemented EKF-SLAM on FPGA, optimizing time-consuming components with custom hardware.	Performance was lower compared to desktop implementations.
Hardware Modification	Yap [21]	Used low-cost sonar sensors with particle filter for SLAM, aiming to make SLAM more accessible to low-resource platforms.	Limited performance in certain scenarios due to sensor quality.
Algorithm Modification	Eade [22]	Introduced techniques to reduce SLAM graph complexity, such as variable elimination and constraint pruning.	Focused on theoretical complexity reduction without empirical data on real-time performance.
Hardware-Software Co-design	Vincke et al. [23]	Developed co-designed hardware architecture for SLAM optimized for low-resource environments.	No direct mention of limitations, but hardware performance remains crucial.
Hardware-Software Co-design	Magnenat [24]	Utilized lightweight SLAM software paired with a slim rotating distance scanner for real-time SLAM on a mini robot.	Performance in real-time scenarios may still be restricted by hardware limitations.
Both Hardware & Algorithm Optimization	Buonocore et al. [25]	Used FPGAs and sensor fusion techniques to achieve robust SLAM in low-resource environments.	Optimization trade-offs in sensor fusion and FPGA limitations.
Mapless Navigation	Study [28]	Mapless navigation using landmark images and CNNs for visual servoing, enabling movement guidance.	Challenges with dynamic environments and retraining the network for new environments, limiting scalability.
Mapless Navigation	ASGDLR Model [29]	Utilized ultrasonic sensors for obstacle detection and infrared sensors for velocity adjustments in dense environments.	Fixed sensor positions may reduce obstacle detection accuracy in dynamic environments.
Mapless Navigation	Tsai et al. [30]	Employed LiDAR data and deep learning for mapless navigation, achieving success in predicting movements.	Requires significant computational resources for real-time processing.
Mapless Navigation	Nguyen et al. [31]	NMFNet used RGB-D cameras and LiDAR to predict steering angles for autonomous navigation.	Potential challenges with real-time sensor fusion in low-resource environments.
Mapless Navigation	Xiong et al. [32]	Proposed Visual Servoing (VS) method using RGB-D data for precise navigation.	Relies on depth map accuracy and may face challenges in more dynamic environments.
Mapless Navigation	Li et al. [33]	Surveyed advancements in vision-based navigation, highlighting mapless approaches emphasizing real-time performance.	Limited use of Monocular Depth Estimation in vision-based navigation.
Monocular Depth Estimation	Study [33]	Leveraged MDE for gap identification and obstacle avoidance, a lightweight alternative to stereo or RGB-D data.	Integration of MDE in visual servoing is still underexplored, and performance in real-time navigation needs further validation.

navigation, particularly in unknown environments. Traditional SLAM systems, however, are resource-intensive, often requiring significant computational power and high-quality sensors, making them less feasible for low-resource devices such as small robots or embedded systems. This section reviews various research efforts aimed at adapting SLAM for low-resource hardware, focusing on how these systems have been implemented and optimized to work within constrained environments. Additionally, the discussion will cover Mapless navigation techniques that provide alternatives to low resource SLAM by relying on different sensor inputs and learning-based approaches. Monocular Depth Estimation models, which allow depth estimation from single RGB images, have also been explored in this context since these models are very promising as a replacement to high cost and resource intensive sensors used in robot navigation. Relevant MDE models such as MiDaS and Depth Anything, which are further analysed in this paper, provide lightweight alternatives to traditional methods and have been discussed in detail (Table 1).

2.1. Map building

In the pursuit of adapting SLAM for low-resource hardware, various strategies have been explored, often involving modifications to algorithms, hardware configurations, or both. For example, when it comes to modifying hardware, Abrate et al. [19] investigated the use of an infrared-only sensor for EKF-SLAM implementation on a mobile mini robot. The study highlighted the limitations of using sparse and noisy data from such sensors, which severely affected SLAM's accuracy and reliability.

Similarly, Bonato et al. [20] targeted embedded systems by implementing EKF-SLAM on an FPGA device. Rather than redesigning the algorithm, the authors optimized its execution by offloading the most time-consuming components to custom hardware, though performance remained significantly lower compared to standard desktop implementations.

Yap [21] explored the use of low-cost sonar sensors for SLAM, which aligned with the goal of making SLAM more accessible to low-resource platforms. The research employed a particle filter and demonstrated that even with low-quality sensors, SLAM could achieve reasonably accurate tracking and mapping, albeit with limitations in certain scenarios.

Now coming to algorithm modification, Eade [22] presented an approach to manage the computational demands of SLAM by introducing techniques to bound the complexity of SLAM graphs. By implementing variable elimination and constraint pruning, the study aimed to maintain SLAM's accuracy while reducing the computational burden. However, the study focused on the theoretical reduction in graph complexity without providing empirical data on the system's real-time performance.

Approaches with both hardware and software modification include Vincke et al. [23], who developed a co-designed hardware architecture for SLAM that integrated various processors and co-processors, optimizing the system for low-resource environments. Magnenat [24] also focused on hardware-software co-design, utilizing a lightweight SLAM software paired with a slim rotating distance scanner to achieve real-time SLAM on a miniature robot.

In more specialized approaches, Buonocore et al. [25] utilized FPGAs and sensor fusion techniques to achieve robust SLAM in environments with limited resources. These studies highlighted the importance of optimizing both the algorithm and hardware to achieve feasible SLAM implementations on low-power devices.

Other latest notable work includes Yi and Guan [26], who explored the use of Deep Reinforcement Learning (DRL) for autonomous robot navigation. Their study evaluates different RL-based approaches, such as Deep Deterministic Policy Gradient (DDPG), Twin Delayed Deep Deterministic Policy Gradient (TD3), and an optimized DRL model, in obstacle-rich environments. The success rate metric, a key evaluation measure in our work, was used to compare navigation performance under varying obstacle densities. Notably, their proposed DRL algorithm achieved a 77 % success rate when navigating through 200 obstacles, aligning closely with the success rates observed in our experiments. This work highlights the importance of adaptive learning-based approaches to enhance navigation efficiency in complex settings. Additionally, Gervet et al. [27] conducted a large-scale empirical study comparing classical, modular, and end-to-end learning-based navigation approaches in real-world environments. Their findings indicate that modular learning transfers well from simulation to real-world settings, achieving a 90 % success rate, whereas end-to-end learning struggles significantly, dropping from 77 % in simulation to 23 % in real-world deployment. This highlights the challenges of direct end-to-end policy learning in complex, uncontrolled environments.

However, while Yi and Guan's work primarily focuses on simulation-based experiments with high-performance computing resources, our work is geared toward real-world deployment on low-resource hardware. By implementing SLAM techniques on a Raspberry Pi 4, we demonstrate the feasibility of efficient and adaptive robot navigation in dynamic environments using computationally lightweight techniques. This makes our approach more applicable for resource-constrained robotic platforms operating in practical scenarios.

2.2. Mapless navigation

While SLAM provides a comprehensive solution for environment mapping and localization, its high computational demands have led to the exploration of alternative navigation strategies, particularly for low-resource platforms. Mapless navigation, which often relies on machine learning and sensor fusion, presents a promising alternative by bypassing the need for explicit map building.

One such approach is demonstrated in [28], where a mapless navigation system utilized landmark images to guide a robot to its target. By employing convolutional neural networks (CNNs) for visual servoing, the system calculated the necessary movements based on the difference between current and target images. However, this approach faced challenges related to dynamic environments and the necessity to retrain the network for new environments, limiting its scalability.

In another notable approach, the ASGDLR model proposed in [29] leveraged ultrasonic sensors for obstacle detection and infrared sensors for wheel velocity adjustments to guide the robot in dense environments. By predicting necessary wheel adjustments and

turning directions, the model successfully enabled real-time collision avoidance and path planning in cluttered spaces. However, a key limitation of this approach lies in its reliance on fixed ultrasonic sensor positions, which may lead to reduced obstacle detection accuracy in dynamic environments where obstacles move in unpredictable patterns.

Tsai et al. [30] explored the use of LiDAR data for mapless navigation. These studies employed deep learning techniques to predict the robot's movements based on the input data, achieving reasonable success rates. Despite the improvements, these methods still require significant computational resources for real-time processing.

Nguyen et al. [31] introduced NMFNet, a mapless navigation solution that integrated RGB-D cameras and LiDAR to predict steering angles for autonomous navigation. This approach demonstrated the potential of fusing multiple sensor inputs to enhance navigation accuracy without relying on pre-built maps.

Xiong et al. [32] proposed another Visual Servoing (VS) method that used RGB-D data to drive a robot by minimizing the depth map error between initial and target positions. This method illustrated how depth information could be leveraged for precise navigation in environments where traditional SLAM might struggle.

Recent surveys, such as those by Li et al. [33] have reviewed advancements in vision-based robot navigation and visual servoing, highlighting the growing trend towards mapless approaches that emphasize real-time performance and adaptability to new environments.

A noteworthy gap in existing research is the limited use of Monocular Depth Estimation in visual servoing systems. While many studies utilize stereo vision or RGB-D data, the integration of MDE offers a lightweight alternative that can significantly reduce the hardware requirements. The system proposed in [33] exemplifies this by using MDE for gap identification and obstacle avoidance, presenting a novel contribution to mapless navigation by leveraging depth estimation from monocular images.

2.3. Monocular depth estimation

Depth estimation is a crucial step toward inferring scene geometry from 2D images. A vast body of research exists on depth estimation, particularly using stereo images. Most stereo-based methods rely on point-matching between left and right images, typically using hand-crafted or learned features. However, monocular depth estimation presents a more challenging problem, where the goal is to predict the depth value of each pixel from a single RGB image. Several models have been developed to address this challenge, with MiDaS, ZoeDepth, and Depth Anything being among the most prominent.

MiDaS (Monocular Depth Estimation using Scale-Invariant Learning) originally released in 2019, became a standard model for monocular depth estimation due to its robustness and widespread applicability [34]. Over the years, several versions of MiDaS have significantly improved its performance. MiDaS v2.1 (2020) was 10 % more accurate than v2.0, and MiDaS v3.0 (2021) achieved 20 % greater accuracy compared to v2.1. The most recent version, MiDaS v3.1 (2022), further improved accuracy by 28 % over v3.0. While MiDaS is no longer considered state-of-the-art, it remains a reliable model due to its backbone architecture, which is still employed in more modern approaches like ZoeDepth. One limitation of MiDaS is that its depth estimation is relative rather than absolute; it excels at predicting which pixels are closer or farther but struggles to provide consistent metric depth values. This limitation becomes apparent when attempting to project depth into 3D space.

Built on top of MiDaS, ZoeDepth addresses MiDaS' limitation by providing depth estimation in metric units [35]. This advancement is especially noticeable when visualizing depth in 3D space, where ZoeDepth produces more realistic depth representations. However, ZoeDepth still suffers from issues such as loss of detail, where sharp corners and textures may blur, and objects like people appear less defined (e.g., as gingerbread-like Fig's). Some recent models like Patch Fusion and Marigold have attempted to address these detail-related challenges.

Depth Anything represents a new frontier in monocular depth estimation, leveraging a vast dataset of 1.5 million labeled images and over 62 million unlabeled images [36]. It offers zero-shot relative depth estimation, surpassing MiDaS v3.1, and zero-shot metric depth estimation, outperforming ZoeDepth. One of its key strengths is its ability to generalize well across different domains and datasets. Depth Anything performs optimally in fine-tuned and zero-shot evaluations on popular benchmarks like NYU Depth V2 and KITTI, setting a new standard for MDE models in terms of both accuracy and versatility. However, like all models, there are areas for improvement, particularly in edge detail and sharpness, which can degrade under certain conditions.

For this work, we selected MiDaS v2.0 as our monocular depth estimation model due to its balance between fast inferencing and robustness. MiDaS is particularly well-suited for edge devices where real-time performance is critical, especially in dynamic environments. Although MiDaS provides relative depth estimations—meaning the depth of one object relative to another, rather than absolute metric depth—this limitation can be addressed by incorporating visual odometry to obtain the relative distance of the robot to objects in the scene. In this setup, MiDaS provides the relative positioning between objects, while VO offers the relative distance between the robot and those objects. Since we are primarily interested in identifying the closest object and the available gap, this combined approach—using MiDaS for fast depth inference and VO for relative depth correction—proves to be highly efficient for deployment on low-resource, edge devices.

While metric depth models like Depth Anything offer improved accuracy in metric units and remove the need for VO, they are too slow to be practical in real-time dynamic environments. This is proved in Experimentation section, where we compare the inference times and memory usage of different MDE models.

3. Methodology

This section outlines the step-by-step process involved in our Hybrd robot navigation system, from goal detection to real-time

navigation. An analogy of this method, illustrating the navigation strategy through a real-world driving scenario, is provided later for better conceptual understanding. The approach integrates deep learning for object detection, visual servoing for orientation correction, a modified A-star algorithm for path planning and PID control for navigation, while leveraging MiDaS depth estimation for obstacle detection, visual odometry for estimating the metric depth of nearest object relative to robot's current state, and wheel odometry for precise localization and motion control. The complete pipeline is shown in Fig. 1

3.1. The pipeline

The complete pipeline consists of total 9 Steps:

Step 1: Goal Location Detection

The first step involves detecting the goal location using a deep learning-based object detection model. For instance, if the robot is required to approach a person, the model detects the person's face, as shown in Fig. 2. The bounding box's dimensions and its angle relative to the robot's camera centre are used to calculate the goal state in terms of real-world coordinates (e.g., meters) using pre-calibrated parameters. The deep learning model we used in this case is YOLO v7, which we trained on our custom head detection dataset, created using Roboflow for accurate model training and evaluation. This approach aligns with the method described in [37], where YOLO v7 was utilized for automatic number plate detection using a Roboflow-annotated dataset.

Step 2: Correcting the Robot's Initial Orientation (optional)

After identifying the goal, the robot adjusts its orientation to face the goal directly. This is achieved using a custom visual servoing algorithm given in Algorithm 1 that ensures that the detected goal object is aligned with the centre of the robot's camera view, thereby orienting the robot in a straight line towards the goal.

Fig. 2a shows the detected face in the right portion of the camera frame, indicating that the goal is not aligned with the robot's current orientation. The robot calculates the error between the centre of the bounding box (goal) and the centre of the camera frame and applies basic yaw correction to reduce this error to zero. The robot will keep turning until the centre of the captured image, in real-time, aligns with the goal's centre—i.e., the centre of the bounding box obtained in the previous step, shown in Fig. 2b thus orienting the robot in a straight line to that of its goal.

Step 3: Initial Path Generation

The robot assumes its initial position as (0,0) with an orientation of $\pi/2$ radians. Using the goal state obtained in Step 1, an optimal path is generated using a modified A* algorithm. During the pathfinding process, obstacle nodes and gap nodes are also considered to ensure safe navigation around obstacles; these will be explained in detail in later steps. The modifications to the A* algorithm include:

1. Adjusted Heuristic for Horizontal Movements:

The cost for horizontal movements is dynamically adjusted based on the heuristic value, ensuring that mostly robot prioritizes a vertical i.e. straight ahead motion and diagonal or horizontal motion are given preference as robot moves closer to the goal state. This ensures that the robot doesn't skid towards the boundaries of the nearest obstacle.

2. Zero Cost for Specific Nodes:

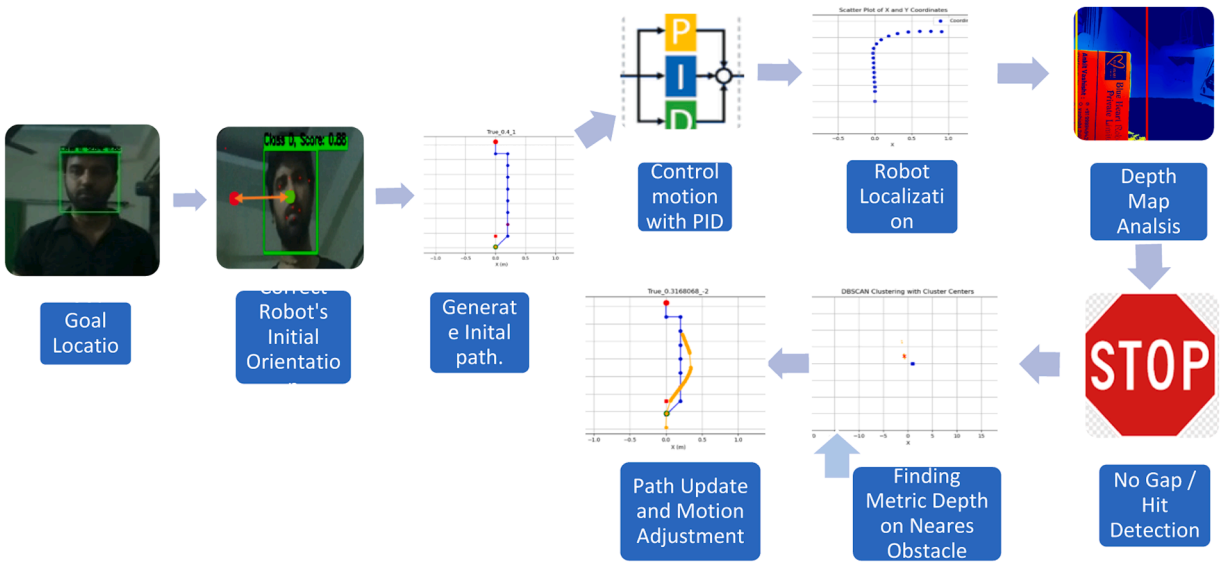


Fig. 1. Pipeline of proposed methodology.

Fig. 2. Visual servoing for goal alignment, (a) initial state, (b) final state.

Algorithm 1

Input: *Real-time camera feed, bounding box centre (C_x, C_y), camera frame centre (F_x, F_y), error threshold (ε).*
Output: *Robot oriented toward the goal.*
Initialize acceptable error threshold ε.
while $|F\_x - C\_x| \leq \varepsilon$:
  Capture a frame from the camera.
  Detect the goal in the frame and get the bounding box center (C_x, C_y).
  Calculate the horizontal error: $E\_x = F\_x - C\_x$.
  if $|E\_x| \leq \varepsilon$:
    Stop turning; the goal is aligned.
  else if $E\_x > 0$:
    Rotate the robot left to reduce the error.
  else:
    Rotate the robot right to reduce the error.
end while
Output: *Robot is now aligned with the goal.*

Algorithm 2
Standard A* Algorithm.

Input: *Graph, Start Node (s), Goal Node (g)*
Output: *Optimal Path from Start to Goal*
Initialize an empty open list (nodes to be evaluated) and a closed list (evaluated nodes). Add the start node to the open list.
While *the open list is not empty:*
  Select the node with the lowest f value (where $f = g + h$*, g is the cost from start to the current node, and h is the heuristic estimate to the goal).*
  If *the selected node is the goal node, construct the path by backtracking and return the path.*
  else*, move the selected node to the closed list.*
  For *each neighbor of the selected node:*
    If *the neighbor is in the closed list, skip it.*
    Calculate the tentative g value for the neighbor.
    If *the neighbor is not in the open list or the new `g` value is lower than its current `g` value.*
      Update the neighbor's g and f values.
      Set the current node as the predecessor of the neighbor.
      Add the neighbor to the open list if it is not already there.
If *the open list becomes empty without finding the goal, return failure (no path found).*

A zero cost is assigned to the 'gap node', ensuring this node is included in the path without penalizing the overall cost. These are the nodes with maximum gap or free space.

The Standard A* algorithm is given in Algorithm 2 and the modified A* Algorithm is given in Algorithm 3. The major differences are in point 3 and 8 of algorithms.

Fig. 3a demonstrates the path obtained using the Non modified A* algorithm, while Fig. 3b demonstrates the path obtained using the modified A* algorithm. As can be observed, the robot's path consists mainly of vertical movements, with horizontal movements only prioritized near the goal.

Additionally, the modified algorithm successfully traverses the gap node (free space), which was not covered in the path generated by the standard A* algorithm shown in the left plot. This adjustment enables the robot to traverse nodes with maximum free space while steadily progressing toward the global goal. Similar to the gradient descent algorithm, where the path adjusts to local minima on the way to the global minimum, this approach emphasizes a direct, efficient path instead of a zigzag pattern, optimizing navigation efficiency. This approach contrasts with the method presented in [38], which improves the Breadth-First Search (BFS) algorithm for navigating dynamic environments by recalculating paths in real-time based on environmental changes. While the BFS-based method efficiently avoids dynamic obstacles, it relies on RFID-based localization, which may be impractical in real-world scenarios. In contrast, our modified A* algorithm incorporates Monocular Depth Estimation for obstacle detection and prioritizes nodes with maximum available depth for more efficient navigation.

Step 4: Motion Control with PID

The robot's movement is controlled using a PID controller. The error between the robot's current state's x coordinate and the target state's x coordinate (next position along the path) is computed based on Euclidean distance. This error is fed to the PID controller, which adjusts the robot's wheel velocities to minimize the error, ensuring the robot follows the planned path.

The tuned PID parameters for the specific robot platform used in the experiments were set as follows:

- Kp = 30 (Proportional gain) – Controls the immediate reaction to the error.
- Ki = 5 (Integral gain) – Helps correct accumulated past errors.
- Kd = 20 (Derivative gain) – Predicts and counteracts future error trends.

These values were selected experimentally to balance stability and responsiveness, preventing oscillations or excessive lag in movement corrections.

Step 5: Robot Localization

The robot's current location is updated through either wheel odometry or visual odometry, with our experiments showing that wheel odometry provides more reliable results in terms of accuracy and consistency. Wheel odometry uses the rotation of the robot's wheels to estimate its position relative to its starting point, continuously updating the robot's location as it moves.

Wheel Odometry

The Wheeled Mobile Robot (WMR) used for experimentation employs a differential drive mechanism. This is a simple and commonly used drive system, particularly for smaller mobile robots. Typically, these robots feature one or more caster wheels to support and prevent tilting. Both primary wheels share a common axis, with each wheel's velocity controlled by a separate motor. the input (control) variables are the velocity of the right wheel V_r and the velocity of the left wheel V_l. These velocities can be calculated by obtaining encoder counts from both motor's encoders:

$$V_{r/l} = \frac{(counts * 2\pi R)}{(T * cpr)} \tag{1}$$

Where $V_{r/l}$ is the linear velocity of the left or right wheel in meters per second, *counts* represent the number of encoder ticks within a fixed time interval *T, R* is the wheel's radius in meters, and *cpr* is the number of counts per wheel revolution. Then, the robot's linear

Algorithm 3
Customized A* Algorithm.

Input: *Graph, Start Node (s), Goal Node (g)*
Output: *Optimized Path from Start to Goal*
Initialize an empty open list (nodes to be evaluated) and a closed list (evaluated nodes). Add the start node to the open list.
1. ***While*** *the open list is not empty:*
2. *Select the node with the lowest f value ($f = g + h$). The heuristic h value is adjusted for horizontal movements*
   - Horizontal costs dynamically increase as the robot nears the goal, prioritizing vertical (straight-ahead) movement.
3. ***If*** *the selected node is the goal node, construct the path by backtracking and return the path.*
4. ***else,*** *move the selected node to the closed list.*
5. ***For*** *each neighbor of the selected node:*
6. ***If*** *the neighbor is in the closed list, skip it.*
7. *Calculate the tentative `g` value for the neighbor.*
   - Assign $g = 0$ for any "gap node" (nodes with maximum free space).
8. ***If*** *the neighbor is not in the open list or the new g value is lower than its current g value:*
9. *Update the neighbor's g and f values.*
10. *Set the current node as the predecessor of the neighbor.*
11. *Add the neighbor to the open list if it is not already there.*
12. ***If*** *the open list becomes empty without finding the goal, return failure (no path found).*

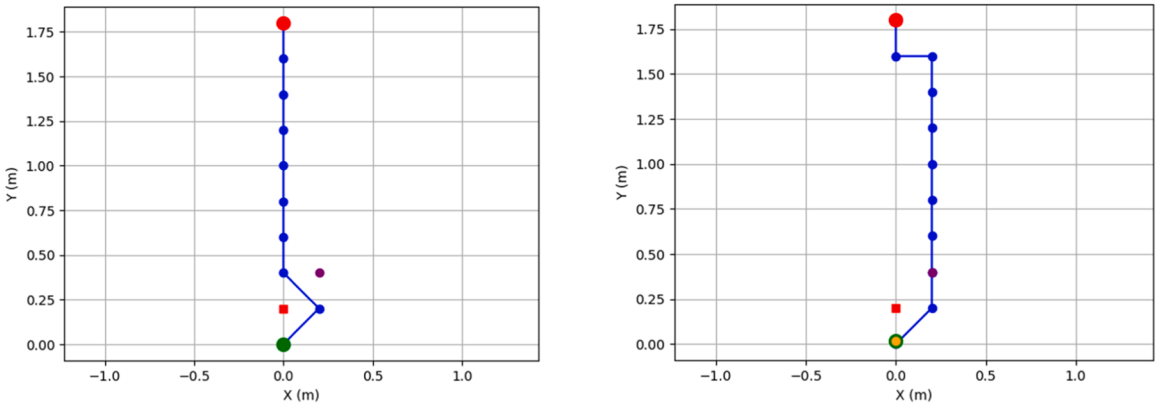


Fig. 3. Path Comparison – Standard A* vs. Modified A*.
(a) Path with Original A* Algorithm, (b) Path with Modified A* Alogirithm.

velocity V and angular velocity w in its local frame of reference are:

$$V = \frac{(V_r + V_l)}{2} \tag{2}$$

$$w = \frac{(V_r - V_l)}{2} \tag{3}$$

Finally the global coordinates for the robot's position (x, y) and heading can be derived as follows:

$$x = x + \Delta x = x + V\Delta t \cos\theta \tag{4}$$

$$y = y + \Delta y = y + V\Delta t \sin\theta \tag{5}$$

$$\theta = \theta + \Delta\theta = \theta + w\Delta t \tag{6}$$

Where:

x and y are the robot's global coordinates,
θ is the robot's heading angle,
V is the linear velocity,
w is the angular velocity,
Δt is the time step.

Visual Odometry

Visual odometry is a technique used in robotics and computer vision to estimate a camera's motion relative to its environment by analysing sequential images. Monocular visual odometry (MVO) and stereo visual odometry (SVO) are two variants that use different cameras to estimate motion. MVO utilizes a single camera to capture successive images, estimating motion by detecting changes in visual features between them. SVO, on the other hand, uses two cameras to capture images from distinct viewpoints and calculates motion by examining disparities between corresponding features in stereo image pairs. SVO provides more accurate estimates as it incorporates depth information and mitigates scale drift but requires more computational power and hardware. In this work, MVO is used due to its lower resource requirements. The steps in Monocular Visual Odometry are:

- **Finding 3D point cloud with 2D-2D feature matching:** The initial step is to match 2D features from the current image to those in the previous image. Using these matches, a set of 3D points is triangulated, forming the 3D point cloud of the scene. To ensure reliable depth estimation in this robotic system, a threshold was applied, If fewer than 200 keypoints were detected in the region near the closest obstacle, VO was skipped to avoid unreliable estimations. This constraint ensures that VO is only performed when sufficient feature matches are available, improving robustness in low-texture or low-light environments.
- **Estimating the camera's motion:** After obtaining the 3D point cloud, the PnP algorithm estimates the camera's motion relative to the previous frame. The PnP algorithm uses the 3D points and their 2D feature positions in the current image to compute the camera's translation and rotation.
- **Refining the camera motion:** Errors due to noise or outliers in the 2D-3D correspondences may be present, so bundle adjustment is used to refine both the camera motion and the 3D point cloud, minimizing reprojection errors.

- **Updating the 3D point cloud:** Once refined, the 3D point cloud is updated by projecting the 3D points onto the current image using the new camera pose. The current image's 2D features are then matched to the projected 3D points, updating the 3D point cloud.
- **Repeating the process:** These steps are repeated for each successive image pair, continuously estimating the camera's motion and updating the 3D point cloud. This trajectory is crucial for localization, mapping, and navigation in robotics and computer vision applications.

Step 6: Depth Map Analysis for Nearest Obstacle and Gap Detection

In this step, the robot uses the MiDaS Monocular Depth Estimation model to understand its environment by analysing the depth information from the camera feed. The depth map shown in Fig. 4 is crucial for detecting obstacles and identifying navigable gaps that guide the robot's movement. The overall Process could be divided in three steps:

(1) **Depth Estimation Using MiDaS Model**

The MiDaS MDE model estimates inverse relative depth from a single camera frame. The output is shown in Fig. 4b .The depth at any pixel $d(x, y)$ is inversely proportional to the MiDaS model's output $z(x, y)$. This relationship can be described as:

$$d(x,y) \propto \frac{1}{z(x,y)} \tag{7}$$

where $z(x, y)$ is the depth value output by the MiDaS model, and $d(x, y)$ is the real-world relative depth. This inverse relationship means that smaller values of $z(x, y)$ indicate further objects, while larger values indicate closer obstacles. as demonstrated in Fig. 4, MiDaS provides a decent approximation of the environment's depth while maintaining lower computational costs compared to other state-of-the-art (SOTA) models.

(2) **Creating a Floor Mask**

K-means clustering with two clusters is applied to the original RGB image (resolution of 640 × 480) to distinguish the floor region from the rest of the scene. This segmentation is based on the colour features of the image. Typically, the floor regions exhibit distinct colour patterns compared to the objects or obstacles placed on it. Therefore, one cluster corresponds to the floor, while the other represents the non-floor regions (e.g., obstacles or objects above the floor). The segmentation process is performed in real-time during robot operation, where the K-means algorithm dynamically clusters the pixels of the image. After segmentation, a binary mask is applied to filter out the floor region from the depth map, allowing the focus to shift solely to potential obstacles. The resulting depth map, with the floor values removed, is shown in Fig. 4c.

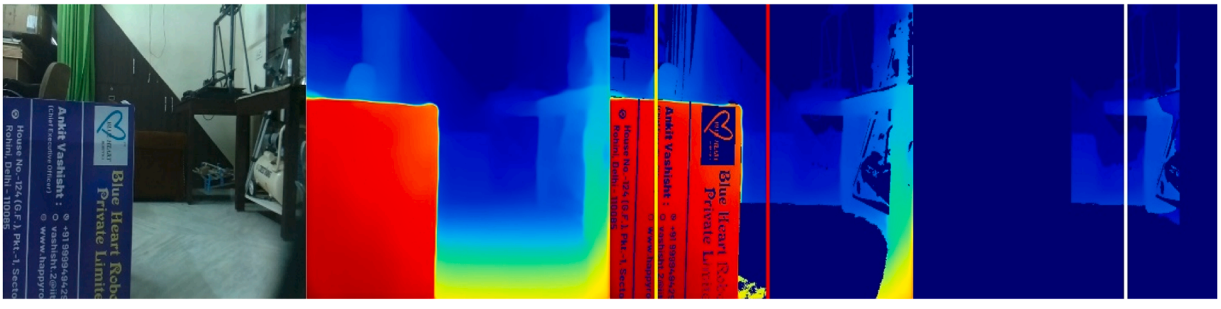
This real-time application of K-means clustering on the robot's operational data is efficient due to the relatively low-resolution images used (640 × 480) and the simplicity of the K-means algorithm.

(3) **Analysing Depth Data for Nearest Obstacle Detection**

To detect obstacles and gaps, depth values are analysed column-wise as shown in Fig. 5. For each column, the maximum inverse depth value is calculated, indicating the nearest obstacle. The curves Fig. 5b and d display the minimum depth values for each column in the depth map, While Fig. 5a and c shows the MiDaS output depth maps after floor segmentation is applied.

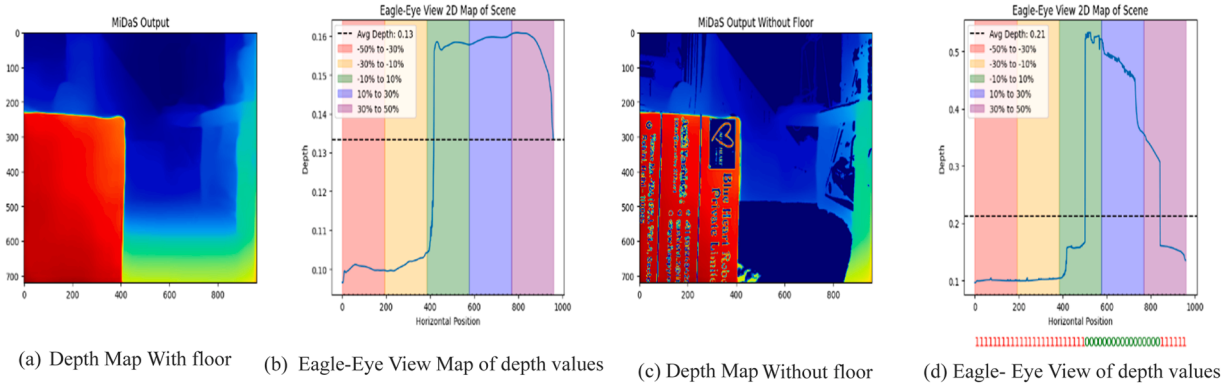
This can be expressed mathematically as:

$$Z_{min}(c) = \min_{y}(z(c,y)) \tag{7a}$$



(a) Original Image (b) Inverse Depth map (c) Depth map without floor (d) the navigable gap

Fig. 4. Process of obtaining the gap block from the depth map.
(a) Original Image (b) Inverse Depth map (c) Depth map without floor (d) the navigable gap.



(a) Depth Map With floor (b) Eagle-Eye View Map of depth values (c) Depth Map Without floor (d) Eagle- Eye View of depth values

Fig. 5. Detection of gaps in the depth map through the MiDaS output.
(a) Depth map with floor (b) Eagle-eye view map of depth values (c) Depth map without floor (d) Eagle- eye view of depth values.

where $z(c, y)$ is the depth value for each pixel in column c, and $Z_{min}(c)$is the minimum depth value for that column. In the non-segmented map shown in Fig. 5b, the floor depth dominates, making it challenging to detect far objects or gaps, with a maximum depth value of only 0.15. In contrast, after floor segmentation shown in Fig. 5d, the maximum depth value increases to over 0.5 units, clarifying the gaps.

The horizontal, black dashed line in the fourth plot indicates the average depth $\overline{d}$, while the binary array at the bottom signifies whether each pixel's depth is above (0) or below (1) the average depth:

$$mask(x,y) = \begin{cases} 0 \text{ if } d(x,y) > \overline{d} \\ 1 \text{ if } d(x,y) \leq \overline{d} \end{cases} \tag{8}$$

A cluster of 0 s in this mask represents the gap area, and the largest cluster signifies the safest direction for the robot to move.

As illustrated in Fig. 5b and d, the depth map's width is divided into multiple vertical segments, each represented by distinct colours. Each segment is treated as a block with x-coordinates of $(-2, -1, 0, 1, 2)$, while the y-coordinate is determined by the robot's state plus one. The gap block is identified as the segment containing the centre of the largest cluster of 0 s, which guides the robot's movement. In the example above, the centre of the 0 s cluster is in the fourth segment, positioning the gap node at (scale * 1, scale * (robot's y state + 1)). This is indicated by the purple circular marker in Fig. 3. Here, "scale" converts continuous space to grid space and is valued at 0.2 in Fig. 3.

So overall the gap detection process involves the following steps:

- Depth values greater than the average are marked as 0 (indicating open space), while those below the average are marked as 1 (indicating near obstacles).
- Clusters of 1 s (representing obstacles) are analysed, with small clusters inverted to prevent noise from being misinterpreted as obstacles.
- The largest cluster of 0 s signifies the gap, and the robot identifies this region as the safest path.
- The vertical region out of the five segments that contains the centre of the largest cluster is taken as the gap block and used in path planning in later steps.

Step 7: No Gap and Hit Detection (optional)

If no navigable gap is detected or the gap detected i.e. pixel length of gap found is less than a threshold, the robot halts its operation. The system also monitors for potential collisions by tracking the distance moved since the last update. If the displacement is minimal, indicating a potential hit, the robot stops to prevent damage.

Experiments with this system revealed that the robot can safely navigate through gaps at least twice its own width. If the detected gap is below this threshold, navigation is halted. Future improvements could refine this approach to allow passage through narrower gaps, mimicking human navigation strategies and abilities.

Step 8: Visual Odometry Processing

Visual Odometry provides a refined estimate of the environment by matching features between frames, estimating rotation and translation, and triangulating 3D points. Using VO, an instantaneous, unscaled, 3D feature-based map of the environment (near the closest obstacle) in front of the robot is obtained. To cluster these 3D map points of the features in the area near the closest depth object, as determined in step 6 effectively, DBSCAN clustering is applied.

The parameters for DBSCAN (eps and min_samples) are selected based on the density and distribution of 3D points: eps=3: This defines the maximum distance between two points to be considered part of the same neighbourhood. The value was experimentally tuned based on the typical spatial distribution of points in the VO-generated 3D map.

- min_samples=6: This specifies the minimum number of points required to form a dense region. It ensures that the algorithm filters out noise and identifies meaningful clusters that correspond to potential obstacles.
- Clusters are formed based on these parameters, and their **centres** are calculated directly as the mean position of points within each cluster. These centres provide a compact representation of obstacles or objects in the environment, as illustrated in Fig. 6b.

It is important to note that only the features in proximity to the closest depth are identified and used for mapping with visual odometry, which optimizes the overall operation of VO.

In this way, we first identify the nearest obstacle by comparing the relative distances between objects in front of the camera, as outputted by the MDE. We then compute the exact distance of this nearest obstacle using visual odometry, scaled by wheel odometry readings. Focusing on only the nearest obstacle optimizes the entire operation, reducing memory usage and increasing the algorithm's overall speed.

This calculation is critical because the robot does not need to alter its path if the nearest obstacle is sufficiently far away, allowing it to safely continue moving straight for the time being.

Step 9: Path Update and Motion Adjustment

The path is updated dynamically based on the detection of obstacles or when the robot is moving straight. The updated path is converted from grid blocks to real-world coordinates. The PID controller then adjusts the motor speeds based on the new error values to ensure the robot continues to follow the optimal path. Fig. 3.7 illustrates the complete motion trajectory using the proposed methodology. In the Fig., the small red square nodes represent obstacles, while the purple nodes indicate the gap nodes . The large red circular node marks the goal state, and the green node represents the start state. The blue path shows the trajectory calculated by the A* algorithm, and the yellow path depicts the actual path followed by the robot. Fig. 7

This methodology combines real-time processing, efficient path planning, and robust obstacle avoidance, allowing the robot to navigate dynamically changing environments with minimal computational resources.

3.2. Methodology explained using a driving analogy

Our robot navigation system can be understood by comparing it to a driver navigating a car on a road. The driver must avoid obstacles while always considering the goal destination. This process involves identifying gaps to safely manoeuvre around the nearest obstacles without veering off course. We break this down into four stages, as illustrated in the columns of Fig. 8:

1. **Approaching the Goal without Immediate Obstacles** (Fig. 8a)

In Fig. 8a, we see two cars: our car, representing the robot, is driving toward its goal, with another car positioned ahead (the closest obstacle) . At this point, the obstacle (the car in front) is at a safe distance, greater than the defined threshold distance for obstacle detection.

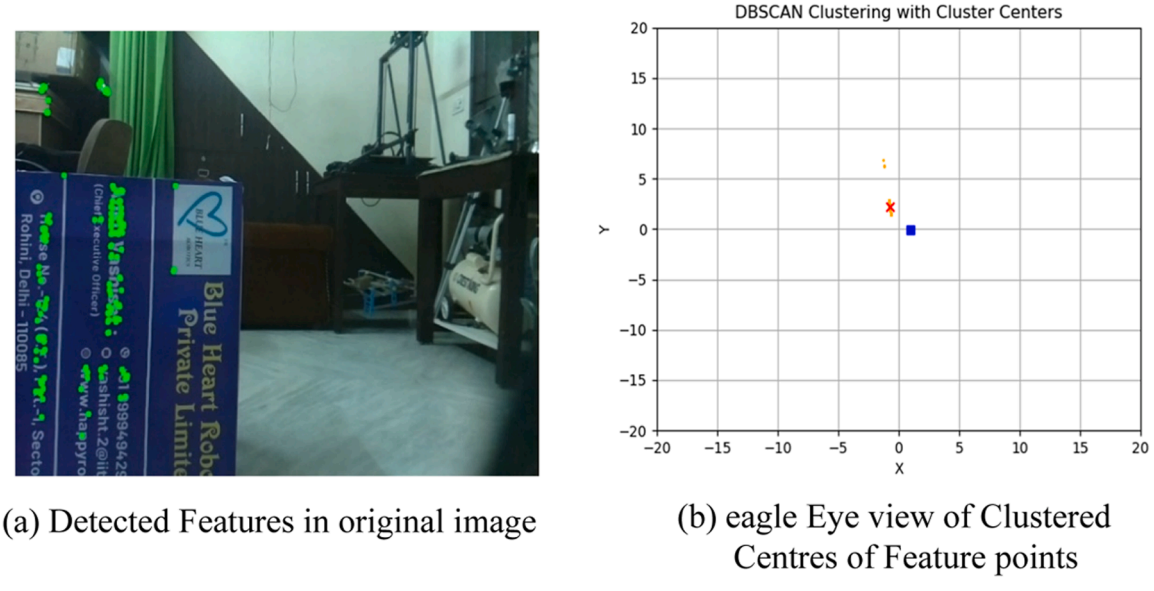


Fig. 6. Metric depth estimation of clustered feature points of the nearest obstacle.
(a) Detected Features in original image (b) eagle Eye view of Clustered Centres of Feature points.

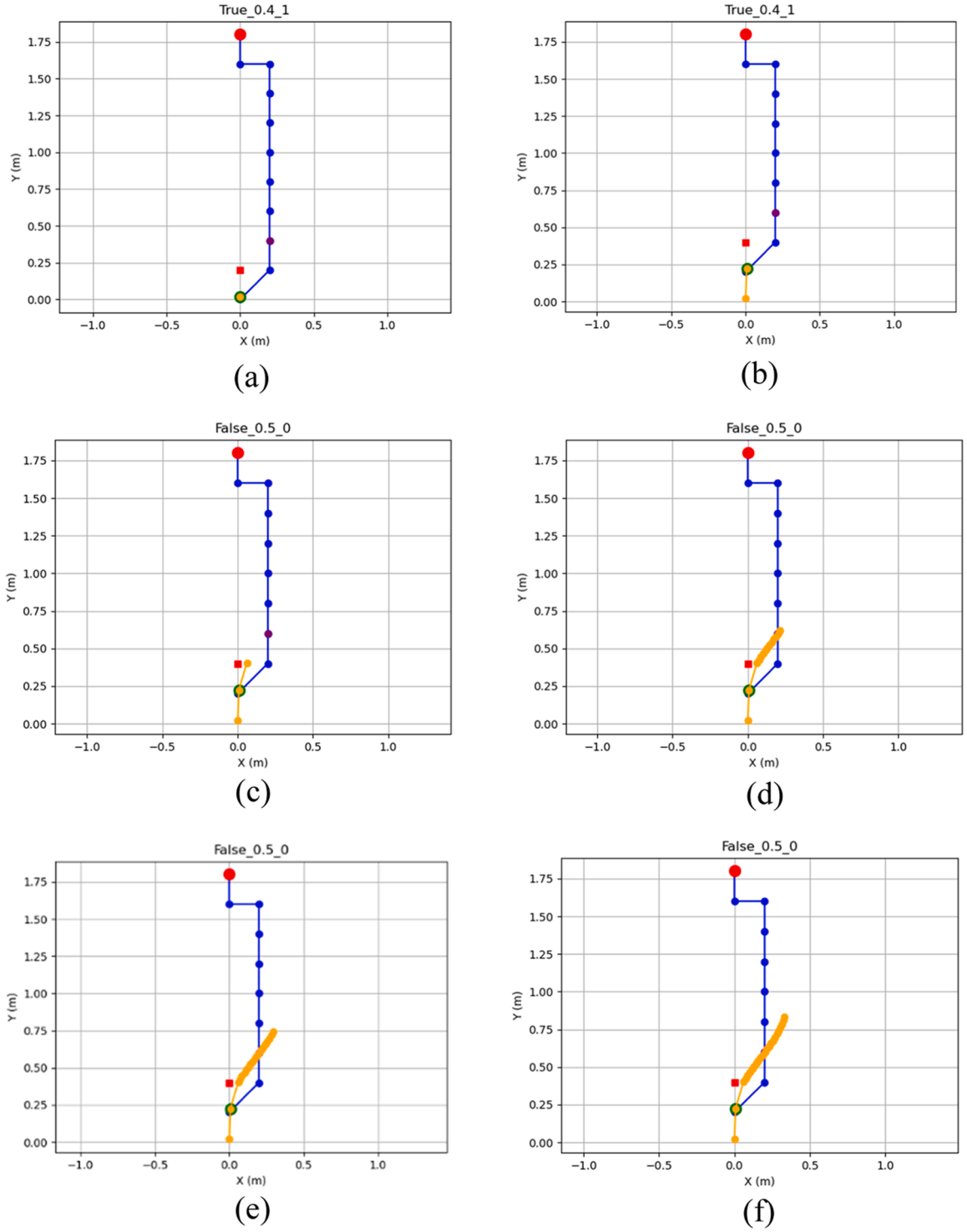


Fig. 7. complete motion trajectory using the proposed methodology. (a) to (l) represent maps selected with a step size of 5 maps.

- Since the front car is far away, the driver (like our robot) continues moving straight towards the goal without making any adjustments to avoid obstacles.
- The robot, similar to the driver, will only start planning for an alternative route once an obstacle crosses the threshold distance.

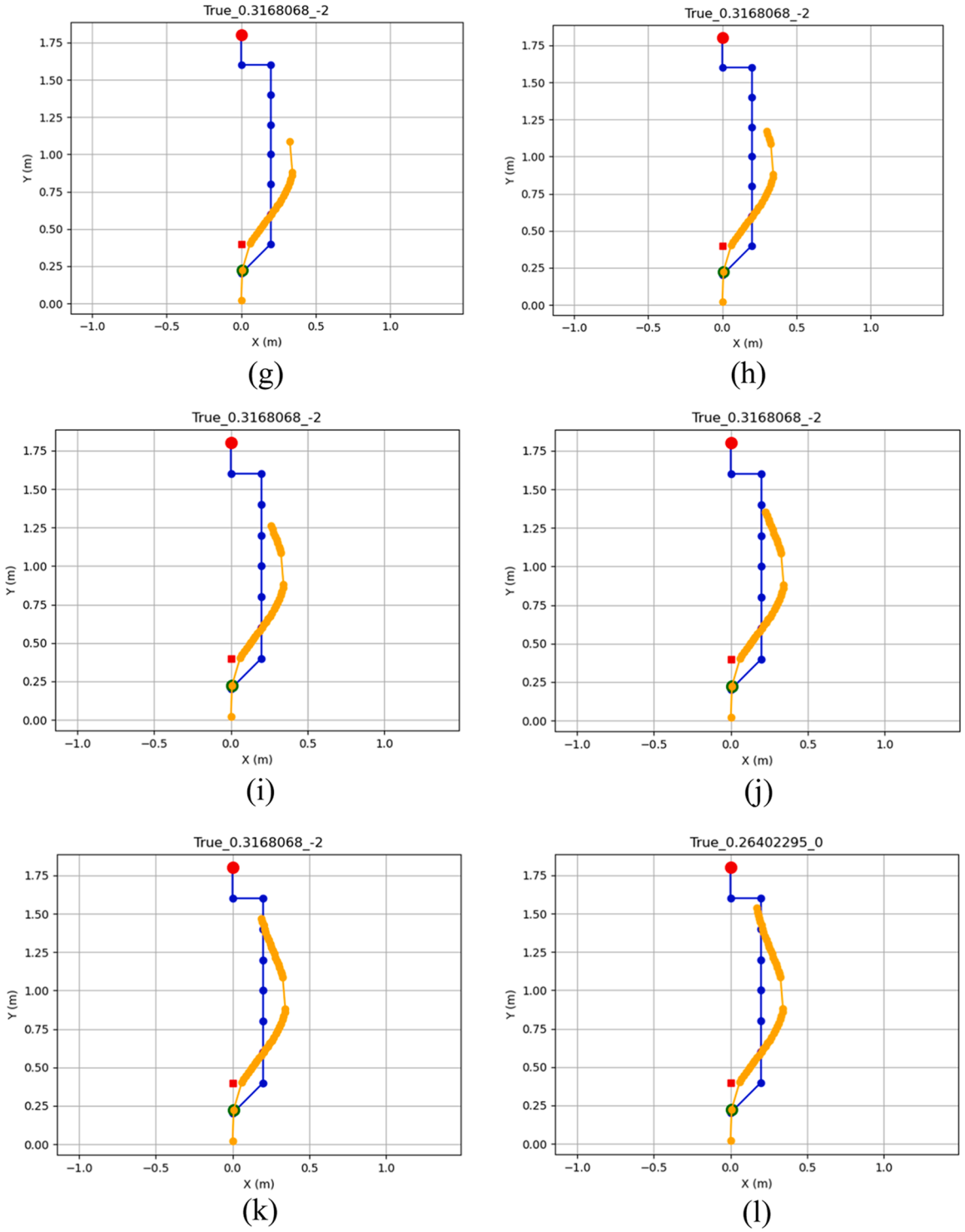


Fig. 7. (*continued*).

2. **Detecting an Obstacle and Choosing the Optimal Gap** (Fig. 8b)

As shown in Fig. 8b, the distance between our car and the front car has decreased, and the obstacle is now within the threshold distance. At this point, the driver must react by identifying the safest gap to avoid the obstacle.

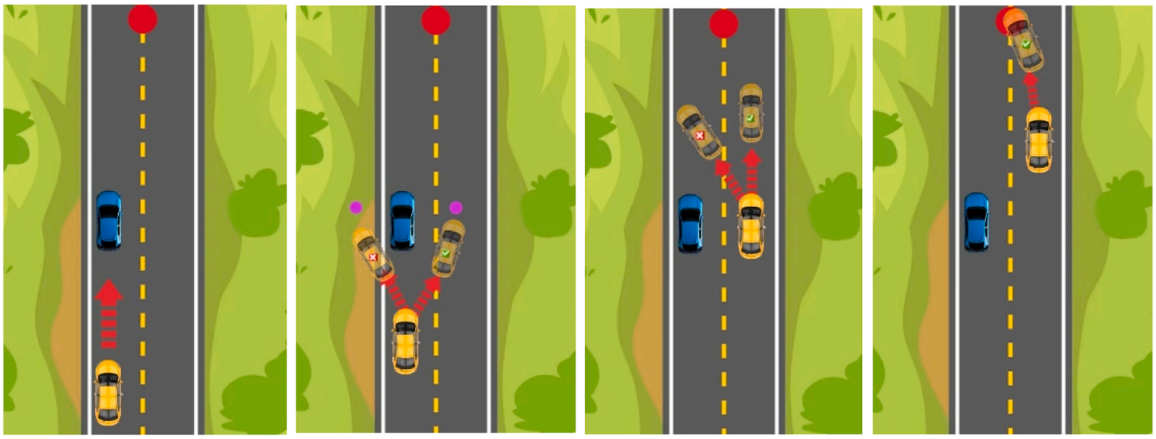


Fig. 8. A simple analogy of the proposed robot navigation pipeline: (a), (b), (c), and (d) collectively illustrate how a car would navigate using the proposed pipeline.

- The driver notices two possible options: a movable gap to the right of the front car (on the road) and a wider gap to the left (in an open field, off the road).
- Although the wider gap to the left seems appealing, the driver recognizes that this direction leads away from the goal and taking it might result in a completely wrong final position.
- Therefore, the driver selects the gap to the right, which keeps them aligned with the global path towards the destination. Our robot does the same by choosing gaps that align with the path generated by the modified A* algorithm, ensuring that obstacle avoidance doesn't pull it off course.

3. **Manoeuvring Through the Chosen Gap** (Fig. 8c)

In Fig. 8c, the car has successfully entered the desired gap and is moving forward.

- At this stage, there is no immediate obstacle in front within the threshold distance, so the driver must continue straight in the new direction.
- The driver must avoid overcorrecting by sharply returning to the previous path, as this could lead to sideswiping another vehicle or obstacle. Similarly, our robot follows the new path without abrupt corrections, ensuring smooth navigation and preventing unnecessary collisions after avoiding an obstacle.

4. **Continuing Towards the Goal** (Fig. 8d)
Finally, in Fig. 8d, the obstacle has been avoided, and the driver is now in a clear, obstacle-free path.

- The driver resumes their journey towards the goal, maintaining focus on reaching the destination without unnecessary detours.
- Our robot, much like the driver, continues its goal-directed navigation using the path generated by the modified A* algorithm. The robot consistently prioritizes the global goal while dynamically avoiding obstacles along the way.

By using these four stages, we can illustrate how our robot intelligently navigates through an environment, always aiming for the goal while ensuring obstacle avoidance to just the nearest obstacle, saving memory in this process, as now instead of storing possible states of all the objects in the scene, we are only concerned about storing the closest obstacle node, the gap node, the goal node and robot's pose node to our memory .

4. Experimentation and results

4.1. Environment

The experiments were conducted in a hall-type indoor lab, filled with various pieces of equipment and furniture. This created a challenging environment for obstacle detection and navigation, where the robot had to deal with a range of obstacles, including static objects like tables and dynamic objects such as moving people.

To evaluate the robustness of the approach, tests were conducted under different lighting conditions:

- **Daylight:** Refers to indoor daylight conditions with an intensity between 500 and 1000 lx, depending on the time of day and external weather conditions. Sunlight was sourced from indirect exposure through windows, ensuring the consistency of natural light without direct sunlight affecting the environment.
- **Dim light:** Defined as environments with lighting below 50 lx, simulating low-visibility conditions that could hinder the robot's navigation ability.
- **Bright tube light:** Defined as lighting conditions with an intensity between 1000 and 2000 lx, providing stable, high-visibility scenarios suitable for testing in controlled lighting conditions.

These varied conditions were designed to test the robot's ability to navigate and detect obstacles effectively under different environmental factors.

Furthermore, experiments were performed in both static and dynamic environments. In the dynamic scenarios, moving obstacles were introduced, testing the robot's responsiveness and speed in adjusting its navigation path in real-time.

4.2. Robot hardware

The hardware used for the experiments consisted of a battery-powered wheeled mobile robot, as shown in Fig. 9. The robot was equipped with:

- Raspberry Pi camera, used for real-time visual input.
- Motors with built-in magnetic encoders, providing odometry data for localization and control.
- A range of Raspberry Pi versions, from Raspberry Pi 2 $A+$ to Raspberry Pi 4B, with varying RAM capacities of 1GB and 2GB.

The robot ran the Raspbian operating system, optimized for low-resource environments. Fig. 9 shows the complete Raspberry pi powered Final Robot used in our experimentation.

4.3. Methods

- Monocular Depth Estimation:

Two state-of-the-art models were considered for depth estimation in this study: MiDaS V2.0 and Depth-Anything. Both models were used in their pretrained versions without additional fine-tuning. The datasets used for training these models are as follows:

- **MiDaS V2.0**: Trained on 12 datasets, including ReDWeb, DIML, Movies, MegaDepth, WSVD, TartanAir, HRWSI, ApolloScape, BlendedMVS, IRS, KITTI, and NYU Depth V2.
- **Depth-Anything**: Trained on approximately 1.5 million labelled images from diverse datasets such as LSUN, Objects365, Open Images V7, and Places365. Additionally, it leverages a vast pool of over 62 million unlabelled images, enabling it to learn from an extensive and varied set of visual data without depending primarily on specific labelled datasets like KITTI or NYU Depth V2, which are commonly used by other depth estimation models.

These original datasets used for pretraining Depth-Anything are publicly available through their respective sources. Both models were evaluated in terms of their accuracy in obstacle detection, as well as their computational efficiency on low-resource hardware like

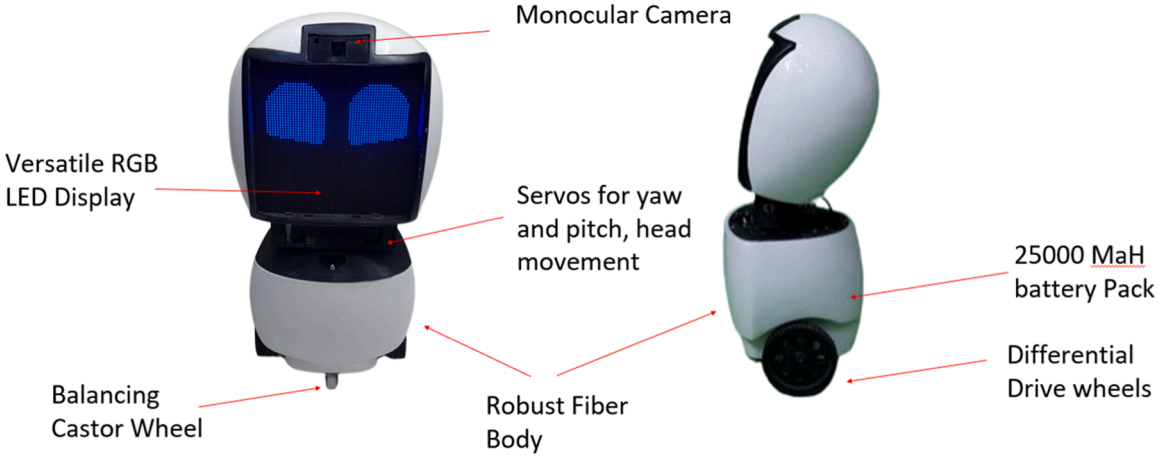


Fig. 9. wheeled mobile robot used for experimentation.

the Raspberry Pi.

Other than these datasets, To assess the performance of these models in our target environment, we created a custom test dataset. This dataset was constructed by placing objects at fixed distances (0.6 m, 0.9 m, 1.2 m, 1.5 m, 1.8 m, 2.1 m, and 2.4 m) and capturing depth estimates from each model. For each distance label, 10 images were captured, ensuring a diverse sample for evaluation.

- Localization:

Three different localization methods were compared:

- Feature Matching: Matching distinct visual features between frames.
- Feature Tracking: Tracking feature points across consecutive frames.
- Wheel Odometry: Using motor encoder data to estimate the robot's position.

- Mapping:

for mapping feature-based VO was used For mapping, the robot used visual odometry data to construct a real-time map of the environment while navigating.

4.3.1. Monocular Depth estimation results

In this section, we compare the inference speed and memory efficiency of two Monocular Depth Estimation models: MiDaS v2.0 (in its TensorFlow Lite format) and Depth Anything. The experiments were conducted on Raspberry Pi 3B and Raspberry Pi 4B models with varying RAM capacities. Our aim was to assess the feasibility of deploying these models on low-resource edge devices, such as Raspberry Pi, which are typically used in real-time dynamic environments.

The specifications of the devices used for testing are as follows:

- Raspberry Pi 3B: 1.2 GHz quad-core ARM Cortex-A53, 1GB RAM
- Raspberry Pi 4B (1GB): 1.5 GHz quad-core ARM Cortex-A72, 1GB RAM
- Raspberry Pi 4B (2GB): 1.5 GHz quad-core ARM Cortex-A72, 2GB RAM

4.3.1.1. Metric depth. To evaluate the performance of these models in a real-world setting, we created a custom test dataset. Objects were placed at predefined distances of 0.6 m, 0.9 m, 1.2 m, 1.5 m, 1.8 m, 2.1 m, and 2.4 m, and depth estimates were captured from each model. For every distance, 10 images were recorded, ensuring a diverse range of conditions for evaluation.

Table 2 presents the accuracy metrics for different depth estimation methods. The three key metrics used for assessment are Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Relative Absolute Error (REL).

- MAE (Mean Absolute Error) measures the average absolute deviation between the predicted and actual depth values. A lower MAE signifies better accuracy. MiDaS V2.0 produces the highest MAE (0.4665), indicating poor predictions. Depth Anything achieves the lowest MAE (0.0515), making it the most precise model, while Scaled MiDaS (VO) significantly improves upon MiDaS V2.0, reducing the error to 0.1086.
- RMSE (Root Mean Squared Error) gives more weight to larger errors due to squaring differences. The unscaled MiDaS V2.0 model has a high RMSE (0.5536), suggesting substantial deviation from the actual depth values. Depth Anything attains the lowest RMSE (0.0614), whereas Scaled MiDaS V2.0 (VO) reduces the RMSE to 0.1315, showing that scaling reduces extreme errors.
- REL (Relative Absolute Error) expresses error as a fraction of the true depth value, making it useful for evaluating performance across various depth ranges. MiDaS V.2 has a significantly high REL (0.3736), confirming its unreliable estimates. Depth Anything achieves the best performance (0.0422), and Scaled MiDaS V2.0 (VO) substantially improves over MiDaS V2.0, lowering REL to 0.0894.

These findings confirm that Depth Anything is the most accurate model, while Scaled MiDaS V2.0 (VO) significantly refines MiDaS V2.0 predictions, bringing them closer to the true depth values.

These results are illustrated in Fig. 10. where:

- Fig. 10a displays the unscaled MiDaS V.2 output, showing notable errors.

Table 2
Accuracy metrics for MiDaS v2.0, Depth Anything and Scaled MiDaS V2.0 on custom test dataset.

Method	MAE (↓)	RMSE (↓)	REL (↓)
MiDaS V2.0	0.4665	0.5536	0.3736
Depth Anything	**0.0515**	**0.0614**	**0.0422**
Scaled MiDaS V2.0 (VO)	0.1086	0.1315	0.0894

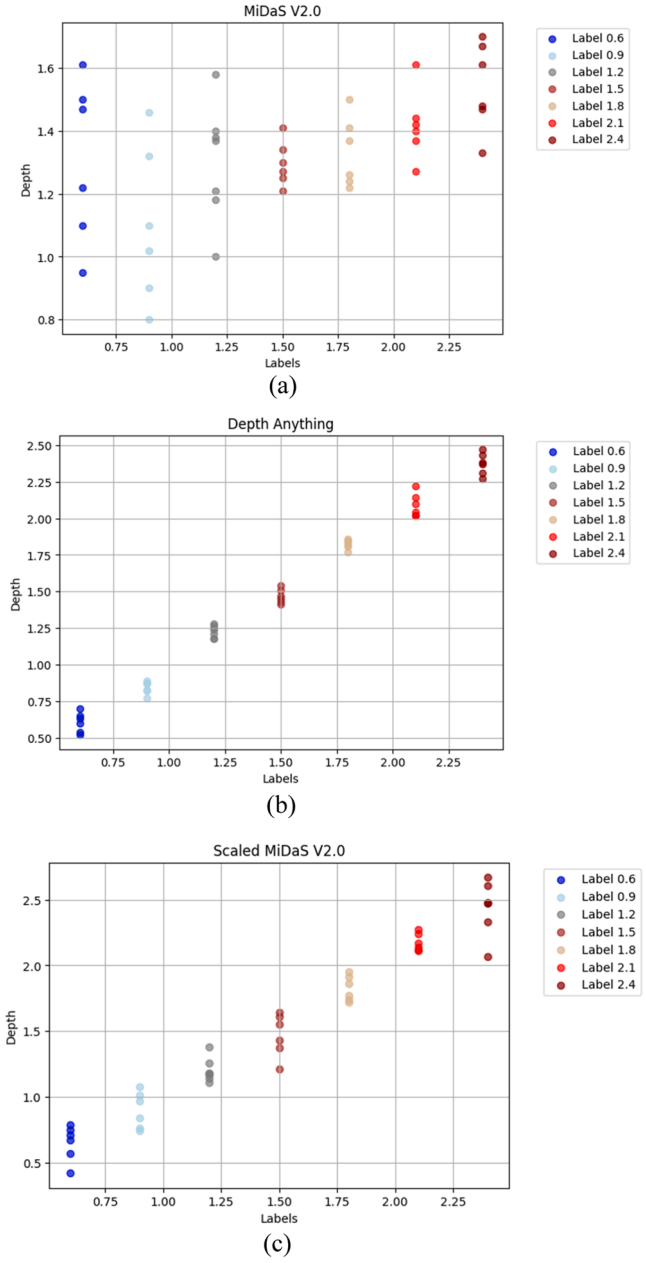


Fig. 10. Depth Results of Closest obstacle in Test Dataset with (a) MiDaS V2.0 (b) Depth Anything (c) MiDaS V2.0 readings scaled with Visual Odometry.

- Fig. 10b presents Depth Anything results, closely matching ground truth.
- Fig. 10c highlights the improved depth estimates from Scaled MiDaS V2.0, demonstrating enhanced accuracy.

4.3.1.2. Inference speed and memory usage. We measured the inference times of both MiDaS v2.0 and Depth Anything on these devices. The inference times on the Raspberry Pi 4B (2GB) model, over 20 such inferences, are plotted in Fig. 11. As shown in Fig. 10, MiDaS v2.0 performs significantly better across all tested devices.

As mentioned in Table 3, on the Raspberry Pi 3B, Raspberry Pi 4B (1GB), and Raspberry Pi 4B (2GB), the MiDaS TensorFlow Lite v2.0 model achieves average inference times of 0.97 s, 0.59 s, and 0.58 s respectively, running on 4 threads. These times (sampling frequency for data collection of 2 frames per second) are well-suited for real-time applications in dynamic environments. In contrast, Depth Anything fails to run on both the Raspberry Pi 3B and Raspberry Pi 4B (1GB) due to insufficient memory (process terminated). Only on the Raspberry Pi 4B (2GB) was Depth Anything able to run, achieving an average inference time of approximately 18.4 s,

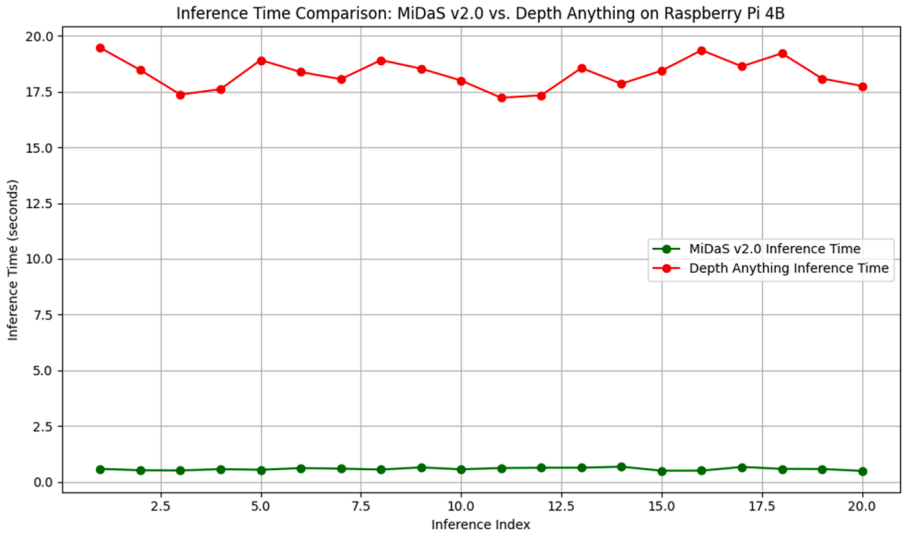


Fig. 11. Inference time comparison between MiDaS v2.0 and depth anything models on Raspberry Pi 4B.

Table 3
Inference times for MiDaS v2.0 and Depth Anything on different Raspberry Pi models.

Device	Processor	RAM	MiDaS v2.0 average Inference time (↓)	Depth Anything average Inference time (↓)
Raspberry Pi 3B	1.2 GHz ARM Cortex-A53	1GB	0.97 s (4 threads)	Process Killed
Raspberry Pi 4B	1.5 GHz ARM Cortex-A72	1 GB	0.593s(4 threads)	Process Killed
Raspberry Pi 4B	1.5 GHz ARM Cortex-A72	2 GB	**0.582s (4 threads)**	**18.39 s**

making it impractical for real-time deployment.

4.3.1.3. Input–Output comparison. We also evaluated the output depth maps produced by both models. On the Raspberry Pi 4B (2GB), where both models could run, we generated depth maps for the same input image. The results show that while Depth Anything

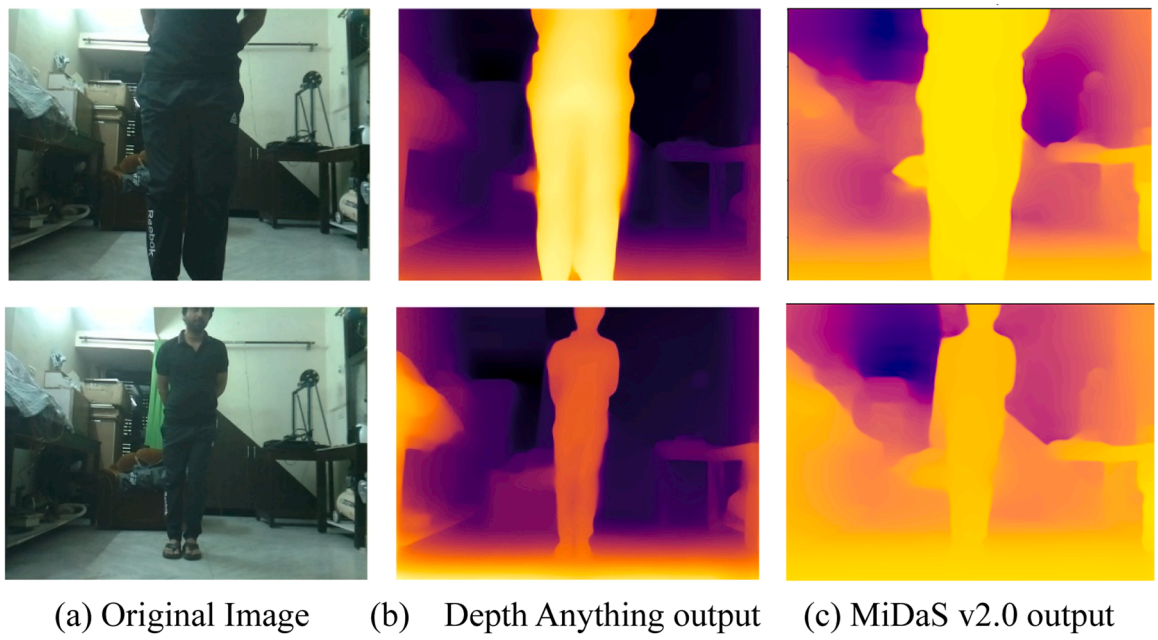


Fig. 12. Comparison of raw camera images and depth maps generated by Depth Anything and MiDaS v2.0.
(a) Original Image (b) Depth Anything output (c) MiDaS v2.0 output.

provides sharper depth details and higher visual fidelity, MiDaS v2.0 outputs sufficiently accurate relative depth maps that are adequate for robot navigation tasks. Fig. 12 displays the comparison between raw RGB images and the corresponding depth maps generated by Depth Anything and MiDaS v2.0. The Fig. is organized into three columns and two rows. The first column shows the raw camera images, the second column displays the depth maps from Depth Anything, and the third column shows the depth maps from MiDaS v2.0.

The depth maps are visualized using the 'plasma' colormap, where brighter colors represent closer proximity to the camera. In the first row, the author is standing closer to the camera compared to the second row, and this is evident from the brighter regions in the depth maps. As apparent, Depth Anything provides accurate metric depth values, showing the actual distance from the camera, while MiDaS v2.0 outputs relative depth between objects. Despite lacking absolute depth measurements, MiDaS v2.0 performs well when combined with visual odometry and wheel odometry, offering fast processing suitable for real-time navigation especially in dynamic environment with low resource compute.

From the experimentation, it is evident that MiDaS v2.0 offers a substantial advantage for real-time applications on low-resource edge devices due to its fast inference times and low memory footprint. While Depth Anything can provide more detailed depth maps, its significantly slower inference time (~17 s) and high memory requirements (unable to run on devices with less than 2GB RAM) make it unsuitable for real-time, resource-constrained environments such as robotic navigation.

Moreover, despite MiDaS v2.0 producing relative depth estimates (rather than absolute, metric depth), this limitation is mitigated by the use of visual odometry. As discussed earlier, MiDaS v2.0 provides depth relationships between objects, and VO complements this by estimating the robot's position relative to those objects. This combined approach is not only computationally efficient but also robust enough for deployment in dynamic environments on edge devices.

4.3.2. Localization results

In this section, we compare three different localization methods: Visual Odometry with Feature Matching, Visual Odometry with Feature Tracking, and Wheel Odometry. Many trials were conducted to evaluate these methods, and for one such trial, the robot first moves straight, then takes a right turn, and finally moves straight again. Images of this trial, taken from the WMR camera at every fifth step, are shown in Fig. 13. For brevity and space limitations, only every fifth frame is shown. The images effectively illustrate the robot's movements within the environment, highlighting the differences in localization results across the three methods.

The localization performance of these methods for this particular case is presented in Fig. 14, where three plots are shown, each representing a different localization approach.

- Fig. 14a: Localization results using Visual Odometry with Feature Matching.
- Fig. 14b: Localization results using Visual Odometry with Feature Tracking.
- Fig. 14c: Localization results using Wheel Odometry.

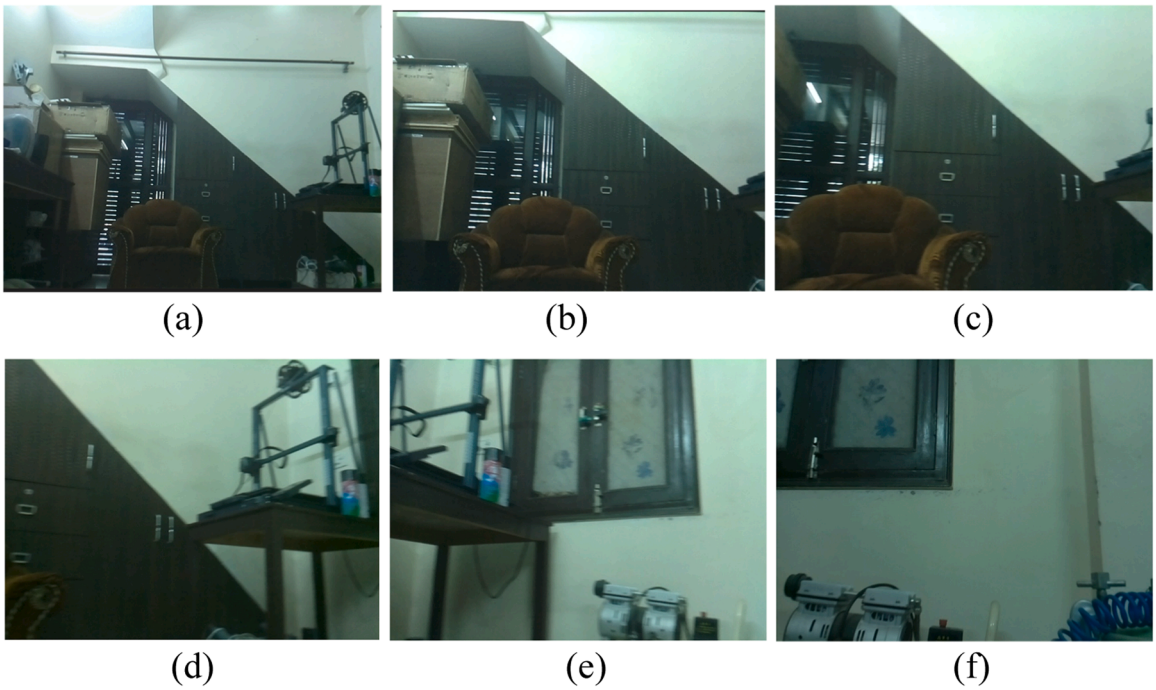


Fig. 13. Images captured by the WMR camera during a trial: (a) to (f) represent frames selected with a step size of 5 images.

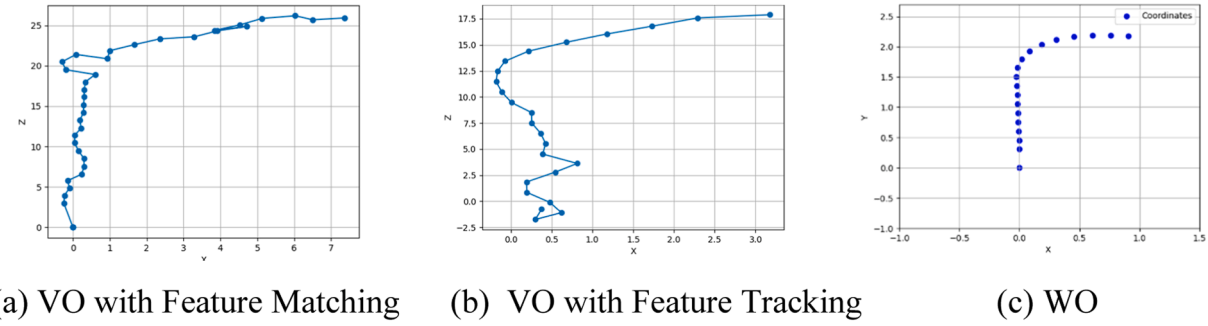


Fig. 14. localization performance of three different methods for the described trial.
(a) VO with Feature Matching (b) VO with Feature Tracking (c) WO.

Throughout the trials, we observed that VO with Feature Matching generally performed well in straight motion but struggled with scale when the robot encountered turns or changes in orientation. In contrast, Feature Tracking provided better performance during orientation changes, handling dynamic adjustments in real-time, but still lacked precise scale accuracy.

Wheel Odometry, however, consistently outperformed both VO approaches in terms of scale accuracy and precision. In all experiments conducted flat indoor surfaces, WO produced nearly perfect results. It was particularly effective for long, straight segments, where the odometry data was invaluable for maintaining scale consistency and accurate tracking. Another significant advantage of WO was its ability to provide scale-correct localization results.

4.3.3. Mapping performance

In this section, we compare the point cloud generation performance of Feature Matching-based Visual Odometry under different lighting conditions. Two of the generated results for the same path are shown in Fig. 15.

- Fig. 15a displays the mapping result under bright room conditions with sufficient lighting.
- Fig. 15b shows the mapping result in low light conditions, where lighting is limited.

As show in Fig. 16a and b The left image was taken in a bright room with sufficient lighting, while the right image was captured in low light conditions, showing reduced visibility but still allowing for functional navigation. It became apparent that, as expected, the mapping results under bright lighting conditions were significantly better than those in low light conditions. The increased illumination allowed for more accurate point cloud generation, leading to a clearer and more defined map.

However, even under low light conditions, the algorithm performed decently well and did not significantly hinder the overall navigation process. While the point cloud generated in low light was less precise compared to bright light conditions, it still provided a usable map that supported successful navigation along the path.

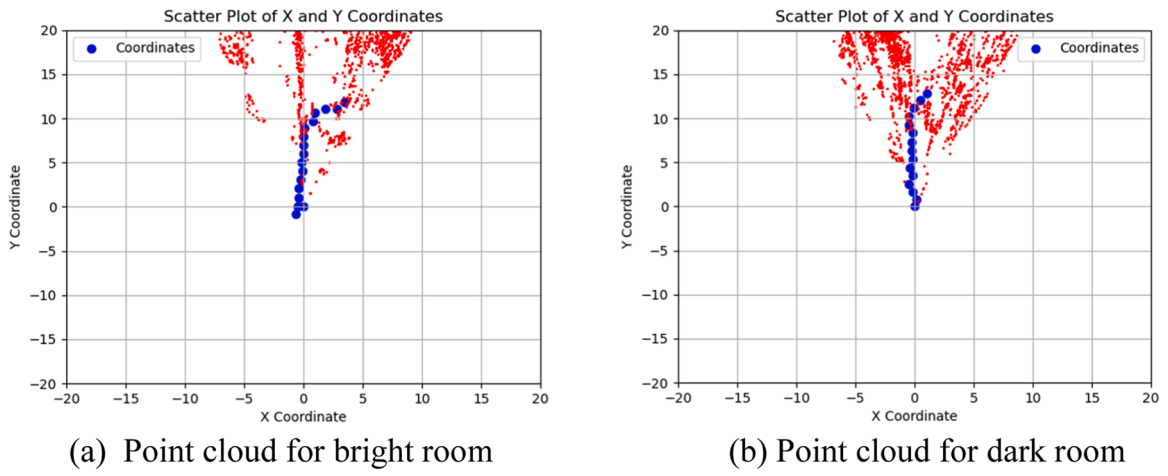
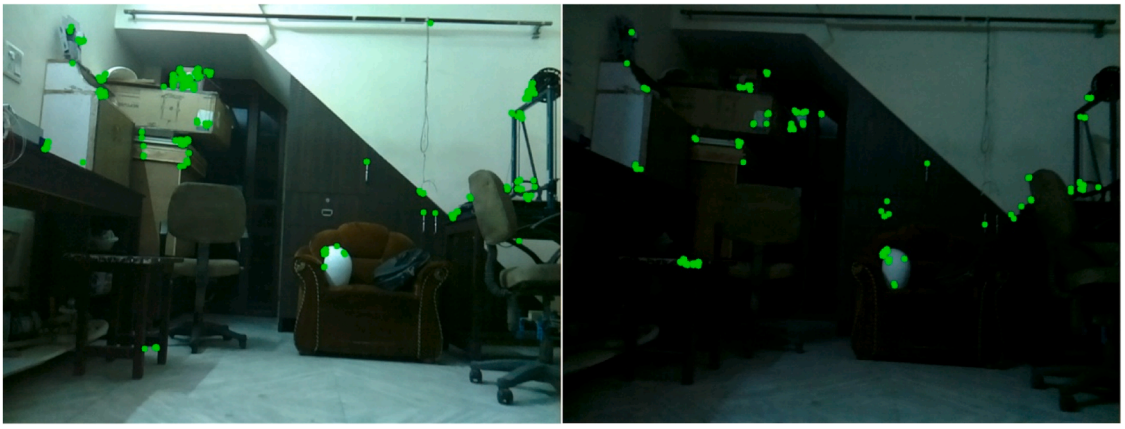


Fig. 15. Comparison of point cloud generation using Feature Matching-based Visual Odometry under different lighting conditions.
(a) Point cloud for bright room (b) Point cloud for dark room.



(a) Detected Features in Bright Room (b) Detected Features in Dark Room

Fig. 16. Actual images captured by the WMR's camera during the trials under different lighting conditions.
(a) Detected Features in Bright Room (b) Detected Features in Dark Room.

4.4. Results

In this section, we discuss the results of the Wheeled Mobile Robot navigating toward a goal location with one or more obstacles in its path. The experiment was conducted 50 times for each of the three different speed settings, and results were evaluated in terms of Success Rate, which is calculated as:

$$\text{Sr} = \frac{\text{Number of Successful completed Navigation Tasks}}{\text{Total number of Navigation Tasks}} \text{ X } 100 \tag{9}$$

Success rate (SR) is a widely recognized metric in autonomous navigation. Rondoni et al. [39] further emphasize its importance by benchmarking robot navigation in hospital environments, reinforcing its relevance as a key measure of task completion reliability across diverse real-world settings, including the dynamic environments considered in our work. Based on this definition, we observed that the success rate varies significantly with the robot's speed, as summarized in Table 4.

Comparison with Prior Studies

To further contextualize our results, we compare our navigation success rates with those from prior studies, as summarized in Table 5.

This comparison highlights that while our approach achieves competitive success rates at lower speeds, motion blur and skidding at higher speeds introduce challenges that are less prominent in modular learning approaches. Unlike Yi and Guan's and Gervet et al.'s work, which focuses on high-performance simulations without considering resource consumption in the overall process, our method is designed for real-world deployment on low-resource hardware.

Statistical Analysis

To assess whether the observed differences in success rates across different speeds are statistically significant, we performed the Friedman test, a non-parametric test for comparing multiple related samples. The test resulted in a Friedman statistic of 15.20 with a p-value of 0.0005.

Since $p < 0.05$, we reject the null hypothesis, indicating that there is a statistically significant difference in success rates among the different speed conditions. This confirms that robot speed plays a critical role in navigation performance, with lower speeds significantly improving success rates compared to higher speeds.

Failure Analysis

Across different speed settings, failures were primarily caused by the following factors:

- VO (Visual Odometry) Failure in Low Lighting Conditions: In low-light environments, the Visual Odometry (VO) system struggled to accurately determine the depth of obstacles. This issue was further exacerbated at higher speeds due to motion blur, which reduced the accuracy of depth estimation and obstacle detection.

Table 4
Effect of robot speed on navigation success rate.

Robot Speed (m/s)	Success Rate (↑)	Observation
0.15 (Low)	**85 % (43/50 trials)**	More stable navigation, fewer failures due to better obstacle detection
0.25 (Medium)	78 % (39/50 trials)	Balanced performance, as observed in our experiment
0.35 (High)	65 % (33/50 trials)	Increased failures due to motion blur & skidding

Table 5
Comparison of navigation success rates with prior studies.

Study	Approach	Success Rate (↑)	Observations
Yi and Guan [26]	DRL-based (Deep Deterministic Policy Gradient (DDPG), Twin Delayed Deep Deterministic Policy Gradient (TD3), Optimized DRL)	77 % (200 obstacles)	Evaluated in simulation; optimized DRL model performed best
Gervet et al. [27]	Modular Learning	**90 %**	Modular learning transfers well from simulation to real-world
Our Work with speed set at 0.15 m/s (Low)	Integrated Monocular Depth Estimation and Visual Odometry	85 % (43/50 trials	Stable navigation, fewer failures

- Failure when Obstacles were Tightly Placed: When obstacles were positioned too close together with navigable gap less than 1.5 times that of robot's width, the robot had difficulties executing precise turns. In some cases, the robot skidded into obstacles when they moved out of the camera's field of view, particularly at higher speeds. Adjusting the PID controller's parameters may improve navigation in such situations.
- The experimentation also included scenarios involving dynamic environments. In many trials, the closest obstacle to the robot was a moving human. The system treats objects as static now of detection, even if they are in motion. If an obstacle is detected as the closest object, its location is added to the grid map as an obstacle node, and the robot generates a new path accordingly.

The system operates at a speed of approximately 1.3 frames per second on a Raspberry Pi 4. This ensures that dynamic objects do not move out of the robot's detection range easily. The system's ability to respond to changes in the environment effectively demonstrates its applicability to real-world dynamic scenarios.

In Fig. 17, the path from one of the experiments is plotted, showing the robot navigating around two obstacles. Initially, the robot moves straight, with the MDE model continuously detecting the closest obstacle. The VO system provides the distance to this obstacle, which is then scaled using the WO's localization data to convert it into metric units. Once the obstacle crosses a certain threshold distance, the robot turns right to avoid it. Further along the path, the robot detects a second obstacle directly ahead and successfully dodges it by turning left, eventually reaching the goal.

During this process, the robot recalculates its path twice using the modified A* algorithm, as described earlier. Overall, the VO system performed adequately, and the MiDaS MDE model successfully detected the closest obstacles, enabling smooth navigation in an indoor environment by our low-compute robot.

5. Conclusion and future work

In this work, we introduced a novel hybrid approach to robotic navigation that seamlessly integrates map-building techniques from classical Visual Odometry (VO) with Monocular Depth Estimation (MDE)-based mapless navigation. This method effectively combines various techniques, including object detection, image segmentation, visual odometry, wheel odometry, monocular depth estimation, path planning, and PID control, in an optimal manner to enable real-time navigation on highly resource-constrained devices.

Through extensive experimentation, the system demonstrated the ability to navigate dynamically changing environments, achieving a success rate of **78 %** across 50 trials. These results highlight the robustness of the approach, especially in scenarios where the closest obstacle was dynamic, such as a moving human. The system's dynamic re-planning capabilities, achieved through instantaneous grid mapping and a modified A* algorithm, enabled effective navigation in real-world conditions.

However, the results also revealed areas for improvement. The **11 unsuccessful trials** were primarily attributed to two factors:

- **VO failure in low lighting conditions (6 cases):** Insufficient lighting impaired the accuracy of depth estimation, causing navigation errors.
- **Tightly placed obstacles (5 cases):** Close-proximity obstacles led to skidding during turns, as the robot's camera lost visibility of these obstacles. This issue underscores the need for improved PID controller tuning and enhanced perception capabilities.
- The current implementation operates at **1.3 FPS**, which, while sufficient for slower navigation tasks, limits its performance in high-speed or highly dynamic environments. Building upon these findings, several challenges and opportunities for future improvements are identified:

1. **Handling Low-Texture and Poor Lighting Conditions**:

The primary challenge lies in the VO component, particularly in accurately estimating the depth of the closest obstacle. This is especially problematic when the actual distance is shorter than the depth output by VO. Developing faster, metric-based MDE models—unlike relative depth models like MiDaS v2—could address this issue. Alternatively, a new MDE model could be trained to take VO, WO, and image inputs and output precise depth measurements of the scene. Future work will focus on exploring these solutions.

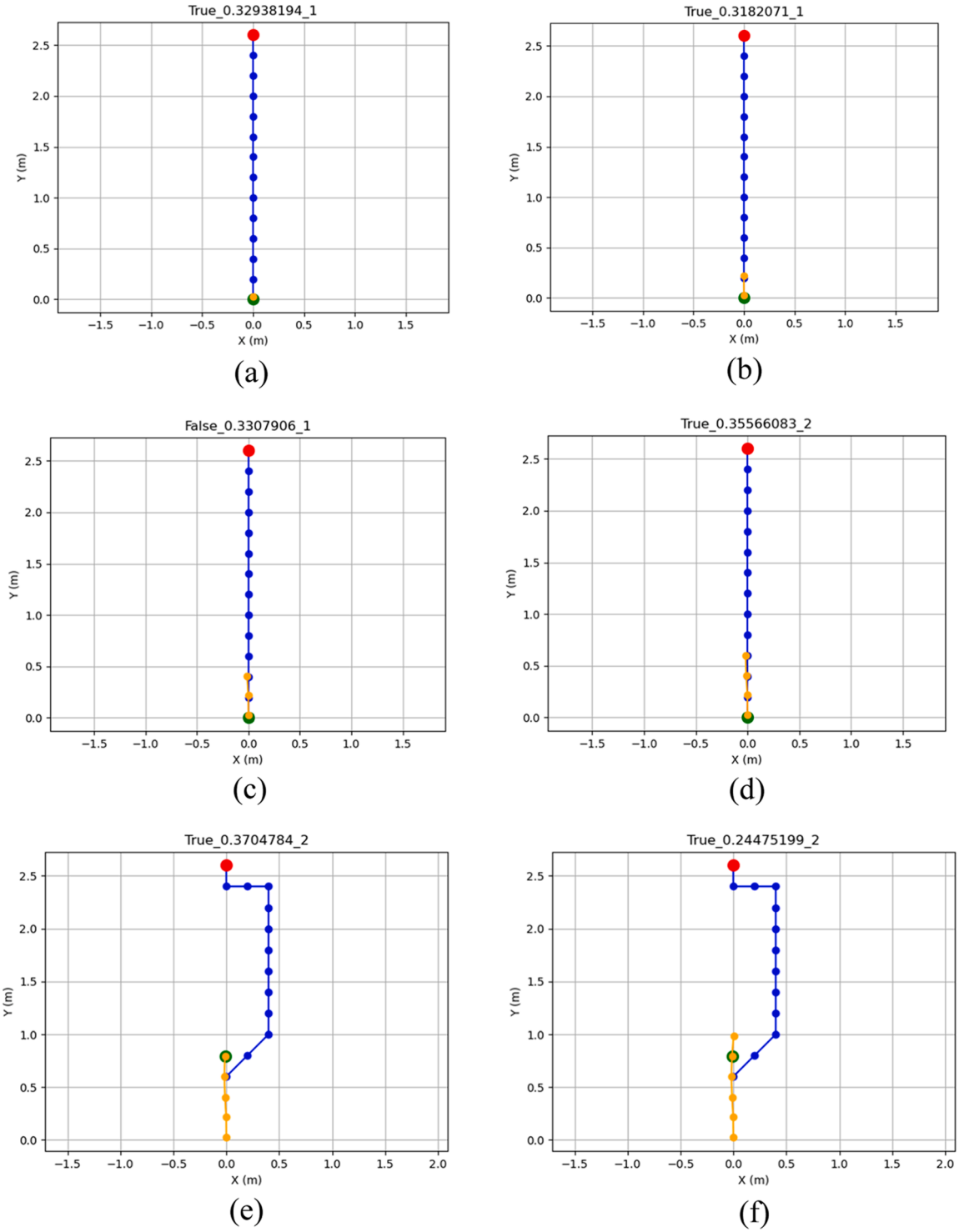


Fig. 17. The robot's navigation path during an experiment with two obstacles in its path. (a) to (n) represent maps selected with a step size of 5 maps.

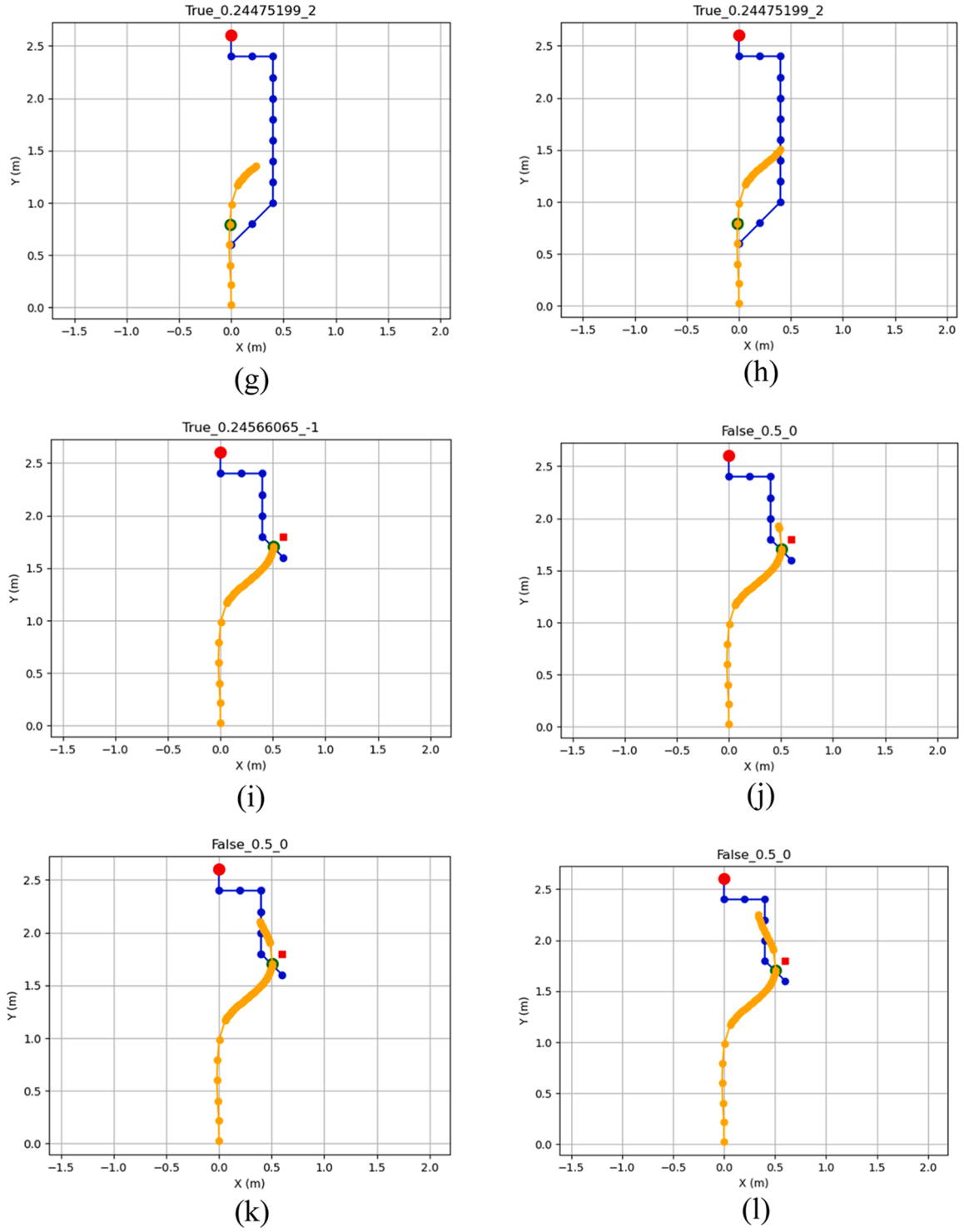


Fig. 17. (*continued*).

2. **Improving Navigation in Tight Spaces**:

Expanding the field of view or incorporating additional cameras could help resolve issues related to skidding and loss of visibility in tight spaces. Additionally, since the platform features a rotating head with the camera mounted on it, integrating the head's yaw motion into the navigation process could further enhance performance.

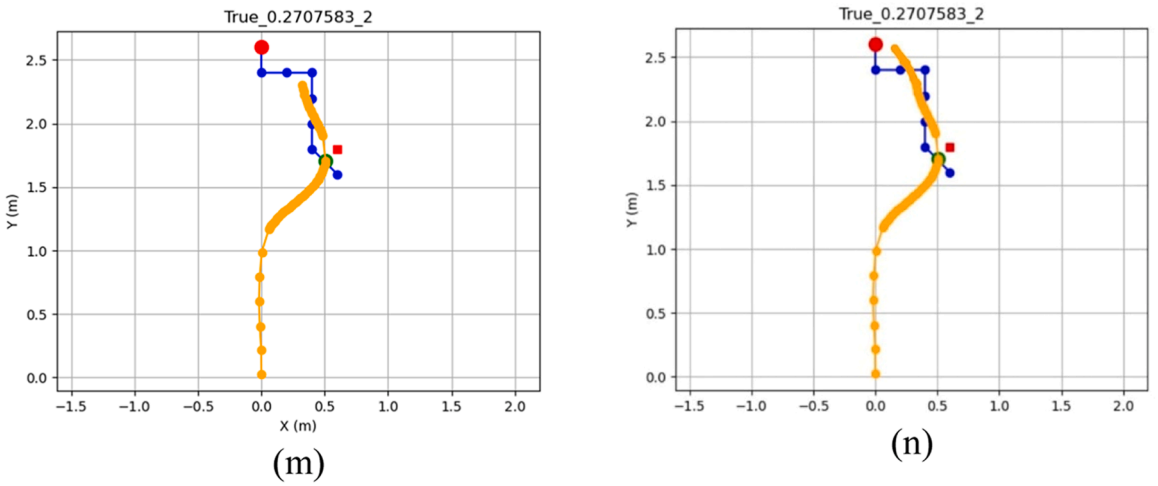


Fig. 17. (*continued*).

3. **Dynamic Object Prediction**:

While the current system treats moving obstacles as static during path planning, integrating motion prediction algorithms could improve decision-making and path optimization in dynamic scenarios.

4. **Broader Applicability**:

Expanding the system for other robotic platforms, such as drones or autonomous vehicles, would require addressing platform-specific dynamics and computational constraints.

5. **Expansion of Experimental Metrics**:

While the current experimental setup focused on the key aspects of the proposed method, future work will explore additional performance metrics, such as navigation accuracy under varying obstacle densities, robot positioning errors, and path planning time costs. These metrics will provide a more comprehensive evaluation of the system's capabilities and performance in different scenarios.

By tackling these challenges, the proposed methodology has the potential to significantly advance robot navigation, particularly for low-resource devices. This could lead to the development of lower-cost robots, ultimately making them more accessible and affordable for everyday use. This hybrid approach lays the groundwork for efficient and adaptable navigation systems, ensuring that sophisticated algorithms can be deployed on low-resource hardware, making them accessible for diverse real-world applications.

This work represents the first integration of MDE-based mapless navigation with VO-based map-building techniques, offering a new paradigm for research in robotic navigation. Future advancements in this direction, such as faster and more accurate metric MDE models or enhanced VO systems, could pave the way for the next generation of navigation systems, characterized by efficiency, adaptability, and scalability.

CRediT authorship contribution statement

Ankit Vashisht: Conceptualization, Methodology, Software, Writing – original draft. **Geeta Chhabra Gandhi:** Data curation, Writing – review & editing. **Sumit Kalra:** Visualization, Investigation. **Dinesh Kumar Saini:** Supervision, Software, Validation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] Rubio F, Valero F, Llopis-Albert C. A review of mobile robots: concepts, methods, theoretical framework, and applications. Int J Adv Robot Syst 2019;16(2): 1729881419839596. https://doi.org/10.1177/17298814198395.

[2] Madhevan B, Sreekumar M. Identification of probabilistic approaches and map-based navigation in motion planning for mobile robots. Sādhanā 2018;43(1):8. https://doi.org/10.1007/s12046-017-0776-8.

[3] Mur-Artal R, Montiel JMM, Tardos JD. ORB-SLAM: a versatile and accurate monocular SLAM system. IEEE Trans Robot 2015;31(5):1147–63. https://doi.org/10.1109/TRO.2015.2463671.

[4] Vakhitov A, Lempitsky V, Zheng Y. Stereo relative pose from line and point feature triplets. In: In proc*eedings of the European conference on computer vision (ECCV)*; 2018. p. 648–63. https://doi.org/10.48550/arXiv.1907.00276.

[5] Engel J, Koltun V, Cremers D. Direct sparse odometry. IEEE Trans Pattern Anal Mach Intell 2017;40(3):611–25. https://doi.org/10.1109/TPAMI.2017.2658577.

[6] Ma FW, Shi JZ, Ge LH, Dai K, Zhong SR, Wu L. Progress in research on monocular visual odometry of autonomous vehicles. J Jilin Univ Eng Technol Ed 2020;50:765–75. https://doi.org/10.1109/ISIVC61350.2024.10577766.

[7] Zhang S, Gong Z, Tao B, Ding H. A visual servoing method based on point cloud. In: 2020 I*EEE international conference on real-time computing and robotics (RCAR)*. IEEE; 2020. p. 369–74. https://doi.org/10.1109/RCAR49640.2020.9303277.

[8] Ye, W., Zhao, Y., & Vela, P.A. (2019). Characterizing SLAM benchmarks and methods for the robust perception age. arXiv preprint arXiv:1905.07808. https://doi.org/10.48550/arXiv.1905.07808.

[9] Delmerico J, Scaramuzza D. A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots. In: 201*8 IEEE international conference on robotics and automation (ICRA)*. IEEE; 2018. p. 2502–9. https://doi.org/10.1109/ICRA.2018.8460664.

[10] Sun K, Mohta K, Pfrommer B, Watterson M, Liu S, Mulgaonkar Y, Kumar V. Robust stereo visual inertial odometry for fast autonomous flight. IEEE Robot Autom Lett 2018;3(2):965–72. https://doi.org/10.1109/LRA.2018.2793349.

[11] Cadena C, Carlone L, Carrillo H, Latif Y, Scaramuzza D, Neira J, Leonard JJ. Past, present, and future of simultaneous localization and mapping: toward the robust-perception age. IEEE Trans Robot 2016;32(6):1309–32. https://doi.org/10.1109/TRO.2016.2624754.

[12] Zhao, Y., Smith, J.S., & Vela, P.A. (2020). Good graph to optimize: cost-effective, budget-aware bundle adjustment in visual slam. arXiv preprint arXiv:2008.10123. https://doi.org/10.48550/arXiv.2008.10123.

[13] De Villiers F, Brink W. Learning fine-grained control for mapless navigation. In: 2020 i*nternational SAUPEC/RobMech/PRASA conference*. IEEE; 2020. p. 1–6. https://doi.org/10.1109/SAUPEC/RobMech/PRASA48453.2020.9041011.

[14] Chen C, Wang B, Lu CX, Trigoni N, Markham A. Deep learning for visual localization and mapping: a survey. IEEE Trans Neural Netw Learn Syst 2023. https://doi.org/10.1109/TNNLS.2023.3309809.

[15] Balntas V, Li S, Prisacariu V. Relocnet: continuous metric learning relocalisation using neural nets. In: In pro*ceedings of the European conference on computer vision (ECCV)*; 2018. p. 751–67. https://doi.org/10.1007/978-3-030-01264-9_46.

[16] Yang N, Wang R, Stuckler J, Cremers D. Deep virtual stereo odometry: leveraging deep depth prediction for monocular direct sparse odometry. In: In proc*eedings of the European conference on computer vision (ECCV)*; 2018. p. 817–33. https://doi.org/10.48550/arXiv.1807.02570.

[17] Balntas V, Li S, Prisacariu V. Relocnet: continuous metric learning relocalisation using neural nets. In: In pr*oceedings of the European conference on computer vision (ECCV)*; 2018. p. 751–67. https://doi.org/10.1007/978-3-030-01264-9_46.

[18] Chen C, Rosa S, Miao Y, Lu CX, Wu W, Markham A, Trigoni N. Selective sensor fusion for neural visual-inertial odometry. In: In proc*eedings of the IEEE/CVF conference on computer vision and pattern recognition*; 2019. p. 10542–51. https://doi.org/10.48550/arXiv.1903.01534.

[19] Abrate F, Bona B, Indri M. Experimental EKF-based SLAM for mini-rovers with IR sensors only. In: EMCR; 2007. https://doi.org/10.1109/TBME.2021.3116514.

[20] Bonato V, Peron R, Wolf DF, de Holanda JA, Marques E, Cardoso JM. An fpga implementation for a kalman filter with application to mobile robotics. In: 200*7 international symposium on industrial embedded systems*. IEEE; 2007. p. 148–55. https://doi.org/10.1109/SIES.2007.4297329.

[21] Yap TN, Shelton CR. SLAM in large indoor environments with low-cost, noisy, and sparse sonars. In: 2009 *IEEE international conference on robotics and automation*. IEEE; 2009. p. 1395–401. https://doi.org/10.1109/ROBOT.2009.5152192.

[22] Eade E, Fong P, Munich ME. Monocular graph SLAM with complexity reduction. In: 2010 *IEEE/RSJ international conference on intelligent robots and systems*. IEEE; 2010. p. 3017–24. https://doi.org/10.1109/IROS.2010.5649205.

[23] Vincke B, Elouardi A, Lambert A. Design and evaluation of an embedded system based SLAM applications. In: 2010 *IEEE/SICE international symposium on system integration*. IEEE; 2010. p. 224–9. https://doi.org/10.1109/SII.2010.5708329.

[24] Magnenat S, Longchamp V, Bonani M, Rétornaz P, Germano P, Bleuler H, Mondada F. Affordable slam through the co-design of hardware and methodology. In: 2010 I*EEE international conference on robotics and automation*. IEEE; 2010. p. 5395–401. https://doi.org/10.1109/ROBOT.2010.5509196.

[25] Buonocore L, Júnior CLN, de Almeida Neto A. Solving the indoor SLAM problem for a low-cost robot using sensor data Fusion and autonomous feature-based exploration. In: ICINCO (2); 2012. p. 407–14. https://doi.org/10.5220/0004000504070414.

[26] Yi Y, Guan Y. Research on autonomous navigation and control algorithm of intelligent robot based on reinforcement learning. Scalab Comput: Pract Exp 2025;26(1):423–31. https://doi.org/10.12694/scpe.v26i1.3841.

[27] Gervet T, Chintala S, Batra D, Malik J, Chaplot DS. Navigating to objects in the real world. Sci Robot 2023;8(79):eadf6991. https://doi.org/10.1126/scirobotics.adf6991.

[28] Kanayama H, Ueda T, Ito H, Yamamoto K. Two-mode mapless visual navigation of indoor autonomous mobile robot using deep convolutional neural network. In: In 2020 IEEE/SICE international symposium on system integration (SII). IEEE; 2020. p. 536–41. https://doi.org/10.1109/SII46433.2020.9025851.

[29] Das S, Mishra SK. A Machine learning approach for collision avoidance and path planning of mobile robot under dense and cluttered environments. Comput Electr Eng 2022;103:108376. https://doi.org/10.1016/j.compeleceng.2022.108376.

[30] Tsai CY, Nisar H, Hu YC. Mapless LiDAR navigation control of wheeled mobile robots based on deep imitation learning. IEEE Access 2021;9:117527–41. https://doi.org/10.1109/ACCESS.2021.3107041.

[31] Nguyen A, Tran QD. Autonomous navigation with mobile robots using deep learning and the robot operating system. In: Robot operating system (ROS) the complete reference. 6; 2021. p. 177–95. https://doi.org/10.1007/978-3-030-75472-3_5.

[32] Xiong Y, Zhang X, Peng J, Yu W. 3d depth map based optimal motion control for wheeled mobile robot. In: 2017 IEEE international conference on systems, man, and cybernetics (SMC). IEEE; 2017. p. 2045–50. https://doi.org/10.1109/SMC.2017.8122920.

[33] Li C, Li B, Wang R, Zhang X. A survey on visual servoing for wheeled mobile robots. Int J Intell Robot Appl 2021;5(2):203–18. https://doi.org/10.1109/TSMC.2020.3044347.

[34] Ranftl R, Lasinger K, Hafner D, Schindler K, Koltun V. Towards robust monocular depth estimation: mixing datasets for zero-shot cross-dataset transfer. IEEE Trans Pattern Anal Mach Intell 2020;44(3):1623–37. https://doi.org/10.1109/TPAMI.2020.3019967.

[35] Bhat, S.F., Birkl, R., Wofk, D., Wonka, P., & Müller, M. (2023). Zoedepth: zero-shot transfer by combining relative and metric depth. arXiv preprint arXiv:2302.12288. https://doi.org/10.48550/arXiv.2302.12288.

[36] Yang L, Kang B, Huang Z, Xu X, Feng J, Zhao H. Depth anything: unleashing the power of large-scale unlabeled data. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition; 2024. p. 10371–81. https://doi.org/10.48550/ARXIV.2401.10891.

[37] Agarwal V, Bansal G. Automatic number plate detection and recognition using YOLO world. Comput Electr Eng 2024;120:109646. https://doi.org/10.1016/j.compeleceng.2024.109646.

[38] Tripathy HK, Mishra S, Thakkar HK, Rai D. CARE: a collision-aware mobile robot navigation in grid environment using improved breadth first search. Comput Electr Eng 2021;94:107327. https://doi.org/10.1016/j.compeleceng.2021.107327.

[39] Rondoni C, Scotto di Luzio F, Tamantini C, Tagliamonte NL, Chiurazzi M, Ciuti G, Zollo L. Navigation benchmarking for autonomous mobile robots in hospital environment. Sci Rep 2024;14(1):18334. https://doi.org/10.1038/s41598-024-69040-z.