Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

E^{2}MN: human-inspired end-to-end mapless navigation with oscillation suppression and short-term memory*#

Yinan YANG[§1], Zhiye WANG[§1], Xuan KONG[2], Peng ZHI[1], Dapeng ZHANG[1], Rui ZHOU[‡1], Qingguo ZHOU[1]

[1]*School of Information Science & Engineering, Lanzhou University, Lanzhou 730000, China*

[2]*Freelance Researcher, Jining 272000, China*

E-mail: yangyn2023@lzu.edu.cn; wzhiye2023@lzu.edu.cn; 3076201331@qq.com; zhip13@lzu.edu.cn; zhangdp22@lzu.edu.cn; zr@lzu.edu.cn; zhouqg@lzu.edu.cn

Received May 29, 2025; Revision accepted Oct. 26, 2025; Crosschecked Nov. 24, 2025

Abstract: Robotic navigation in unknown environments is challenging due to the lack of high-definition maps. Building maps in real time requires significant computational resources. Nevertheless, sensor data can provide sufficient environmental context for robots' navigation. This paper presents an interpretable and mapless navigation method using only two-dimensional (2D) light detection and ranging (LiDAR), mimicking human strategies to escape from dead ends. Unlike traditional planners, which depend on global paths or vision-based and learning-based methods, requiring heavy data and hardware, our approach is lightweight and robust, and it requires no prior map. It effectively suppresses oscillations and enables autonomous recovery from local minimum traps. Experiments across diverse environments and routes, including ablation studies and comparisons with existing frameworks, show that the proposed method achieves map-like performance without a map—reducing the average path length by 50.51% when compared to the classical mapless Bug2 algorithm and increasing it by only 17.57% when compared to map-based navigation.

Key words: Vector field histogram; Density-based spatial clustering of applications with noise (DBSCAN); Oscillation suppression; Temporary goal prediction

https://doi.org/10.1631/FITEE.2500348 **CLC number:** TP242

1 Introduction

"Where can a wine shop be found to drown his sad hours? A cowherd boy points to a cot amid apricot flowers."

This classical Chinese poem reflects that, long before the advent of high-definition maps, humans could still reach destinations by following general directions. Even in unfamiliar environments, humans navigate effectively using short-term memory. Typically, humans tend to move in a straight line toward their goal. When encountering a dead end or realizing that the target is blocked by an obstacle, they briefly remember that the path is inaccessible and seek alternatives. One of the primary goals of robotic navigation is to replicate such human behavior, making this human-like navigation strategy suitable for robots as well.

Traditional autonomous robot navigation relies on coarse-grained global planning over prior maps to guide the robot toward its goal. The navigation

‡ Corresponding author
§ These two authors contributed equally to this work
* Project supported by the Ling Chuang Research Project of China National Nuclear Corporation (No. CNNC-LCKY-2025-098), the Gansu Province Science and Technology Major Project-Industrial Project (Nos. 22ZD6GA048 and 23ZDGA006)
# Electronic supplementary materials: The online version of this article (https://doi.org/10.1631/FITEE.2500348) contains supplementary materials, which are available to authorized users
ORCID: Yinan YANG, https://orcid.org/0009-0007-1626-5141; Zhiye WANG, https://orcid.org/0009-0002-0116-2477; Rui ZHOU, https://orcid.org/0000-0002-9968-6190

process usually includes two stages: global planning and local planning. First, a global planner provides a global solution based on prior maps, and then a local planner continuously refines the path under its guidance (Zhong et al., 2023). However, in fully unknown scenarios, such as forest rescue or battlefield operations, where only the approximate direction and distance are known, robots must navigate directly using perception without global maps. Under such conditions, the navigation task faces the following key challenges:

1. Planning a short and smooth path toward the goal without a prior map and with limited hardware;

2. Preventing the system from falling into local optima during navigation;

3. Ensuring safe, collision-free navigation without global information.

In previous research, various advances have been made in mapless navigation, and several influential studies have been published. Many existing methods have achieved navigation in relatively simple environments, and some have enabled escape from local minima. However, these approaches still have notable limitations. Traditional local planners and reinforcement learning methods that directly output linear and angular velocities can navigate non-maze scenarios but tend to fall into local optima and struggle to escape from dead ends. Human-inspired navigation techniques often rely on visual input, which imposes high hardware requirements, whereas two-dimensional (2D) light detection and ranging (LiDAR) sensors already capture sufficient environmental information. Therefore, this work adopts LiDAR-based navigation. Existing strategies for escaping from local minima generally fall into two categories. One constructs topological maps from perception data, which suffer from step size constraints (long steps may overlook narrow passages, while short steps increase memory costs). The other predicts temporary goals directly, offering better spatial efficiency, but often lacks interpretability and produces unreasonable goal selection. This paper adopts the temporary goal-based approach and improves both the interpretability and rationality of goal prediction. Based on this, the main contributions of this paper are as follows:

1. It proposes an end-to-end mapless navigation method, namely, E^{2}MN, which mimics human behaviors for collision-free navigation in unknown environments;

2. It improves local navigation by predicting and suppressing oscillations at frequently repeated positions;

3. It designs a temporary goal generation strategy based on 2D LiDAR data to help the robot escape from dead ends or narrow obstacle-induced local minima.

2 Related works

2.1 Commonly used local planning algorithms

The commonly used local planning algorithms include timed elastic band (TEB) (Rösmann et al., 2012), artificial potential field (APF) (Khatib, 1985), dynamic window approach (DWA) (Fox et al., 1997), virtual force field (VFF) (Borenstein and Koren, 1989), and velocity obstacles (VO) (Fiorini and Shiller, 1998; Chaudhary and Ghose, 2017). These algorithms are typically implemented based on grid maps, and the conversion from raw sensor data to a usable map requires additional spatial and temporal resources. Therefore, this work selects the vector field histogram (VFH) algorithm (Borenstein and Koren, 1991), which shares the same origin as APF and VFF, as the baseline. VFH computes navigation decisions based on angular sectors formed from surrounding environments, allowing it to operate using only 2D LiDAR data even in the absence of a map (Ortiz et al., 2019). The VFH algorithm also improves upon its predecessor VFF by resolving the issue of blocking at narrow passages (Borenstein and Koren, 1989). There are also algorithms specifically designed for mapless scenarios, such as the Bug family of algorithms (Lumelsky and Stepanov, 1987), which suffer from issues like unsmooth paths and frequent wall collisions. Kamon et al. (1996) proposed the TangentBug algorithm, which directly guides the robot toward the edge of obstacles within the sensing boundary, partially improving corner handling. Traditional local planners are prone to local minima or oscillatory behaviors and rely heavily on maps, while Bug-type algorithms often lead to excessive detours in indoor navigation. The method proposed in this paper incorporates the Bug-style idea to detect and suppress oscillations while improving path efficiency.

2.2 Modifications to the VFH algorithm

The traditional VFH algorithm considers robot dimensions inadequately, and most implementations still rely on grid maps. Many improvements have been proposed to address these limitations. Ulrich and Borenstein (1998) introduced the VFH+ algorithm, which applies multi-level processing to the histogram and incorporates robot width, although not rigorously. Balan et al. (2019) employed fuzzy evaluation of perception data to generate smoother paths. Ulrich and Borenstein (2000) proposed VFH*, which achieves a global-planning-like behavior through path prediction, but it requires a prior map. Wu et al. (2021) combined VO with VFH by masking histogram sectors corresponding to collision cones. Li et al. (2016) bypassed map construction and used raw sensor data for real-time detection of new obstacles, improving responsiveness. Similarly, Meng et al. (2018) used three-dimensional (3D) sensor input to compute polar histograms based on the distance to the nearest obstacle in each sector, avoiding grid-based maps. Ortega et al. (2024) enhanced the obstacle avoidance safety by directly masking sectors within the robot's minimum turning radius. From a human–robot interaction perspective, Pappas et al. (2020) introduced a framework to arbitrate between sudden user inputs and autonomous commands, reducing the communication frequency while improving the efficiency. To address the issue of ignoring narrow passages in mapless conditions, Zhang YF and Wang (2017) proposed guiding the robot toward potential openings between obstacles. This paper draws inspiration from strategies for detecting hidden paths and sensor-driven navigation, while further addressing limitations such as prior map dependency and underestimated feasible space in previous approaches.

2.3 Deep learning and reinforcement learning models

In addition to traditional methods, mapless navigation commonly leverages perception-based deep reinforcement learning (DRL) models. The combination of deep learning and reinforcement learning was first introduced by Mnih et al. (2013). In the navigation domain, Jin et al. (2020) designed reward functions based on multi-sensor perception, assigning different penalties for proximity to pedestrians and dynamic and static obstacles, encouraging the robot to stay away from moving agents. Tai et al. (2017) formulated input scenes as sparse matrices to reduce training cost and enhance the generalization. Zhou et al. (2022) integrated long short-term memory (LSTM) into reinforcement learning to introduce long-term memory in mapless navigation. Zou et al. (2024) applied a double deep Q-network (DDQN) model to generate navigation actions, while Marchesini and Farinelli (2020) used discrete deep reinforcement learning with parallel asynchronous training and prioritized experience replay to shorten the training time. Cheng et al. (2023) addressed local minima by rewarding the robot for escaping from dead zones. To further handle dead ends caused by missing maps, various reinforcement learning-based approaches have been proposed. Jang et al. (2022) introduced the hindsight-intermediate-target-based reinforcement learning (HIT-RL) model, which selects intermediate goals to avoid dead ends by repeatedly choosing from left, front, or right based on learned parameters. Bektaş and Bozma (2022) combined APF with reinforcement learning using only 2D LiDAR, where the model generates potential field parameters such as temporary goals and attraction/repulsion values instead of direct control commands. This work draws inspiration from these approaches that infer temporary goals based on perception. Human behavior imitation algorithms are often vision-based, as human spatial awareness primarily relies on visual input. For example, Wapnick et al. (2021) and Liang et al. (2024) directly used images to determine feasible paths, while Xie et al. (2020) combined geographic instructions with visual navigation to reach the target. Some methods also mimic biological navigation; Mathews et al. (2009) proposed a memory-based coordinate marking strategy inspired by insects to navigate without maps. This work adopts the idea of memory-based coordinate references, while reducing hardware requirements and enhancing the interpretability of decision-making.

2.4 Emerging mapless navigation methods beyond conventional approaches

Beyond commonly used methods such as DRL and Bug-based algorithms, several less common but effective approaches have emerged. Ort et al. (2018) constructed topological maps based on sensor data in the form of Voronoi diagrams, replacing grid maps

without relying on prior maps. Zhong et al. (2023) also built a topological map using a dynamic visible topology tree (DVT-Tree) to plan paths, allowing the robot to backtrack in dead zones and expand branches into unexplored areas. Chaudhary and Ghose (2017) proposed calculating minimum covering circles and extracting collision cones by estimating the maximum and minimum angles to handle irregular obstacles. Chen et al. (2025) used visual data to generate topological representations. Xu et al. (2025) also employed topological maps in traffic flow prediction research. Yan et al. (2023) applied imitation learning to derive navigation strategies from expert demonstrations. Ali and Liu (2023) proposed boundary-based navigation using Gaussian probability models to select boundary points, although the model requires training data and is unsuitable for environments where the map size is smaller than the boundary distance. In fully unknown environments, key geographic information is often extracted from point clouds. For instance, Przewodowski and Osório (2022) used coarse surface features for localization without requiring fine-grained sensors, though the method still depends on a map. Miao et al. (2021) applied density-based spatial clustering of applications with noise (DBSCAN) to classify obstacles, while Zhang W et al. (2021) used deep learning to directly identify navigation-critical points from point clouds. This work adopts a DBSCAN-inspired approach to extract key target points and addresses the limitations of prior methods, including poor performance in detecting hidden paths, high hardware requirements, and limited adaptability in narrow spaces.

2.5 Explainability of DRL

DRL methods in complex tasks usually rely on deep neural networks to approximate policy functions or value functions. The highly nonlinear and multi-layered structures of these networks make their decision processes exhibit significant "black-box" characteristics, which are difficult to interpret intuitively (Rudin, 2019). At present, our understanding of the quality of explainable and transparent DRL methods has remained limited, and the challenge of making DRL models interpretable is substantial (Vouros, 2022). Moreover, the availability of limited data has been shown to lead to epistemic uncertainty (Clements et al., 2020). In contrast, the algorithm proposed in this paper does not rely on large-scale training data but instead makes decisions based on a set of deterministic rules. During the path planning process, decisions such as moving forward or halting, updating the navigation goal, and selecting temporary sub-goals all follow clear logical rationales, thereby providing strong interpretability. Furthermore, the design concept of this algorithm is inspired by the human "route memorization" behavior: When a path is found to be blocked, one immediately abandons that direction; When a potential exit is recognized, one promptly turns to explore it. In the algorithmic implementation, this process is reflected in the judgment and recording mechanisms of obstacle regions (ob regions) and candidate goal regions (end2 regions). These mechanisms render the decision-making process intuitively understandable.

3 Problem statement

A typical robot navigation system works as follows: First, a grid map is built from sensor data. An inflation layer is added to avoid hits. A global path is made using this map. If new blocks are seen from new sensor data, local planning starts to avoid them in real time.

This method needs high-definition maps and renders prior knowledge of the area indispensable. However, in real tasks such as outdoor rescue, prescanning and map-making are often not possible. Here, navigation must work without a map. Humans, however, can find paths in new areas using memory and new area information. Thus, this paper works on the following tasks:

1. Determining whether a given coordinate lies in free space using only the current LiDAR sensor data, without relying on an inflated grid map;

2. Navigating in complex environments without global path guidance, using only sensor data and short-term memory;

3. Perceiving oscillatory behaviors in the navigation path and proactively predicting and suppressing such oscillations.

Detailed solutions to the above tasks are presented in Section 4.

4 Method

4.1 Underlying algorithm

The proposed method uses real-time perception and short-term memory. Upon receiving a goal point, the robot heads straight for the target. Although VFH enables easy obstacle avoidance in simple cases, complex environments often cause local minima. This paper thus presents strategies to prevent, detect, and eliminate local minima.

In APF, local minima are found using the zero resultant force. However, in VFH, local minima, which cause small-area oscillations, can be detected by spotting oscillations (as detailed in Section 4.2). Once found, the method mimics human behaviors: It marks the area as a local minimum zone and uses sensor data to find a temporary escape goal. If found, the robot goes there, memorizes the spot, marks its neighborhood as a temporary goal zone, applies a penalty, and adjusts thresholds to avoid re-entry. Section 4.3 details the temporary goal selection.

To prevent local minima, this method incorporates the Bug family algorithms, which avoid local minima in mapless navigation when a path exists. It adopts Bug-style wall-following without constant turns. The integration method is described in Section 4.4.

The algorithm framework is shown in Fig. 1.

4.1.1 Passability assessment and sector binarization

The algorithm works directly on 2D LiDAR polar data without converting to grid maps. Sector i's binarization depends on whether distance l along its center line is collision-free. Common methods only check radial distances in adjacent sectors, raising crash risks with side obstacles. Thus, this paper uses an improved approach that compares radial distances at multiple points along the sweeping path with an obstacle threshold to detect potential collisions. Sector i's binarization rule is shown in Eq. (1) at the bottom of this page.

Here, B_i is the binarization result of sector i,

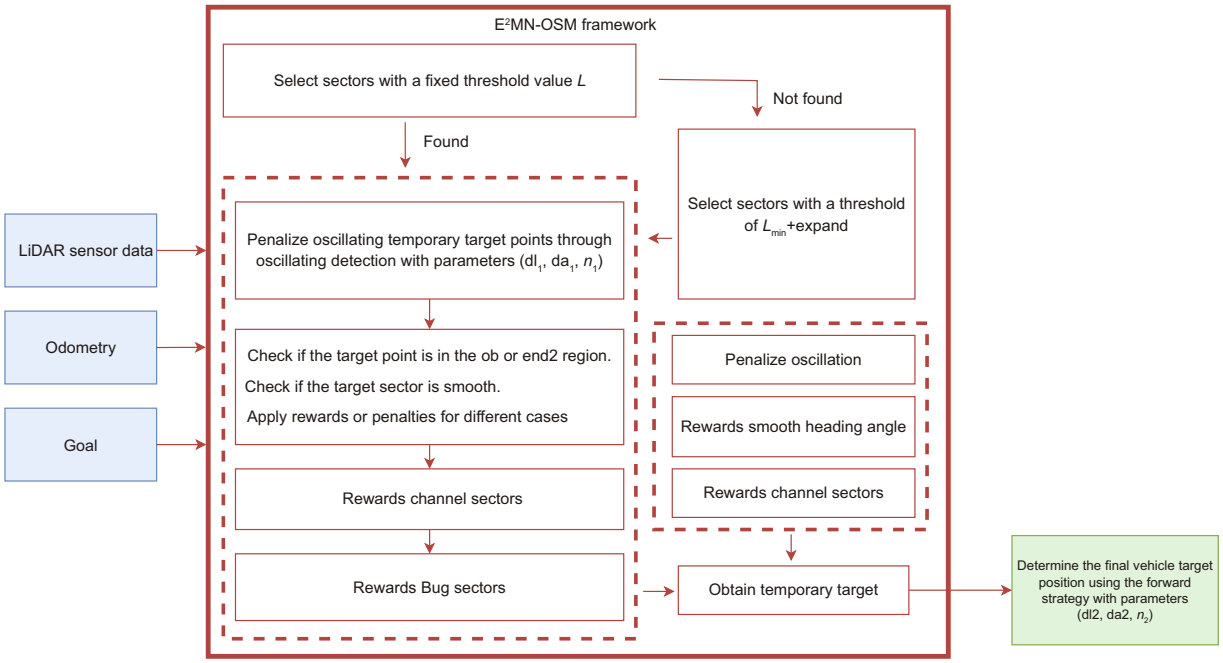

Fig. 1 End-to-end mapless navigation (E^2MN) framework

$$B_i(l) = \left(\wedge^{j}_{|\theta_j-\theta_i|<\arctan(\frac{\text{expand}}{l})} \text{len}_j \geq l\cos|\theta_j-\theta_i| + \sqrt{\text{expand}-(l\sin(\theta_j-\theta_i))^2} \right) \wedge \left(\wedge^{j}_{\arctan(\frac{\text{expand}}{l})<|\theta_j-\theta_i|<\frac{\pi}{2}} \text{len}_j \geq \frac{\text{expand}}{\sin|\theta_j-\theta_i|} \right). \tag{1}$$

θ_i is the center line angle (radian) of sector i, len_i is the obstacle distance in that sector, expand is the robot's radius, and l is a fixed threshold for judging safe passage. Sector i is a candidate valley if B_i is true.

4.1.2 Underlying implementation of the VFH algorithm

The target point is determined by selecting the most suitable valley from all candidate valleys. The basic evaluation metric is the Euclidean distance (length) from the valley to the goal, which will be subsequently adjusted with rewards and penalties.

As shown in Fig. 2, if the step size is too large, hidden paths may be missed. Therefore, the forward distance in this method is limited and does not exceed one-fifth of the maximum detection range. On the other hand, a small step size may lead to excessive memory usage when storing path information or generating vector maps (Zhong et al., 2023). This issue is addressed using short-term memory. To select a sector direction that is both open and as close as possible to the goal, the forward distance must also be less than or equal to the Euclidean distance between the current point and the goal, preventing repeated overshooting near the goal. For each sector, the radial length is set as $L_i = \min(\text{shrink}\cdot\text{len}_i, ||\boldsymbol{X}_{\text{end}} - \boldsymbol{X}_{\text{now}}||)$, where $\boldsymbol{X}_{\text{end}}$ is the planar coordinate of the final goal, $\boldsymbol{X}_{\text{now}}$ is the current position, and shrink is a scaling factor between 0 and 1. Since L_i is not identical to l, an additional binarization check is required. If L_i is found to be not passable, the forward distance is reverted to l, and the optimal sector is selected accordingly, as shown in Eq. (2):

$$\begin{cases} \boldsymbol{X}_{\text{pur}_i} = L_i(\cos\theta_i, \sin\theta_i) | L_i = \min(\text{shrink}\cdot\text{len}_i, \\ \qquad\qquad \|\boldsymbol{X}_{\text{end}} - \boldsymbol{X}_{\text{now}}\|), \\ \boldsymbol{X}_{\text{pur}} = \boldsymbol{X}_{\text{pur}_i} | \text{length}_i = \min\limits_i ||\boldsymbol{X}_{\text{pur}_i} - \boldsymbol{X}_{\text{end}}||, \\ \boldsymbol{X}_{\text{pur}_{\text{true}}} = l\left(\frac{||\boldsymbol{X}_{\text{pur}}||}{l}\right)^{B(||\boldsymbol{X}_{\text{pur}}||)} \hat{\boldsymbol{X}}_{\text{pur}}. \end{cases} \tag{2}$$

Here, $\boldsymbol{X}_{\text{pur}}$ denotes the planar coordinate of the temporary goal, $\boldsymbol{X}_{\text{pur}_i}$ denotes the planar coordinate of the temporary goal represented by sector i, length_i is the evaluation metric for each candidate goal, $\boldsymbol{X}_{\text{pur}_{\text{true}}}$ is the final selected target point, and $\hat{\boldsymbol{X}}_{\text{pur}}$ represents $\boldsymbol{X}_{\text{pur}}/||\boldsymbol{X}_{\text{pur}}||$. $B(\cdot)$ is the binarization function used to determine whether a given distance is safely passable; The candidate valleys mentioned earlier are selected based on whether $B_i(l)$ is true. However, in extreme cases, all sectors may return false after binarization. In such situations, this method introduces another key mechanism: dual thresholds. Since all obstacle distances across sectors have already been evaluated, the minimum absolute distance $L_{\min}$ can be easily obtained. The original l is then redefined as $L_{\min}$, and if a feasible path objectively exists, a candidate valley will be found under this new threshold. In other words, the requirement for path openness is relaxed to ensure the continuity of algorithm execution. This method chooses to binarize using a fixed threshold l instead of $L_i = \min(\text{shrink}\cdot\text{len}_i, ||\boldsymbol{X}_{\text{end}} - \boldsymbol{X}_{\text{now}}||)$, because L_i depends on obstacle distance len_i and may guide the robot into narrow areas along the direct path to the goal, causing it to stall, as shown in Fig. 3.

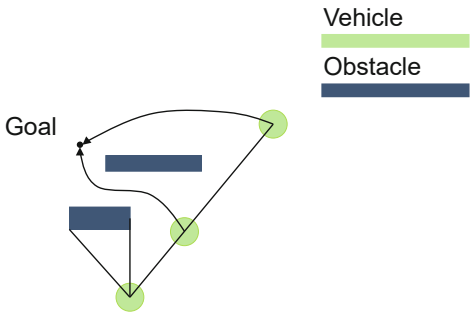

Fig. 2 Difference in navigation paths between long and short step sizes

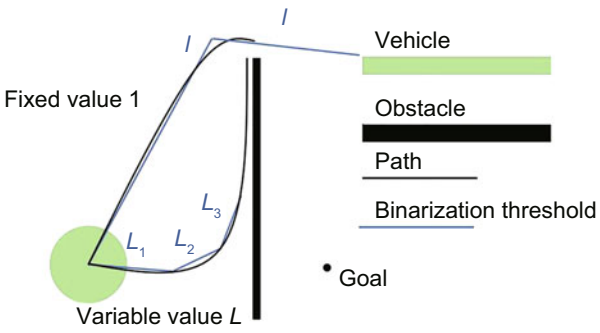

Fig. 3 Paths under different threshold selections

4.2 Detection, prevention, and elimination of local minima

4.2.1 Detection of local minima

VFH's persistent optimal sector selection causes oscillatory paths during local optima. This method detects oscillations by comparing the current trajectories with historical paths. Two oscillation patterns exist: (1) one involves sharp directional oscillation; (2) the other involves looping around several nearby points. These oscillatory paths show spatial proximity, temporal continuity, and repetition, demonstrating clear spatiotemporal locality. Thus, storing only recent path nodes suffices for local minima detection.

Local minima are thus identified through the procedure described in Algorithm 1.

Here, path denotes the sequence of past path nodes, ra is the current heading angle, and L is the forward distance in the direction of ra; dl is the threshold distance used to determine the looping behavior in the oscillatory case (b), and da is the angular threshold used to detect sharp oscillation in oscillatory case (a); Given the locality of oscillatory paths, n specifies the number of recent path nodes used for backtracking.

Algorithm 1 Is_collision(path, L, ra, dl, da, n)

Require: Path, double L, double ra, double dl, double da, int n
Ensure: bool is_collision

$\boldsymbol{X}_{\text{pur}} \leftarrow (L\cos(\text{ra}),\ L\sin(\text{ra}))$
is_collision $\leftarrow$ false
len $\leftarrow$ path.size
$\text{ra}_{\text{last}} \leftarrow \arctan\left(\frac{\text{path[len]}.y-\text{path[len}-1].y}{\text{path[len]}.x-\text{path[len}-1].x}\right)$
is_collision $\leftarrow$ | ra $-$ ra_{last} | $<$ da
if is_collision **then**
    return is_collision
else
    for $i = \text{len}-1$ to $\max(0,\ \text{len}-n)$ **do**
        $\boldsymbol{X}_{\text{path}} \leftarrow (\text{path}[i].x,\ \text{path}[i].y)$
        is_collision $\leftarrow$ is_collision $\vee$ $\|\boldsymbol{X}_{\text{pur}} - \boldsymbol{X}_{\text{path}}\|^2 < \text{dl}^2$
        if is_collision **then**
            return is_collision
        end if
    end for
end if
return is_collision

4.2.2 Prevention of local minima

To prevent local minima during the selection of suitable valleys, this method introduces reward and penalty coefficients as a preventive strategy, as shown in Eq. (3) at the bottom of this page.

Here, repeat is a penalty coefficient > 1; dl_1, da_1, and n_1 are input parameters of Algorithm 1; Is_collision$(\cdot)$ is the function used in Algorithm 1 to determine whether the robot has entered a local minimum.

After entering a navigation dead zone, humans typically respond by walking straight in one direction to explore possible exits rather than retracing their steps, unless no other options are available. They also tend to remember areas that are not passable and avoid returning to them if lost again.

Human behavior is reflected in the algorithm by marking the neighborhood around the oscillation point as the ob region, followed by recomputing the optimal sector. This time, target points within the ob region are penalized, while sectors with headings close to the previous direction are rewarded to reflect the human tendency not to backtrack. If the newly computed target point leads the robot out of the oscillation state, it proceeds toward that point. If oscillation still occurs after this reward–penalty adjustment, a temporary goal must be selected (see Section 4.3 for details). After reaching the temporary goal, the robot re-navigates toward the final goal, and the area around the temporary goal is marked as the end2 region. To avoid revisiting, target points in the end2 region are also penalized in future planning. The temporary goal is expected to guide the robot quickly out of the local minimum while minimizing unnecessary detours, so during this process, penalties on the ob and end2 regions are ignored. The heading deviation from previous directions is evaluated, and similar directions are rewarded to suppress type-A oscillations. Among candidates with the same score, the algorithm favors those in open areas. Therefore, after applying rewards and penalties, the final score is further reduced by the obstacle distance detected in the temporary goal's sector, and the improved formula is updated from Eq. (3) to Eq. (4) at the top of the next page.

Here, $\boldsymbol{X}_{\text{origin}}$ represents the initially specified global goal; beside_ob_i indicates whether the target point in sector i lies within the ob region, and beside_end2_i indicates whether it lies within the

$$\boldsymbol{X}_{\mathrm{pur}}=\boldsymbol{X}_{\mathrm{pur}_i}\,\big|\,\mathrm{length}_i=\min_i\Big(\mathrm{repeat}^{\,\mathrm{Is\_collision(path},L_i,\mathrm{ra}_i,\mathrm{dl}_1,\mathrm{da}_1,n_1)}\cdot\|\boldsymbol{X}_{\mathrm{pur}_i}-\boldsymbol{X}_{\mathrm{end}}\|\Big). \tag{3}$$

$$\begin{aligned}
&\mathrm{beside_ob}_i=\{\boldsymbol{X}_{\mathrm{end}}=\boldsymbol{X}_{\mathrm{origin}}\}\wedge\left(\vee_{j\le\mathrm{OB_size}}\|\boldsymbol{X}_{\mathrm{pur}_i}-\boldsymbol{X}_{\mathrm{ob}_j}\|<\mathrm{len_ob}\right),\\
&\mathrm{beside_end2}_i=\{\boldsymbol{X}_{\mathrm{end}}=\boldsymbol{X}_{\mathrm{origin}}\}\wedge\left(\vee_{j\le\mathrm{end2_size}}\|\boldsymbol{X}_{\mathrm{end}_j}-\boldsymbol{X}_{\mathrm{pur}_i}\|<\mathrm{len_end2}\right),\\
&\mathrm{is_smooth}_i=\mid\theta_i-\theta_{\mathrm{last}}\mid<\mathrm{da},\\
&\boldsymbol{X}_{\mathrm{pur}}=\boldsymbol{X}_{\mathrm{pur}_i}\Big|\mathrm{length}_i=\min_i\Big(\big(\mathrm{repeat}^{\mathrm{Is_collision(path},L_i,\mathrm{ra}_i,\mathrm{dl}_1,\mathrm{da}_1,n_1)}\cdot\mathrm{pun_end2}^{\mathrm{beside_end2}_i}\cdot\mathrm{pun_ob}^{\mathrm{beside_ob}_i}\\
&\qquad\cdot\mathrm{smooth}^{\mathrm{is_smooth}_i}\cdot\|\boldsymbol{X}_{\mathrm{pur}_i}-\boldsymbol{X}_{\mathrm{end}}\|\big)-0.01\cdot\min(5,\mathrm{len}_i)\Big).
\end{aligned} \tag{4}$$

end2 region; OB_size and end2_size represent the maximum number of ob and end2 points considered for penalty evaluation respectively, reflecting the limit of short-term memory; len_ob and len_end2 are the radii of the circular ob and end2 regions, respectively; θ_i is the angle of the i^{th} sector, and θ_{last} is the current heading angle; da is the threshold for determining whether the path is smooth; pun_end2 and pun_ob are penalty coefficients >1 for target points falling within the end2 and ob neighborhoods; smooth is a reward factor between 0 and 1, which encourages path smoothness.

4.2.3 Elimination of local minima

This paper eliminates oscillations caused by local minima by predicting them in advance and replanning the trajectory accordingly.

Even with the preventive reward–penalty strategy introduced in the previous subsection, various extreme cases still occur during experiments. The following presents solutions for each of these cases.

If the target point still causes oscillation after penalizing potential oscillatory candidates, the robot does not proceed. Instead, the current position is marked as the ob point, and a new suitable target point is recalculated.

If oscillation still occurs after marking the current position as an ob point and selecting a new target, a temporary goal must be selected to help the robot escape from the local minimum as quickly as possible. In this paper, the terms "target" $\boldsymbol{X}_{\mathrm{pur}}$ and "goal/destination" $\boldsymbol{X}_{\mathrm{end}}$ should be distinguished. A target is a dynamic, short-term objective generated from sensor data for local guidance, while a goal (or destination) is a long-term objective for the overall navigation task.

If oscillation still occurs after switching to a new temporary goal, or if no suitable temporary goal can be found after oscillation, resulting in the robot being unable to move forward, this situation is considered as having no viable path. In such cases, similar to human behaviors when all paths appear blocked, the robot is allowed to tolerate oscillation temporarily and proceed, revisiting previously explored areas in search of undiscovered routes.

If the goal $\boldsymbol{X}_{\mathrm{origin}}$ could be directly reached but the algorithm suggested another direction, the robot ignores the algorithm's guidance and proceeds straight to the goal. If the currently reachable goal is not the global goal end but rather a temporary goal end2, then it is necessary to check whether the number of current candidate temporary goals is <3. Since an exit may contain a pair of goal points, a candidate number <3 represents the fact that the position of end2 is likely located in the direction of a unique temporary exit. In this case, the robot proceeds directly to end2.

If changing the goal temporarily relieves oscillation but leads to new oscillations in other areas, there are two options: selecting another temporary goal or switching back to the original goal. In this method, temporary goals are searched within the visible range (see Section 4.3), and to avoid unnecessary detours, penalties for target points in the ob and end2 regions are disabled during this process. If oscillation still occurs under these conditions, it indicates that the robot has deviated from the intended trajectory due to various factors. In such cases, continuing to search for new temporary goals may lead to further deviation from the final objective. Therefore, this method chooses to revert to the original goal.

To quantify the above situations, this method introduces a global variable numofover to represent the severity of the current oscillation. This variable is initialized whenever the navigation goal is

changed and remains effective throughout the navigation process. If the target point computed in the current iteration causes oscillation, numofover is incremented by 1. If the current position is within the influence range of an obstacle and the current goal is the original goal $\boldsymbol{X}_{\text{origin}}$, then regardless of the computed temporary goal $\boldsymbol{X}_{\text{pur}}$, the system carries a certain risk of oscillation. At this point, the overflow counter flag numofover is set to 1, and the navigation goal is switched to the temporary goal. This change of the target becomes effective in the next navigation cycle, while the previous temporary goal $\boldsymbol{X}_{\text{pur}}$ remains valid. Regardless of whether the current position is within the obstacle influence range, the system updates the target $\boldsymbol{X}_{\text{pur}}$ and numofover uniformly according to the following rules based on the actual values:

When the current navigation goal is the original goal $\boldsymbol{X}_{\text{origin}}$ and numofover >1 while a temporary goal is available, the temporary goal replaces the navigation goal and a new target point $\boldsymbol{X}_{\text{pur}}$ is selected. To ensure rapid escape from the current local minimum, numofover is reset to 0. If the current navigation goal is a temporary goal and numofover>2, the navigation goal is switched back to the original goal. Since the main purpose of the temporary goal is to escape from the local minimum, continued oscillation under this condition indicates a high level of severity, and numofover is reinitialized to 1.

A variable numofstop is introduced to represent the number of times the robot fails to move due to invalid target points $\boldsymbol{X}_{\text{pur}}$. When numofstop accumulates beyond a certain threshold, the robot accepts and proceeds toward the oscillatory target point $\boldsymbol{X}_{\text{pur}}$. When the navigation goal is a temporary goal, the priority is to escape from the current environment, so the acceptance threshold for numofstop is lower, making the robot more tolerant of oscillation.

The pseudocode of the above algorithm is provided in the supplementary materials; since the algorithm is described in detail within the main text, it is not included here.

4.2.4 Dual verification mechanism for local minima

As described in the previous sections, both the penalty for oscillatory targets and the final decision rely on two levels of local minimum detection. For the same L and ra, using different values of n, dl, and da may lead to different results. To minimize the occurrence of oscillation, different parameter sets are used for the two checks, with the final detection being stricter than the penalty detection. Specifically, the condition $\text{dl}_2 > \text{dl}_1$, $\text{da}_2 > \text{da}_1$, and $n_2 > n_1$ must be satisfied. The rationale is that if the robot is still guided toward an oscillatory target despite a lenient penalty check, it suggests that oscillation is difficult to avoid in that scenario. Therefore, the elimination strategy described in Section 4.2.3 must be applied.

4.3 Temporary goal selection strategy

This section describes in detail the process of selecting the temporary goal, as referenced multiple times in previous sections.

4.3.1 Sensor data reconstruction

Temporary goals are categorized into two types as shown in Fig. 4: wall-edge type and entrance type. The detection logic is straightforward; Identify prominent discontinuities within long-range sequences of consecutive point clouds to compute a temporary goal. However, this simple method is vulnerable to small discrete obstacles. For example, in Fig. 4b, the gap between table legs may be mistakenly identified as a temporary goal, though navigating under a table typically leads nowhere. Therefore, the first step in selecting a temporary goal is to reconstruct the sensor data by filtering out small discrete objects and retaining only continuous wall-like structures.

This method uses DBSCAN clustering to reconstruct sensor data by dividing the point clouds into noise points and clusters. Small discrete obstacles, representing valid sensor readings, are treated as small obstacles and walls. Their distances, reflecting

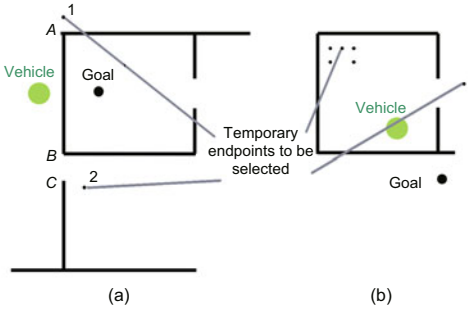

Fig. 4 Intermediate target identification patterns: (a) dual-mode temporary destinations; (b) false-positive recognition at table leg intervals

the actual LiDAR measurements, are replaced by the nearest wall-type point's distance. This substitutes table leg distances with those of nearby walls, as shown in Algorithm 2.

Algorithm 2 Reconstruction(scanOri)

Require: LaserScan scanOri
Ensure: LaserScan scanRe
DBSCAN(scanOri, 0.5, 5)
for $i = 1$ to scanOri.size **do**
  num[i] $\leftarrow$ scanOri[i].AssociatedCluster.num
end for
scanRe $\leftarrow$ new LaserScan
for $i = 1$ to scanOri.size **do**
  if num[i] < 5 **then**
    $j \leftarrow 0$
    while num[$i + j$] ≤ 5 and num[$i - j$] ≤ 5 **do**
      if num[$i + j$] $>$ num[$i - j$] **then**
        $k \leftarrow i + j$
      else
        $k \leftarrow i - j$
      end if
      num[i] $\leftarrow k$
      scanRe[i] $\leftarrow$ scanOri[k]
      break
    end while
  else
    scanRe[i] $\leftarrow$ scanOri[i]
  end if
end for
return scanRe

4.3.2 Temporary goal selection based on the reconstructed sensor data

As shown in Fig. 4, the obstacle distance in the sector containing the temporary goal is noticeably greater than that in at least one neighboring sector. For example, the wall-edge temporary goal 1 has a longer distance than edge point A, and the passage-type temporary goal 2 is much farther than edge points B and C. Additionally, sectors 1 and 2 are not directly adjacent to A, B, or C, and there are multiple consecutive sectors with significantly greater distances. Therefore, the procedure for finding a temporary goal starts by checking each sector and its immediate neighbors. If a neighbor has a significantly longer distance, the current sector is marked as an edge point, and a fixed number of consecutive sectors in that direction are examined. If multiple consecutive sectors have longer distances, the farthest one is chosen as the temporary goal. This procedure, outlined in Algorithm 3, generates a set of candidate temporary goals.

After obtaining all candidate temporary goals, the most suitable one must be selected. As discussed earlier, each sector's target point is evaluated using a scoring metric, which is also applied to temporary goals. The primary objective is to stay close to the original goal, so the evaluation is based on the Euclidean distance between the temporary goal and the original one. To ensure path smoothness, the heading angle of the candidate goal is compared with the previous heading. Additionally, the goal should avoid inducing oscillation, particularly type 2

Algorithm 3 FindEnd2(scanRe)

Require: LaserScan scanRe
Ensure: double $\boldsymbol{X}_{\text{end2}}$[][2]
width $\leftarrow \left\lfloor \frac{|\operatorname{atan2}(\text{expand}, l)|}{\text{ScanRe.angle\_increment}} \right\rfloor$
for $i = 2$ to scanRe.size $- 1$ **do**
  dLast $\leftarrow$ scanRe.len[$i - 1$] $-$ scanRe.len[i]
  dFuture $\leftarrow$ scanRe.len[$i + 1$] $-$ scanRe.len[i]
  if ($|\text{dFuture}| > 4 \cdot |\text{dLast}| \vee |\text{dLast}| > 4 \cdot |\text{dFuture}|) \wedge ((\text{dFuture} > 1 \wedge \text{dLast} < 1) \vee (\text{dFuture} < 1 \wedge \text{dLast} > 1))$ **then**
    if dFuture > 1 **then**
      for $j = i$ to $i +$ width **do**
        if scanRe.len[j] $-$ scanRe.len[i] > 1 **then**
          $x \leftarrow$ scanRe.len[j] $\cdot$ cos(scanRe.ra[j])
          $y \leftarrow$ scanRe.len[j] $\cdot$ sin(scanRe.ra[j])
          $\boldsymbol{X}_{\text{end2}}$.PushBack($(x, y)$)
        end if
      end for
    end if
    if dLast > 1 **then**
      for $j = i -$ width to i **do**
        if scanRe.len[j] $-$ scanRe.len[i] > 1 **then**
          $x \leftarrow$ scanRe.len[j] $\cdot$ cos(scanRe.ra[j])
          $y \leftarrow$ scanRe.len[j] $\cdot$ sin(scanRe.ra[j])
          $\boldsymbol{X}_{\text{end2}}$.PushBack($(x, y)$)
        end if
      end for
    end if
  end if
end for
return $\boldsymbol{X}_{\text{end2}}$

oscillation, since the robot may already be tolerating some level of oscillation. Type 1 temporary goals are not penalized because they may guide the robot back to the original path direction. As shown in Fig. 5, the algorithm prefers path a–b–c over a–b–d, as the latter leads to redundancy. The local minimum detection function Algorithm 1 from Section 4.2.1 is invoked using parameters dl_3, 0, and n_3.

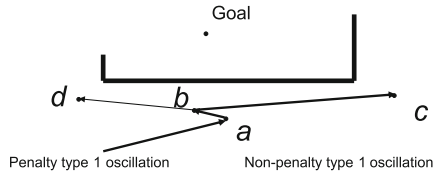

Fig. 5 Selection of temporary goal after receiving oscillation

This method encourages the robot to navigate toward open and spacious areas, so the evaluation score decreases as the obstacle distance in the temporary goal's sector decreases. In addition, it is undesirable for the temporary goal to be repeatedly selected or for the robot to enter regions where oscillations have previously occurred. Therefore, when oscillation occurs, the temporary goal is selected according to Eq. (5) at the bottom of this page. The neighborhood range for determining whether a temporary goal lies within the ob or end2 regions should be larger than the range used for finding $\boldsymbol{X}_{\mathrm{pur}}$, because it is necessary to stay as far as possible from these two regions. If this range is smaller than or equal to that used for $\boldsymbol{X}_{\mathrm{pur}}$, then navigating to the temporary goal may result in a renewed oscillation. Moreover, the smoothing encouragement parameter should be slightly smaller than the one used for $\boldsymbol{X}_{\mathrm{pur}}$, because in this algorithm an unsmoothed path of $\boldsymbol{X}_{\mathrm{pur}}$ represents a minor deviation over a short distance, whereas an unsmoothed direction of $\boldsymbol{X}_{\mathrm{end2}}$ leads to a deviation over a longer distance.

Here, $\mathrm{beside\_ob}_i$ denotes whether the i^{th} candidate temporary goal lies $\boldsymbol{X}_{\mathrm{end2\_near}_i}$ within the ob region, $\mathrm{beside\_end2}_i$ denotes the criterion for determining whether the i^{th} candidate temporary goal has been repeatedly selected, $\mathrm{is\_smooth}_i$ indicates whether the next heading angle of the path is smooth, is_collision determines whether oscillation occurs, $\boldsymbol{X}_{\mathrm{end2}}$ represents the final selected temporary goal, and $\boldsymbol{X}_{\mathrm{end2\_near}_i}$ denotes the i^{th} candidate temporary goal. $\boldsymbol{X}_{\mathrm{end2}_i}$ represents the i^{th} temporary goal that has already been visited. Variable "near" is the evaluation metric for the temporary goal, smooth_near is the reward factor for path smoothness, repeat_near is the penalty term for oscillation, pun_end2_near is the penalty coefficient for repeatedly selecting a temporary goal, and pun_ob_near is the penalty coefficient for selecting a temporary goal within the ob region. $\mathrm{len}_{\boldsymbol{X}_{\mathrm{end2\_near}_i}.\mathrm{sector}}$ is the obstacle distance in the sector where the i^{th} candidate temporary goal lies, and $\theta_{\boldsymbol{X}_{\mathrm{end2\_near}_i}.\mathrm{sector}}$ is the heading angle of the centerline of that sector. $\boldsymbol{X}_{\mathrm{end2\_near}_i}.\mathrm{sector}$ refers to the sector where $\boldsymbol{X}_{\mathrm{end2\_near}_i}$ is located. len_ob_near and len_end2_near are the detection range for the ob and end2 regions, respectively.

4.4 Passage direction reward and Bug mechanism integration

When navigating through complex indoor and outdoor environments, humans tend to explore by entering visible passages or entrances. In the proposed algorithm, this behavior is mapped to detecting and rewarding passage-oriented heading angles.

If informed that the target lies on the other side of a wall, humans tend to follow along the wall to

$$
\begin{aligned}
&\mathrm{beside\_ob}_i = \vee_{j\le \mathrm{OB\_size}} \|\boldsymbol{X}_{\mathrm{end2\_near}_i} - \boldsymbol{X}_{\mathrm{ob}_j}\| < \mathrm{len\_ob\_near},\\
&\mathrm{beside\_end2}_i = \vee_{j\le \mathrm{end2\_size}} \|\boldsymbol{X}_{\mathrm{end}_j} - \boldsymbol{X}_{\mathrm{end2\_near}_i}\| < \mathrm{len\_end2\_near},\\
&\mathrm{is\_smooth}_i = \mid \theta_{\boldsymbol{X}_{\mathrm{end2\_near}_i}.\mathrm{sector}} - \theta_{\mathrm{last}} \mid < \mathrm{da},\\
&\mathrm{is\_collision}_i = \mathrm{Is\_collision}(\mathrm{path},\ \|\boldsymbol{X}_{\mathrm{end2}_i}\|,\ \mathrm{ra}_{\mathrm{end2}_i},\ \mathrm{dl}_3,\ 0,\ n_3),\\
&\boldsymbol{X}_{\mathrm{end2}}=\boldsymbol{X}_{\mathrm{end2\_near}_j}\Big|\mathrm{near}_j=\min_i\big(\mathrm{repeat\_near}^{\mathrm{is\_collision}_i}\cdot\mathrm{smooth\_near}^{\mathrm{is\_smooth}_i}\cdot\mathrm{pun\_end2\_near}^{\mathrm{beside\_end2}_i}\\
&\qquad\cdot\mathrm{pun\_ob\_near}^{\mathrm{beside\_ob}_i}\cdot\|\boldsymbol{X}_{\mathrm{end2\_near}_i}-\boldsymbol{X}_{\mathrm{end}}\|-0.01\cdot\min(5,\ \mathrm{len}_{\boldsymbol{X}_{\mathrm{end2\_near}_i}.\mathrm{sector}})\big).
\end{aligned}
\tag{5}
$$

its end; however, if the traveled distance becomes too long, they begin to doubt the chosen direction and may turn back. This behavior is reflected in the algorithm as a Bug mechanism with a distance threshold.

4.4.1 Passage direction reward

The passage-type heading angle exhibits distinct characteristics: It lies between two edge points and the sector's distance measurement is significantly greater than those of the neighboring edge sectors. Similar to temporary goals, passage headings must be identified through edge points. However, while temporary goals typically show one-to-one correspondence with edge points, passage headings require a matched pair of edge points. First, all sectors are traversed in a clockwise order. If the sector counterclockwise to an edge point has a significantly greater distance measurement, the edge point is labeled as a counterclockwise edge. Similarly, clockwise edges are defined. To determine a passage heading, each detected clockwise edge point searches forward for the nearest counterclockwise edge, and vice versa. If the nearest matched edge point lies within a threshold distance and an angular range, the two edge points are considered the sides of a passage, and all heading angles between them are marked as passage headings, as described in Algorithm 4.

As this subsection focuses on rewarding specific sectors regardless of distance, filtering out small obstacles may cause the vehicle to collide with minor objects in the passage direction. Therefore, passage detection is based on the original sensor input ScanOri, rather than the reconstructed version ScanRe introduced in Section 4.3.1. Once passage sectors are identified, reward terms are incorporated into the selection process of both the optimal target point $\boldsymbol{X}_{\text{pur}}$ and the optimal temporary goal $\boldsymbol{X}_{\text{end2}}$, as shown in Eqs. (6) and (7) respectively, at the top of the next page.

Here, is_pass determines whether the sector is a passage sector, and pass is a reward coefficient <1. pass_near is the passage reward parameter for selecting $\boldsymbol{X}_{\text{end2}}$, and should be slightly smaller than pass, meaning that the selection of a temporary goal has a stronger tendency toward passage choice. Since $\boldsymbol{X}_{\text{pur}}$ is the actual trajectory point taken at each step, an excessively small pass may easily cause oscillations of entering and then immediately exiting a passage. In contrast, $\boldsymbol{X}_{\text{end2}}$ is selected after oscillation occurs, which may indicate that the initial path choice is incorrect; in such cases, it is acceptable to re-enter the passage that is just exited.

Algorithm 4 FindPass(scan)

Require: LaserScan scan
Ensure: double array pass
Initialize Direction[i] $\leftarrow 0$ for all i's
for $i = 1$ to scan.size $- 1$ **do**
  dLast $\leftarrow$ scan.len[$i-1$] $-$ scan.len[i]
  dFuture $\leftarrow$ scan.len[$i+1$] $-$ scan.len[i]
  if |dFuture| $> 4\cdot$|dLast|$\wedge$dFuture $> 1\wedge$dLast < 1 **then**
    Direction[i] $\leftarrow 1$
  else if |dLast| $> 4 \cdot$ |dFuture| $\wedge$ dLast $> 1 \wedge$ dFuture < 1 **then**
    Direction[i] $\leftarrow -1$
  end if
end for
Initialize pass[i] $\leftarrow 0$ for all i's
for $i = 1$ to scan.size **do**
  if |Direction[i]| $= 1$ **then**
    for $j = i$ to $i + 20 \cdot$ Direction[i] **do**
      if Direction[j] $=-$Direction[i] **then**
        for $k = i$ to j **step** Direction[i] **do**
          pass[k] $\leftarrow 1$
        end for
        break
      end if
    end for
  end if
end for
return pass

4.4.2 Bug mechanism integration

Bug algorithm is a classic mapless navigation method, and if a feasible path objectively exists, the Bug family algorithms are guaranteed to achieve obstacle avoidance. Their core idea is to follow along the obstacle boundary upon encountering it. Many of the oscillation scenarios in the VFH algorithm occur because the goal is blocked by a long wall, causing the robot to oscillate near the perpendicular projection point. However, by integrating the Bug strategy and encouraging the robot to persist in following the wall, it is more likely to find a valid path.

The main idea of this section is to determine whether to activate the Bug mechanism based on the relative positions of the wall, goal, and robot. In this work, the Bug mechanism is activated only under

$$\begin{aligned}&\text{is\_pass}_i = \theta_i \in \text{FindPass(scan)},\\ &\boldsymbol{X}_{\text{pur}} = \boldsymbol{X}_{\text{pur}_j} \;\Big|\; \text{length}_j = \min_i \Big(\text{repeat}^{\text{is\_collision}_i} \cdot \text{pun\_end2}^{\text{beside\_end2}_i} \cdot \text{pun\_ob}^{\text{beside\_ob}_i}\\ &\qquad \cdot \text{smooth}^{\text{is\_smooth}_i} \cdot \text{pass}^{\text{is\_pass}_i} \cdot \|\boldsymbol{X}_{\text{pur}_i} - \boldsymbol{X}_{\text{end}}\| - 0.01 \cdot \min(5,\ \text{len}_i)\Big),\end{aligned} \tag{6}$$

$$\begin{aligned}&\text{is\_pass}_i = \theta_{\boldsymbol{X}_{\text{end2\_near}_i.\text{sector}}} \in \text{FindPass(scan)},\\ &\boldsymbol{X}_{\text{end2}} = \boldsymbol{X}_{\text{end2\_near}_j} \;\Big|\; \text{near}_j = \min_i \Big(\text{repeat\_near}^{\text{is\_collision}_i} \cdot \text{smooth\_near}^{\text{is\_smooth}_i} \cdot \text{pass\_near}^{\text{is\_pass}_i}\\ &\qquad \cdot \text{pun\_end2\_near}^{\text{beside\_end2}_i} \cdot \text{pun\_ob\_near}^{\text{beside\_ob}_i} \cdot \|\boldsymbol{X}_{\text{end2\_near}_i} - \boldsymbol{X}_{\text{end}}\|\\ &\qquad - 0.01 \cdot \min(5,\ \text{len}_{\boldsymbol{X}_{\text{end2\_near}_i.\text{sector}}})\Big).\end{aligned} \tag{7}$$

certain conditions, which include verifying whether the wall edge lies on the side closer to the goal, as shown in Fig. 6a. To prevent the robot from deviating excessively, the mechanism also requires that the distance to the goal falls within a critical threshold, as illustrated in Fig. 6b.

The Bug mechanism in this work is essentially a reward-based mechanism that encourages the robot to follow the direction suggested by the Bug algorithm, rather than a mandatory one. The final decision-making still relies on the valley selection of the VFH. The method for detecting walls adopts DBSCAN clustering. If there exists a continuous cluster near the robot with a distance below a certain threshold, it is identified as a wall. The robot may not only be adjacent to a single wall but may also be surrounded by walls on both sides. Therefore, in this section, clustering is performed separately for point clouds on either side of the x axis in the baselink coordinate frame, and the closer wall is used to determine whether the robot is following the left or the right wall. Once the side of the wall is identified, a reward is applied to the Bug sectors as shown in Fig. 7.

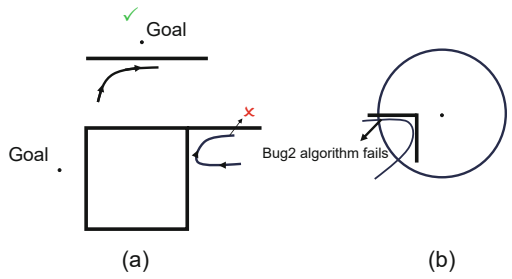

Fig. 6 Triggering mechanisms of the Bug algorithm: (a) wall-target directional proximity activation; (b) ego-vehicle threshold-proximity activation

This work does not apply rewards to areas directly in front or behind the robot, since rewards for smooth heading angles have already been introduced earlier, and hidden passages are often located beside walls. Excessive reward for the forward direction may cause the robot to miss those lateral passages.

The process of determining the wall direction based on the distance and computing the set of rewarded sectors according to the wall orientation and previous velocity is shown in Algorithm 5.

Thus, an additional reward term is introduced in the candidate valley selection, as shown in Eq. (8) at the bottom of the next page.

Here, is_bug denotes whether the sector belongs to the reward set defined by the Bug mechanism, and Bug is a reward coefficient less than 1. FindBug denotes the method for identifying reward sectors (as described in Algorithm 5), and v_{last} represents the current velocity.

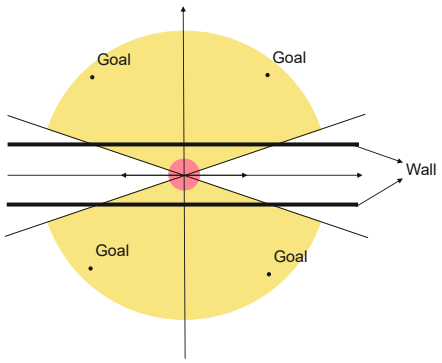

Fig. 7 Rewarded sectors

Algorithm 5 FindBug reward sector based on wall orientation and velocity

Require: LaserScan scan, double v_{last}
Ensure: double array raBugReward
scanLeft $\leftarrow$ scan$[0:\pi]$
scanRight $\leftarrow$ scan$[0:-\pi]$
clusterLeftList $\leftarrow$ DBSCAN(scanLeft)
clusterRightList $\leftarrow$ DBSCAN(scanRight)
lenLeft $\leftarrow -1$, lenRight $\leftarrow -1$
lenLeft $\leftarrow \min_i(\min_j(\|\text{clusterLeftList}[i].\text{element}[j]\|) \mid \text{clusterLeftList}[i].\text{num} > 10)$
lenRight $\leftarrow \min_i(\min_j(\|\text{clusterRightList}[i].\text{element}[j]\|) \mid \text{clusterRightList}[i].\text{num} > 10)$
if lenLeft $>$ lenRight and lenLeft > 0 **then**
    WallDirection $\leftarrow 1$
else if lenRight $>$ lenLeft and lenRight > 0 **then**
    WallDirection $\leftarrow -1$
else
    WallDirection $\leftarrow 0$
end if
if WallDirection > 0 and $v_{\text{last}} > 0$ **then**
    return $[\pi/12, \pi/2]$
else if WallDirection < 0 and $v_{\text{last}} > 0$ **then**
    return $[-\pi/2, -\pi/12]$
else if WallDirection > 0 and $v_{\text{last}} < 0$ **then**
    return $[\pi/2, 11\pi/12]$
else if WallDirection < 0 and $v_{\text{last}} < 0$ **then**
    return $[-11\pi/12, -\pi/2]$
end if

5 Experiment and analysis

5.1 Time and space complexity analysis

5.1.1 Time complexity of E^{2}MN

Let the total number of sectors be n, the number of temporarily memorized ob points be o, the number of visited end2 points be e, and the number of traced back path points for oscillation detection be n_1 for sector detection, n_2 ($n_2 > n_1$) for target point detection, and n_3 for temporary endpoint detection. Before selecting the optimal sector, it is necessary to identify passage sectors and determine wall locations. Both algorithms are based on post-processing after DBSCAN clustering. Generally, the time complexity of DBSCAN in 2D scenarios is $O(n^2)$, where n is the number of 2D data points, but in 2D LiDAR scans, neighboring point clouds are sequentially ordered, so proximity search requires checking only adjacent sector points, reducing the time complexity of these two steps to $O(n)$.

After acquiring the prerequisite data, each sector undergoes passability assessment, requiring $n/2$ computations. If a sector is passable, further evaluations are needed to check whether the target point lies within the ob region or the end2 region, or causes oscillation by comparing with all points in the path, involving $o+e+n_1$ computations. Therefore, the total complexity of the selection process is $O((n/2+o+e+n_1)n)$. Since in most cases $n/2$ is significantly greater than $o+e+n_1$, the overall time complexity can be considered as $O(n^2)$. This process

$$
\begin{aligned}
&\text{is\_bug}_i = \theta_i \in \text{FindBug}(\text{scan},\ v_{\text{last}}),\\
&\boldsymbol{X}_{\text{pur}} = \boldsymbol{X}_{\text{pur}_j} \ \Big|\ \text{length}_j = \min_i \Big(\text{repeat}^{\text{is\_collision}_i} \cdot \text{pun\_end2}^{\text{beside\_end2}_i} \cdot \text{pun\_ob}^{\text{beside\_ob}_i}\\
&\qquad \cdot \text{smooth}^{\text{is\_smooth}_i} \cdot \text{pass}^{\text{is\_pass}_i} \cdot \text{bug}^{\text{is\_bug}_i} \cdot \|\boldsymbol{X}_{\text{pur}_i} - \boldsymbol{X}_{\text{end}}\| - 0.01 \cdot \min(5,\ \text{len}_i)\Big).
\end{aligned}
\tag{8}
$$

may be executed twice, but the time complexity remains unchanged.

Under the conventional passability assessment strategy, which involves examining the adjacent sectors, let the number of adjacent sectors inspected be denoted as num. Then, each sector requires num $+ o + e + n_1$ operations, resulting in an overall complexity of $O((\text{num} + o + e + n_1)n)$. Since num$+o+e+n_1$ is generally a small constant, the overall time complexity can also be simplified as $O(n)$.

Subsequently, oscillation detection for the target point is performed, requiring n_2 comparisons, with a time complexity of $O(n_2)$. If oscillation is detected, a temporary endpoint must be selected. The selection process is based on DBSCAN and involves a constant number of computations per iteration, resulting in a time complexity of $O(n)$.

In summary, since the resolution parameter n constitutes the dominant term while all other variables remain small in magnitude, the overall time complexity of the algorithm can be simplified as $O(n^2)$. If the conventional passability assessment strategy with fixed neighboring sectors is adopted, the time complexity becomes $O(n)$.

Since this method allows for accepting oscillation in certain cases, the above procedure may be executed multiple times, up to a predetermined maximum, and the time complexity remains unchanged.

After preliminary experiments, the variation curve of computational latency with increasing resolution gradually approaches a parabolic trend, consistent with the theoretically derived time complexity. Since device configurations vary, the latency results from a single device have limited reference value; therefore, the detailed latency data are not presented in the main text but provided in the supplementary materials.

5.1.2 Space complexity of E^{2}MN

The data used by the proposed algorithm include raw LiDAR data, reconstructed sensor data with small obstacles removed, edge point flags for each sector, and passability indicators. All of these are proportional to n. Additionally, the algorithm requires storage of ob points and end2 points, whose sizes are o and e, respectively. The path information also needs to be stored, which could increase significantly over time. However, based on the locality principle, the algorithm only traces a fixed number of recent path points, so it only requires storing $\max(n_1, n_2, n_3)$ points. Therefore, the overall space complexity is $O(n + o + e + \max(n_1, n_2, n_3))$. In most cases, the values of o, e, n_1, n_2, and n_3 are much smaller than n, so the space complexity can be approximated as $O(n)$.

5.2 Experimental data

The algorithm implementation, scenario construction, and performance evaluation are conducted on the Ubuntu-based Robot Operating System (ROS) platform. The experiment environment is built using the six-robot simulated vehicle and the Gazebo simulator, while the path-planning algorithm is tested through the RViz software. For visualization purposes, navigation trajectories are overlaid onto the map using image processing tools such as Photoshop, although the mapless algorithms do not use any map data during the actual experiments. The experimental computing platform uses an AMD Ryzen 7 7700 8-core processor, and the physical vehicle tests are conducted using a self-developed vehicle with a HarmonyOS-based mainboard.

5.2.1 Experimental scenario configuration

Figs. 8a and 8b represent outdoor simple obstacle scenarios and are used to test the basic functionalities of the algorithm—obstacle avoidance and escape from local minima. The remaining experimental scenarios are used in evaluating path performance, parameter sensitivity, and extreme environment experiments. Specifically, as shown in Figs. 8c, 8d, 8g, and 8h, complex indoor evaluation space, outdoor spacious environment, large dead-end area, and small dead-end area are used for parameter sensitivity experiments, while maze environment (Fig. 8e) and indoor narrow environment (Fig. 8f) are used to test the algorithms' performance under extreme conditions.

In addition, we conduct experiments in dynamic environments, with the scenarios and results presented in the supplementary materials. However, since the proposed algorithm in this paper mainly targets static complex environments, these are omitted from the main text.

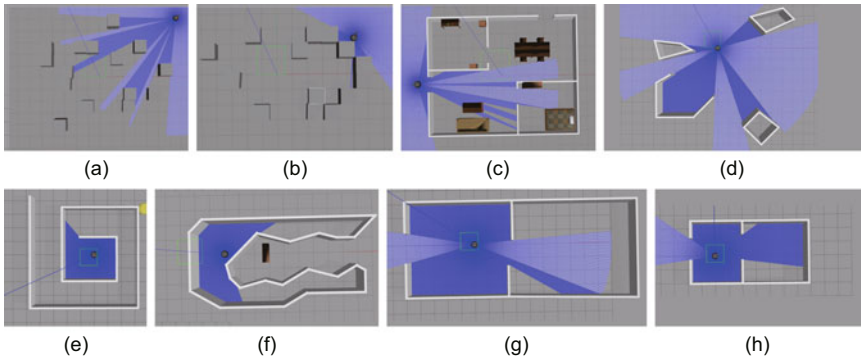

Fig. 8 Experimental scenarios: (a) obstacle avoidance testing field; (b) local minima resolution arena; (c) complex indoor evaluation space; (d) outdoor spacious environment; (e) maze environment; (f) indoor narrow environment; (g) large dead-end area; (h) small dead-end area

5.2.2 Basic functionality testing

The following presents the test of the algorithm's basic obstacle avoidance functionality.

As shown in Fig. 9, the algorithm successfully guides the vehicle to avoid all obstacles without collision.

The following presents the algorithm's response after falling into a local optimum.

As shown in the experimental results of Fig. 10, when the vehicle enters a local optimum, the algorithm is able to guide it out of the local optimum and eventually reach the target.

5.2.3 DBSCAN parameter selection and its effect on the temporary goal choice

In this section, a systematic evaluation is conducted on the influence of varying the two key parameters in the DBSCAN algorithm—the neighborhood radius eps and the minimum number of samples $\min_{\text{samples}}$—on the point cloud reconstruction performance in Algorithm 2 and the temporary goal selection strategy in Algorithm 3. The clustering and navigation goal generation results under different parameter configurations are shown in Fig. 11.

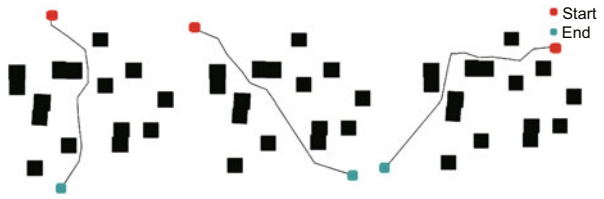

Fig. 9 Obstacle avoidance function test

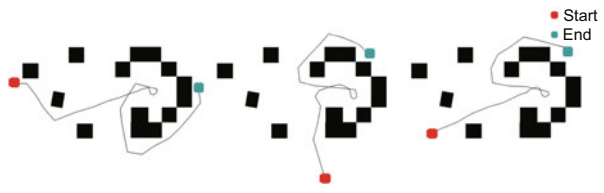

Fig. 10 Local minimum trap handling function test

As shown in Fig. 11, when eps is large (e.g., 1.0 or 1.5), objects such as table legs and stools are incorrectly merged into wall clusters, reducing structural distinction. When eps is too small (e.g., 0.1 or 0.3), wall points are split into multiple clusters due to sparse density or large spacing, causing oversegmentation and misidentification of distant walls as temporary goal points. Overall, eps = 0.5 provides a good balance. For $\min_{\text{samples}}$, small values (e.g., 1 or 3) retain small obstacles and degrade the reconstruction quality, while larger values (5, 7, or 9) yield no significant difference between clusters. To avoid misclassifying near-wall areas as isolated

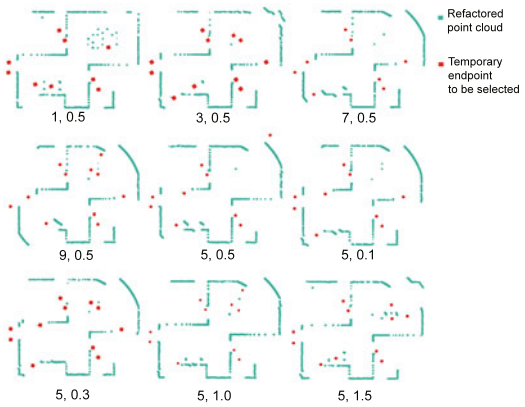

Fig. 11 Reconstructed point clouds and candidate temporary goal points under different $\min_{\text{samples}}$ and eps values

obstacles, $\min_{\text{samples}}$ is set to 5.

5.2.4 Sensitivity analysis of key parameters

This algorithm involves the following main parameters: the local minimum determination conditions da and dl; the penalty coefficients for target points falling within the neighborhoods of end2 and ob, namely pun_end2, pun_ob, pun_end2_near, and pun_ob_near; the channel reward parameters pass and pass_near; the reward parameter associated with the obstacle-edge bug, which is named bug as it is inspired by the Bug algorithm; the local minimum penalty coefficients repeat and repeat_near; the neighborhood range criteria for end2 and ob, namely len_ob, len_end2, len_ob_near, and len_end2_near; the smoothness reward parameters smooth and smooth_near. The baseline values, i.e., the parameters used in Sections 5.2.5 and 5.2.2, are set as follows: da=0.8 rad, dl=0.205 m, pun_end2=7.5, pun_ob=10, pun_end2_near=7.5, pun_ob_near=15, pass=0.85, pass_near=0.8, bug=0.85, repeat=4.5, repeat_near=4.5, len_ob=2 m, len_end2=1 m, len_ob_near=4 m, len_end2_near=2 m, smooth=0.75, and smooth_near=0.65.

This section presents a sensitivity analysis of key parameters. The parameters are grouped based on their correlations, and a one-at-a-time (OAT) method is used to assess each group's influence on the algorithm performance. For groups with correlated parameters, their values are varied proportionally to preserve internal consistency.

The analysis first focuses on fundamental parameters that have a decisive impact on algorithm performance, where significant differences can be observed within a single experiment.

In this experiment, two scenarios with different structures, one wide and one narrow, are selected, each exhibiting distinct optimal neighborhood judgment ranges. As shown in the experimental results (Fig. 12), when the neighborhood range is set too small (Fig. 12a), the path planning process experiences repeated oscillations (oscillations 1 and 2) because the temporary goal frequently falls outside the neighborhoods of the obstacle (ob) or the endpoint (end2), until the theoretically feasible region is eventually covered. Conversely, when the neighborhood range is set too large (Fig. 12d), the majority of temporary goals are judged to lie within the neighborhoods of ob or end2 (Fig. 12d), leading to reduced discriminability among goals and likewise inducing persistent oscillations (oscillations 1 and 2). In contrast, when the parameters are set appropriately (e.g., Figs. 12b and 12c), only a single oscillation is triggered within the same region, thereby significantly enhancing the planning stability.

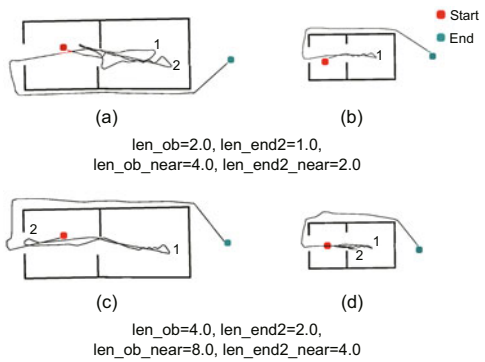

Fig. 12 Sensitivity experimental results of the judgment range for the end2 and ob neighborhoods with smaller judgment ranges in two scenarios (a, b), and with larger judgment ranges in two scenarios (c, d)

As shown in Fig. 13, when the local minimum determination conditions are too lenient (Fig. 13a), noticeable oscillations occur. Conversely, when the oscillation criterion is set too strictly (Fig. 13d), the oscillation determination becomes overly arbitrary. Since the algorithm tolerates a certain degree of oscillation under extreme conditions, excessively strict criteria actually induce oscillations.

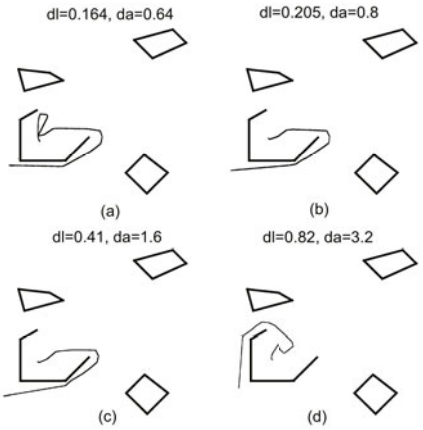

Fig. 13 Sensitivity experimental results of the local minimum determination conditions with four different values (a–d)

Next are the reward–penalty parameters. These parameters do not decisively affect algorithm outcomes and show no significant differences in a single trial, but they influence the stability and average performance of path planning. To fully reveal their effects, experiments are conducted in scenarios selected to highlight each parameter's impact.

This experimental scenario involves navigating around table legs. Due to the narrow and complex environment, oscillations are likely to occur, making it suitable for testing the parameters repeat and repeat_near. As shown in Fig. 14, when these parameters are entirely disabled (set to 1), the stability is the worst and the path length is the longest. However, as the penalty strength increases, the differences among the best case, worst case, and stability become less significant. Therefore, the sensitivity of these parameters is low, yet they remain indispensable.

In this experimental scenario, the chosen start and goal positions require passing through a complex area with multiple interconnected corridors, making it suitable for testing the parameters pass and pass_near. As shown in Fig. 15, when these parameters are entirely disabled (set to 1), the stability is the worst, and the suboptimal paths occur as the robot repeatedly circles around the room after missing the correct corridor. With increasing reward strength, the stability of the optimal path improves. However, when the reward strength becomes too large, it overshadows other reward–penalty terms, causing the robot to repeatedly enter incorrect corridors and thus leading to excessively long suboptimal paths. Therefore, these parameters are indispensable and exhibit a certain degree of sensitivity, with values of 0.8 or higher being the most effective.

As shown in Fig. 16, the optimal path in this experimental scenario requires the robot to follow obstacle boundaries for a considerable distance, making it suitable for evaluating the performance of the parameter bug, which encourages boundary-following behaviors. When this parameter is entirely disabled (set to 1), the stability is the worst, with numerous extreme outcomes. As the reward strength increases, the results become more stable; however, when the reward strength is too high (0.5), stability declines again, and the excessive reliance on obstacle boundaries causes the robot to follow incorrect edges, leading to redundant detours.

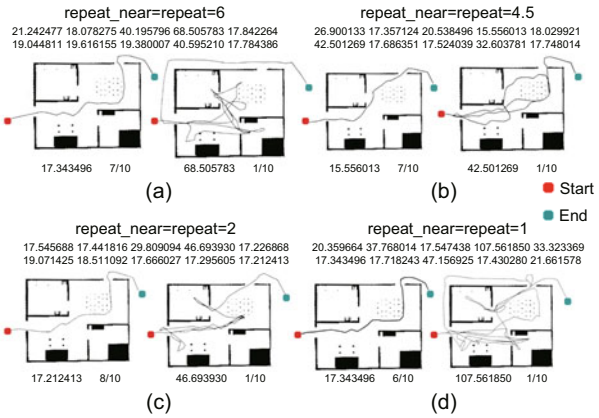

Fig. 14 Sensitivity experimental results of the local minimum penalty coefficients repeat and repeat_near with four different values (a–d)

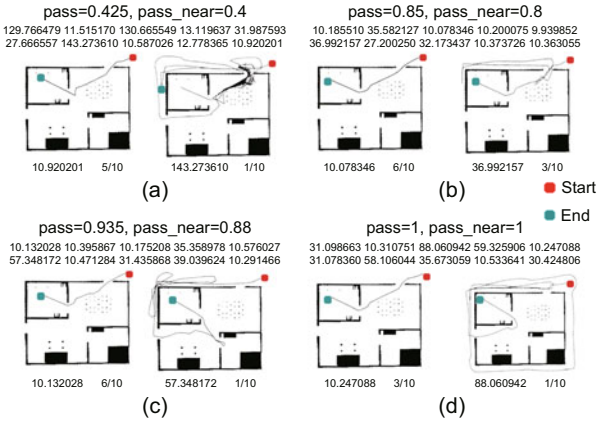

Fig. 15 Sensitivity experimental results of the channel reward parameters pass and pass_near with four different values (a–d)

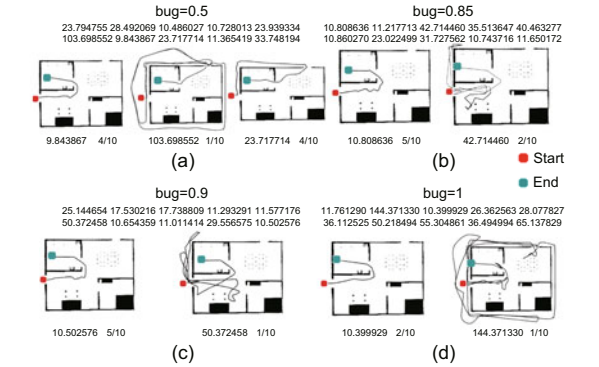

Fig. 16 Sensitivity experimental results of the reward parameter associated with the obstacle-edge bug with four different values (a–d)

As shown in Fig. 17, this experimental scenario involves passing through a doorway and easily encountering oscillations near wall boundaries, leaving an ob neighborhood. Moreover, the start position lies in an end2 neighborhood, where oscillations

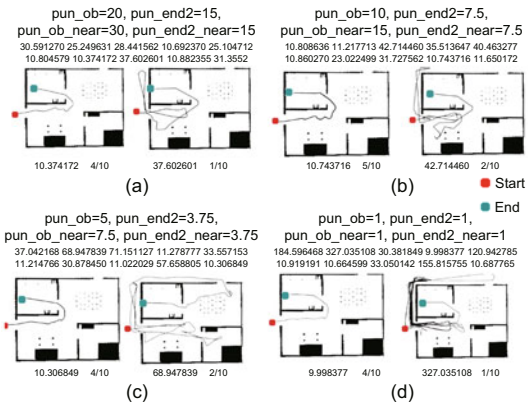

Fig. 17 Sensitivity experimental results of the penalty coefficients for target points falling within the neighborhoods of end2 and ob (pun_end2, pun_ob, pun_end2_near, and pun_ob_near) with four different values (a–d)

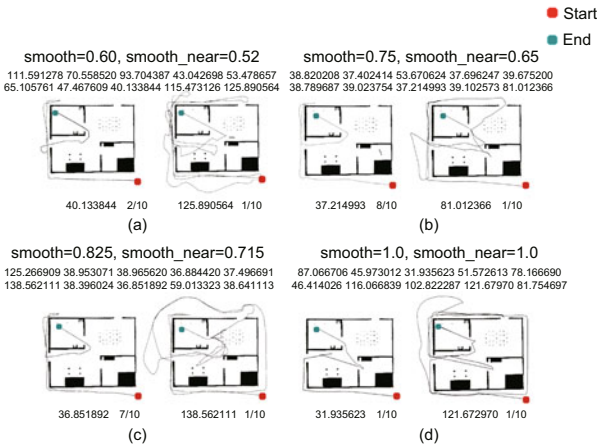

Fig. 18 Sensitivity experimental results of the smoothness reward parameters smooth and smooth_near with four different values (a–d)

often cause the robot to retrace its path. This makes the scenario suitable for testing the neighborhood penalty parameters. When these parameters are entirely disabled (all set to 1), the robot repeatedly enters the ob and end2 regions, resulting in oscillatory behaviors, elongated suboptimal paths, and poor stability. With stronger penalties, although the stability of the optimal path improves only marginally, the robot maintains robustness by avoiding large deviations even when it fails to follow the optimal path. As the penalty strength increases, the differences among the best case, the worst case, and stability remain limited; however, overly large penalties cause a certain decline in stability. Therefore, these parameters exhibit low sensitivity but remain indispensable.

This experimental scenario requires sacrificing a certain degree of smoothness to enter the doorway while simultaneously maintaining the smoothness when navigating past long obstacle segments at intersections. Therefore, it is well-suited for evaluating the optimal values of the parameters smooth and smooth_near. As shown in Fig. 18, when this parameter group is completely ineffective, the robot can enter the doorway but, due to the lack of smoothness, takes an incorrect branch, resulting in poor stability. As the smoothness reward increases, both the stability of the optimal path and that of the suboptimal paths improve. Among the four tested parameter settings, stability is maximized when the values are set to 0.75 and 0.65, with fewer detours in the failure cases. Either increasing or decreasing the reward beyond these values reduces stability and lengthens the suboptimal paths. Hence, these parameters demonstrate high sensitivity, but all parameter configurations eventually allow the robot to reach the goal, and stability is maintained to a certain degree unless the parameters are entirely ineffective.

The comprehensive experimental results demonstrate that all categories of parameters play indispensable roles in the overall performance of the algorithm, though their sensitivities vary. Among them, the fundamental decision parameters (e.g., len_ob, len_end2, da, and dl) exert the most significant influence on stability and robustness, exhibiting high sensitivity and thus requiring precise tuning within moderate ranges. Reward–penalty parameters, while not directly determining task success or failure, strongly affect path stability, length, and smoothness. In particular, the pass and smooth series parameters show high sensitivity in optimal path generation: Excessively low or high values degrade the performance, whereas moderate settings (e.g., pass $\approx$ 0.8, smooth $\approx$ 0.75) yield superior results. The bug parameter effectively suppresses extreme oscillations at moderate levels, though overly strong values may induce excessive dependence on obstacle edges. The repeat and pun series parameters exhibit relatively poor sensitivity but remain essential for mitigating persistent oscillations and ensuring robustness. Overall, the effectiveness of parameter configurations relies on balanced tuning, as overly small or large values undermine algorithmic stability and global optimality, whereas moderate values achieve an optimal performance in complex

environments. Accordingly, the parameter settings adopted in Section 5.2.5 are as follows: da = 0.8 rad, dl = 0.205 m, pun_end2 = 7.5, pun_ob = 10, pun_end2_near = 7.5, pun_ob_near = 15, pass = 0.85, pass_near = 0.8, bug = 0.85, repeat = 4.5, repeat_near = 4.5, len_ob = 2m, len_end2 = 1 m, len_ob_near = 4 m, len_end2_near = 2 m, smooth = 0.75, and smooth_near = 0.65.

5.2.5 Path performance test

This subsection aims to evaluate the obstacle avoidance performance of the vehicle under different path-planning algorithms given various start and goal positions. The experiments adopt both ablation and comparative approaches by progressively removing optimization modules from the proposed E^{2}MN framework to analyze each component's contribution to the overall performance. To comprehensively assess the effectiveness of the proposed method, the experiments also include comparisons with the representative map-based navigation framework MoveBase and the classical mapless navigation algorithm Bug2. Since the proposed method does not rely on map data and the coordinate updates may not be timely, it occasionally fails to reach the intended target, leading to variability in navigation outcomes from the same initial position. Therefore, the proposed method and its ablation variants are tested 10 times in each scenario, and both the best and worst results are reported. The number of runs with similar outcomes (where the path length difference is <5) is also documented. For Bug2, different outcomes can result from turning left or right upon encountering obstacles; both choices are shown. As MoveBase performs navigation based on map data with a fixed path, only one result is presented, using the Dijkstra algorithm for global planning and the DWA algorithm for local planning, serving as the reference for optimal paths.

This study designs a stepwise ablation experiment framework to systematically validate the optimization effects of algorithmic modules. The specific configurations of each experimental group are as follows:

1. Baseline algorithm. It serves as the reference using the classical VFH architecture.

2. First-level optimized algorithm. It integrates local minimum detection and prevention modules into the baseline algorithm and introduces a simple interim target point selection strategy determined by Eq. (9).

3. Second-level optimized algorithm. It extends the first-level optimized algorithm by incorporating the predictive elimination module for local minima.

4. Full-version algorithm. It replaces the interim target selection strategy in the second-level optimized algorithm with the strategy proposed in Section 4.3.

$$\begin{aligned}&\boldsymbol{X}_{\mathrm{end2}_i}=\begin{pmatrix}\mathrm{len}_i\cdot\cos(\theta_i)\\ \mathrm{len}_i\cdot\sin(\theta_i)\end{pmatrix},\\ &\boldsymbol{X}_{\mathrm{end}}=X_{\mathrm{end2}_j}\left|\mathrm{length}_j=\min_i\left(\mathrm{len}_i-\|\boldsymbol{X}_{\mathrm{pur}}-\boldsymbol{X}_{\mathrm{end2}_i}\|\right)\right.\end{aligned}\tag{9}$$

In the scenario of Fig. 19, since the goal is close to the entrance direction, the path planning process tends to exhibit exploration and trial-and-error behaviors due to the absence of map information. As shown in Fig. 19, the baseline algorithm experiences repeated oscillations at a certain location, resulting in a local deadlock. After the oscillation detection mechanism is introduced, the algorithm identifies and exits the deadlock, and then searches for a new feasible path. With the addition of the oscillation prediction mechanism, the generated path becomes smoother. When the interim goal point search strategy is updated, the overall path stability, smoothness, and length are significantly improved. Compared with the Bug2 algorithm, the proposed method achieves a shorter path even in the worst-case trial. In the vast majority of cases, the path length of the full-version algorithm is reduced by 19.72%, compared to the average length of the two directions of the Bug2 algorithm. Due to the unavoidable exploration process, the path length is increased by 110.54%, compared to the reference path length generated by the MoveBase.

In the scenario of Fig. 20, most of the paths had to pass through areas with a large number of small-sized obstacles. After sensor data reconstruction described in Section 4.3 was applied, the stability of the path generated by the full-version algorithm is significantly improved. In the vast majority of cases, the path length of the full-version algorithm is reduced by 52.695%, compared to the average length of the two directions of the Bug2 algorithm, and is only increased by 12.101%, compared to the reference path length generated by the MoveBase.

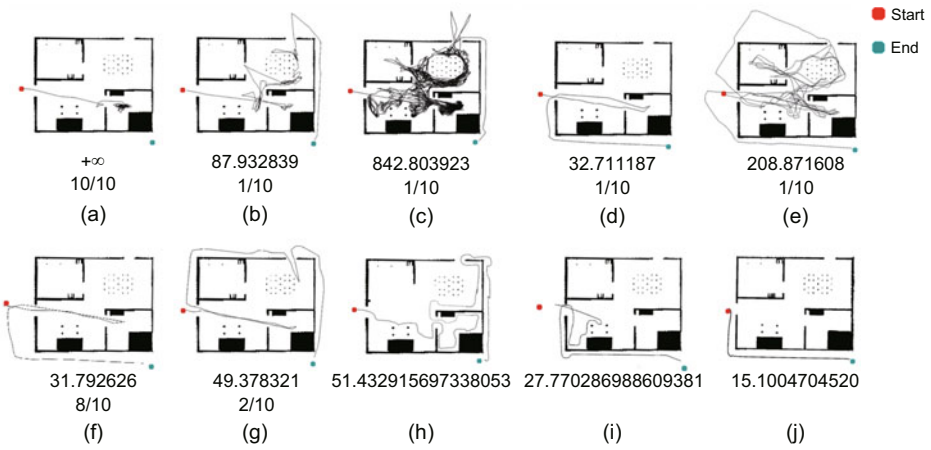

Fig. 19 Route 1 results: (a) baseline algorithm; (b, c) first-level optimized algorithm; (d, e) second-level optimized algorithm; (f, g) full-version algorithm; (h) Bug2 algorithm (left turn); (i) Bug2 algorithm (right turn); (j) MoveBase framework

Fig. 20 Route 2 results: (a) baseline algorithm; (b, c) first-level optimized algorithm; (d, e) second-level optimized algorithm; (f, g) full-version algorithm; (h) Bug2 algorithm (left turn); (i) Bug2 algorithm (right turn); (j) MoveBase framework

In the scenario in Fig. 21, the goal is located in the corner farthest from the entrances. The distance between the goal and any entrance is longer than the direct distance between the start and the goal, resulting in a path that has to detour significantly before reaching the destination. During this detour, the trajectory initially moves farther from the goal before turning toward it. A human may suspect that they are on the wrong path under such circumstances, and the vehicle also has a high probability of abandoning the correct route in search of alternatives. As a result, the first three algorithms perform poorly in terms of stability and path length. However, after incorporating the Bug reward mechanism and the interim goal selection strategy described in Section 4.3, the full-version algorithm achieves significantly improved stability and produces a path close to the optimal one. In more than half of the cases, the path length of the full-version algorithm is reduced by 55.75% compared to the average length of the two directions of the Bug2 algorithm, and by 10.52% compared to the reference path length generated by the MoveBase.

The scenario in Fig. 22 requires the robot to find a way out of an indoor environment. The drawbacks of the initial VFH-based baseline algorithm are evident, as it remains stuck in place during this task. With the progressive addition of various optimization modules, the resulting path becomes increasingly stable and short in length. In the vast majority of cases, the path length of the full-version algorithm is reduced by 58.15% compared to the average of the

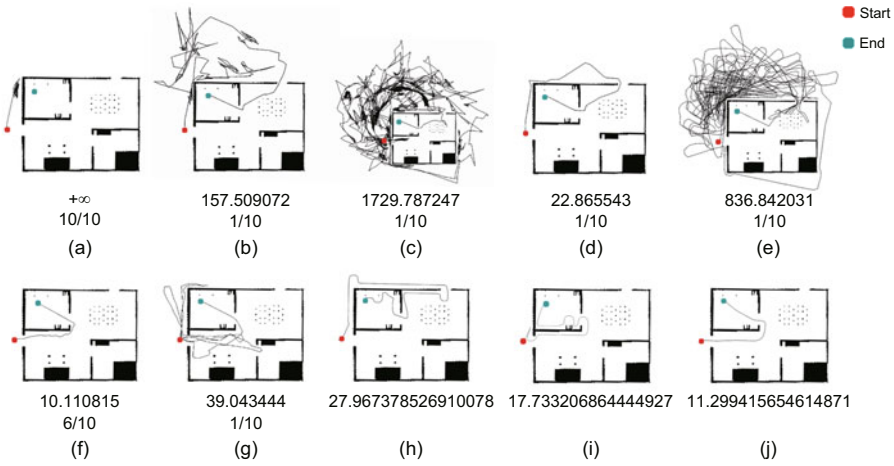

Fig. 21 Route 3 results: (a) baseline algorithm; (b, c) first-level optimized algorithm; (d, e) second-level optimized algorithm; (f, g) full-version algorithm; (h) Bug2 algorithm (left turn); (i) Bug2 algorithm (right turn); (j) MoveBase framework

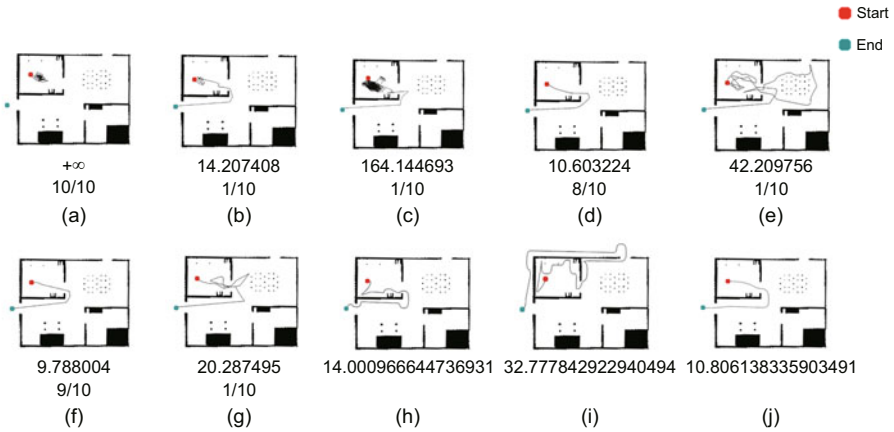

Fig. 22 Route 4 results: (a) baseline algorithm; (b, c) first-level optimized algorithm; (d, e) second-level optimized algorithm; (f, g) full-version algorithm; (h) Bug2 algorithm (left turn); (i) Bug2 algorithm (right turn); (j) MoveBase framework

two directions of the Bug2 algorithm, and by 9.42% compared to the reference path length generated by the MoveBase.

In summary, the contributions of each module to the algorithm can be outlined as follows: By integrating the local minimum detection and prevention modules and introducing a simple interim target point selection strategy determined in Eq. (9), this module primarily addresses the most critical "complete stagnation" problem during navigation. When the algorithm detects that the robot is trapped in a local minimum region (e.g., dead ends or U-shaped corridors), this mechanism is triggered to dynamically generate a temporary escape target based on the environmental point cloud density. Experimental results demonstrate that the majority of scenarios that would have otherwise failed due to deadlock are successfully reversed; although the robot's path may be tortuous, it ultimately finds a feasible route. This verifies the effectiveness of this module as a fundamental escape guarantee.

After introducing the predictive elimination module for local minima, this module builds upon the previous one and aims to optimize the path quality and efficiency of the escape process. Through online learning and state memory, it actively predicts and suppresses control commands that may lead to oscillations. After its introduction, although the robot still needs to perform necessary exploration to exit the local minimum region (paths may have

twists and turns), trajectory smoothness is significantly improved, and repetitive back-and-forth oscillations are greatly reduced.

After introducing the complex interim target point selection module proposed in Section 4.3, both path length and stability are significantly improved. This module represents a qualitative leap from "successful escape" to "efficient navigation." It comprehensively uses long- and short-term historical information, environmental topology, and a more refined cost function, aiming not merely to generate a feasible escape point but to select a temporary sub-goal that guides the robot toward the globally optimal direction. The results manifest as significantly short path lengths, further the enhanced trajectory stability, and the overall navigation performance approaching or in some cases surpassing existing methods that rely on prior maps. This marks a key advancement in both the intelligence and the practical applicability of the algorithm.

In addition to the routes presented in the main text, we have several other route experimental results and analyses that are not shown in the main text due to space limitations. Interested readers are referred to the supplementary materials for details.

The average route lengths of each algorithm across seven complex models in the main text and supplementary materials are calculated, and the ratios of the algorithmic route outcomes are compared. It is found that, compared with the average path length of the two directions of the classical mapless Bug2 algorithm, the proposed method reduces the average path length by 50.51%, while it increases the path length by only 17.57% relative to the navigation results of MoveBase.

5.2.6 Performance in extreme scenarios

In this section, multiple experiments are conducted in two extreme scenarios to evaluate the algorithm's capability in handling extreme environments. The focus is primarily on testing the algorithm's performance in narrow corridors and maze scenarios. Except for dl=0.13 m and da=0.51, all other parameters use the baseline values, allowing for a certain degree of oscillation. The experimental results are shown in Figs. 23 and 24.

In the maze environment tests for both directions, it can be observed that due to the constraints of environmental complexity and the absence of a prior map, some experimental results exhibit the phenomenon of misidentifying feasible paths as obstacle regions (ob). This leads to the generation of numerous ineffective detour paths. Nevertheless, the algorithm still demonstrates a certain degree of stability and is ultimately able to converge to the optimal path.

In the experiments conducted in the narrow scenario, the limited size of the entrance region requires the algorithm to tolerate local oscillations and attempt multiple coordinate points to obtain an observation angle sufficient for entering the passage. Although differences in path length exist,

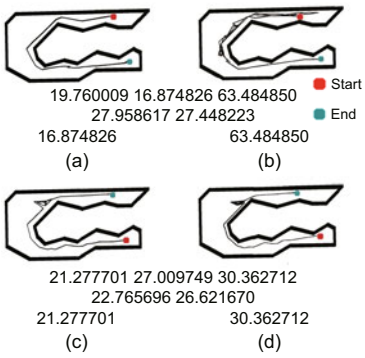

Fig. 23 Experimental results in the extreme narrow corridor scenario: (a, b) more successful and less successful trials in direction 1 (top to bottom); (c, d) more successful and less successful trials in direction 2 (bottom to top)

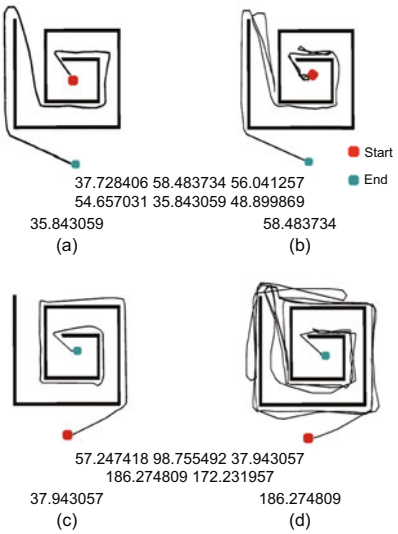

Fig. 24 Experimental results in the extreme maze scenario: (a, b) more successful and less successful trials in direction 1 (inside to outside); (c, d) more successful and less successful trials in direction 2 (outside to inside)

all experiments successfully reach the target point and the algorithm still retains a certain probability of finding the optimal path even under extreme conditions.

Furthermore, in both scenarios, the following phenomenon is observed: The algorithm tends to achieve transitions more easily from narrow regions to wide ones, whereas the reverse traversal proves more challenging. The underlying reason is that the algorithm prefers motion paths through wide areas, while the sweeping-based feasibility determination conditions are relatively strict, thereby increasing the decision-making difficulty of transitioning from wide to narrow regions.

5.2.7 Performance in real environments

This part presents the algorithm's performance in real environments. The experiments are conducted in an office scenario using our in-development HarmonyOS LiteOS differential vehicle, as shown in Fig. 25.

The experimental results are shown in Fig. 26. In real environments, each odometry frame update contains offsets and errors. Even under such conditions, the algorithm guides the vehicle to the vicinity of the target point, demonstrating its effectiveness and robustness.

To further evaluate the performance of the algorithm, a large obstacle is introduced into the aforementioned spacious real-world environment to create a narrow passage scenario. The algorithm's performance under these constrained conditions is depicted in Fig. 27.

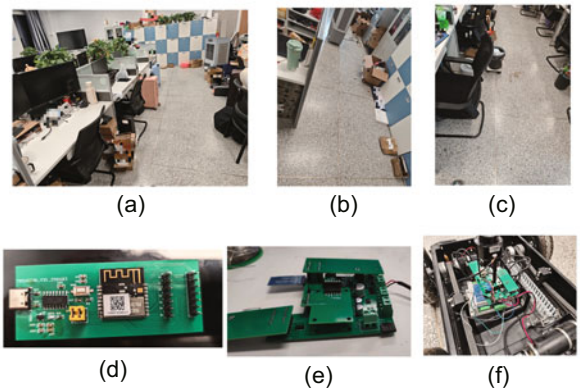

Fig. 25 Real-world experimental scene and vehicle illustration: (a–c) office corridor as the experimental environment; (d) HarmonyOS vehicle drive core board; (e) vehicle control board, composed of the HarmonyOS core board with motor control and sensor extension boards; (f) experimental vehicle, consisting of the vehicle control board and chassis

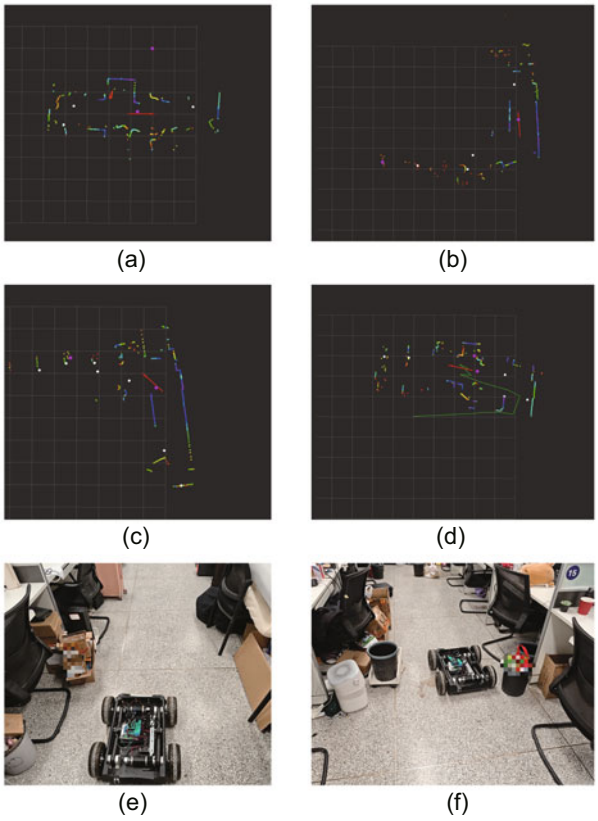

Fig. 26 Experimental results in real environments: (a–d) RViz screenshots showing sensor data and trajectory; (e) experiment photo at the starting point; (f) experiment photo at the destination

In the constrained environment, the increased number of obstacles leads to a significant rise in the density and complexity of point cloud data. As shown in Fig. 27c, this increase results in noticeable drift in the odometry data, which primarily stems from the accumulated pose estimation errors caused by repetitive or occluded features in the environment. Despite these challenges, the proposed algorithm successfully guides the vehicle to complete the obstacle avoidance task, demonstrating its strong robustness in handling complex perceptual noise and uncertainties in the state estimation.

5.3 Performance comparison with recent models

This section analyzes the performance of existing models and compares them with the proposed

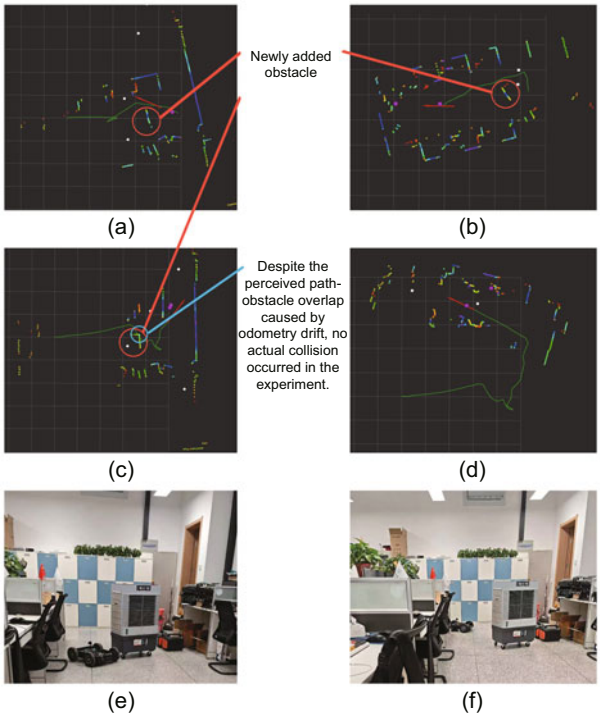

Fig. 27 Experimental results in the constrained real-world environment: (a) obstacle avoidance experiment during forward navigation; (b) obstacle avoidance experiment in the reverse direction (return path corresponding to (a)); (c, d) trajectory diagrams of the vehicle successfully crossing the introduced obstacle and navigating through the U-shaped corridor to reach the target point; (e) experimental photograph of the vehicle avoiding the newly added obstacle in the actual scene; (f) experimental photograph of the vehicle crossing the introduced obstacle and circumventing the U-shaped bend

algorithm. Specifically, we compare the widely used map-based framework MoveBase (with the DWA local planner (Fox et al., 1997)), the mapless classical Bug2 algorithm (Fox et al., 1997), the recent topology-based method DVT_Tree (Zhong et al., 2023), and the recent learning-based methods APF-RL (Bektaş and Bozma, 2022) and HIT-RL (Jang et al., 2022), focusing on the spatiotemporal complexity, interpretability, and related performance aspects.

Let the sensor resolution be n, the number of global map grid cells be N, the linear and angular velocity resolutions be v and ω, and the number of grid cells in the local map be N_1. In the DVT_Tree method, denote the average number of leaf nodes per parent node as t and the number of path nodes as P. For learning-based methods, let the network size C_{net} be the total number of neurons across all layers.

MoveBase uses the classical Dijkstra and DWA algorithms, whose performance metrics have been extensively studied and widely adopted. The derivations of these metrics are omitted in the main text for brevity. Readers may refer to the supplementary materials for detailed reasoning.

Both reinforcement learning algorithms exhibit $O(C_{\text{net}})$ time and space complexity. Designed for mapless scenarios, neither requires a predefined map, yet both inherit the common limitation of learning-based methods: weak interpretability. The APF_RL method introduces no fundamental improvements to the underlying APF mechanism and thus lacks the ability to escape from local optimum traps. In contrast, HIT_RL incorporates an intermediate node mechanism that effectively handles local optima. Detailed derivations of these metrics are provided in the supplementary materials.

The DVT_Tree method, selected for comparison among topology-based algorithms, operates without a global map but requires a local grid map. Constructing the local map involves processing sensor data with a time complexity of $O(n)$. Leaf nodes are then generated by traversing the local map, requiring $O(N_1)$ time complexity. Subsequent pruning of invalid leaves involves evaluating each leaf's local environment, resulting in $O(tN_1)$ time complexity. Local minima are escaped by backtracking to the parent node. The method stores the local map, sensor data, and the DVT tree—with the tree size being a constant multiple of P—yielding a space complexity of $O(N_1 + P)$. The approach features clear rules and high interpretability.

Comparison of the performances among the above algorithms and the proposed method is summarized in Table 1.

6 Conclusions

This paper proposes an end-to-end mapless navigation method that mimics human behaviors and relies solely on 2D LiDAR, addressing the practical need for autonomous navigation in unknown environments without high-definition map support. The method aims to solve key challenges in path planning, oscillation suppression, and local minimum handling. The proposed algorithm is interpretable, with each decision in the planning process based on explicit rules and perceptual data, providing logical

Table 1 Performance comparison of different methods

Method	Time complexity	Space complexity	Map handling	Local optimum handling	Interpretability
MoveBase	$O(N^2)$	$O(N)$	Y	Y	Strong
Bug2	$O(n)$	$O(n)$	N	N	Strong
DVT_Tree	$O(tN_1)$	$O(N_1+P)$	Requires rasterization	Y	Strong
APF_RL	$O(C_{\text{net}})$	$O(C_{\text{net}})$	N	N	Weak
HIT_RL	$O(C_{\text{net}})$	$O(C_{\text{net}})$	N	Y	Weak
E^2MN	$O(n)/O(n^2)$	$O(n+o+e)$	N	Y	Strong

clarity and traceability. It also incorporates the advantages of the Bug algorithms in specific scenarios (route 3 in Section 5.2.5), while overcoming their limitations in concave spaces, effectively addressing the common issue of local planners falling into local minima in mapless settings. Experimental results have shown that the proposed algorithm performs well in terms of obstacle avoidance and handling of local minima in simple environments. In complex scenarios, it generates smoother paths and exhibits more stable performance compared to the original VFH algorithm. Compared to the classical mapless Bug2 algorithm, the proposed method reduces the average path length by 50.51%, and increases it by only 17.57% compared to the MoveBase algorithm results. The current algorithm still requires prior knowledge of whether the environment is narrow to adjust the influence ranges of the ob and end2 regions. Future work will focus on detecting passage width and adaptively determining the values of beside_end2 and beside_ob.

Acknowledgments

The authors thank the Top Leading Talent in Gansu Province, and the Supercomputing Center of Lanzhou University.

Contributors

Yinan YANG designed the algorithms and the schematic and printed circuit board (PCB) layout of the real vehicle control board, and was responsible for assembly and soldering. Zhiye WANG created the visualizations and analyzed the data. Yinan YANG and Zhiye WANG conducted the experiments and drafted the paper. Xuan KONG created the graphical illustrations. Peng ZHI and Dapeng ZHANG revised the paper while providing supervision and guidance. Rui ZHOU and Qingguo ZHOU handled project administration and managed funding acquisition. Yinan YANG finalized the paper.

Conflict of interest

All the authors declare that they have no conflict of interest.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

Ali M, Liu LT, 2023. GP-Frontier for local mapless navigation. IEEE Int Conf on Robotics and Automation, p.10047-10053. https://doi.org/10.1109/ICRA48891.2023.10161230

Balan K, Manuel MP, Faied M, et al., 2019. A fuzzy based accessibility model for disaster environment. Int Conf on Robotics and Automation, p.2304-2310. https://doi.org/10.1109/ICRA.2019.8793602

Bektaş K, Bozma HI, 2022. APF-RL: safe mapless navigation in unknown environments. Int Conf on Robotics and Automation, p.7299-7305. https://doi.org/10.1109/ICRA46639.2022.9811537

Borenstein J, Koren Y, 1989. Real-time obstacle avoidance for fast mobile robots. *IEEE Trans Syst Man Cybern*, 19(5):1179-1187. https://doi.org/10.1109/21.44033

Borenstein J, Koren Y, 1991. The vector field histogram—fast obstacle avoidance for mobile robots. *IEEE Trans Robot Autom*, 7(3):278-288. https://doi.org/10.1109/70.88137

Chaudhary KL, Ghose D, 2017. Path planning in dynamic environments with deforming obstacles using collision cones. Indian Control Conf, p.87-92. https://doi.org/10.1109/INDIANCC.2017.7846457

Chen JG, Pan LL, Ji SP, et al., 2025. Online temporal fusion for vectorized map construction in mapless autonomous driving. *IEEE Robot Autom Lett*, 10(4):3948-3955. https://doi.org/10.1109/LRA.2025.3544456

Cheng X, Wang ST, Hu YM, et al., 2023. Autonomous mapless navigation via maximum entropy learning. 42nd Chinese Control Conf, p.4562-4567. https://doi.org/10.23919/CCC58697.2023.10240697

Clements WR, Van Delft B, Robaglia BM, et al., 2020. Estimating risk and uncertainty in deep reinforcement learning. https://doi.org/10.48550/arXiv.1905.09638

Fiorini P, Shiller Z, 1998. Motion planning in dynamic environments using velocity obstacles. *Int J Rob Res*, 17(7):760-772. https://doi.org/10.1177/027836499801700706

Fox D, Burgard W, Thrun S, 1997. The dynamic window approach to collision avoidance. *IEEE Robot Autom Mag*, 4(1):23-33. https://doi.org/10.1109/100.580977

Jang Y, Baek J, Han S, 2022. Hindsight intermediate targets for mapless navigation with deep reinforcement learning. *IEEE Trans Ind Electron*, 69(11):11816-11825. https://doi.org/10.1109/TIE.2021.3118407

Jin J, Nguyen NM, Sakib N, et al., 2020. Mapless navigation among dynamics with social-safety-awareness: a reinforcement learning approach from 2D laser scans. IEEE Int Conf on Robotics and Automation, p.6979-6985. https://doi.org/10.1109/ICRA40945.2020.9197148

Kamon I, Rivlin E, Rimon E, 1996. A new range-sensor based globally convergent navigation algorithm for mobile robots. Proc IEEE Int Conf on Robotics and Automation, p.429-435. https://doi.org/10.1109/ROBOT.1996.503814

Khatib O, 1985. Real-time obstacle avoidance for manipulators and mobile robots. Proc IEEE Int Conf on Robotics and Automation, p.500-505. https://doi.org/10.1109/ROBOT.1985.1087247

Li XZ, Gong Y, Jia SM, et al., 2016. VFHF: a combining obstacle avoidance method based on laser rangefinder. IEEE Int Conf on Information and Automation, p.825-830. https://doi.org/10.1109/ICInfA.2016.7831933

Liang J, Payandeh A, Song D, et al., 2024. DTG: diffusion-based trajectory generation for mapless global navigation. IEEE/RSJ Int Conf on Intelligent Robots and Systems, p.5340-5347. https://doi.org/10.1109/IROS58592.2024.10802055

Lumelsky VJ, Stepanov AA, 1987. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2(1):403-430. https://doi.org/10.1007/BF01840369

Marchesini E, Farinelli A, 2020. Discrete deep reinforcement learning for mapless navigation. IEEE Int Conf on Robotics and Automation, p.10688-10694. https://doi.org/10.1109/ICRA40945.2020.9196739

Mathews Z, Lechón M, Blanco Calvo JM, et al., 2009. Insect-like mapless navigation based on head direction cells and contextual learning using chemo-visual sensors. IEEE/RSJ Int Conf on Intelligent Robots and Systems, p.2243-2250. https://doi.org/10.1109/IROS.2009.5354264

Meng XR, Cai J, Wu YL, et al., 2018. A navigation framework for mobile robots with 3D LiDAR and monocular camera. 44th Annual Conf of the IEEE Industrial Electronics Society, p.3147-3152. https://doi.org/10.1109/IECON.2018.8591329

Miao YM, Tang Y, Alzahrani BA, et al., 2021. Airborne LiDAR assisted obstacle recognition and intrusion detection towards unmanned aerial vehicle: architecture, modeling and evaluation. *IEEE Trans Intell Transp Syst*, 22(7):4531-4540. https://doi.org/10.1109/TITS.2020.3023189

Mnih V, Kavukcuoglu K, Silver D, et al., 2013. Playing Atari with deep reinforcement learning. https://doi.org/10.48550/arXiv.1312.5602

Ort T, Paull L, Rus D, 2018. Autonomous vehicle navigation in rural environments without detailed prior maps. IEEE Int Conf on Robotics and Automation, p.2040-2047. https://doi.org/10.1109/ICRA.2018.8460519

Ortega M, Paredes M, Cruz PJ, 2024. Control architecture based on the VFH+ obstacle avoidance algorithm for collective behaviors in swarms of differential drive mobile robots. Argentine Conf on Electronics, p.86-91. https://doi.org/10.1109/CAE59785.2024.10487126

Ortiz S, Yu W, Li XO, 2019. Autonomous navigation in unknown environments using robust SLAM. 45th Annual Conf of the IEEE Industrial Electronics Society, p.5590-5595. https://doi.org/10.1109/IECON.2019.8926620

Pappas P, Chiou M, Epsimos GT, et al., 2020. VFH+ based shared control for remotely operated mobile robots. IEEE Int Symp on Safety, Security, and Rescue Robotics, p.366-373. https://doi.org/10.1109/SSRR50563.2020.9292585

Przewodowski A, Osório FS, 2022. A Monte Carlo particle filter formulation for mapless-based localization. IEEE Intelligent Vehicles Symp, p.1782-1788. https://doi.org/10.1109/IV51971.2022.9827064

Rösmann C, Feiten W, Wösch T, et al., 2012. Trajectory modification considering dynamic constraints of autonomous robots. 7th German Conf on Robotics, p.1-6.

Rudin C, 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat Mach Intell*, 1(5):206-215. https://doi.org/10.1038/s42256-019-0048-x

Tai L, Paolo G, Liu M, 2017. Virtual-to-real deep reinforcement learning: continuous control of mobile robots for mapless navigation. IEEE/RSJ Int Conf on Intelligent Robots and Systems, p.31-36. https://doi.org/10.1109/IROS.2017.8202134

Ulrich I, Borenstein J, 1998. VFH+: reliable obstacle avoidance for fast mobile robots. Proc IEEE Int Conf on Robotics and Automation, p.1572-1577. https://doi.org/10.1109/ROBOT.1998.677362

Ulrich I, Borenstein J, 2000. VFH*: local obstacle avoidance with look-ahead verification. Proc IEEE Int Conf on Robotics and Automation, p.2505-2511. https://doi.org/10.1109/ROBOT.2000.846405

Vouros GA, 2022. Explainable deep reinforcement learning: state of the art and challenges. *ACM Comput Surv*, 55(5):92. https://doi.org/10.1145/3527448

Wapnick S, Manderson T, Meger D, et al., 2021. Trajectory-constrained deep latent visual attention for improved local planning in presence of heterogeneous terrain. IEEE/RSJ Int Conf on Intelligent Robots and Systems, p.460-467. https://doi.org/10.1109/IROS51168.2021.9636422

Wu CF, Wang YL, Ma L, et al., 2021. VFH+ based local path planning for unmanned surface vehicles. IEEE Int Conf on Recent Advances in Systems Science and Engineering, p.1-6. https://doi.org/10.1109/RASSE53195.2021.9686856

Xie LH, Markham A, Trigoni N, 2020. SnapNav: learning mapless visual navigation with sparse directional guidance and visual reference. IEEE Int Conf on Robotics and Automation, p.1682-1688. https://doi.org/10.1109/ICRA40945.2020.9197523

Xu JP, Zhao CN, Yang J, et al., 2025. FDDSGCN: fractional decoupling dynamic spatiotemporal graph convolutional network for traffic forecasting. IEEE Int Conf on Acoustics, Speech and Signal Processing, p.1-5. https://doi.org/10.1109/ICASSP49660.2025.10888084

Yan CZ, Qin JH, Liu QC, et al., 2023. Mapless navigation with safety-enhanced imitation learning. *IEEE Trans Ind Electron*, 70(7):7073-7081. https://doi.org/10.1109/TIE.2022.3203761

Zhang W, Liu N, Zhang YF, 2021. Learn to navigate maplessly with varied LiDAR configurations: a support point-based approach. *IEEE Robot Autom Lett*, 6(2):1918-1925. https://doi.org/10.1109/LRA.2021.3061305

Zhang YF, Wang G, 2017. An improved RGB-D VFH+ obstacle avoidance algorithm with sensor blindness assumptions. 2nd Int Conf on Robotics and Automation Engineering, p.408-414. https://doi.org/10.1109/ICRAE.2017.8291420

Zhong YC, Liu JY, Jian ZQ, et al., 2023. DVT-Tree: dynamic visible topology tree for efficient mapless navigation in maze environments. IEEE 26th Int Conf on Intelligent Transportation Systems, p.4717-4724. https://doi.org/10.1109/ITSC57777.2023.10422402

Zhou QL, Lyu L, Liu H, 2022. Deep reinforcement learning with long-time memory capability for robot mapless navigation. IEEE 25th Int Conf on Computer Supported Cooperative Work in Design, p.1215-1220. https://doi.org/10.1109/CSCWD54268.2022.9776137

Zou Q, Liu JY, Cong M, et al., 2024. Research on continuous control approach of mobile robots for mapless navigation. IEEE 14th Int Conf on CYBER Technology in Automation, Control, and Intelligent Systems, p.185-189. https://doi.org/10.1109/CYBER63482.2024.10749299

List of supplementary materials

1 Path performance test
2 Performance in dynamic environments
3 Details of algorithm time and space complexity analysis
4 Computation latency under different resolutions
5 Pseudocode format for the local minima elimination algorithm
Fig. S1 Route 3 results
Fig. S2 Route 5 results
Fig. S3 Route 6 results
Fig. S4 Dynamic scenario design
Fig. S5 Latency line chart
Table S1 Dynamic environment experimental results
Table S2 Statistical results under different resolutions
Algorithm S1 Proceed_Condition(path, dl_2, da_2, n_2, numofover)