MARS-SLAM: Marker-Assisted Region Scanning for Simultaneous Localization and Mapping

Luigi Maciel Ribeiro[1(✉)], Nadia Nedjah[1], and Paulo Victor R. de Carvalho[2]

[1] State University of Rio de Janeiro (UERJ), Rio de Janeiro, Brazil
{luigi,nadia}@eng.uerj.br
[2] Federal University of Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil
paulov@ien.gov.br

Abstract. This paper presents Marker-Assisted Region Scanning for Simultaneous Localization and Mapping (MARS-SLAM), a novel approach to optimizing the Simultaneous Localization and Mapping process in unknown environments. The method was specifically designed to address the challenges of autonomous exploration in extreme conditions, to enable efficient navigation and offer a systematic approach to determine the completion of mapping. The approach uses markers to indicate unexplored regions, ensuring an organized and complete exploration. During the process, the robot places markers in free areas identified by the LiDAR sensor, located at the sensor's range limit, building a list of regions yet to be explored. The mapping is considered complete when the marker list is empty, indicating that no unexplored regions remain. Target marker selection during navigation is based on age and distance. Age refers to the chronological order in which markers are created, while distance refers to the length of the route from the robot to the marker. The method is validated in two virtual environments of varying complexity. Experimental results demonstrate the effectiveness of MARS-SLAM in achieving complete mapping and accurately identifying mapping completion. Compared to alternative navigation methods, including predefined zigzag routes and routes generated by Ant Colony Optimization (ACO) algorithms, MARS-SLAM shows superior performance, particularly in reducing the number of poses required to complete the mapping, achieving a 64.39% reduction in poses compared to ACO and a 71.07% reduction compared to zigzag.

Keywords: SLAM · Robotic Navigation · Path Planning · Mapping optimization · 2D Mapping

1 Introduction

Simultaneous Localization and Mapping (SLAM) [1] is a cornerstone of mobile robotics [7], enabling autonomous navigation in unknown environments. SLAM

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2026
O. Gervasi et al. (Eds.): ICCSA 2025 Workshops, LNCS 15888, pp. 19–36, 2026.
https://doi.org/10.1007/978-3-031-97596-7_2

allows robots to build and update maps while tracking their position within them, a capability essential for applications ranging from autonomous vehicles [11] to space exploration robots [3]. However, ensuring the completeness of mapping in complex environments remains a critical challenge in SLAM research. Traditional SLAM approaches rely on random or exploratory navigation, which often result in gaps, shadow zones, and redundant paths due to unplanned trajectories [2,13]. These methods typically lack a deterministic process for verifying mapping completeness, leading to inefficient use of time, energy, and computational resources, particularly in scenarios requiring real-time mapping.

This research addresses the challenge of ensuring complete and efficient exploration during SLAM. Marker-Assisted Region Scanning for SLAM (MARS-SLAM) is proposed as a method that utilizes virtual markers derived from Light Detection and Ranging (LiDAR) data to identify and guide the exploration of unmapped regions. Markers are strategically placed in frontier areas and serve as navigation targets, optimizing trajectories based on their age and distance. Mapping is deemed complete when all markers have been visited and removed, ensuring no unexplored regions remain. By optimizing exploration, MARS-SLAM minimizes resource consumption while improving map accuracy and completeness, with applications ranging from industrial robotics to exploration missions in extreme environments. While MARS-SLAM offers significant improvements over traditional methods, it faces challenges in dynamic environments, where moving obstacles may interfere with marker placement. Additionally, its performance depends on the quality of LiDAR data and the computational cost of updating markers in cluttered spaces. Despite these challenges, the method provides a robust and objective solution for achieving mapping completeness.

This paper seeks to answer: How can marker-based exploration optimize the SLAM process to ensure complete and accurate mapping in unknown environments? To address this question, Sect. 2 reviews related works, identifying gaps in current methodologies. Section 3 details the proposed method, while Sect. 4 presents experimental results. Finally, Sect. 5 concludes with contributions, limitations, and future research directions.

2 Related Works

This section presents an overview of significant studies selected to showcase distinct approaches related to path planning and SLAM optimization, highlighting advances and limitations in each domain. Table 1 provides a comparative view of the characteristics each method addresses, facilitating a clear understanding of the innovation that MARS-SLAM brings to the field, particularly in terms of completion detection and uncertainty reduction.

Path planning approaches play a crucial role in enhancing SLAM performance by optimizing navigation and localization strategies. In [4], a method integrating inertial SLAM with trajectory planning for Unmanned Aerial Vehicles (UAVs) is proposed, leveraging entropy and observability to improve navigation accuracy in GPS-denied environments. This approach enhances map quality and localization by utilizing dynamic UAV maneuvers and real-time computational solutions, proving highly effective for applications in surveillance and exploration.

Table 1. Characteristics's comparison addressed in related works.

Feature	[4]	[5]	[9]	[12]	[14]	[8]	[10]	MARS-SLAM
Indoor Environment	✗	✓	✓	✓	✓	✓	✗	✓
Outdoor Environment	✓	✗	✗	✓	✓	✓	✓	✓
Path Planning	✓	✓	✓	✓	✓	✓	✓	✓
Dynamic Environments	✗	✗	✓	✗	✗	✗	✗	✗
Exploration Optimization	✓	✓	✗	✓	✗	✓	✓	✓
Real-Time Processing	✓	✓	✓	✓	✓	✓	✗	✓
Conclusion Detection	✗	✗	✗	✗	✗	✗	✗	✓
Map Building	✓	✓	✓	✓	✓	✓	✓	✓
Uncertainty Reduction	✓	✓	✓	✗	✓	✓	✗	✓

Similarly, in [9], an adaptation of the D* algorithm for active SLAM addresses the challenge of handling negative edge weights, a significant limitation in traditional path finding algorithms. This adaptation enables efficient path recalculation, improves loop closure, and reduces localization uncertainty, demonstrating robust performance in both simulations and real-world scenarios. Lastly, [14] evaluates the performance of SLAM algorithms (GMapping, Hector-SLAM, and Cartographer) when combined with the A* algorithm for global path planning and DWA for local planning in indoor rescue scenarios. The study highlights the specific strengths and limitations of these algorithmic combinations, offering valuable insights into selecting optimal solutions for rescue tasks.

SLAM optimization approaches, on the other hand, focus on improving the efficiency, accuracy, and overall applicability of SLAM in diverse environments. In [5], the P-SLAM algorithm predicts environmental structures in unexplored regions using a Rao-Blackwellized particle filter, which significantly reduces exploration time and accelerates the mapping process. This Bayesian formulation, combined with frontier cell strategies, proves effective for real-time applications, as validated through experimental results. In [12], an exploration EM algorithm, adapted for pose SLAM, is introduced, utilizing point cloud segmentation to construct maps and enhance place recognition. The method also predicts future uncertainty, achieving higher exploration rates and improved accuracy in real-time applications. Furthermore, [8] presents a boundary exploration method that integrates Double-Layer Rapid-Exploration Random Tree (DL-RRT), Affinity Propagation clustering, and Bayesian Optimization. This approach reduces execution time and path length while maintaining high stability, offering notable improvements over traditional RRT methods. Finally, [10] proposes FIT-SLAM, which combines Fisher information and navigability estimates for 3D exploration. By transforming 3D exploration spaces into 2D navigability maps, this method reduces computational costs, improves exploration efficiency, and enhances SLAM accuracy, as demonstrated through extensive simulations and experiments.

3 Marker-Assisted Region Scanning

To ensure the effective and accurate completion of mapping, we propose a methodology that uses markers to identify and track unexplored regions on the map. The primary goal of this method is to ensure that all areas of the environment are mapped, avoiding gaps that could compromise the navigability and usefulness of the generated map. Markers play a key role in this process, acting as indicators of regions that still need to be explored. As the robot progresses in building the map, it adds markers to unexplored areas based on sensor readings. These markers are stored in a list, which over time reflects the progress of the mapping. The process is considered complete when the list of markers is empty, indicating that no unexplored regions remain.

Initially, the robot performs a full 360° rotation, analyzing the surrounding environment and placing markers in locations corresponding to unexplored areas. Once the rotation is completed, the robot selects a marker as a target and moves toward it. The robot then chooses the next most relevant marker and repeats the process. This cycle of marker selection and navigation continues until all markers are removed, signaling the completion of the environment mapping. Algorithm 1 describes the process in detail.

Algorithm 1. Marker-Assisted Region Scanning

Ensure: Grid map of the environment (G)
$G \leftarrow$ unknown map ▷ Grid map with all cells as unknown (gray pixels)
$P, M, L, \Psi \leftarrow$ empty list ▷ Pose list, marker list, LiDAR measurements and path
$\eta \leftarrow$ null ▷ Target marker
$k \leftarrow 0$
$P_k \leftarrow$ initial position
while full rotation not completed **do**
    $P_{k+1}, L \rightarrow$ UpdateRobotState$(P_k, \eta, rotate, G, \Psi)$ ▷ Algorithm 2
    $G \rightarrow$ UpdateGridMap(P_{k+1}, L)
    $M \rightarrow$ MarkerAddition(G, L, M) ▷ Algorithm 3
    $k \leftarrow k + 1$
end while
while $M \neq$ empty list **do**
    if η is null **then**
        $\eta \rightarrow$ MarkerSelection(M) ▷ Algorithm 4
    end if
    $P_{k+1}, L \rightarrow$ UpdateRobotState$(P_k, \eta, towards, G, \Psi)$
    $G \rightarrow$ UpdateGridMap(P_{k+1}, L)
    if η is reached **then**
        $\eta \leftarrow$ null;
    end if
    $M \rightarrow$ MarkerRemoval(G, M) ▷ Algorithm 5
    $M \rightarrow$ MarkerAddition(G, L, M)
    $k \leftarrow k + 1$
end while

3.1 Update Robot State

This process continuously updates the robot's state during navigation, determining its new pose and LiDAR measurements based on its current action and the specified target. It plays a crucial role in ensuring that the robot remains aligned with its goal, dynamically adjusting its trajectory as needed while collecting essential data to support the ongoing mapping of the environment. The algorithm enables the robot to adapt its trajectory in real time, for effective decision-making that enhances its ability to navigate complex environments.

Algorithm 2 requires the following inputs: the robot's current pose (P_k), the target marker (η), the action to be executed (A), the grid map (G), and the path list (Ψ). It outputs the robot's updated pose (P_{k+1}) and the new LiDAR measurements (L). The process is divided into three main steps: First, the algorithm identifies the action assigned to the robot, which may involve either a rotation or movement toward the target. In the case of rotation, the algorithm updates the robot's pose and gathers new LiDAR measurements. If the action involves moving toward the marker, the algorithm checks for obstacles along the path. If the path is clear, the robot moves directly toward the target, updates the LiDAR measurements, and resets the path list to empty. If obstacles are detected, the algorithm employs a path-planning approach to determine an alternative route, stores the intermediate points in the list Ψ, and guides the robot along these points to bypass the obstacle and reach the target marker.

Algorithm 2. Update Robot State

Require: Current Pose (P_k), Target marker (η), Action (A), Grid Map (G), Path (Ψ)
Ensure: Updated Pose (P_{k+1}), Updated LiDAR Measurements (L)

if A is rotate **then**
    $P_{k+1} \leftarrow$ rotate robot
    $L \leftarrow$ LiDAR measurements
else if A is towards η **then**
    if way to η is free **then**

$L \leftarrow$ LiDAR measurements
        $\Psi \leftarrow$ empty list
    else
        if Ψ is empty **then**

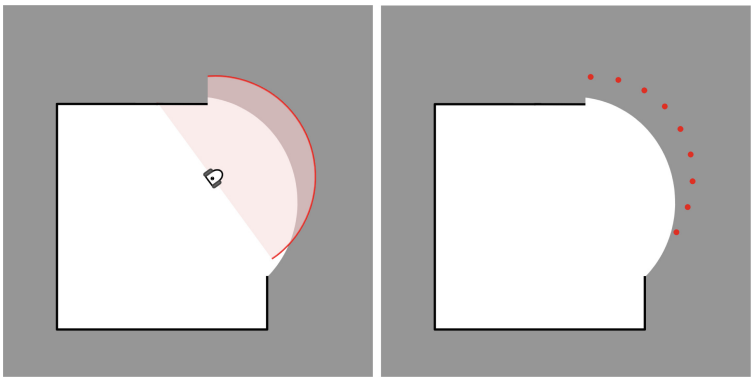
$\Psi \leftarrow$ PathFinding(P, k, z)) ▷ Algorithm 6
        end if

$L \leftarrow$ LiDAR measurements
        if $\Psi[0]$ has been reached **then**
            remove $\Psi[0]$ of the Ψ
        end if
    end if
end if

3.2 Marker Addition

Markers are created based on readings from the LiDAR, a sensor used by the robot to map the environment. Each measurement provides the distance to the first obstacle detected in the respective direction. The LiDAR has a maximum range, and measurements beyond this limit are not considered. During the robot's movement, at each registered pose, all LiDAR measurements are analyzed to identify potential obstacle-free areas. To ensure efficient distribution and avoid redundancies, the algorithm imposes a restriction: a new marker can only be added if no other marker exists within a predefined minimum distance. Figure 1 illustrates the process of identifying free areas and adding markers.

Algorithm 3 details the marker addition procedure. It takes as input the current grid map (G), the LiDAR measurements (L), and the list of existing markers (M), while also defining the minimum distance between markers, denoted as γ. For each LiDAR measurement, the algorithm checks whether it is within the sensor's maximum range. If the measurement exceeds this limit, the corresponding coordinates (x_l, y_l) are determined. Then, a loop checks whether a marker already exists within the minimum distance γ. If a marker is found too close, the creation of a new marker is avoided. Otherwise, if there are no nearby markers and the corresponding area on the map is unknown (represented by a gray pixel), the point (x_l, y_l) is added as a new marker. However, if the area already contains a marker, the algorithm checks the neighboring cells around the detected point. If any of these neighboring cells are also unknown, a new marker is added at that location.



(a) Identification of free areas (b) Marker inclusion

Fig. 1. Process of identifying free areas and adding markers during the robot's navigation. Figure (a) shows how free areas are identified based on LiDAR readings; figure (b) illustrates the addition of markers in these areas, ensuring a minimum distance between them.

Algorithm 3. Marker Addition

Require: Grid map (G), LiDAR measurements (L), Marker list (M)
Ensure: Updated Marker List (M)

$\gamma \leftarrow$ minimum distance between markers
for each l in L **do**
  if l is outside LiDAR's maximum range **then**
    $x_l, y_l \leftarrow$ coordinates of l
    $\kappa \leftarrow$ true;
    while κ **do**
      for each m in M' **do**
        $x_m, y_m \leftarrow$ coordinates of m
        if $\sqrt{(x_l - x_m)^2 + (y_l - y_m)^2} < \gamma$ **then**
          $\kappa \leftarrow$ false;
        end if
      end for
    end while
    if κ **then**
      if $G[x_l, y_l]$ is unknown **then**
        add (x_l, y_l) to M as a new marker;
      else
        $\tau \leftarrow$ true;
        $N \leftarrow \{(x_l + i, y_l + j) \mid i, j \in \{-1, 0, 1\}, (i, j) \neq (0, 0)\}$
        for each n in N **do**
          if $G[n]$ is unknown and τ **then**
            add (x_l, y_l) to M as a new marker
            $\tau \leftarrow$ false;
          end if
        end for
      end if
    end if
  end if
end for

3.3 Marker Selection

After the initial navigation period, the robot must choose a marker as a target to map an unexplored area. For this selection, the robot identifies markers with free access, meaning those for which a direct path can be traced from the robot's current position without encountering obstacles. Once the accessible markers are identified, the target marker is chosen based on two criteria: age and distance. Each marker is associated with a specific pose, recorded in a pose list that starts at pose 0 and progresses up to the robot's current pose. For example, a marker created at pose 10 is considered older than one created at pose 50. The robot then decides between the oldest marker and the closest marker. If the distance to the oldest marker exceeds μ times the distance to the closest marker, the closest marker is selected as the target (μ is a parameter adjustable according to the desired navigation behavior). Otherwise, the oldest marker is chosen. When

there are no markers with free access, the robot uses the tournament selection method to determine the target.

Algorithm 4 describes this logic in a structured way, starting with the filtering of markers with free access (M_f) and, in their absence, applying the tournament selection method. After this filtering, the algorithm considers the distance (d_o) and the age of the oldest marker (η_o), comparing them with the distance to the closest marker (η_c). Based on this comparison and the parameter μ, the algorithm determines the target marker (η), which will guide the robot in the next exploration step. This efficient selection process allows the robot to balance the exploration of new areas while prioritizing older markers, while also optimizing the path by selecting closer markers when appropriate.

Algorithm 4. Marker Selection

Require: Marker list (M), Robot's current pose (P_k)
Ensure: Target marker (η)
$M_f \leftarrow$ filter markers from M with free access
$\mu \leftarrow$ navigation balance parameter
if M_f is empty **then**
    $M_t \leftarrow$ randomly select ρ markers from M
    $\eta \leftarrow$ marker with the shortest distance in M_t
    return η
end if
$\eta_o \leftarrow$ oldest marker in M_f
$\eta_c \leftarrow$ closest marker in M_f
$x_{P_k}, y_{P_k} \leftarrow$ coordinates of P_k
$x_{\eta_o}, y_{\eta_o} \leftarrow$ coordinates of η_o
$x_{\eta_c}, y_{\eta_c} \leftarrow$ coordinates of η_c
$d_o \leftarrow \sqrt{(x_{P_k} - x_{\eta_o})^2 + (y_{P_k} - y_{\eta_o})^2}$
$d_c \leftarrow \sqrt{(x_{P_k} - x_{\eta_c})^2 + (y_{P_k} - y_{\eta_c})^2}$
if $d_o > \mu \times d_c$ **then**
    $\eta \leftarrow \eta_c$
else
    $\eta \leftarrow \eta_o$
end if

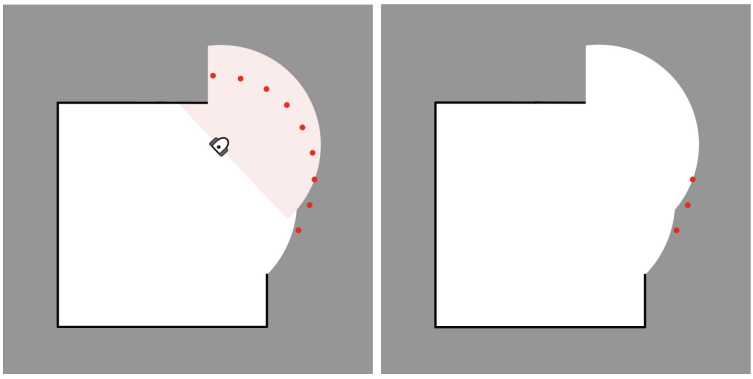
3.4 Marker Removal

Marker removal is an essential step to ensure that the map remains up to date, reflecting only the areas that still need to be explored. During the robot's navigation, at each step, it is checked whether any of the previously added markers are located in a region that has already been mapped. Specifically, a marker is removed when its cell, as well as its neighboring cells, are no longer unknown. Figure 2 illustrates the process of map updating and the subsequent removal of markers during the robot's navigation. Figure 2(a) shows the map being updated

as the robot explores new areas. During these updates, previously unknown regions are converted into mapped areas, indicating that the robot has already explored these cells. Figure 2(b) demonstrates the marker removal process.

Algorithm 5 details the marker removal procedure. It takes as input the current grid map G and the list of markers M, aiming to return an updated marker list M' containing only those located in unexplored regions of the map. The process begins by creating an empty list M', where the markers that remain after evaluation will be stored. For each marker m in the original list M, the algorithm retrieves its coordinates (x_m, y_m), which indicate its position on the grid map. It then checks whether the corresponding cell in the map $G[x_m, y_m]$ is still unknown (represented by a gray value on the grid). If this cell remains unexplored, the marker m is kept in the updated list M'.

If the cell has already been explored, the algorithm proceeds to evaluate the neighboring cells around (x_m, y_m). These neighboring cells are represented by the set N, which includes the eight adjacent cells surrounding marker m. The algorithm then checks whether any of the cells in N are still unknown on the map G. If at least one of these neighboring cells remains unexplored, the marker m is retained in the updated list M'. Otherwise, if all neighboring cells have already been explored, the marker is removed, as it no longer points to an unexplored area. At the end of the procedure, the algorithm returns the updated marker list M', ensuring that only those markers located in or near unexplored regions are retained. This process guarantees the removal of markers in fully explored areas, optimizing both the marker distribution and the efficiency of the mapping process.



(a) Map update after robot navigation (b) Marker removal

Fig. 2. Process of map updating and marker removal during navigation. Figure(a) illustrates the map update with new information obtained by the robot; figure(b) shows the removal of markers from areas that are no longer unknown, ensuring that only unexplored regions retain markers.

Algorithm 5. Marker Removal

Require: Grid map (G), Marker list (M)
Ensure: Updated marker list (M')
$M' \leftarrow$ empty list
for each m in M **do**
  $x_m, y_m \leftarrow$ coordinates of m
  if $G[x_m, y_m]$ is unknown **then**
    add m to M'
  else
    $\alpha \leftarrow$ true
    $N \leftarrow \{(x_m + i, y_m + j) \mid i, j \in \{-1, 0, 1\}, (i, j) \neq (0, 0)\}$
    for each n in N **do**
      if $G[n]$ is unknown and α **then**
        add m to M'
        $\alpha \leftarrow$ false
      end if
    end for
  end if
end for

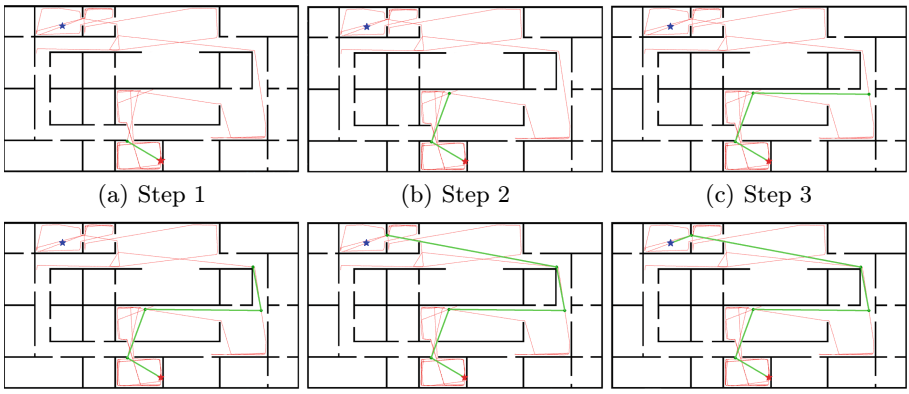
3.5 Path Finding

In situations where the target marker is not directly accessible, the robot must return to the pose where the marker was created to ensure a clear path. While backtracking through the pose list is an option, it can be inefficient. To optimize this, a method was developed to reduce the set of poses needed to reach the target. By leveraging the recorded path, the method identifies the smallest set of poses connecting the robot's current position to the target marker. Starting from the current position, the robot traverses poses at intervals defined by a constant until the target is reached. At each step, the closest pose with a clear path to the current position is selected, ensuring an efficient route.

The specifics of this procedure are detailed in Algorithm 6. Initially, the algorithm sets the path Ψ as an empty list and defines a search step size s. A loop continues until a valid path is found. In each iteration, the algorithm generates a set of candidate poses Ω by calculating the indices based on the target marker's pose index δ and the current pose index k, considering increments defined by the search step size. If there are no candidate poses available, the loop terminates. For each pose in the candidate set Ω, the algorithm checks for obstacles between the current pose and the candidate pose. If a clear path is identified (*i.e.*, no obstacles), the algorithm updates the path Ψ by adding the candidate pose and then updates the current pose index k to the candidate pose. This process repeats until a direct, obstacle-free route is established between the robot's current position and the target marker. Figure 3 visually illustrates this path optimization process, demonstrating the stages of checking and selecting intermediate poses until the final path is defined.

Algorithm 6. Path finding

Require: Pose list (P), Robot's pose index (k), Target marker's pose index (z)
Ensure: Path (Ψ)
$\Psi \leftarrow$ empty list
$s \leftarrow$ search step size
$\Omega = \{i \mid i = z + ns,\ n \in \mathbb{Z}^+ \text{ e } i \leq k\}$
while Ω is not empty **do**
    $\tau \leftarrow$ true;
    for each ω in Ω **do**
        obstacle $\leftarrow$ check for obstacles between $P[\omega]$ and $P[k]$;
        if without *obstacle* and τ **then**
            add $P[\omega]$ to Ψ;
            $k \leftarrow \omega$;
            $\tau \leftarrow$ false;
        end if
    end for
    $\Omega = \{i \mid i = z + ns,\ n \in \mathbb{Z}^+ \text{ e } i \leq k\}$
end while



(a) Step 1 (b) Step 2 (c) Step 3

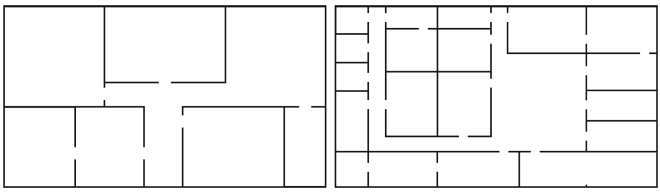
(d) Step 4 (e) Step 5 (f) Step 6

Fig. 3. The set of images shows the process of locating the optimized path using the list of poses. The goal is to identify a path from the red star (starting point) to the blue star (target point). The red points represent the list of posed already traversed by the robot. The green path represents the so-far constructed route. (Color figure online)

4 Results

In this section, the results of the experiments, conducted to evaluate the efficiency and stability of the proposed MARS-SLAM method, are presented. The experiments were performed in two distinct virtual environments to analyze the performance of the approach under different conditions. The maps used in the simulations are shown in Fig. 4, with increasing levels of complexity: Map 1 is the least complex, followed by the more challenging Map 2. These maps are designed

to thoroughly test the adaptability of the MARS-SLAM method. The simulations were carried out on a computer with the following specifications: Intel(R) Core(TM) i7-10750H CPU @ 2.59GHz, 24 GB of RAM, and Microsoft Windows 11 Home operating system (version 10.0.22631). The development environment and tools used for the implementation and simulations included Python 3.7.7 and the necessary libraries for calculations and visualization of the results.



(a) Map 1 (b) Map 2

Fig. 4. Maps of the virtual environments used in the simulations, designed with different levels of complexity to test the adaptability of the MARS-SLAM method. Map 1 presents a simple layout, while Map 2 has intermediate complexity.

To compare the performance of MARS-SLAM, two alternative routes are created, serving as benchmarks to establish a solid basis for evaluation. The goal of these simulations is to provide a comparative analysis of MARS-SLAM's efficiency in mapping environments with no prior knowledge. A pragmatic approach is adopted, assuming prior knowledge of the map, which allows for the distribution of markers in a grid format. The spacing between the markers is proportional to the LiDAR scan limit, ensuring that, by visiting all markers, the mapping process is completed. These routes will provide a reference for evaluating MARS-SLAM's efficiency. By comparing the performance of MARS-SLAM with these routes, we aim to determine if the method can map the entire environment with a similar number of poses, even without prior knowledge. If MARS-SLAM achieves comparable results to the alternative routes, it would indicate that the method is efficient and capable of producing similar results despite lacking map knowledge.

Two alternative routes are developed for comparison with MARS-SLAM, using markers distributed in a grid:

1. Standard Zigzag Route: In this route, the robot starts at the first marker in the first row and navigates left to right on even-numbered rows and right to left on odd-numbered rows, creating a zigzag pattern.
2. ACO-Optimized Route: In this route, a heuristic based on the Ant Colony Optimization (ACO) algorithm [6] is applied to construct a route that closely approximates the optimal path.

These routes are generated for the three virtual scenarios shown in Fig. 4. Figure 5 illustrates the mapping progress on Map 1, showing how each method

(Zigzag, ACO, and MARS-SLAM) covers the environment at three different stages: 0%, 50%, and 100%. Similarly, Fig. 6 presents the same stages for Map 2, demonstrating how each method performs in a more complex layout. In each of these three figures, the areas already mapped are shaded in pink, red dots represent the markers added by the robot, and the small pink dots indicate the robot's recorded poses throughout the mapping process.

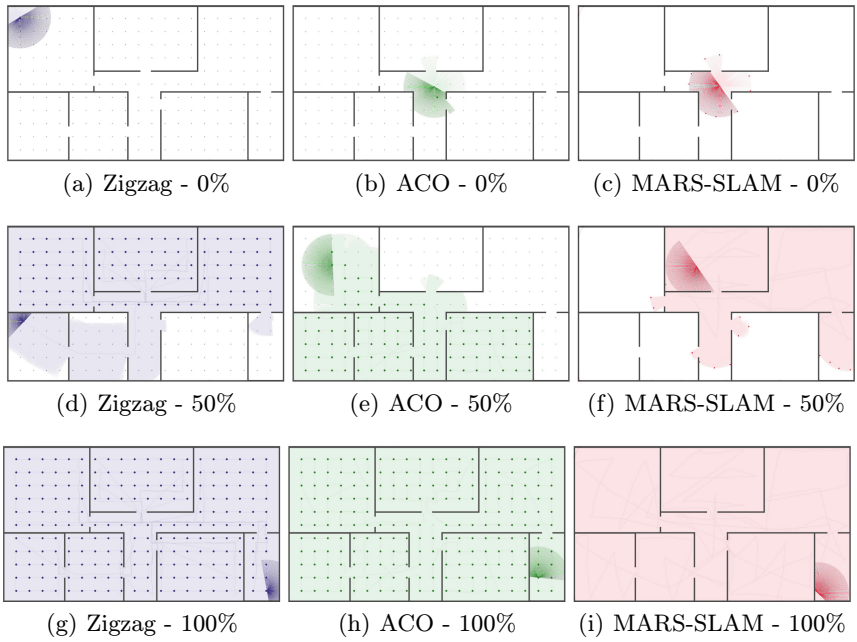
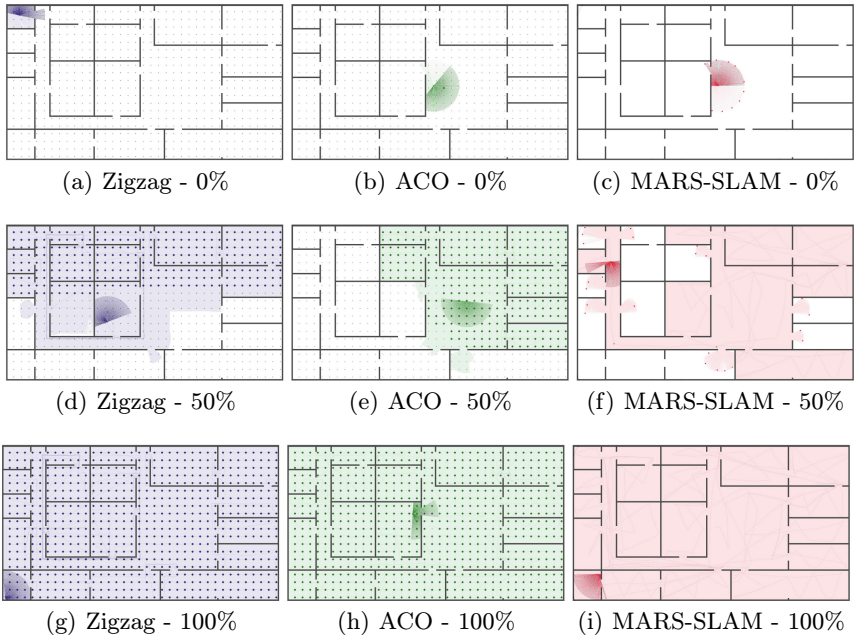


Fig. 5. Mapping progress on Map 1 using the three methods (Zigzag, ACO, and MARS-SLAM) at the stages of 0%, 50%, and 100%.

The simulations demonstrate that MARS-SLAM successfully achieves its main objective, which is to perform a complete mapping of the environment, utilizing an efficient stopping criterion to identify the conclusion of the navigation task. Each simulation is repeated 10 times to ensure the robustness of the results. Table 2 presents the averages and standard deviations for the number of poses required for complete mapping by each method, as well as the mapping completeness. MARS-SLAM achieves the lowest number of poses in both maps, with 4404.00($\pm$280.85) poses in Map 1 and 7618.30($\pm$1277.47) poses in Map 2, significantly outperforming Zigzag and ACO. Table 3 presents the average times and standard deviations related to three time categories: preprocessing, navigation, and total simulation time. The preprocessing time refers to the initial phase before navigation, which includes setting up specific parameters for each method. MARS-SLAM eliminates preprocessing time, unlike ACO, which



(a) Zigzag - 0% (b) ACO - 0% (c) MARS-SLAM - 0%

(d) Zigzag - 50% (e) ACO - 50% (f) MARS-SLAM - 50%

(g) Zigzag - 100% (h) ACO - 100% (i) MARS-SLAM - 100%

Fig. 6. Mapping progress on Map 2 using the three methods (Zigzag, ACO, and MARS-SLAM) at the stages of 0%, 50%, and 100%.

has the highest preprocessing time, reaching 1750.5(±227.7) seconds in Map 1 and 12503.2(±2501.3) seconds in Map 2. Navigation time represents the period in which the robot actively explores the environment. MARS-SLAM completes navigation in 5725.2(±364.1) seconds for Map 1 and 13331.0(±2554.9) seconds for Map 2, achieving a reduction compared to Zigzag, 10764.6(±23.0) seconds and 28676.6(±400.5) seconds, respectively, and ACO 6381.2(±48.6) seconds and 18730.5(±353.3) seconds, respectively.

Table 2. Data showing the number of poses (NP) and mapping completeness (MC) used by each method in the two maps: Zigzag, ACO, and MARS-SLAM.

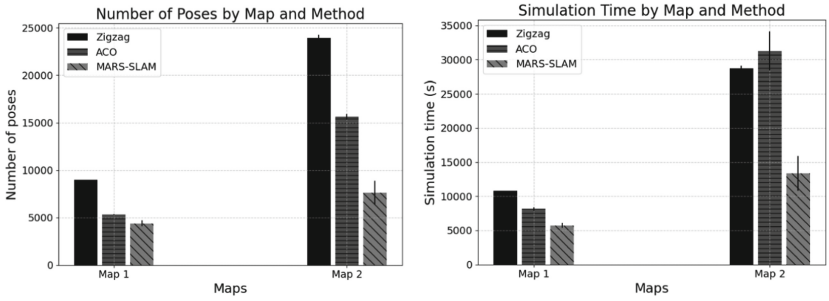
	Map	Zigzag	ACO	MARS-SLAM
NP	M1	8970.50 ± 19.16	5317.70 ± 40.49	$\mathbf{4404.00 \pm 280.85}$
	M2	23897.20 ± 333.76	15608.73 ± 294.43	$\mathbf{7618.30 \pm 1277.47}$
MC %	M1	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
	M2	100.0 ± 0.0	100.0 ± 0.0	99.7 ± 0.3

Figure 7 illustrates the comparison between the evaluated methods based on two criteria: the number of poses and the total simulation time. Figure 7(a)

Table 3. Data showing the preprocessing time (PT), navigation time (NT), and total simulation time (TT) used by each method in the two maps: Zigzag, ACO, and MARS-SLAM.

	Map	Zigzag	ACO	MARS-SLAM
PT (s)	M1	6.2 ± 1.1	1750.5 ± 227.7	$\mathbf{0.0 \pm 0.0}$
	M2	35.3 ± 5.2	12503.2 ± 2501.3	$\mathbf{0.0 \pm 0.0}$
NT (s)	M1	10764.6 ± 23.0	6381.2 ± 48.6	$\mathbf{5725.2 \pm 364.1}$
	M2	28676.6 ± 400.5	18730.5 ± 353.3	$\mathbf{13331.0 \pm 2554.9}$
TT (s)	M1	10770.8 ± 24.1	8131.7 ± 276.3	$\mathbf{5725.2 \pm 364.1}$
	M2	28711.9 ± 405.7	31233.7 ± 2854.6	$\mathbf{13331.0 \pm 2554.9}$

compares the number of poses used by each method across the three maps; Fig. 7(b) compares the required simulation time. It is observed that for Maps 1 and 2, MARS-SLAM demonstrates consistency, showing a significantly lower number of poses compared to the other methods.



(a) Number of poses (b) Simulation time (s)

Fig. 7. Comparison between the methods for the number of poses (a) and total simulation time (b) across three different maps.

Simulation time impacts resource costs during SLAM, but the number of poses is a more critical indicator of mapping quality. A lower number of poses reflects higher efficiency, avoiding redundancies while producing accurate and compact maps suitable for reuse. Table 4 shows the percentage variations in the number of poses across methods and scenarios, highlighting MARS-SLAM's superior performance. In Map 1, MARS-SLAM achieves a 50.9% reduction compared to Zigzag and 17.1% compared to ACO, while in Map 2, the reductions reach 68.1% and 48.8%, respectively. Unlike ACO and Zigzag, which require prior map knowledge for marker grid distribution, MARS-SLAM autonomously explores unknown environments, adapting in real time. This adaptability makes MARS-SLAM a robust solution for unpredictable scenarios, where pre-mapped data is unavailable.

Table 4. The number of poses variation in Maps 1 and 2, comparing Zigzag, ACO, and MARS-SLAM methods.

		The Number of Poses Variation (%) Compare To		
		Zigzag	ACO	MARS-SLAM
Map 1	Zigzag	0	+68.7	+103.7
	ACO	−40.7	0	+20.8
	MARS-SLAM	**−50.9**	−17.1	0
Map 2	Zigzag	0	+53.1	+213.6
	ACO	−34.7	0	+105.0
	MARS-SLAM	**−68.1**	−48.8	0

In terms of completeness, as presented in Table 2, all methods fully map the environment in Map 1, achieving 100.0% completeness. In Map 2, MARS-SLAM attains 99.7% completeness, with a slight variation due to the shadow effect, illustrated in Fig. 8. This phenomenon occurs when the LiDAR's limited reading angle (–90° to 90°) and marker placement strategy fail to detect narrow regions, such as doorways or dense structures. For example, when the robot passes perpendicularly through a narrow doorway, it may only detect the doorframe, missing inner spaces and leaving areas unmapped. This issue intensifies in dense environments, where marker dispersion is insufficient, creating shadow zones that falsely appear as mapped.

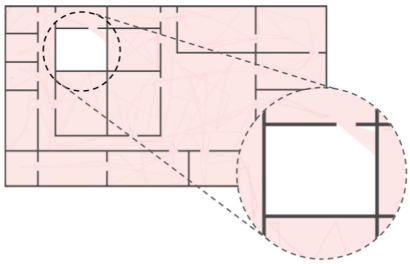


Fig. 8. Example of the shadow effect caused by the robot navigating through narrow areas. In such cases, the LiDAR may fail to accurately detect unmapped regions, leading to shadow zones where the robot incorrectly assumes the area has been fully mapped.

Overall, MARS-SLAM yields superior results, significantly reducing the number of poses while maintaining mapping completeness. This efficiency translates into more effective navigation and higher-quality mapping, making MARS-SLAM a valuable tool for real-world applications. A crucial distinction is that MARS-SLAM is designed to operate in completely unknown environments, unlike ACO and Zigzag, which require prior knowledge of the map for marker placement. This autonomy enables MARS-SLAM to adapt dynamically to the

environment, making it a more robust and versatile solution for scenarios where no pre-mapped data is available.

5 Conclusion

This paper introduces MARS-SLAM, a novel method leveraging markers to optimize SLAM in unknown environments. MARS-SLAM ensures efficient and accurate mapping by identifying and tracking unexplored regions while minimizing resource usage, such as the number of poses. Experiments in two virtual scenarios demonstrated its superior performance compared to methods like Zigzag and ACO, achieving high mapping completeness while significantly reducing the number of poses and total simulation time. In Map 1, MARS-SLAM reduced the number of poses by 50.9% compared to Zigzag and 17.2% compared to ACO. In Map 2, the reduction was even more pronounced, with MARS-SLAM using 68.1% fewer poses than Zigzag and 51.2% fewer than ACO. This efficiency translates into lower computational cost and improved mapping quality, making MARS-SLAM particularly valuable for autonomous mobile robotics.

The use of markers provides a robust mechanism for optimizing the mapping process and verifying completeness, though challenges such as the shadow effect in narrow areas, caused by LiDAR limitations, led to slight reductions in mapping accuracy. In Map 2, MARS-SLAM achieved 99.7% mapping completeness, with minor errors due to undetected regions. Future improvements in sensor configuration, such as increasing measurement density or refining marker placement strategies, could mitigate these limitations.

While MARS-SLAM excelled in virtual environments, its performance in real-world settings remains to be validated. Factors like dynamic obstacles, varying obstacle density, and alternative sensor types (e.g., RGB-D cameras or radars) may impact results. Adapting MARS-SLAM to dynamic environments and integrating it with real-time autonomous navigation systems could broaden its applicability, including challenging missions like search and rescue. Additionally, optimizing preprocessing time and further refining the marker selection algorithm could enhance computational efficiency. These adaptations could strengthen MARS-SLAM's robustness and utility across diverse scenarios, solidifying its role as a powerful tool for autonomous exploration.

References

1. Aulinas, J., Petillot, Y., Salvi, J., Lladó, X.: The slam problem: a survey. Artifi. Intell. Res. Developm., 363–371 (2008)
2. Blochliger, F., Fehr, M., Dymczyk, M., Schneider, T., Siegwart, R.: Topomap: topological mapping and navigation based on visual slam maps. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 3818–3825. IEEE (2018)
3. Bogue, R.: Robots for space exploration. Indust. Robot: Inter. J. **39**(4), 323–328 (2012)

4. Bryson, M., Sukkarieh, S.: Active airborne localisation and exploration in unknown environments using inertial slam. In: 2006 IEEE Aerospace Conference, pp. 13–pp. IEEE (2006)
5. Chang, H.J., Lee, C.G., Lu, Y.H., Hu, Y.C.: P-slam: Simultaneous localization and mapping with environmental-structure prediction. IEEE Trans. Rob. **23**(2), 281–293 (2007)
6. Dorigo, M., Birattari, M., Stutzle, T.: Ant colony optimization. IEEE Comput. Intell. Mag. **1**(4), 28–39 (2006)
7. Garcia, E., Jimenez, M.A., De Santos, P.G., Armada, M.: The evolution of robotics research. IEEE Robot. Autom. Mag. **14**(1), 90–103 (2007)
8. Liu, Y., Zhou, Z., Sang, H., Yu, S., Yan, Y., Er, M.J.: Efficient exploration of mobile robot based on dl-rrt and ap-bo. IEEE Trans. Instrument. Measure. (2024)
9. Maurović, I., Seder, M., Lenac, K., Petrović, I.: Path planning for active slam based on the d* algorithm with negative edge weights. IEEE Trans. Syst. Man Cybernet. Syst. **48**(8), 1321–1331 (2017)
10. Saravanan, S., Chauffaut, C., Chanel, C., Vivet, D.: Fit-slam–fisher information and traversability estimation-based active slam for exploration in 3d environments. arXiv preprint arXiv:2401.09322 (2024)
11. Takleh, T., Bakar, N.A., Rahman, S.A., Hamzah, R., Aziz, Z.: A brief survey on slam methods in autonomous vehicle. Int. J. Eng. Technol **7**(4), 38–43 (2018)
12. Wang, J., Shan, T., Englot, B.: Virtual maps for autonomous exploration with pose slam. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4899–4906. IEEE (2019)
13. Zhan, Z., Jian, W., Li, Y., Yue, Y.: A slam map restoration algorithm based on submaps and an undirected connected graph. IEEE Access **9**, 12657–12674 (2021)
14. Zhang, X., Lai, J., Xu, D., Li, H., Fu, M.: 2d lidar-based slam and path planning for indoor rescue using mobile robots. J. Adv. Transp. **2020**(1), 8867937 (2020)