This paper has been accepted in ***Journal of Intelligent and Robotic Systems (JIRS)****, kindly cite accordingly.*

Efficient Multi-robot Active SLAM

Muhammad Farhan Ahmed[1*†], Matteo Maragliano[2†], Vincent Frémont[1], Carmine Tommaso Recchiuto[2]

[1*]Laboratoire des Sciences du Numérique de Nantes (LS2N), CNRS, Ecole Centrale de Nantes (ECN), 1 Rue de la Noë, Nantes, 44300, France.
[2]University of Genoa, 5 via Balbi, Genoa, 16126, Italy.

*Corresponding author(s). E-mail(s): Muhammad.Ahmed@ec-nantes.fr;
Contributing authors: 4636216@studenti.unige.it;
vincent.fremont@ec-nantes.fr; carmine.recchiuto@dibris.unige.it;
†These authors contributed equally to this work.

Abstract

Autonomous exploration in unknown environments remains a fundamental challenge in robotics, particularly for applications such as search and rescue, industrial inspection, and planetary exploration. Multi-robot active SLAM presents a promising solution by enabling collaborative mapping and exploration while actively reducing uncertainty. However, existing approaches often suffer from high computational costs and inefficient frontier management, making them computationally expensive for real-time applications. In this paper, we introduce an efficient multi-robot active SLAM framework that incorporates a frontier-sharing strategy to enhance robot distribution in unexplored environments. Our approach integrates a utility function that considers both pose graph uncertainty and path entropy, achieving an optimal balance between exploration coverage and computational efficiency. By filtering and prioritizing goal frontiers, our method significantly reduces computational overhead while preserving high mapping accuracy. The proposed framework has been implemented in ROS and validated through simulations and real-world experiments. Results demonstrate superior exploration performance and mapping quality compared to state-of-the-art approaches.

Keywords: SLAM, Frontier Detection, Mapping, Entropy

1 Introduction

Simultaneous Localization and Mapping (SLAM) involves techniques where a robot simultaneously localizes itself and maps its environment while navigating. SLAM is divided into localization, which estimates the robot's pose relative to the map, and mapping, which reconstructs the environment using information from sensors like cameras, inertial measurement units, and lidars.

Modern approaches, as described by [1], [2] and [3] formulate the SLAM problem using a bipartite graph, where nodes represent robot or landmark poses, and edges represent measurements between them. By considering a robot with state $x \in \mathbb{R}^2$ describing its position and orientation (pose), the objective is to find the optimal state vector $\mathbf{x}^*$ which minimizes the measurement error $\mathbf{e}_i(\mathbf{x})$ weighted by the covariance matrix $\Omega_i \in \mathbb{R}^{l \times l}$ which encapsulates the measurement uncertainty. Where i is the index of edge measurement and l is the dimension of the state vector as shown in Equation 1.

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_i \mathbf{e}_i^T(\mathbf{x}) \Omega_i \mathbf{e}_i(\mathbf{x}) \tag{1}$$

Most SLAM algorithms are passive, meaning that the robot's movement is either manually controlled and directed toward pre-defined waypoints by a human operator or external agent. In these passive systems, the navigation or path planning components do not play an active role in deciding how the robot should move to improve its mapping or localization process. The robot follows a set path or user-defined objectives without any dynamic decision-making that could optimize exploration based on the current state of the map or the robot's pose.

In contrast, Active SLAM (A-SLAM) approaches aim to address this limitation by tackling the optimal exploration problem within an unknown environment. It introduces a navigation strategy that generates future goals or target positions for the robot, with the primary objective of reducing map and pose uncertainty. Rather than passively following a fixed trajectory, the robot in A-SLAM actively selects actions that contribute to better map coverage and more accurate localization. This involves continuously assessing the current state of the map and the robot's pose, then making decisions about where the robot should move next to improve both mapping and localization.

A-SLAM systems enable fully autonomous exploration and navigation, allowing the robot to intelligently map its environment without human intervention. This makes A-SLAM a more effective solution for autonomous robotic navigation, particularly in unknown or dynamic environments where flexibility and decision-making are crucial. In Active Collaborative SLAM (AC-SLAM) multiple robots interchange information to improve their localization estimation and map accuracy to achieve some high-level tasks such as exploration.

In this article, we extend the A-SLAM approach proposed in [4], to a multi-agent AC-SLAM framework and present a system for efficient environment exploration using frontiers detected over an Occupancy Grid (OG) map. In particular, in this work we aim to:

1. Incorporate a utility function that leverages frontier path entropy to compute the rewards of goal candidate frontiers.
2. Introduce an effective method for distributing robots within the environment to enhance exploration. This approach optimizes goal selection by minimizing frontiers based on reward, distance, and merged map information gain metrics.
3. Efficiently maximize environment exploration while maintaining a good SLAM estimates and provide a computationally inexpensive solution by reducing the number of goal frontiers.

We implemented the proposed method in ROS, leveraging its client-server modular architecture, and made our code publicly accessible[1]. Our approach is validated through both simulations and real robot experiments, demonstrating improved performances compared to state-of-the-art methods.

The article is organized as follows: Section 2 summarizes the related literature from selected articles. Section 3 presents a thorough explanation of our proposed system, with specific emphasis on the methodologies employed for frontier filtering and management, implementation of the utility function, and coordination among robots. We show the usefulness and application of the system in simulations and real robot experiments in Sections 4.1 and 4.2. Finally, in Section 5 we summarize and conclude this work. Throughout this article, we will use the words *robots* or *agents* interchangeably, and the same applies to *frontiers* and *points*, as they imply the same meaning in the context.

2 Related Work

As previously mentioned, A-SLAM is designed for situations in which a robot must navigate in an environment that is only partially observable or unknown. In this context, the robot must choose a sequence of future actions while dealing with noisy sensor measurements that impact its understanding of both its state and the map of the environment. This scenario is typically formalized as a specific case of the Partially Observable Markov Decision Process (POMDP), as presented in [5], [6], and [7].

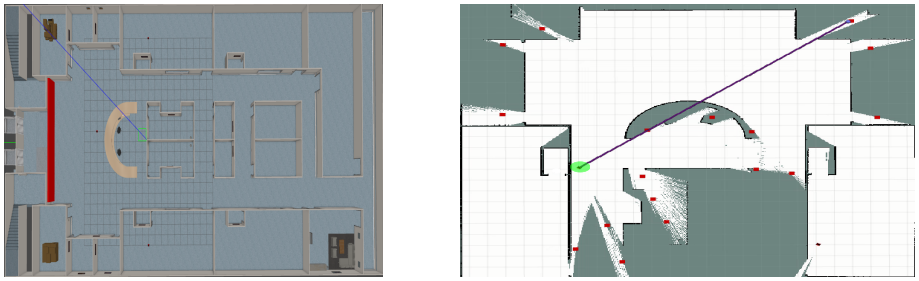
The POMDP formulation, while widely adopted, is computationally intensive due to its consideration of planning and decision-making under uncertainty. To streamline computation, A-SLAM is usually divided into three key steps: 1) identifying potential goal positions (frontiers, i.e., boundaries between visited and unexplored areas), 2) calculating their associated costs using a utility function where utility is computed using Information Theory (IT) [8] or Theory of Optimal Experimental Design (TOED) [9], hence selecting the next action to be performed, and 3) executing the action, eventually moving the robot to the chosen goal position.

Regarding the first step, a typical approach involves identifying potential exploration targets, such as frontiers. A frontier is the border between known and unknown map locations. Figure 1 illustrates frontier detection using lidar measurements within a simulated AWS Modified Hospital (HOS) environment[2].

[1] https://github.com/MF-Ahmed/ACSLAM_Karto
[2] https://github.com/aws-robotics/aws-robomaker-hospital-world

As the robot explores the environment, the uncertainties related to the map and robot pose increase over time, the goal is to reduce uncertainty in belief space [10]. The existing approaches propose solutions that make use of IT and TOED approaches to quantify this uncertainty. In IT, entropy measures the amount of uncertainty associated with a random variable or random quantity. Higher entropy leads to less information gain and vice versa. The authors in [11] formulate the Shannon entropy of the Map $\mathcal{M}$ as in Equation 2 where the map is represented as an OG and each cell $c_{i,j}$ is associated with a Bernoulli distribution $p(c_{i,j})$. The objective is to reduce the map entropy.

$$\mathcal{H}[p(\mathcal{M})] = -\sum_{i,j}(p(c_{i,j})log_2(p(c_{i,j})) + (1 - p(c_{i,j}))log_2(1 - p(c_{i,j})) \tag{2}$$



(a) Simulation environment (b) Computed OG map and frontiers detection

Fig. 1: (1a) AWS Modified Hospital environment. (1b) Frontier detection on the OG map, green = robot, red = detected frontiers (centroids), white = free space, gray = unknown map area, black = obstacles

Alternatively, TOED defines many optimally criteria defining the mapping of the covariance matrix Ω_i to a scalar value. Hence, the priority of a set of actions is based on the amount of covariance in the joint posterior. Less covariance contributes to a higher weight of the action set. Optimality criterion deals with the minimization of average variance, reducing the volume of the covariance ellipsoid (D-optimality), and the maximum Eigenvalue as described by [12]. After identifying the goal positions and the utility or cost associated to reach them, the next step is to execute the optimal action that will ultimately guide the robot to the goal position. The exchanged information can be localization information [13], entropy [11], visual features [14], [15], and frontier points [16].

In AC-SLAM multiple robots interchange information to improve their localization estimation and map accuracy to achieve some high-level tasks such as exploration. This collaboration raises some challenges regarding the usage of computational resources, communication resources, and the ability to recover from network failure. AC-SLAM parameters may include a) localization information [13], visual features [14], and frontier points [16], b) parameters relating to exploration and re-localization (to gather

at a predefined meeting position) of robots as described by [13], c) 3D Mapping info (OctoMap) used by authors in [16], d) path and map entropy as used in [17] and relative entropy, as mentioned in [18]. Typical application scenarios include collaborative localization [19][20], exploration and exploitation strategies [21] and trajectory planning [17][18].

The approach used by the authors in [15] presents a multi-layer approach where the first layer selects utility criterion based on Shannon's Entropy to goal locations (frontier points). While the second and third layers actively re-plan the path based on the updated OG map, non-linear Model Predictive Control (MPC) is applied for local path execution. In a similar approach presented by [17], a decentralized method for a long planning horizon of actions for exploration. The action is chosen that best minimizes the entropy change per distance traveled. At first, entropy in short horizons is computed using Square-Root Information Filter updates and that of the long horizon is computed considering a reduction in loop closures in robot paths. The main advantage of this approach is that it maintains good pose estimation and encourages loop closure trajectories.

The approach mentioned in [22] and [23] uses MPC to solve the area coverage and uncertainty reduction in A-SLAM. A control switching mechanism is formulated and SLAM uncertainty reduction is treated as a graph topology problem and planned as a constrained nonlinear least-squares problem. The area coverage task is solved via the sequential quadratic programming method and Linear SLAM is used for sub-map joining.

The authors in [24] present a centralized method in which a Deep Reinforcement Learning (DRL) [25] based task allocation is used to assist agents in a relative observation task. Each agent can choose to perform its independent ORB-SLAM [26] or help localize other agents. The reward function incorporates the influence of other agent's transition errors in decision making. The observation function is derived from ORB-SLAM and consists of map points, keyframes, and loop closure detection components. To compute the relative observation between agents, a nonlinear optimization problem is solved using the Gauss-Newton algorithm to estimate the pose of the target agent. The large associated computational cost of this method lacks real-time application. In a similar approach by [27], the authors present a Next Best View (NBV) planner that facilitates loop closure based on information gained from visual features. The authors in [14] propose frontiers-based coverage approaches that divide the perception task into an exploration layer and a detailed mapping layer, making use of heterogeneous robots to carry out the two tasks and solving a Fixed Start Open Traveling Salesman Problem (FSOTSP). In [16] the authors propose frontier-based viewpoints as a valid solution to build a volumetric model of the environment with multiple agents.

In a less computationally expensive approach, the method proposed by [28] uses a Convolutional Neural Network (CNN) for fusing the aerial and ground robots maps in a traversability mapping scenario. It formulates an active perception module that uses conditional entropy to guide the robots toward high entropy paths.

In an interesting approach, the method presented in [19] uses multiple humanoid robots, where each robot has two working modes, independent and collaborative. Each

robot has two threads running simultaneously: a) the motion thread and b) the listening thread. During the motion thread, it will navigate the environment via the trajectory computed by the organizer (central server) using a D* path planner and a control strategy based on DRL and a greedy algorithm. During the listening thread, it will receive its updated pose from the organizer and may receive the command to help other robots in the vicinity improving their localization using a chained localization method. In this method, each robot's localization is improved by its preceding robot, and its covariance is updated depending on the measurement error between the two robots.

Recently, a new method has been developed to measure uncertainty in pose graph SLAM by using graph connectivity measures. Graph connectivity indices are computationally less expensive to measure SLAM uncertainty as compared to TOED and IT approaches discussed previously. In [29], [30], and [31], the authors debate how the graphical topology of SLAM has an impact on estimation reliability. They establish a relationship between the Weighted number of Spanning Trees (WST) and the D-Optimality criterion and show that the graph Laplacian is closely related to the Fisher Information Matrix (FIM). The authors in [32] and [4] extend [31] by debating that the maximum number of Weighted Spanning Trees (WST) is directly related to the Maximum Likelihood (ML) estimate of the underlying graph SLAM problem. Instead of computing the D-optimality criterion defined over the entire slam sparse information matrix, it is computed over the weighted graph Laplacian, and it is proven that the maximum number of WST of this weighted graph Laplacian is directly related to the underlying pose graph uncertainty.

In [20], the authors utilize the graph connectivity indexes and propose a method for identifying weak connections in pose graphs to enhance information exchange when robots are in proximity. The proposed system identifies the weak connections in the target robot pose graph, and when the covariance increases to a certain threshold, other agents help to rectify these weak connections and generate trajectories using Rapidly Exploring Random Trees (RRT) to decrease uncertainty and improve localization. This method uses continuous refinement along with the D-optimality criterion to collaboratively plan trajectories. A bidding strategy is defined, which selects the winning host robot based on the least computational cost, feasible trajectory, and resource-friendly criteria.

The approaches discussed above exhibit several significant limitations that warrant attention. First, these methods often incur high computational costs associated with processing a large number of frontiers. As the number of frontiers increases, the computational burden can escalate, leading to higher processing times and reduced efficiency. Second, many of these approaches fail to effectively promote the distribution of robots throughout the environment. This lack of effective distribution can reduce exploration tasks. Furthermore, the uncertainty in these methods is typically quantified using a scalar mapping of the entire pose graph covariance matrix. This matrix can become quite large, especially in landmark-based SLAM methods, which further exacerbates the computational cost. The size of the covariance matrix can lead to challenges in real-time processing and may limit the scalability of these approaches in more complex environments.

In addition to these issues, existing AC-SLAM methods often do not explicitly incorporate strategies for the efficient management of frontiers. This oversight can slow down map discovery and impede robot localization, ultimately reducing the overall effectiveness of the SLAM process. Addressing these limitations could lead to significant improvements in the efficiency and effectiveness of AC-SLAM strategies.

3 Methodology

3.1 Overview of the proposed approach

Summarizing the AC-SLAM approaches discussed in Section 2, we observe that they quantify the uncertainty using the entire map entropy and by using the full covariance matrix which renders them computationally expensive. Further, they do not favor the distribution of agents for maximum coverage requirements. For these reasons, here we propose an AC-SLAM method that encourages sparsity between agents while utilizing a utility function that takes into account the uncertain propagation not only of the pose graph (D-optimality) but also of the map (frontier path). Using our approach we manage to distribute the agents while maintaining a good SLAM estimate. Additionally, when compared to the state-of-the-art methods described in Section 2, our method provides a computationally efficient solution by working on fewer frontiers, maximizing the exploration while using a utility function incorporating modern D-Optimality along with path entropy.

Figure 2 shows the architecture and communication pipeline of our proposed approach. We have built our method upon [4] which uses a lidar based SLAM back-end and proposes a utility function based on the modern D-Optimality criterion as a maximum number of spanning trees of the graph Laplacian of the pose graph. Each robot performs its SLAM using Open Karto [3] and detects local frontiers relative to its map. A map merging node [4] (map-merging-node) merges local maps into a global map, so that all the computed frontiers are referenced to the global map. Frontiers from each agent are concatenated into a list and further processed by the Filtering and Classification (*merge-points-server*) module (Section 3.2). This module removes redundant frontiers (already chosen goals from previous iteration) and further filters the frontiers/points by keeping only those points that are at (or near to) the border of the merged map (global map). This new list of points is sent back to each agent, which computes its utility and reward matrix for each point as we will see in the Utility Computation (*assigner* node) module (Section 3.3). This reward matrix is further processed by the Update Rewards & Goal Selection (*choose-goals-server*) module (Section 3.4) which updates the rewards keeping into account the sparsity and number of already selected goal points for each agent. Finally, the selected goal for each agent is sent to the Path planning & control module (ROS Navigation stack) which uses Dijkstra's algorithm [33] for global and Dynamic Window Approach (DWA) [34] as local planners.

[3] https://github.com/ros-perception/slam_karto.
[4] https://github.com/robo-friends/m-explore-ros2

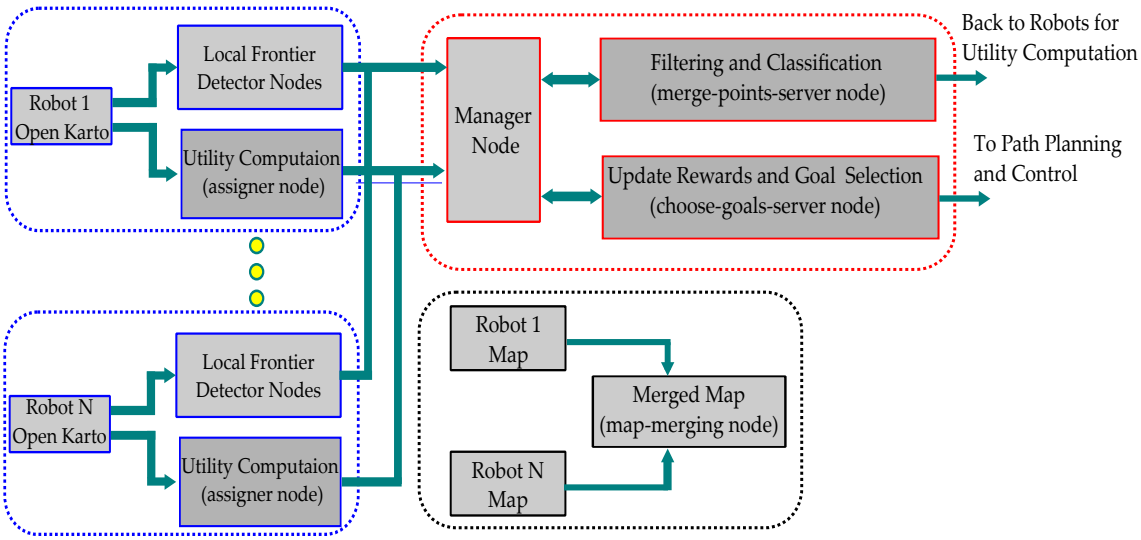


Fig. 2: Architecture of the proposed method. Local nodes of each robot (blue), ROS server (red), map-merging mode (black)

Since the approach has been implemented in ROS and involves a centralized frontier-sharing system as shown in Figure 2 each agent uses two main nodes responsible for utility computation and frontier detection: the *assigner* node computes the utility function and assigns the goal points to the agent, while the *detector* nodes use OpenCV and RRT-based frontier detection from [4]. The following nodes are a part of the *central server* of the system: 1) the *manager* node, acts as a communication gateway between robots and server, managing agent priority and frontier information, 2) the *merge-points-server* node, responsible for merging lists of points acquired by different agents, and 3) the *choose-goals-server* node, which chooses a specific target point from the list. By adopting a specific policy (Section 3.4), the server also aims to optimize robot distribution in the environment and minimize exploration time.

Regarding the following Sections and Algorithms, we summarize the workflow as:

1. For a set of agents $R = \{r_1, \ldots, r_M\} \subset \mathbb{R}^2$, each agent r_m detects a set of frontier points $p^m_{\text{list}} = \{p^m_1, ..., p^m_N\}$, where each point is defined as $p^m_n = (x^m_n, y^m_n) \in \mathbb{R}^2$, and transmits them to a central server (subject to its availability) on a dedicated topic.
2. The manager node takes the list of points p_{list} passed by each agent and sends it to *merge-points-server*.
3. The *merge-points-server* takes as input the lists received by all robots, merging the points into a unique list using Algorithm 1, also checking the actual frontiers on the merged map $M^{merged} = \{res, origx, origy, w, h\}$, corresponding its resolution, origin position along x, y axis, width and height respectively.
4. The *merge-points-server* through the Algorithm 2 and 3 limits the dimension of p_{list} And eventually, it gives back the list to each agent.

5. Each agent computes the reward matrix H based on the received list, using the approach described in Section 3.4, and sends it to the *choose-goals-server*.
6. The *choose-goals-server* server updates the reward through Algorithms 4 and 5 to take into account all the points already assigned. The selected target point is fed back to the robot.
7. The global and local planners of the ROS package *move_base* are responsible for driving each agent to the selected frontier. Once the agent reaches the target, the workflow restarts from step 1.

In the following Sections, we describe comprehensively steps 3, 4, and 5, which represent the core of the proposed approach.

3.2 Filtering and classification

Each agent is responsible for building its map and merging local frontier points as shown in Algorithm 1. Some parts of the map from each agent can be overlapped and the frontiers could lie in an already mapped area when considering the merged map. Since usually the goal of an exploration task is to cover the entire area by minimizing the exploration time, the frontiers lying in the middle of the merged map are not significant because moving an agent to them would not increase the overall discovered area.

To avoid the need to consider these points as *goal-like* points, we decided to filter the points considering only the actual frontiers of the merged map. To this purpose, Algorithm 1 takes a list of points p_{list} and the merged map M^{merged} and it checks if each point has enough unknown cells (`PER_UNK`) around it, in a radius (`RAD`) (line 3), i.e., if the point is near the border. If so, the point is added to the global list uni_{pts}. This process (line 3) of Algorithm 1 is detailed in Algorithm 2. In particular, it may be observed how the percentage `PER_UNK` of unknown cells (Line 18) contained in the radius `RAD` is used to decide whether to keep a point in the list or not (Line 19).

The Algorithm 2 starts by merging the point p from the lists passed to the server. To determine if p is a frontier in merged map M^{merged}, it first converts it into grid coordinates (c_x, c_y) of M^{merged} then for each cell in `RAD` it calculates the grid cell coordinates by adding its offsets to the grid coordinates of p (line 7). We take a radius as shown in Figure 3a, with length `RAD` around the point and check the percentage, with a value `PER_UNK`, of unknown cells in this radius. Taking this percentage in Line 18, the algorithm can filter out or keep this point in Line 19. If this percentage unk_{perc} exceeds a given threshold, the point is considered near the map's border, and the function returns "True". Otherwise, it returns "False".

To illustrate this approach, Figure 3b presents an example featuring two points, P and Q, on a partially explored map. Algorithm 1, in conjunction with Algorithm 2, identifies a circular region around each point, as shown in Figure 3a, and calculates the percentage of unknown cells within the total area of the circle. Based on this computed percentage, the point is either retained or discarded according to the `PER_UNK` threshold set during program execution. In this specific case, by appropriately configuring `PER_UNK`, point P is added to the global list as a border point, while point Q is discarded.

Algorithm 1 Merge Points

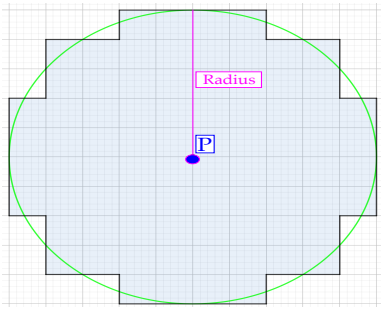
Require: p_{list}, M^{merged} ▷ $p_{list} = \{x_1, y_1, ...x_N, y_N\}$, merged_map = M^{merged}
Ensure:
$uni_{pts} \leftarrow 0$
for $p \in p_{list}$ **do**
  if $N_{bdr}(p, M^{merged}, \texttt{RAD}, \texttt{PER\_UNK})$ **then**
    if $p \notin uni_{pts}$ **then**
      $uni_{pts} \leftarrow p$ ▷ add p to unique list of points
    end if
  end if
end for

Algorithm 2 Check if a Point is Near the Map Border

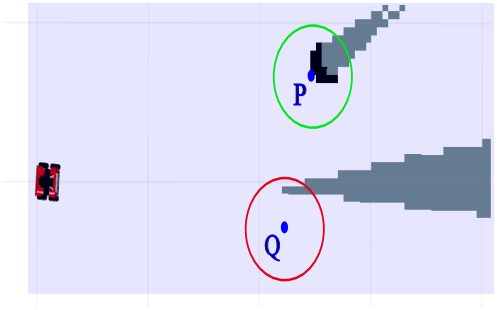
Require: $p, M^{merged}, \texttt{RAD}, \texttt{PER\_UNK}$:
Ensure: Return **True** if p is near the border
function $N_{bdr}(p, M^{merged}, \texttt{RAD}, \texttt{PER\_UNK})$
  $c_x \leftarrow (p_x - M^{merged}_{origx})/M^{merged}_{res}$
  $c_y \leftarrow (p_y - M^{merged}_{origy})/M^{merged}_{res}$
  $Rad_c \leftarrow \texttt{RAD}/M^{merged}_{res}$
  $Cir_c \leftarrow \{\}$, $unk_{cnt} \leftarrow 0$, $tot_{cells} \leftarrow 0$
  for $i, j \in Rad_c$ **do**
    $cel_i \leftarrow c_x + i$, $cel_j \leftarrow c_y + j$
    if $cel_i, cel_j \geq 0$ **&** $< M^{merged}_w, M^{merged}_h$ **then**
      $idx \leftarrow cel_i + cel_j \times M^{merged}_w$
      $Cir_c \leftarrow cel_i, cel_j$
      if $M^{merged}_{data}[idx] = -1$ **then**
        $unk_{cnt} \leftarrow unk_{cnt} + 1$
      end if
      $tot_{cells} \leftarrow tot_{cells} + 1$
    end if
  end for
  if $tot_{cells} \neq 0$ **then**
    $unk_{perc} \leftarrow \left(\frac{unk_{cnt}}{tot_{cells}}\right) \times 100$
    if $unk_{perc} \geq \texttt{PER\_UNK}$ **then**
      return True
    end if
  end if
  return False
end function

Algorithm 3 Check list dimension

Require: uni_{pts}, M^{merged}, RAD, PER_UNK
Ensure:
while $uni_{pts} \leq \texttt{MIN\_PTS}$ **or** $\geq \texttt{MAX\_PTS}$ **do**
  if $uni_{pts} \leq \texttt{MIN\_PTS}$ **then**
    $uni_{ptsN} \leftarrow L$,$uni_{pts} \leftarrow \{\}$
    $perc = perc - 10$
    for $p \in uni_{ptsN}$ **do**
      if $N_{bdr}(p, M, \texttt{RAD}, \texttt{perc})$ **then**
        if $p \notin uni_{pts}$ **then**
          $uni_{pts} \leftarrow uni_{pts} \cup \{p\}$
        end if
      end if
    end for
  else if $uni_{pts} \geq \texttt{MAX\_PTS}$ **then**
    $uni_{ptsN} \leftarrow uni_{pts}$
    $uni_{pts} \leftarrow \{\}$
    $rad = rad + 0.25$
    for $p \in uni_{ptsN}$ **do**
      if $N_{bdr}(p, M, rad, \texttt{PER\_UNK})$ **then**
        if $p \notin uni_{pts}$ **then**
          $uni_{pts} \leftarrow uni_{pts} \cup \{p\}$
        end if
      end if
    end for
  end if
end while



(a) Discretized circle around a point P.



(b) Discretized circle around points P & Q.

Fig. 3: General 3a and OG Map Representation 3b of the Discretized circle

Even discarding points that are not on the border, having many agents may lead to having extensive lists of points that need to be processed; to avoid this problem, we decided to bind the number of points to process using Algorithm 3. After the list uni_{pts} is created, there is another check to validate the boundaries of the list dimension as described in Algorithm 3. If the list has fewer points than the minimum required (i.e., `MIN_PTS`), the same is recomputed by decreasing the threshold by 10% as shown in Line 3, and 4 (where L represents the length of uni_{pts}). Conversely, if the list has more points than a maximum threshold (line 12) `MAX_PTS`, it is reprocessed by increasing the radius `RAD` by 0.25m (Line 15 of Algorithm 3).

3.3 Utility computation

Once the global list of frontier points is created, it is sent to the *assigner* node of each agent for the computation of the utility function for each frontier candidate. We incorporate the utility function from our previous work [35] to this multi-agent system and will briefly describe it here. It takes into account the amount of uncertainty in the map measured as path entropy and Euclidean distance to each frontier candidate. For each robot r_m, the path entropy E_m^n for each frontier candidate p_n is computed using Equation 3 where G^n represents the set of grid cells in the path of the frontier candidate.

$$E_m^n = E_m^n[p(c)]_{c \in G^n} = -\sum_{c \in G^n} (p(c_{i,j}) \log_2(p(c_{i,j})) + p(1 - c_{i,j}) \log_2(1 - p(c_{i,j})), \forall c_{i,j} \in \mathcal{M}) \quad (3)$$

We assign probability values to the OG map cells which favor exploration of unknown areas of the environment. Hence, we assign high information gain to obstacles and free space, as we are not interested in places already known to the robot.

The path entropy is then normalized with the number of pixels/cells within the frontier path L. To penalize frontiers that are further away, we apply an exponential decay operator γ_m^n as shown in Equation 4, where $d(r_m, p_n)$ is the Euclidean distance between the agent and a frontier candidate.

$$\gamma_m^n = \exp(-\lambda \cdot d(r_m, p_n)) \tag{4}$$

Finally, the proposed utility $U_{2,m}^n$ as shown in Equation 5 is computed by weighing normalized entropy E^n with ρ_m^n, which depends on the number of spanning trees of the weighted graph Laplacian L_w. Equation 6 represents the modern D-Optimality criterion adopted from [4], which proposes a utility function to quantify the best frontier candidates that have a maximum number of spanning trees in the weighted pose graph Laplacian $L_{w,m}$ towards them. Eventually, we obtain the proposed utility function U_m^n in Equation 7, which not only provides a good SLAM estimate but also increases the coverage of the unknown map by reducing the frontier path entropy.

$$U_{2,m}^n = (1 - E^n/L^n) * \rho_m^n + \gamma_m^n \tag{5}$$

$$U_{1,m}^n = \text{Spann}(L_{w,m}) \tag{6}$$

$$U_m^n = \max(U_{1,m}^n + U_{2,m}^n) \tag{7}$$

$$H_m = [p_{list}^m, U_m^n] \tag{8}$$

The reward matrix H_m for each agent is computed as shown in Equation 8. Where $U_m^n \subset \mathbb{R}$ is the associated reward for each frontier and is computed using Equation 7.

3.4 Update Rewards and Goal Selection

Once each agent has computed its reward matrix, it is sent to the *choose-goals-server* (hereafter referred to as *server*) to opportunely update the rewards and select the goal point of each agent. Since the server can manage one reward matrix at a time, it is asynchronously handled by the various agents, with an ascending priority: assuming to have a system of $M > 1$ agents, taking two general agents r_1 and r_2, which request the server at the same time, the goal of r_1 is processed before r_2 if $r_1 < r_2$. Agents access the server in priority order, with the server managing one reward matrix at a time.

Moreover, the server also stores the already assigned locations, to avoid reusing them. The server has to manage M different matrices, one for each agent. Indeed, to explore the biggest possible area of the environment, the goals for the agents need to be spread. To foster this sparsity, since there are many agents, we decided to update the reward function using Equations 9-11, where d is the Euclidean distance between the highest reward point and all other points.

$$K = \frac{\texttt{max reward in matrix}}{\texttt{number of already chosen points}} \tag{9}$$

$$k = \frac{K}{d^2} \tag{10}$$

$$R_{new} = R_{old} - k \tag{11}$$

K in Equation 9 is motivating by two reasons:

- `max reward in matrix` allows for scaling K with respect to the reward matrix of each agent.
- `number of already chosen points` allows for distributing the reward update also taking into account the number of already selected points. When the number of targets already explored becomes significant, each point will only receive a smaller portion of the total reward.

It may be observed (Equation 11) how k represents a subtracting factor for the reward matrix elements, updated when a target goal for one agent is selected. Since k is inversely dependent on the distance computed between the last chosen goal and the considered frontier point (Equation 10), the closer the point is to the already chosen goal, the higher k will be, decreasing the probability that the point will be chosen as the next goal, thus achieving the task of spreading the goals into the environment. The server then updates the reward for specific points using a negative additive factor k, as in Equation 11.

The complete procedure for the selection of points (goals) is described in Algorithm 4, also including the relative function to update rewards when a goal is selected, described in Algorithm 5.

Algorithm 4 Select Points

function $sel_{pts}(uni_{pts}, uni_{ptsU})$
  $goals \leftarrow \{\}$
  $H \leftarrow$ using uni_{pts}, uni_{ptsU}
  if $H \neq \{\}$ **then**
    $p \leftarrow$ **Point2D**() ▷ **Point2D** is ROS message type
    if $cho_{cords} \neq \{\}$ **then**
      $H \leftarrow upd\_rewards(cho_{cords}, H)$ ▷ Algorithm 5
    end if
    $H_{MaxID} \leftarrow$ get maximum reward index in H
    $p.x, p.y \leftarrow$ x,y value in H at H_{MaxID}
    for $pt \in cho_{cords}$ **do**
      if $pt.x, pt.y = p.x, p.y$ **then** ▷ if already chosen
        $H[H_{MaxID}, 0] \leftarrow -\infty$
        $H_{MaxID} \leftarrow$ get maximum reward index in H
        $p.x, p.y \leftarrow$ x,y value at H_{MaxID}
      end if
    end for
    $cho_{cords} \leftarrow p$
    $goals \leftarrow p$
  end if
  return $goals$
end function

Algorithm 5 Update Rewards

1: **function** $upd\_rewards(cho_{cords}, H)$
2: $H_{MaxVal} \leftarrow$ max. reward in H
3: $K \leftarrow$ Equation 9
4: **for** $c \in cho_{cords}$ & $g \in H$ **do**
5: **if** $c_{x,y} = g_{x,y}$ **then** ▷ If goal already chosen then select reward to -infinity
6: $H[g] = -\infty$
7: **end if**
8: **if** $H[g] \neq -\infty$ **then** ▷ compute distance from chosen goals
9: $d2 \leftarrow \sqrt{(c_x - g_x)^2 + (c_y - g_y)^2}$
10: **if** $d2 \neq 0$ **then**
11: $H[g] \leftarrow$ Equation 11
12: **else**
13: $H[g] = -\infty$
14: **end if**
15: **end if**
16: **end for**
17: **return** H
18: **end function**

Algorithm 4 takes as input a list of points uni_{pts} along with their rewards uni_{ptsU} (computed with Equation 5), and updates the H matrix using Algorithm 5 (line 7). It checks whether the frontier corresponds with the already selected goal, discarding the point in this case, and then updates the reward as described in Equations 9-11. The goal corresponding to the higher reward for the current agent, updated with the described policies, is eventually passed back to the agent.

Algorithm 5 takes as input the chosen frontier points cho_{cords} which are the previous assigned goal points from the previous iteration and H matrix and starts by finds the maximum reward in H and calculating a parameter K. It then iterates over each chosen frontier coordinate and every goal g in H. If a goal's coordinates match a chosen coordinate, its reward is set to $-\infty$, effectively removing it from consideration. For other goals, it calculates the Euclidean distance to the chosen coordinate and updates the reward using Equation 11 if the distance is non-zero. Finally, the updated reward matrix H returned.

As described before, agents are managed asynchronously with a priority approach. The priority assigned to robots can lead to having one or more robots with low priority being stuck because they are always prioritized by higher-priority agents. To avoid this issue, the server also considers the number of requests unrelated to each agent. Once this number exceeds a certain predefined threshold `GOAL_SKIP_WAIT`, the corresponding agent will be associated with the higher priority. This approach prevents robots from getting stuck and distributes goals more uniformly.

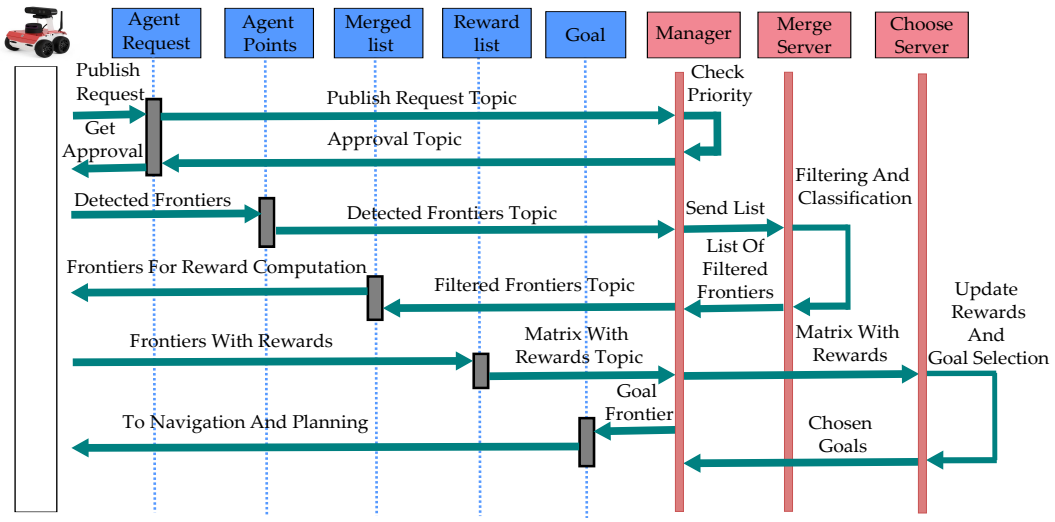


Fig. 4: Sequence diagram. Local nodes of each robot (blue) and server (red).

3.5 Program Execution Flow

In this section, we provide a comprehensive explanation of the detailed execution flow involved in the goal frontier assignment process, a critical aspect of our approach. This process is represented in Figure 4, which offers a clear depiction of the various steps and decision points within the workflow. It is essential to emphasize that the methodology described here is designed to function without any explicit coordination or communication between the agents. In this context, each agent operates fully autonomously, making decisions independently based solely on its local information and reward computations. This lack of coordination implies that the agents do not share information or collaborate in the goal assignment process, thereby underscoring the decentralized nature of the approach. As a result, the decision-making process for each agent is entirely self-contained, with no reliance on the actions or states of the other agents within the system. The sequence of actions unfolds as follows:

1. *Publish Request*: Initially, the robot sends a request to the server by publishing a message on a dedicated topic. This publication process is uniform for all agents in the system, allowing the server to prioritize robot assignments effectively.
2. *Approval and Execution*: Once the agent receives approval from the server, it proceeds with task execution. It gathers all the points it has detected and publishes them on a dedicated topic. The server, through subscription, collects the lists of points recently published by all agents. Subsequently, it employs Algorithm 1 to eliminate duplicate points from the lists and Algorithm 2 to distinguish frontier points from others. The resultant list, once compiled, is then published on a designated topic. This list typically contains a set of points, constrained by parameters such as `MIN_PTS` and `MAX_PTS`.

3. *Reward Computation*: With the obtained list, the agent proceeds to calculate the reward according to the utility function in Equation 7.
4. *Server-side Reward Update*: Upon receiving the reward matrix from an agent, the server updates its records using Algorithm 5. This process ensures that the server maintains an up-to-date assessment of the associated rewards.
5. *Goal Selection*: Finally, the server selects the goal that offers the highest reward from among the updated rewards as described in Algorithm 5. This chosen goal becomes the target that the robot is assigned to reach and explore.

This entire procedure, encompassing steps 1 through 5, is executed each time an agent reaches its previous goal.

4 Experimental Evaluation

4.1 Simulation Environment

To evaluate the proposed approach, simulations were performed with the simulation environment Gazebo on a PC with an Intel Core i7® (32GB RAM) and NVIDIA RTX 1000 GPU, equipped with Ubuntu 20.04 and ROS Noetic. The OG maps have a resolution of $0.1m$/cell. Practically, a team of RosBots 2, equipped with lidar sensors were deployed in a modified version of the Willow Garage (W.G)[5] environment and that of HOS (from Section 2), with an area of 2071 m^2 and 1243 m^2, respectively.

We compared our proposed approach against 1) Frontier Detection based Exploration (Frontier) [36] which uses a greedy frontier exploration strategy without any SLAM uncertainty quantification. 2) The method of [4] by converting it into a multi-robot system namely MAGS.

Since our proposed approach aims at environment exploration while working on minimum frontier points with efficient AC-SLAM, we decided to use the following performance metrics:

- *percentage of map coverage*, to quantify the evolution of the covered map concerning the ground truth map.
- *number of frontier points*, to measure the average points reduction (corresponding to a decreased computational cost) achieved with the method described in Section 3.2.
- *Map quality*, we compared metrics measuring Structural Similarity Index Measurement (SSIM) $\in [0, 1]$, Root Mean Square Error (RMSE), and Alignment Error (AE) with reference to ground truth maps.

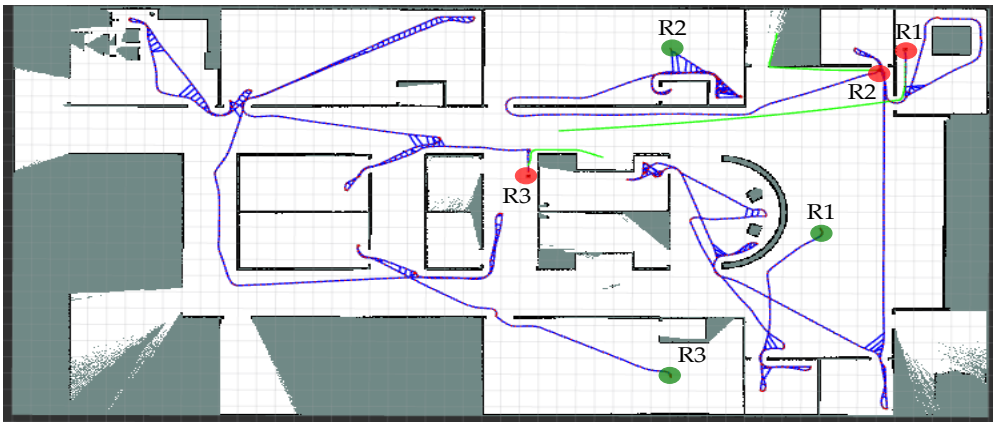
We conducted 10 simulations of 20 minutes each for both W.G and HOS using Frontier, MAGS, and our method resulting in a total simulation time of 10 hours. `PER_UNK`, `RAD`, `MIN_PTS`, `MAX_PTS` and `GOAL_SKIP_WAIT` were initialized to 60 %, $1m$, 0, 10 and 5 respectively.

Figures 5, 6, and 7 show the resulting OG maps and pose graphs generated using Open Karto SLAM, including loop closures, for the AWS Hospital environment. These maps correspond to our approach, MAGS, and Frontier, respectively. The Figures also

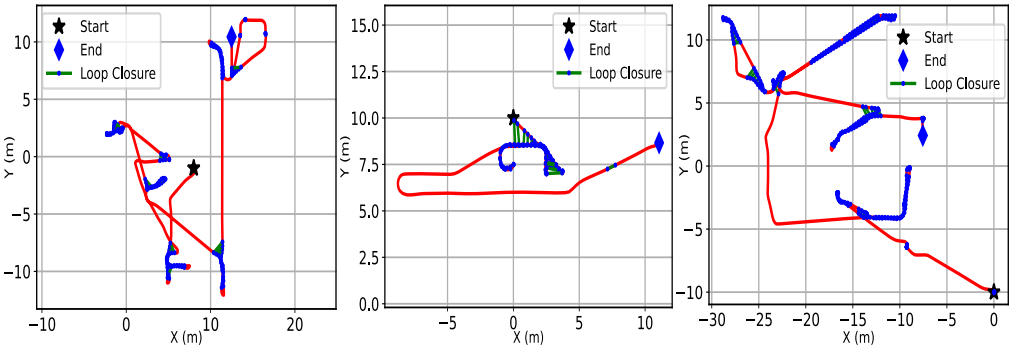
[5] https://github.com/arpg/Gazebo/

highlight the starting positions (in green) and final positions (in red) of the three agents after 30 minutes of exploration.

From Figure 5, we observe that our method achieves accurate mapping with 80% coverage while maintaining good SLAM accuracy, as the agents actively seek loop closures. In Figure 6, we also observe good localization and mapping accuracy, though the coverage is lower at 60%, and fewer loop closures are performed. Finally, in Figure 7, the agents explore 70% of the area—more than MAGS (Figure 6)—but with reduced SLAM accuracy. This results in poor localization and mapping performance, as illustrated by the dotted rectangle where agents R1 and R2 fail to accurately map the environment due to limited localization accuracy.

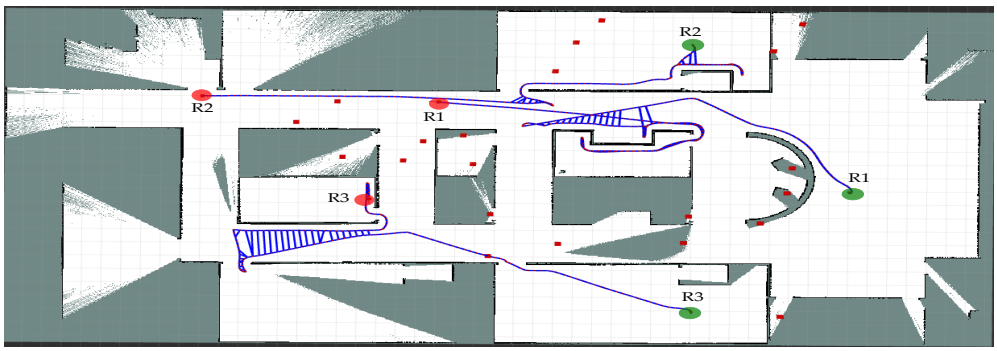


(a) HOS environment OG map and pose graphs.

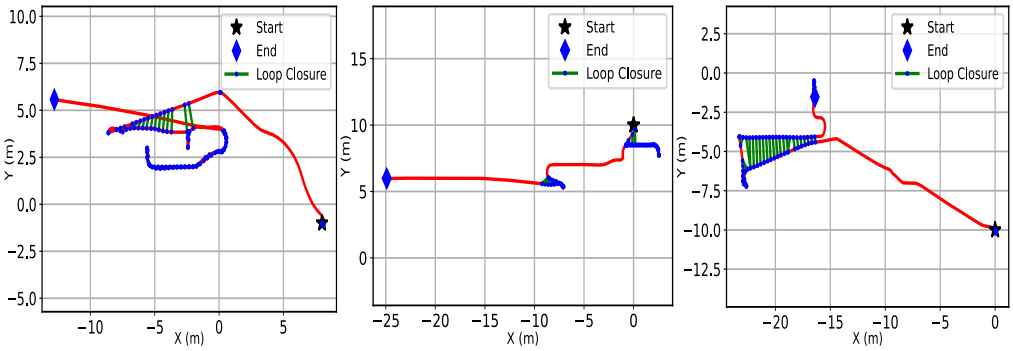


(a) Robot 1 (R1) (b) Robot 2 (R2) (c) Robot 3 (R3)

Fig. 5: Top: Final OG map using our method. Bottom: Individual pose graphs.

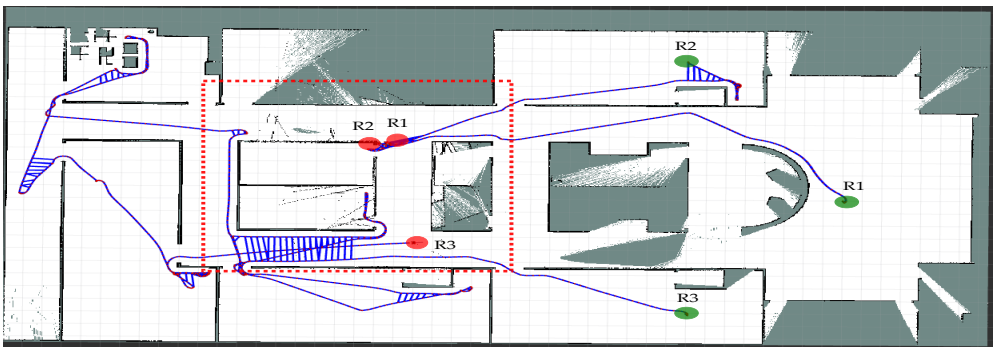


(a) HOS environment OG map and pose graphs.

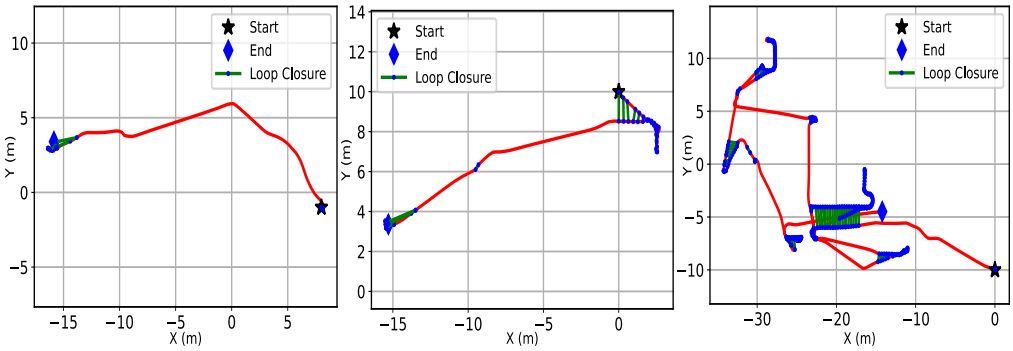


(a) R1 (b) R2 (c) R3

Fig. 6: Top: Final OG map using MAGS method. Bottom: Individual pose graphs.



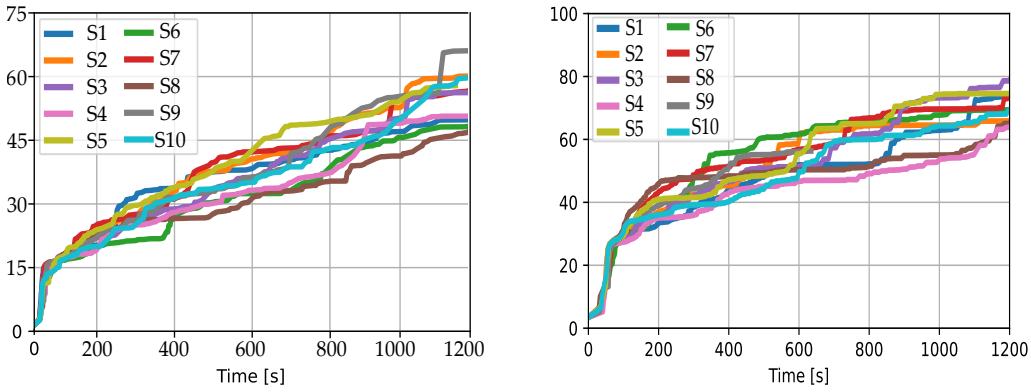
(a) HOS environment OG map and pose graphs.



(a) R1 (b) R2 (c) R3

Fig. 7: Top: Final OG map using Frontier method. Bottom: Individual pose graphs.

Figure 8a shows the evolution of the percentage [%] of map area discovered in W.G and HOS environments in 10 simulations using our approach and 3 robots. In both environments, the agents initially discover 15% and 20% area and eventually manage to cover 48% ($994m^2$) and 70% ($870m^2$) area respectively.



(a) W.G. (b) HOS.

Fig. 8: Evolution of % area coverage

Figures 9 and 10 show the average percentage of the map explored by three robots in the W.G. and HOS environments, respectively, using our approach (blue), MAGS (red), and Frontier (green). In Figure 9, our approach explores an area of $1040m^2$, which is 43% and 16% more than the areas covered by MAGS and Frontier, respectively. Similarly, in Figure 10, our method covers $870m^2$, representing 40% and 12% more area than MAGS and Frontier, respectively.

Additionally, we observe that in both environments, the Frontier method outperforms MAGS in terms of exploration area. This is because Frontier follows a greedy exploration strategy without incorporating uncertainty quantification, leading to more extensive exploration but at the expense of SLAM map quality and localization. Figure 11 plots the average percentage of maps discovered using two and three agents (subscript XR denotes the number of robots). It can be deduced that our approach outperforms other approaches considerably resulting in more average area coverage in both environments.

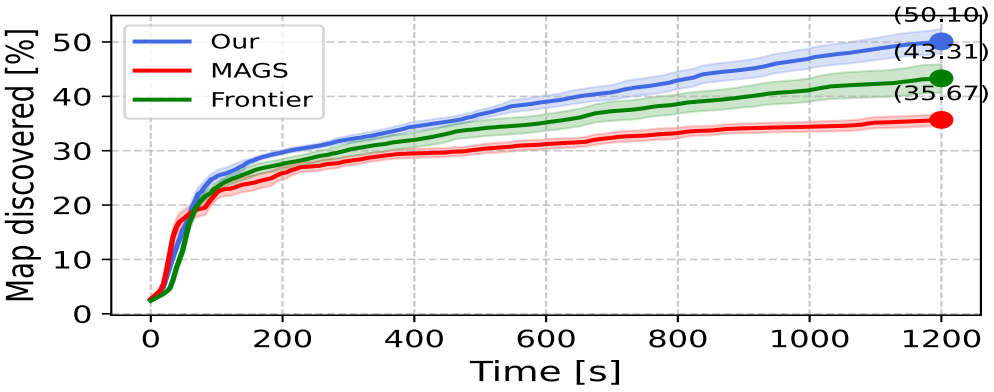


Fig. 9: The average percentage along with the standard deviation of the explored area in the W.G environment.

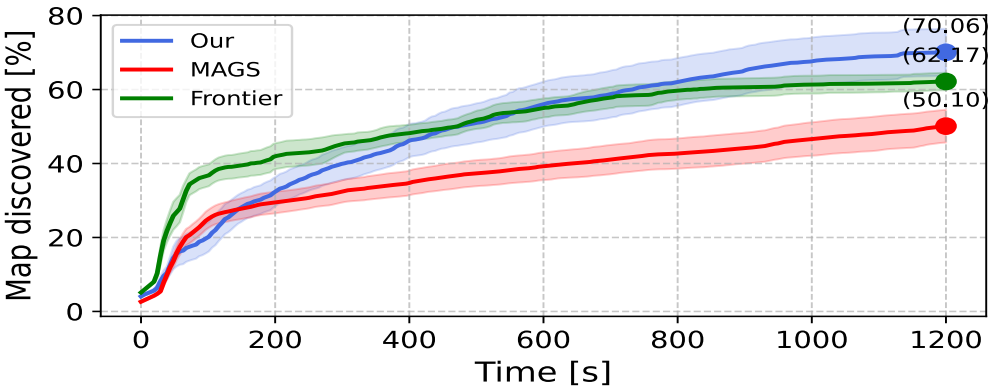
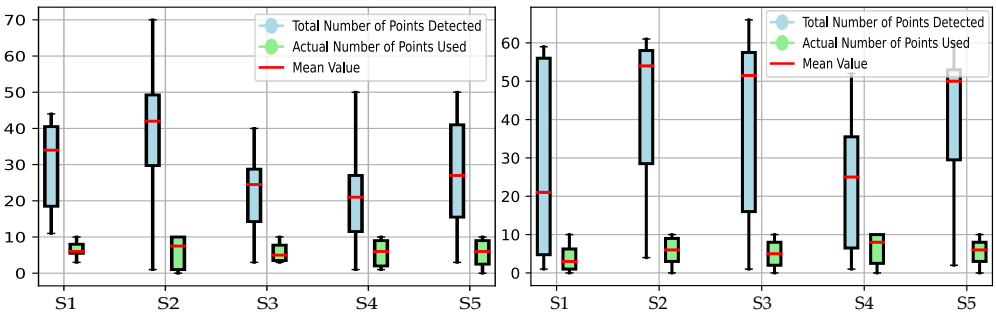


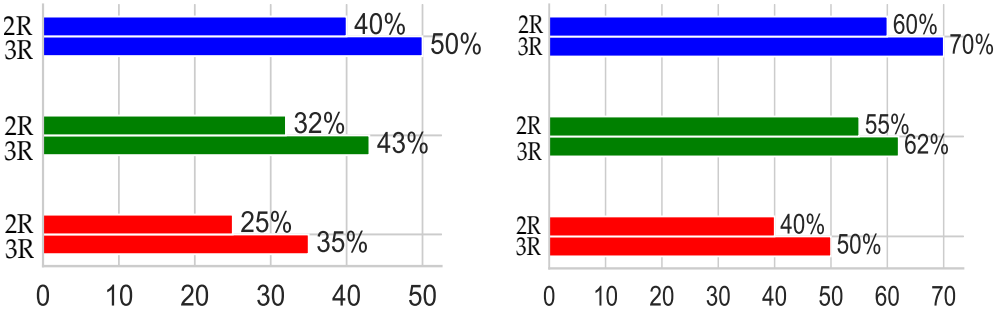
Fig. 10: The average percentage along with the standard deviation of the explored area in the HOS environment.

Regarding computational complexity and reducing the number of processed frontier points, Figure 12a provides an insight into how the number of points used is reduced in the W.G environment. We can deduce that using the methods in Section 3.2 and setting `PER_UNK` = 60%, `RAD`= $1m$ we manage to drastically reduce the average number of points processed from 35 to 8 for S1 (77%), 41 to 9 for S2 (78%), 27 to 7 for S3 (74%), 20 to 8 for S4 (60%) and 28 to 8 for S5 (71%) respectively, also exploring the environment more efficiently. This reduction in points is directly related to the computational complexity, as fewer points require computing the utility function with lesser frequency, hence accelerating the overall performance of the proposed system.



(a) W.G. (b) HOS.

Fig. 12: Total points detected and actual points used in both environments.



(a) W.G. (b) HOS.

Fig. 11: Average percentage of explored map in both simulation environments.

The same comparison has been carried out in the smaller HOS environment, in Figure 12b the usage of the *Filtering and Classification* method leads to a meaningful reduction in the number of points processed from 21 to 4 for S1 (80%), 55 to 6 for S2 (89%), 51 to 5 for S3 (90%), 25 to 9 for S4 (64%) and 50 to 7 for S5 (86%) respectively. The average values are higher compared to those in Figure 12a since also the environment has more obstacles than W.G.

Table 1 presents the usage of `PER_UNK`, which represents the percentage of unknown cells considered within a given radius when computing the information gain of a frontier candidate. The table also includes `RAD`, showing how the radius values change when the list of points is recomputed using three robots with our approach.

We observe that in W.G, `PER_UNK` decreases to $\leq 40\%$ (from an initial threshold of 60%), indicating the re-computation of the list. This occurs because W.G has fewer obstacles than HOS, leading to increased computational effort to maximize uni_{pts}, as explained in Algorithm 3.

In contrast, for HOS, `PER_UNK` remains at $\leq$ 60%, suggesting fewer list recomputations on the server, thereby reducing computational cost. Additionally, in both environments, `RAD` remains predominantly at $1m$, indicating that the list size

remains $\leq$ MAX_PTS, which reflects lower computational effort required by Algorithm 3 to reduce the number of points.

Env.	PER_UNK [%]			RAD [m]		
	60	50	≤ 40	1.00	1.25	≤ 1.50
W.G	34.5	1.4	**64.0**	**87.0**	1.8	9.7
HOS	**67.3**	4.3	28.2	**76.2**	5.1	8.5

Table 1: Impact of PER_UNK threshold and RAD values on frontier list computation in both environments.

Env.	Method	SSIM	RMSE	AE
W.G	Our	0.80	4.53	21.68
	MAGS	**0.86**	6.34	28.39
	Frontier	*0.20*	**10.04**	**40.89**
HOS	Our	**0.78**	4.02	21.39
	MAGS	0.72	6.39	29.98
	Frontier	0.35	**12.67**	**42.89**

Table 2: Comparison of map quality metrics (SSIM, RMSE, AE) for both environments.

Regarding the visual analysis of the maps and the evaluation of map quality metrics, the results, on average, appear promising, as shown in Table 2 using three robots. In almost all cases, our method yielded lower RMSE and AE while achieving higher SSIM compared to the MAGS and Frontier methods. Furthermore, we observe that the Frontier method, in comparison to MAGS, explores the environment more extensively, as illustrated in Figure 9, but at the cost of higher RMSE and AE, as well as lower SSIM.

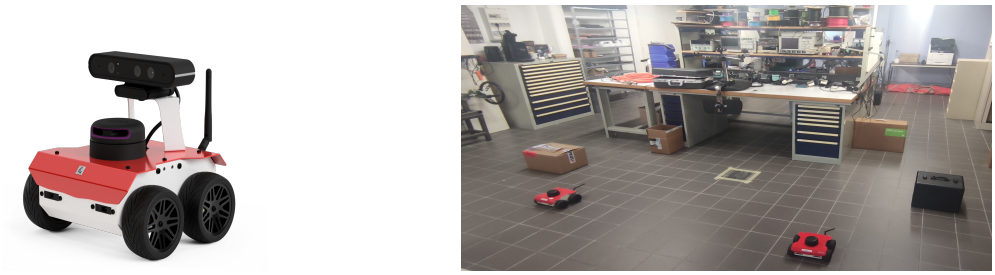
The aforementioned simulation results highlight the efficiency of our approach compared to state-of-the-art methods. To further validate our methodology, we conducted experiments with a team of ground robots in a real-world environment. The results of these experiments are presented in the following Section.

4.2 Real Environment

Experiments in a real environment are performed using two ROSBot 2R robots[6] with RPLidar A2 (Figure 13a) with ROS on Ubuntu 20.04.6 (LTS). The robots are equipped with an Intel Xeon® W-2235 CPU (3.80GHz x 12), with 64Gb RAM and Nvidia Quadro RTX 4000 GPU. The environment consists of a room and two corridors measuring $81m^2$ in total as shown in Figure 13b. Figure 14 presents the computed OG map

[6] https://husarion.com/manuals/rosbot/

and SLAM pose graph, illustrating the start (green) and final (red) positions of both robots across four experiments using the MAGS method and our proposed approach.

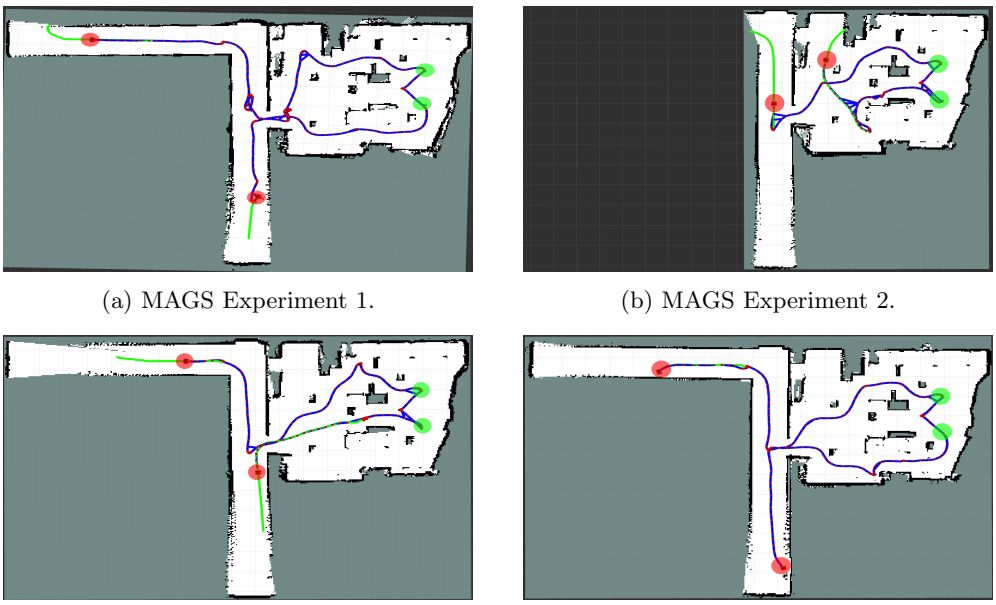


(a) ROSbot 2R.

(b) Experimental environment.

Fig. 13: Robot and experimental environment used.

In the MAGS method (Figures 14a and 14b), the robots frequently revisit previously explored areas, leading to excessive loop closures. This behavior reduces exploration efficiency and limits overall environment coverage. In contrast, our approach (Figures 14c and 14d) minimizes loop closures, allowing the robots to focus on unexplored areas and achieve broader coverage. This further demonstrates that while both methods produce accurate maps, the proposed approach enhances SLAM performance by prioritizing efficient exploration.



(a) MAGS Experiment 1.

(b) MAGS Experiment 2.

(c) Our method Experiment 1.

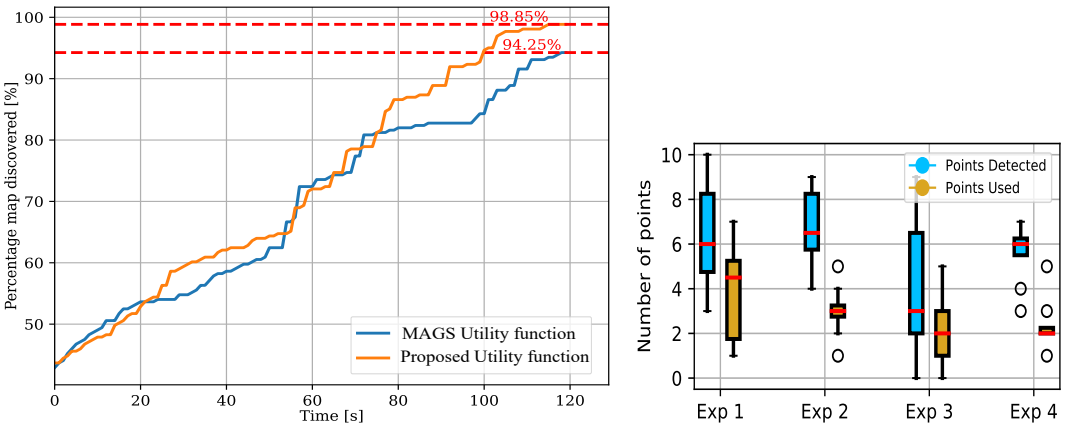
(d) Our method Experiment 2.

Fig. 14: Comparison of explored area in MAGS and our methods using two robots.

Figure 15a shows the percentage of maps discovered over time. As for simulation results, we compared the proposed utility function with the 'MAGS' in four experiments, considering an exploration time of 2 minutes. It can be shown that the proposed approach achieves a coverage of 98.85%, higher than the coverage percentage achieved with MAGS (94.25%). This implies a higher portion of map covered with the proposed approach (4.6%) and hence proves the effectiveness of the proposed method.

Figure 15b compares the number of points processed and detected across the four experiments. A significant reduction in the number of used points is observed: from 6 to 5 in Exp 1 (17%), from 7 to 3 in Exp 2 (58%), from 3 to 2 in Exp 3 (34%), and from 6 to 2 in Exp 4 (67%), leading to a notable decrease in computational load.

Due to the smaller size of the experimental environment, the exploration time was limited to just 2 minutes. Nevertheless, the increase in map coverage (4.6%) was comparable to that achieved in the simulation (3%–11%). Additionally, the maximum number of detected and processed points was lower than in the simulation, with an average reduction of 44% in the number of frontier points used.



(a) Comparison of mean values.

(b) Points filtering.

Fig. 15: (15a) Comparison of mean values of map discovered with and without the proposed utility function. (15b) Total points detected vs the actual points used.

5 Conclusion

In this article, we presented a multi-robot collaborative active SLAM framework for efficient environment exploration. The proposed framework implements a utility function that incorporates the SLAM uncertainty and path entropy for the efficient selection of goal frontier candidates. We also proposed an efficient frontier filtering method that encourages sparsity while working on a reduced number of frontier candidates, hence providing a less computationally expensive solution. The overarching goal of our proposed system is to minimize map entropy, reduce SLAM uncertainty, and maximize the efficiency and coverage of the exploration process. By doing so, our system ensures that the robots generate accurate maps with low SLAM uncertainty while exploring unknown environments in the most efficient manner possible. The implementation of our framework leverages a ROS based client-server paradigm, which is integrated into a modular software architecture. This modularity facilitates ease of development, testing, and deployment, allowing each module to be independently developed and seamlessly integrated into the overall system. A thorough and detailed explanation of each module, along with the associated algorithms, is presented and discussed.

Extensive simulations on publicly available environments, along with real-world experiments, demonstrate the effectiveness of our approach. Compared to state-of-the-art methods, our framework achieves superior exploration efficiency, higher mapping accuracy, and a substantial reduction in computational costs. The results show that our method accelerates exploration while maintaining high-quality maps with reduced SLAM uncertainty, confirming its robustness and applicability in real-world scenarios.

Looking ahead, we plan to extend our approach to encompass visual active collaborative SLAM using heterogeneous robots. This extension will allow us to exploit visual features and leverage the unique perspectives provided by different types of robots, thereby further enhancing the robustness and applicability of our framework in diverse and dynamic environments.

Acknowledgement

This work was carried out in the framework of the NExT Senior Talent Chair DeepCoSLAM, which was funded by the French Government, through the program Investments for the Future managed by the National Agency for Research ANR-16-IDEX-0007, and with the support of Région Pays de la Loire and Nantes Métropole. This research was also supported by the DIONISO project (progetto SCN_00320-INVITALIA), which is funded by the Italian Government.

References

[1] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., Leonard, J.J.: Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. IEEE Transactions on Robotics **32**(6), 1309–1332 (2016) https://doi.org/10.1109/TRO.2016.2624754 . Accessed 2023-05-31

[2] Saeedi, S., Paull, L., Trentini, M., Li, H.: Multiple robot simultaneous localization and mapping. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 853–858. IEEE, San Francisco, CA (2011). https://doi.org/10.1109/IROS.2011.6094709

[3] Zamora, E., Yu, W.: Recent advances on simultaneous localization and mapping for mobile robots. IETE Technical Review **30**(6), 490 (2013) https://doi.org/10.4103/0256-4602.125671

[4] Placed, J.A., Castellanos, J.A.: Fast autonomous robotic exploration using the underlying graph structure. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 6672–6679 (2021). https://doi.org/10.1109/IROS51168.2021.9636148

[5] Fox, D., Burgard, W., Thrun, S.: Active markov localization for mobile robots. Robotics and Autonomous Systems **25**(3), 195–207 (1998) https://doi.org/10.1016/S0921-8890(98)00049-9 . Autonomous Mobile Robots

[6] Placed, J.A., Castellanos, J.A.: A deep reinforcement learning approach for active slam. Applied Sciences **10**(23) (2020) https://doi.org/10.3390/app10238386

[7] Placed, J.A., Strader, J., Carrillo, H., Atanasov, N., Indelman, V., Carlone, L., Castellanos, J.A.: A survey on active simultaneous localization and mapping:

State of the art and new frontiers. IEEE Transactions on Robotics **39**(3), 1686–1705 (2023) https://doi.org/10.1109/TRO.2023.3248510

[8] Shannon, C.E.: A mathematical theory of communication. The Bell System Technical Journal **27**(3), 379–423 (1948)

[9] Fundamentals of Experimental Design, pp. 43–76. Springer, Berlin, Heidelberg (2007)

[10] Indelman, V.: Towards multi-robot active collaborative state estimation via belief space planning. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4620–4626 (2015). https://doi.org/10.1109/IROS.2015.7354035

[11] Stachniss, C., Grisetti, G., Burgard, W.: Information gain-based exploration using rao-blackwellized particle filters. In: Robotics: Science and Systems I, pp. 65–72 (2005). https://doi.org/10.15607/RSS.2005.I.009

[12] Carrillo, H., Reid, I., Castellanos, J.A.: On the comparison of uncertainty criteria for active slam. In: 2012 IEEE International Conference on Robotics and Automation, pp. 2080–2087 (2012). https://doi.org/10.1109/ICRA.2012.6224890

[13] Pham, V.-C., Juang, J.-C.: An improved active slam algorithm for multi-robot exploration. In: SICE Annual Conference 2011, pp. 1660–1665 (2011)

[14] Qin, H., Meng, Z., Meng, W., Chen, X., Sun, H., Lin, F., Ang, M.H.: Autonomous exploration and mapping system using heterogeneous uavs and ugvs in gps-denied environments. IEEE Transactions on Vehicular Technology **68**(2), 1339–1350 (2019) https://doi.org/10.1109/TVT.2018.2890416

[15] Bonetto, E., Goldschmid, P., Pabst, M., Black, M.J., Ahmad, A.: irotate: Active visual slam for omnidirectional robots. Robotics and Autonomous Systems **154**, 104102 (2022) https://doi.org/10.1016/j.robot.2022.104102

[16] Meng, Z., Sun, H., Qin, H., Chen, Z., Zhou, C., Ang, M.H.: Intelligent robotic system for autonomous exploration and active slam in unknown environments. In: 2017 IEEE/SICE International Symposium on System Integration (SII), pp. 651–656 (2017). https://doi.org/10.1109/SII.2017.8279295

[17] Ossenkopf, M., Castro, G., Pessacg, F., Geihs, K., De Cristóforis, P.: Long-horizon active slam system for multi-agent coordinated exploration. In: 2019 European Conference on Mobile Robots (ECMR), pp. 1–6 (2019). https://doi.org/10.1109/ECMR.2019.8870952

[18] Kontitsis, M., Theodorou, E.A., Todorov, E.: Multi-robot active slam with relative entropy optimization. In: 2013 American Control Conference, pp. 2757–2764 (2013). https://doi.org/10.1109/ACC.2013.6580252

[19] Pei, Z., Piao, S., Souidi, M.E.H., Qadir, M.Z., Li, G.: Slam for humanoid multi-robot active cooperation based on relative observation. Sustainability **10**(8) (2018) https://doi.org/10.3390/su10082946

[20] Chen, Y., Zhao, L., Lee, K.M.B., Yoo, C., Huang, S., Fitch, R.: Broadcast your weaknesses: Cooperative active pose-graph slam for multiple robots. IEEE Robotics and Automation Letters **5**(2), 2200–2207 (2020) https://doi.org/10.1109/LRA.2020.2970665

[21] Arvanitakis, I., Tzes, A.: Collaborative mapping and navigation for a mobile robot swarm. In: 2017 25th Mediterranean Conference on Control and Automation (MED), pp. 696–700 (2017). https://doi.org/10.1109/MED.2017.7984199

[22] Chen, Y., Huang, S., Fitch, R.: Active slam for mobile robots with area coverage and obstacle avoidance. IEEE/ASME Transactions on Mechatronics **25**(3), 1182–1192 (2020) https://doi.org/10.1109/TMECH.2019.2963439

[23] Chen, Y., Huang, S., Fitch, R., Yu, J.: Efficient Active SLAM Based on Submap Joining, Graph Topology and Convex Optimization. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 5159–5166. IEEE, Brisbane, QLD (2018). https://doi.org/10.1109/ICRA.2018.8460864

[24] Pei, Z., Piao, S., Quan, M., Qadir, M.Z., Li, G.: Active collaboration in relative observation for multi-agent visual simultaneous localization and mapping based on Deep Q Network. International Journal of Advanced Robotic Systems **17**(2), 172988142092021 (2020) https://doi.org/10.1177/1729881420920216

[25] Phueakthong, P., Varagul, J., Pinrath, N.: Deep reinforcement learning based mobile robot navigation in unknown environment with continuous action space. In: 2022 5th International Conference on Intelligent Autonomous Systems (ICoIAS), pp. 154–158 (2022). https://doi.org/10.1109/ICoIAS56028.2022.9931272

[26] Campos, C., Elvira, R., Rodríguez, J.J.G., M. Montiel, J.M., D. Tardós, J.: Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. IEEE Transactions on Robotics **37**(6), 1874–1890 (2021) https://doi.org/10.1109/TRO.2021.3075644

[27] Zhao, Y., Xiong, Z., Zhou, S., Wang, J., Zhang, L., Campoy, P.: Perception-aware planning for active slam in dynamic environments. Remote Sensing **14**(11) (2022) https://doi.org/10.3390/rs14112584

[28] Li, J., Cheng, Y., Zhou, J., Chen, J., Liu, Z., Hu, S., Leung, V.C.M.: Energy-efficient ground traversability mapping based on uav-ugv collaborative system. IEEE Transactions on Green Communications and Networking **6**(1), 69–78 (2022) https://doi.org/10.1109/TGCN.2021.3107291

[29] Khosoussi, K., Sukhatme, G.S., Huang, S., Dissanayake, G.: In: Goldberg, K., Abbeel, P., Bekris, K., Miller, L. (eds.) Designing Sparse Reliable Pose-Graph SLAM: A Graph-Theoretic Approach, pp. 17–32. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-43089-4_2

[30] Khosoussi, K., Huang, S., Dissanayake, G.: Novel insights into the impact of graph structure on slam. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2707–2714 (2014). https://doi.org/10.1109/IROS.2014.6942932

[31] Khosoussi, K., Giamou, M., Sukhatme, G.S., Huang, S., Dissanayake, G., How, J.P.: Reliable graphs for slam. The International Journal of Robotics Research **38**(2-3), 260–298 (2019) https://doi.org/10.1177/0278364918823086

[32] Placed, J.A., Rodríguez, J.J.G., Tardós, J.D., Castellanos, J.A.: ExplORB-SLAM: Active Visual SLAM Exploiting the Pose-graph Topology. In: Tardioli, D., Matellán, V., Heredia, G., Silva, M.F., Marques, L. (eds.) ROBOT2022: Fifth Iberian Robotics Conference vol. 589, pp. 199–210. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-21065-5_17 . Series Title: Lecture Notes in Networks and Systems

[33] Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1**(1), 269–271 (1959)

[34] Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. IEEE Robotics & Automation Magazine **4**(1), 23–33 (1997) https://doi.org/10.1109/100.580977

[35] Ahmed, M.F., Frémont, V., Fantoni, I.: Active slam utility function exploiting path entropy. In: 2023 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI), pp. 1–7 (2023). https://doi.org/10.1109/SOLI60636.2023.10425063

[36] Yamauchi, B.: A frontier-based approach for autonomous exploration. In: Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation', pp. 146–151 (1997). https://doi.org/10.1109/CIRA.1997.613851

Statements and Declarations

- Funding: This work was carried out in the framework of the NExT Senior Talent Chair DeepCoSLAM, which was funded by the French Government, through the program Investments for the Future managed by the National Agency for Research ANR-16-IDEX-0007, and has been supported by the DIONISO project (progetto SCN_00320 - INVITALIA) which is funded by the Italian Government.

- Competing interests: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.
- Ethics approval and consent to participate: Not applicable.
- Consent for publication: Not applicable.
- Author contribution: Conceptualization, Muhammad Farhan Ahmed (M.F.A.), Matteo Maragliano (M.M.), Vincent Frémont (V.F.), Carmine Tomasso Recchiuto (C.T.R.); methodology, M.F.A. and M.M.; validation, V.F. C.T.R.; formal analysis, M.F.A.; investigation, M.M.; resources, V.F. and C.T.R.; data curation, M.F.A. and M.M.; writing original draft preparation, M.F.A. and M.M.; writing review and editing, C.T.R. and V.F.; visualization, M.F.A. and M.M.; supervision, V.F and C.T.R.; project administration, V.F., C.T.R; and funding acquisition, V.F. All authors have read and agreed to the published version of the manuscript.