2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 27 - October 1, 2021. Prague, Czech Republic

Prioritized Indoor Exploration with a Dynamic Deadline

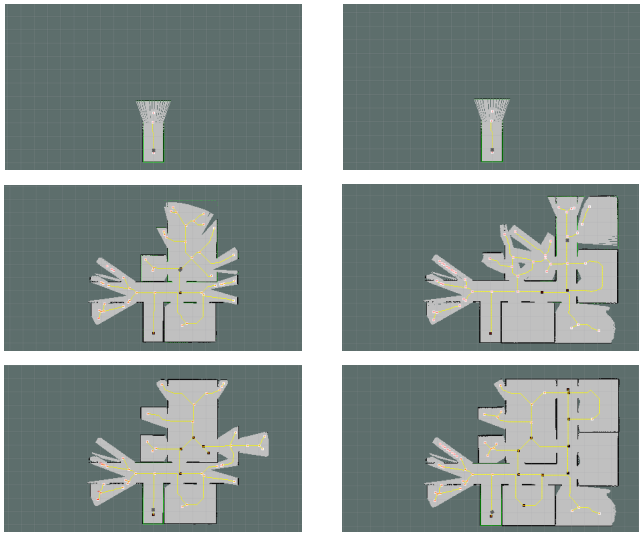
Sayantan Datta and Srinivas Akella

Abstract— **Indoor exploration using mobile robots has typically focused on exploring the entire environment without considering deadlines. This paper introduces a priority-based exploration algorithm for situations with an initially unknown and dynamically assigned deadline. The goal of our exploration strategy is to determine the geometric structure of an unknown environment as rapidly as possible and return to the home location. This is necessary for dangerous environments where an initial rapid robot exploration provides critical information about the layout for subsequent operations. For example, firefighters, for whom time is of the essence, can utilize the map generated by this robotic exploration to navigate a building on fire. We present a three-part strategy to solve this problem. First, we represent the physical environment as an exploration graph, whose vertices represent the local environment with its geometric and semantic information. Second, we assign priority values to these vertices based on their environment regions. Third, we present a graph exploration algorithm that employs the vertex priorities. Simulation experiments on a set of graph environments and Gazebo environments demonstrate that, in contrast to prior approaches that ignore the semantic information, our priority-based exploration algorithm enables the robot to efficiently explore more of the environment while satisfying its deadline constraints.**

I. Introduction

This paper presents a novel variant of the indoor exploration problem. Prior work has primarily focused on exploration without a time limit, intended to completely explore an indoor environment [1], [2], [3], [4]. We focus on indoor exploration with dynamic deadlines and present a prioritized approach to address this problem. Our algorithm applies to scenarios where the robot does not have prior knowledge of a deadline when it begins its exploration; the robot is informed of the time remaining while exploring the environment. We develop an approach to explore a building to determine its layout (i.e., connectivity) to the maximum extent possible and return to the home location, all within the deadline.

Situations with deadlines arise when the environment or the robot imposes constraints on the exploration time. For example, during exploration of an indoor radioactive environment (e.g., a nuclear power plant during a radiation leak), the radiation the robot receives reduces the time that the robot can function in the environment. Similarly, while exploring a building on fire, the robot's exploration time depends on how damaging the fire is to the robot. It is also difficult to exactly estimate the remaining battery life in a robot, especially when using an unmanned aerial vehicle (UAV). UAVs have a short battery life, and depending on the trajectory, the battery can deplete faster than expected. In all these scenarios, the robot typically starts the exploration with an unknown deadline. During exploration, it receives a time limit and must come back to the starting location within the time limit. So the robot must efficiently explore the environment in limited time to gather information about its connectivity and geometric layout.



(a) (b)

Fig. 1: Snapshots of exploration of an environment using (a) a cost-based greedy algorithm and (b) our priority-based algorithm. The cost-based algorithm explores the nearest vertex first, while the priority-based algorithm prioritizes exploring corridors over rooms. In the top row, both the robots start without a deadline. The middle row shows the explored environments at 500 seconds, when the robots are informed of a deadline of 500 seconds. The bottom row shows the explored environments at 1000 seconds after the robots return home. The cost-based algorithm explored 37% of the environment while the priority-based algorithm explored 56% of the environment.

To achieve this, we prioritize the exploration of building features with high connectivity, such as corridors and large rooms. See Figure 1 for an illustrative example. We demonstrate the effectiveness of our approach by comparing its performance against a cost-based greedy strategy with the same deadline.

Our contribution is a prioritized exploration approach to plan a robot's motion so it can efficiently explore the environment with a dynamically set deadline. The exploration is based on first visiting the high priority regions of the unknown environment and visiting lower priority regions if time permits. Our technique assigns a priority to each discovered location and follows a greedy approach to visit

The authors are with the Department of Computer Science, University of North Carolina at Charlotte, NC 28223, USA. This work was partially supported by NSF Award IIP-1919233.
sdatta3@uncc.edu, sakella@uncc.edu

high priority regions of the environment through the least cost paths to them. Finally, the robot returns to the home location within the deadline. In our approach, the robots prioritize exploring corridors over large rooms, and large rooms over smaller rooms.

The paper is organized as follows: In Section II, we briefly review the related work in indoor exploration by mobile robots. After stating the problem in Section III, we introduce our prioritized exploration strategy in Section IV. Our simulation environments and metrics are illustrated in Section V, and we discuss our results in Section VI.

II. Related Work

Autonomous robotic exploration is an important and well-studied problem. A widely used approach for robotic exploration is frontier-based exploration, which greedily directs robots to regions likely to provide new information about the environment. It can build occupancy grid representations of environments using either a single robot [1] or multiple robots [5]. Decision theoretic approaches for exploration consider the information gain of exploration actions and robot pose uncertainty reduction of loop-closing actions [6]. See Thrun et al. [7] for an overview of probabilistic mapping techniques that compute occupancy grids using Simultaneous Localization and Mapping (SLAM) approaches.

Robotic exploration was modeled as graph construction by Dudek et al. [8] for a robot that uses portable markers instead of distance and orientation sensors. An exploration and mapping strategy using a topological model based on a spatial semantic hierarchy was developed by Kuipers and Byun [9]. Cowley et al. [10] advocate an efficient exploration approach that uses both topological and metric data to identify places with semantic significance for exploration. The idea was improved by Wang et. al. [3] by using a topological map and semantic information from RGBD sensors to improve indoor exploration. The technique uses semantic information from the environment to distinguish between rooms and corridors. Since their exploration algorithm is directed towards rooms, the robot revisits corridors several times. This makes it less suitable for dynamic deadline environments. Graph-based exploration for subterranean environments was addressed in [11]. The authors implement a two-stage local and global planner that enables the UAV to return to its starting location based on the current remaining flight time, information that is available from the start of the exploration. However, the approach does not consider dynamic deadlines.

Communication and energy constraints during exploration have been addressed by [12] for a multi-robot setup. Energy constraints serve as deadlines; however the focus is on multi-robot coordination, and the robots do not return to their home locations. Semantic information-based coordinated multi-robot exploration has been introduced in [2]. Prior information, in the form of rough topometric graphs provided by a user [13], or predictions of unexplored regions of the environment based on environment structure and a library of previously seen maps [14], has recently been leveraged for more efficient exploration. Recent work uses deep reinforcement learning (DRL) to learn exploration information over office blueprints [15]. DRL-based exploration [16], [15] has not considered deadlines.

III. The Exploration with Dynamic Deadlines Problem

A single robot explores an unknown indoor environment and models the explored region as an undirected graph $G = (V, E)$. V is the set of vertices representing the physical locations in the explored environment, and E is the set of connecting edges. Edge $e_{ij} \in E$ is a collision-free path between vertices v_i and v_j of weight w_{ij}, where w_{ij} is the time required for the robot to traverse the edge. A priority $p(v_i)$ is associated with each vertex v_i depending on the geometry of the physical location it represents. The set of vertices visited by the robot during exploration is V_{vis}, and the vertices adjacent to V_{vis} are the set of discovered vertices, V_d. At each iteration, the robot selects a target vertex $v_t \in V_d$ with the highest priority, and computes the least cost path to it. We assume that the environment is static, the robot has a 360° field-of-view sensor, and a vertex once visited will offer no additional information about the unknown environment.

The robot, during the course of its exploration, is provided a *time remaining* deadline, t_r. The time instant at which the robot is provided this deadline is referred to as *alarm time*, t_a. For example, if t_r is 100 seconds and t_a is 50 seconds, it means that the robot is provided a deadline of 100 seconds after 50 seconds of exploring the environment.

The robot's goal is to identify the environment connectivity and maximize the number of explored vertices V, which is the union of the vertices the robot has discovered and visited, $V_d \cup V_{vis}$. The robot is also subject to the constraint of returning back to the home vertex within the total exploration time of $t_r + t_a$. In the above example, the robot should explore the unknown environment and be back at the home location within a total of 150 seconds.

We consider two types of environments: graph environments and Gazebo environments. The graph environments represent an ideal scenario for the exploration algorithm without sensor or control errors. For the graph environments, we make the following assumptions: We are given a function $p(v_i)$ that assigns the priority of vertex v_i. The graph environment reported by the robot is accurate and the position of the robot in the environment is known exactly.

However, these assumptions do not hold for real-world experiments. Hence in our Gazebo simulations, we assume that the robot is equipped with sensors (e.g., Lidar) to detect the obstacles and classify the type of physical location of a vertex. As the generated graph here is not incrementally updated, but regenerated at every iteration of the exploration algorithm, the set of visited vertices V_{vis} refers to specific points on the occupancy grid.

IV. Prioritized Exploration Strategy

The prioritized exploration strategy consists of three stages. (1) Represent the exploration environment as a graph.

(2) Assign priorities to the discovered vertices. (3) Depending on whether the deadline t_r has been received or not, follow the exploration algorithm for exploring with a deadline or the exploration algorithm without a deadline.

The robot uses the exploration algorithm for exploring without a deadline, until a deadline t_r is received. Once the robot has a deadline, it uses a different exploration algorithm that ensures that the robot can explore and return to its home location within the deadline. If the robot is not provided with any deadline, it explores all of the unknown region and returns to the home location.

A. Creating the Exploration Graph

For the graph environments, we assume that the robot discovers the vertices along with their semantic information. The semantic information is the type of region the vertex is in (e.g., corridor, room). We also assume that two vertices are connected by an edge if they have mutual line-of-sight visibility. These simplifying assumptions allow greater focus on the exploration algorithms.

We next discuss creation of the exploration graphs for Gazebo environments. The agent senses an initially unknown environment using its Lidar sensor. By running a SLAM algorithm on the robot odometry and Lidar data, we create an occupancy grid map [17] of the environment. Each cell of this occupancy grid shows the probability of it being occupied. This occupancy grid map is morphed into a skeleton image [18]. This image is further transformed into a skeleton graph, which serves as the exploration graph G. The vertices V in this skeleton graph indicate two types of locations of interest in the environment. The first is a frontier region between the explored and unexplored regions. The second is an intersection or branching of possible paths that the robot can take, such as an intersection of two corridors or a corridor and a room. An edge connects a pair of vertices if there is a collision-free path between them. As the robot traverses the environment, the exploration map is updated. A new skeleton graph is generated after the robot reaches the next vertex during exploration.

Our exploration algorithm can also be used, with minimal modifications, with other 2D and 3D environment representations such as topometric graphs [10] and OctoMap [19].

Each vertex can represent the local environment and its associated semantic information observed by the robot's sensor at that position. The shape and size of the available floor region are used to assign a label to a given vertex. For example, if a vertex is in a region that is a narrow passageway or a passageway that leads to different doors, the vertex is labeled as a "corridor". Similarly, if a vertex is in a room with small dimensions, it is labeled as a "small room". On the other hand, if a vertex is in an area with large open spaces, it is labeled as a "large room".

We have implemented a rule-based classification of corridors, small rooms, and large rooms. For every vertex, we extract a local map, a 3 m $\times$ 3 m window of the occupancy grid map centered on the vertex. This local map provides information on the obstacles and free space of the environment. We employ a geometry-based approach to identify the location and orientation of the obstacles in the local map. The vertices are classified based on the number of obstacles, and the distance and orientation of the obstacles relative to one another.

Algorithm 1: Prioritized Exploration with a Dynamic Deadline

Input: starting vertex: v_0; deadline: $t_r \leftarrow$ *unknown*
Result: Explored graph G

```
G ← ∅ ;                          // initialize exploration graph
V_d ← ∅ ;   // discovered and unvisited vertices
V_vis ← {v_0} ;                  // visited vertices
Add (v_0, E(v_0)) to G ;     // E(v_i) is set of edges
 incident on vertex i
v_c ← v_0 ;                          // current vertex
v_h ← v_0 ;                            // home vertex
V_n ← N_G(v_c) ;     // all vertices adjacent to v_c
V_d ← V_d ∪ V_n;
while V_d ≠ ∅ do
    t_r ← Query_Deadline() ;   // Check deadline
    if t_r is unknown then
        v_t ← NextVertexWODeadline(V_d, v_c);
    else if t_r is 0 then
        return G
    else
        v_t ← NextVertexWDeadline(V_d, v_c, v_h, t_r);
    Move agent to v_t through graph G;
    V_d ← V_d \ v_t; v_c ← v_t; V_vis ← V_vis ∪ v_c ;
    Add (v_c, V_n, E(v_c)) to G;
    V_n ← N_G(v_c) \ V_vis; // exclude visited vertices
    V_d ← V_d ∪ V_n ;   // update discovered vertices
if t_r is unknown then
    Move agent to v_h through G;
return G;
```

B. Vertex Prioritization

We introduce two agents, Alice and Bob, that represent different exploration strategies. Both agents explore an unknown indoor environment, and if time permits, return to the home location. However, they order their visits to discovered vertices differently.

Alice uses a priority-based greedy exploration strategy. It assigns a level of priority for each type of discovered vertex region. Corridors are assigned a higher priority over rooms, and large rooms are assigned a higher priority over small rooms. The agent chooses to visit the vertex v_i with the highest priority, breaking ties by selecting the lower cost vertex. Alice's goal is to rapidly compute the building layout by prioritizing the exploration of corridors over other regions.

Bob uses a cost-based greedy exploration strategy, as in [1]. All vertices have the same priority, and they are visited based on their costs.

C. Prioritized Exploration Algorithm

The input to the prioritized exploration algorithm (Algorithm 1) is the starting position v_0 of the robot, also called the home vertex v_h. The output is the explored graph G under the constraint that the robot should return to the home position by the end of exploration.

Algorithm 2: `NextVertexWODeadline`: Vertex selection without deadline

Input: V_d, v_c
Result: Next vertex to visit
$v_t \leftarrow$ initialize from V_d;
for v_d *in* V_d **do**
  if $p(v_d) > p(v_t)$ **then**
    $v_t \leftarrow v_d$;
  if $p(v_d) = p(v_t)$ **then**
    $cost_{cd} \leftarrow w_{cd}$; $cost_{ct} \leftarrow w_{ct}$;
    if $cost_{cd} < cost_{ct}$ **then** $v_t \leftarrow v_d$;
return v_t;

The robot iteratively visits vertices while adding the set of visited vertices V_{vis} as well as the set of discovered vertices V_d to the explored graph G. It will explore the environment as long as the set of discovered vertices V_d is not empty and there is time remaining to explore.

1) Exploration without a Deadline: When exploring without a deadline, the agent performs a greedy selection of the vertex with the highest priority from the set of discovered vertices V_d (Algorithm 2). If there are multiple vertices with the highest priority, the agent chooses the one with the lowest cost. The cost is the estimated time to traverse the shortest path between the current vertex v_c and the candidate target vertex.

2) Exploration with a Deadline: When the deadline is known to the robot during exploration, it computes whether the deadline provides enough time for it to return to the home vertex. If the time limit t_r is adequate, the robot checks if the time limit allows it to explore additional vertices in V_d before returning to the home vertex v_h. It identifies a subset $V_{eligible}$ of V_d that consists of the vertices the robot would be able to visit and still return home within the deadline. The robot selects the vertex with the highest priority in this subset $V_{eligible}$. If there are several such vertices with equal priority, the robot chooses to visit the vertex with the lowest path costs $cost_{cd}$ and $cost_{dh}$; $cost_{cd}$ is the path cost to travel from the current vertex v_c to vertex $v_d \in V_{eligble}$, and $cost_{dh}$ is the path cost to travel from v_d to the home vertex v_h. If the time is inadequate for the robot to reach the home vertex, the robot continues exploring the environment until the deadline so it can communicate the explored map back to its base. This behavior can be modified to instead make the robot return as close as possible to the home location. The $cost_{dh}$ is used as a tiebreaker when return is not possible, so the robot can come closer to the home location. See the pseudocode in Algorithm 3.

Algorithm 3: `NextVertexWDeadline`: Vertex selection with deadline

Input: V_d, v_c, v_h, t_r
Result: Vertex to visit next
$cost_{ch} \leftarrow w_{ch}$;
if $cost_{ch} < t_r$ **then** `// Return home is possible`
  $V_{eligible} \leftarrow \emptyset$;
  for v_d *in* V_d **do**
    $cost_{cd} \leftarrow w_{cd}$; $cost_{dh} \leftarrow w_{dh}$;
    if $(cost_{cd} + cost_{dh}) < t_r$ **then**
      add v_d to $V_{eligible}$;
  if $V_{eligible}$ *is* $\emptyset$ **then return** v_h;
  $v_t \leftarrow \arg\max_{v_d \in V_{eligible}} p(v_d)$, tiebreakers: $cost_{cd}, cost_{dh}$;
  return v_t
else `// Return home not possible`
  $V_{eligible} \leftarrow \emptyset$;
  for v_d *in* V_d **do**
    $cost_{cd} \leftarrow w_{cd}$;
    if $cost_{cd} < t_r$ **then**
      add v_d to $V_{eligible}$;
  if $V_{eligible}$ *is* $\emptyset$ **then**
    return v_c
  $v_t \leftarrow \arg\max_{v_d \in V_{eligible}} p(v_d)$, tiebreakers: $cost_{cd}, cost_{dh}$;
  return v_t

V. Simulation Environments

A. Simulation Environment Setup

Typically indoor environments employ corridors to connect multiple rooms and other corridors in a building. A corridor is an architectural element that functionally leads to rooms, or that loops back to itself while connecting to rooms, or that branches off to several corridors, which in turn connect to rooms. Usually, in a large indoor environment, we find a combination of all the above three layouts.

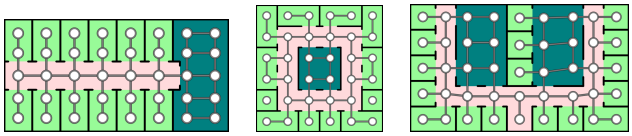


Fig. 2: Three indoor graph environments showing different corridor layouts used in our exploration algorithm simulations. The respective graph environments are overlaid on them. Light green represents small rooms, dark green represents large rooms, and pink represents corridors. (a) Straight Corridor ends in a large room, with rooms on either side. (b) Looped Corridor wraps around a large room, connecting multiple rooms on its periphery. (c) Branched Corridor splits into two, connecting large and small rooms.

1) Graph Environments: These simulated environments, represented as graphs, are based on the three types of corridor layouts (Figure 2). Figure 2(a) shows an environment with a straight line corridor ending in a large room while connecting small rooms on either side. Figure 2(b) shows an environment where a corridor wraps around a large room while connecting to several rooms on its outer periphery. Figure 2(c) shows an environment with a corridor that branches into two separate corridors. Each of these separated corridors has large and small rooms connected to them. Since corridors are not always necessary to connect rooms, this layout has the large room on the right connecting to several smaller rooms.

2) Gazebo Environments: A set of three environments (Figure 3) similar to the graph environments has been created

TABLE I: Exploration results for the graph environments.

t_r	Agent	Straight corridor				Looped corridor				Branched Corridor			
		Corridor	Large Room	Small Room	All Vertices	Corridor	Large Room	Small Room	All Vertices	Corridor	Large Room	Small Room	All Vertices
500	Alice	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	90.0%	94.4%	100.0%	100.0%	100.0%	100.0%
	Bob	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	90.0%	94.4%	100.0%	100.0%	100.0%	100.0%
60	Alice	100.0%	**100.0%**	61.4%	**76.8%**	**100.0%**	100.0%	**65.0%**	**80.6%**	**100.0%**	**75.0%**	**75.0%**	**82.8%**
	Bob	99.3%	22.6%	**73.0%**	64.4%	93.8%	99.0%	59.3%	75.2%	72.6%	62.1%	46.6%	58.8%
30	Alice	**100.0%**	**66.8%**	**50.0%**	**61.7%**	**83.3%**	**85.8%**	**40.0%**	**59.5%**	**64.3%**	32.3%	**36.8%**	**44.2%**
	Bob	75.2%	0.3%	43.5%	37.4%	63.1%	71.5%	28.5%	44.8%	51.6%	32.1%	21.8%	33.8%

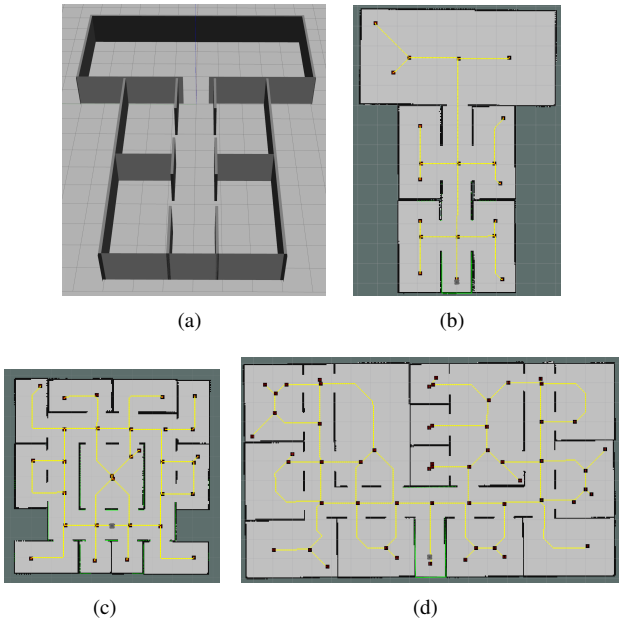


Fig. 3: Three Gazebo environments with different corridor layouts used in simulating our exploration algorithm. (a) Straight Corridor Gazebo environment in 3D. (b, c, d) Occupancy grid maps of the environments, where yellow lines represent edges and black points represent vertices. (b) Straight Corridor environment. (c) Looped Corridor environment. (d) Branched Corridor environment.

in Gazebo [20]. These environments are explored using a simulated Turtlebot3 robot equipped with a single scan 360° Lidar with a range of 6 meters, matching the Slamtec RPLidar A1M8 sensor. The robot starts from a specified location. We use the GMapping SLAM algorithm [21] to create an occupancy grid map from the sensor data. This occupancy grid map is converted to G as discussed in Section IV-A. A new skeleton graph is created after the robot reaches the target vertex while exploring the map. This adds a few challenges: First, the graph needs to be recomputed every time the map is updated as some of the frontier vertices may cease to exist once the environment is explored further past the current frontier. Second, a list of visited vertices V_{vis} cannot be maintained as previously existing vertices may not exist later on during the exploration. Third, as the robot follows the edges of the skeleton graph, the edges and vertices should be at a safe distance from the obstacles to avoid collisions. Finally, the SLAM algorithm ignores Lidar sensor values of infinity that are reported when there are no obstacles within the sensor range in a region. So such regions are not reported on the map.

We address these challenges with the following steps. We update the graph once the robot reaches the target vertex, instead of every time the map is updated by the SLAM algorithm. We maintain the locations that the robot has visited as a substitute for the set of visited vertices. If there are new vertices that are created within a threshold distance from the stored locations, we ensure that the vertices are not a part of V_d. To ensure collision-free edges between vertices, we make a binary map of the occupancy grid, with the unoccupied space marked with a different value than the rest of the map. This unoccupied space is eroded by the robot's width. This eroded map is then converted to a skeleton graph. This ensures that the edges of the skeleton graph are at a safe distance from the obstacles.

B. Metrics

The performance of the exploration algorithms is measured using the extent of the explored environment. We compare our priority-based greedy exploration strategy of Alice to the cost-based greedy strategy of Bob. The cost-based exploration strategy of Bob is motivated by a baseline greedy exploration algorithm [1]. As the agents are informed of the deadline at time instant t_a, we compare the performance of the agents at time $t_a + t_r$.

For the graph environments, explored regions of the map are represented by the set of explored vertices V, where $V = (V_d \cup V_{vis})$. The performance metric for the graph environments is $|V|$, the number of explored vertices.

Since the number of vertices may change during exploration for the Gazebo environments, the performance metric used is the percentage of floor area explored. Here the exploration time includes the computation time, and the times t_a and t_r are measured in seconds.

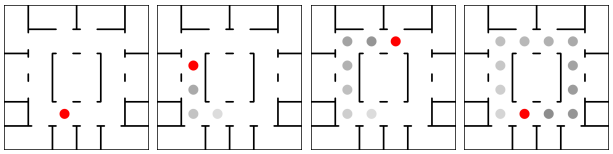
VI. Results and Discussion

A. Graph Environments

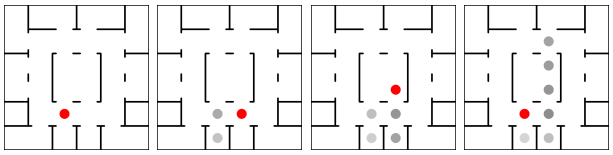
Agents Alice and Bob are evaluated on the three graph environments (Figure 2) in Table I. For these experiments, we provided the agents with the deadline t_r at the start of exploration. So t_a is zero. The deadline t_r has been set to three different values: a large deadline, an intermediate deadline, and a short deadline represented by the values 500, 60, and 30 respectively. Since Bob randomly chooses

TABLE II: Exploration results for the Gazebo environments.

t_a	t_r	Agent	Straight corridor				Looped Corridor				Branched Corridor			
			Corridor	Large Room	Small Room	Total	Corridor	Large Room	Small Room	Total	Corridor	Large Room	Small Room	Total
500	1000	Alice	99.8%	**98.7%**	96.7%	**98.1%**	100.0%	99.7%	97.6%	98.7%	**80.3%**	**66.7%**	**64.8%**	**68.6%**
		Bob	99.6%	77.2%	98.6%	88.7%	100.0%	99.3%	97.8%	98.9%	77.5%	54.7%	62.6%	61.4%
500	500	Alice	**100.0%**	**24.1%**	96.6%	**63.0%**	100.0%	99.2%	96.1%	97.9%	**68.2%**	54.6%	**48.5%**	**54.3%**
		Bob	97.7%	6.1%	96.5%	54.3%	100.0%	99.6%	96.8%	98.3%	55.6%	54.8%	21.5%	36.8%
0	500	Alice	85.3%	0.0%	**65.8%**	37.7%	**89.0%**	99.6%	**76.0%**	**83.5%**	**50.4%**	**19.7%**	**21.5%**	**27.4%**
		Bob	84.7%	0.0%	62.5%	36.3%	85.6%	99.5%	57.7%	72.6%	43.3%	18.7%	14.2%	21.6%



(a) Exploration by Alice.



(b) Exploration by Bob.

Fig. 4: Exploration of the Looped Corridor graph environment. Each row shows snapshots of exploration at time steps 0, 6, 12, and 25. The deadline t_r of 10 time steps was provided at a t_a of 15 time steps. Each edge cost is 2 time steps. The red circle denotes the current position of the robot and the trail of circles show the visited vertices. (a) Alice's strategy, which prioritizes corridors over large rooms and small rooms. (b) Bob's strategy, which gives equal priority to all vertices.

between two vertices of the same cost, the values in Table I are averaged over 100 simulations for each deadline.

A comparison of exploration by Alice and Bob in the Looped Corridor graph environment is shown in Figure 4. It shows a trail of explored vertices as an agent explores the environment of Figure 2(b).

For all three environments, in the case of a large deadline $t_r = 500$, we observe that both agents have explored the entire graph environment. For an intermediate deadline of $t_r = 60$, Alice explores a larger percentage of all vertices than Bob. Since Alice prioritizes corridors over other regions, its corridor exploration percentage is higher than for Bob in all the environments. For the Straight Corridor environment, Bob explores a higher percentage of small rooms than Alice while Alice explores a higher percentage of the large room than Bob. For the short deadline of $t_r = 30$, Alice explores a higher portion of the environment than Bob for all three graph environments. As small rooms and large rooms are adjacent to the corridor, Alice eventually explores a higher percentage of these sections while visiting the corridor vertices.

To summarize, Alice's prioritization of corridor vertices has allowed it to discover more vertices, thereby increasing the overall exploration percentage.

B. Gazebo Environments

Agents Alice and Bob were evaluated, based on the percentage of explored floor area, on the three Gazebo environments shown in Figure 3. The deadline t_r to return to the home location is provided to the robot after t_a seconds. If the deadline is provided at the start, $t_a = 0$, else $t_a > 0$.

For our experiments, we have three sets of values for t_a and t_r. The values $t_a = 0$ and $t_r = 500$ signify a short exploration time provided at the start of the exploration. The other two cases $t_a = 500$, $t_r = 500$ and $t_a = 500$, $t_r = 1000$ correspond to intermediate and long exploration times. The exploration results for the three environments, averaged over three simulations for each exploration time, are presented in Table II.

For the Straight Corridor environment (Figure 3(b)), with the longest exploration time, $t_a = 500$, $t_r = 1000$, both Alice and Bob almost completely explore the environment. Alice explores a higher percentage of the large room that lies at the end of the corridor while Bob explores a higher percentage of small rooms. For the intermediate exploration time, $t_a = 500$, $t_r = 500$, Alice explores a higher percentage of the large room, while Bob explores the smaller rooms before returning to the home location. On average, Alice has explored 24% of the large room, while Bob has explored 6% of the large room. When the exploration time is short, $t_a = 0$, $t_r = 500$, Alice explores 38% of the total environment while Bob explores 36% of the total environment. For this case, for every graph update, the nearest frontier vertex to the robot is coincidentally the vertex with the highest priority. So Alice and Bob visit almost the same set of locations during exploration. That the area of the corridor explored by both Alice and Bob is equal (85% of the corridors) quantifies this observation.

The Looped Corridor environment of Figure 3(c) is much smaller than the other two environments. For exploration with the longest exploration time, $t_a = 500$, $t_r = 1000$, both Alice and Bob have explored the entire environment. With an intermediate exploration time of $t_a = 500$, $t_r = 500$, both exploration agents explore almost all of the unknown environment. When the exploration time is short, $t_a = 0$, $t_r = 500$, Alice explores more of the corridor and of the total environment (83.5%) compared to Bob's total exploration of 72.6%.

The Branched Corridor environment of Figure 3(d) is

much larger than the other two environments. Hence, both Alice and Bob could not explore the entire environment even with the longest exploration time, $t_a = 500$, $t_r = 1000$. The performance difference between Alice and Bob is the lowest for this case because the cost of returning back home limits the exploration in the second branch for both agents. When $t_a = 500$, $t_r = 500$, the difference is more pronounced as Alice explores one complete branch of the corridor and a part of the other corridor, while Bob explores significantly less. However, for the shortest exploration time, $t_a = 0$, $t_r = 500$, Alice completely explores one of the two branches of the corridor, while Bob's bias towards the closest vertex limits the extent of its exploration.

We observe that the explored percentage for Alice is consistently higher than or equal to that for Bob. By visiting the corridors first, Alice often explored the neighboring regions of small and large rooms. The performance on the Gazebo environments is not as high as on the graph environments due to the additional challenges discussed earlier. We observed that the robots spend time to travel back to the skeleton graph, instead of traveling directly to a target vertex. Even though this impacts the exploration performance, it ensures collision-free paths. Additionally, changes in the locations of obstacle boundaries in the occupancy map from the SLAM algorithm affect the accuracy of priority classification of the environment regions.

VII. Conclusion

We have presented a prioritized exploration algorithm for indoor environments under the constraint of a dynamic deadline. The goal is to rapidly determine the geometric structure and connectivity of the environment. The explored environment is modeled as a graph, and sensor data and odometry serve as inputs to the algorithm. Our algorithm prescribes a prioritized order for exploring discovered vertices to maximize exploration of the graph. It also ensures, when there is sufficient time, that the robot can return to the home location at the end of the exploration. We have tested our algorithm in realistic environments in Gazebo and show significant improvement over non-prioritized exploration. Our algorithm performs well despite the inaccuracies of SLAM, mobile robot sensor noise, and errors due to misclassification of building regions in the simulation. It can be easily extended to handle multiple dynamic deadlines.

There are several directions for future work. In our simulations, we have calculated vertex priorities from local occupancy grid maps. Instead of a geometry-based classification, we can utilize a neural network based classification of the occupancy grid maps. Similarly, we can use computer vision techniques to recognize building regions and compute vertex priorities. We plan to demonstrate our exploration algorithm using real robots in a office-like environment. We also plan to develop a multi-robot exploration approach for dynamic deadline constraints.

References

[1] B. Yamauchi, "A frontier-based approach for autonomous exploration.," in *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pp. 146–151, July 1997.

[2] C. Stachniss, Ó. M. Mozos, and W. Burgard, "Efficient exploration of unknown indoor environments using a team of mobile robots," *Annals of Mathematics and Artificial Intelligence*, vol. 52, no. 2-4, pp. 205–227, 2008.

[3] C. Wang, D. Zhu, T. Li, M. Q.-H. Meng, and C. W. de Silva, "Efficient autonomous robotic exploration with semantic road map in indoor environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2989–2996, 2019.

[4] M. T. Ohradzansky, A. B. Mills, E. R. Rush, D. G. Riley, E. W. Frew, and J. S. Humbert, "Reactive control and metric-topological planning for exploration," in *IEEE International Conference on Robotics and Automation*, pp. 4073–4079, 2020.

[5] B. Yamauchi, "Frontier-based exploration using multiple robots," in *Second International Conference on Autonomous Agents*, pp. 47–53, 1998.

[6] C. Stachniss, G. Grisetti, and W. Burgard, "Information gain-based exploration using Rao-Blackwellized particle filters," in *Robotics: Science and Systems*, vol. 2, pp. 65–72, 2005.

[7] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, Sept. 2005.

[8] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, "Robotic exploration as graph construction," *IEEE Transactions on Robotics and Automation*, vol. 7, pp. 859–865, Dec. 1991.

[9] B. Kuipers and Y.-T. Byun, "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations," *Journal of Robotics and Autonomous Systems*, vol. 8, no. 1/2, pp. 47–63, 1993.

[10] A. Cowley, C. J. Taylor, and B. Southall, "Rapid multi-robot exploration with topometric maps," in *IEEE International Conference on Robotics and Automation*, pp. 1044–1049, 2011.

[11] T. Dang, F. Mascarich, S. Khattak, C. Papachristos, and K. Alexis, "Graph-based path planning for autonomous robotic exploration in subterranean environments," in *IEEE International Conference on Intelligent Robots and Systems*, pp. 3105–3112, 2019.

[12] K. Cesare, R. Skeele, S.-H. Yoo, Y. Zhang, and G. Hollinger, "Multi-UAV exploration with limited communication and battery," in *IEEE International Conference on Robotics and Automation*, pp. 2230–2235, 2015.

[13] S. Oßwald, M. Bennewitz, W. Burgard, and C. Stachniss, "Speeding-up robot exploration by exploiting background information," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 716–723, 2016.

[14] A. J. Smith and G. A. Hollinger, "Distributed inference-based multi-robot exploration," *Autonomous Robots*, vol. 42, no. 8, pp. 1651–1668, 2018.

[15] D. Zhu, T. Li, D. Ho, C. Wang, and M. Q.-H. Meng, "Deep reinforcement learning supervised autonomous exploration in office environments," in *IEEE International Conference on Robotics and Automation*, pp. 7548–7555, 2018.

[16] O. Zhelo, J. Zhang, L. Tai, M. Liu, and W. Burgard, "Curiosity-driven exploration for mapless navigation with deep reinforcement learning," *arXiv preprint arXiv:1804.00456*, 2018.

[17] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 116–121, 1985.

[18] T. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Communications of the ACM*, vol. 27, pp. 236–239, 1984.

[19] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on Octrees," *Autonomous Robots*, vol. 34, pp. 189–206, 2013.

[20] C. E. Aguero, N. Koenig, I. Chen, H. Boyer, S. Peters, J. Hsu, B. Gerkey, S. Paepcke, J. L. Rivero, J. Manzo, E. Krotkov, and G. Pratt, "Inside the Virtual Robotics Challenge: Simulating Real-Time Robotic Disaster Response," *IEEE Transactions on Automation Science and Engineering*, vol. 12, pp. 494–506, April 2015.

[21] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with Rao-Blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.