2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Online On-Demand Multi-Robot Coverage Path Planning

Ratijit Mitra[1] and Indranil Saha[2]

***Abstract*— We present an online centralized path planning algorithm to cover a large, complex, unknown workspace with multiple homogeneous mobile robots. Our algorithm is horizon-based, synchronous, and on-demand. The recently proposed horizon-based synchronous algorithms compute all the robots' paths in each horizon, significantly increasing the computation burden in large workspaces with many robots. As a remedy, we propose an algorithm that computes the paths for a subset of robots that have traversed previously computed paths entirely (thus on-demand) and reuses the remaining paths for the other robots. We formally prove that the algorithm guarantees complete coverage of the unknown workspace. Experimental results on several standard benchmark workspaces show that our algorithm scales to hundreds of robots in large complex workspaces and consistently beats a state-of-the-art online centralized multi-robot coverage path planning algorithm in terms of the time needed to achieve complete coverage. For its validation, we perform ROS+Gazebo simulations in five 2D grid benchmark workspaces with 10 Quadcopters and 10 TurtleBots, respectively. Also, to demonstrate its practical feasibility, we conduct one indoor experiment with two real TurtleBot2 robots and one outdoor experiment with three real Quadcopters.**

I. Introduction

Coverage Path Planning (CPP) deals with finding conflict-free routes for a fleet of robots to make them completely visit the obstacle-free regions of a given workspace to accomplish some designated task. It has numerous applications in indoor environments, e.g., vacuum cleaning [1], [2], industrial inspection [3], etc., as well as in outdoor environments, e.g., precision farming [4], surveying [5], search and rescue operations [6], etc. A CPP algorithm, often called a Coverage Planner (CP), is said to be *complete* if it guarantees coverage of the entire obstacle-free region. Though a *single* robot is enough to achieve complete coverage of a small environment (e.g., [7], [8], [9], [10], [11]), *multiple* robots (e.g., [12], [13], [14], [2], [15], [16], [17]) facilitate complete coverage of a large environment more quickly. However, the design complexity of the CP grows significantly to exploit the benefit of having multiple robots.

For many CPP applications, the workspace's obstacle map is unknown initially. So, the *offline* CPs (e.g., [8], [1]) that require the obstacle map beforehand are not applicable here. Instead, we need an *online* CP (e.g., [18], [19], [20], [21], [15]) that runs through multiple rounds to cover the entire workspace gradually. In each round, the robots explore some unexplored regions using attached sensors, and the CP subsequently finds their subpaths to cover the explored obstacle-free regions not covered so far, a.k.a. *goals*.

[1]Ratijit Mitra and [2]Indranil Saha are with the Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, Uttar Pradesh - 208016, India {ratijit,isaha}@iitk.ac.in

Based on where the CP runs, we can classify it as either *centralized* or *distributed*. A centralized CP (e.g., [22], [18], [19], [8], [20], [9], [23]) runs at a server and is responsible for finding all the paths alone. In contrast, a distributed CP (e.g., [24], [25]) runs at every robot as a local instance, and these local instances *collaborate* among themselves to find individual paths. The distributed CPs are computationally faster as they deal with only the local state spaces. However, they find highly inefficient paths due to the lack of global knowledge about the state space. Despite having high computation time, the centralized CPs can provide a shorter coverage completion time as they can find highly efficient paths by exploiting the global state space. The recently proposed *receding horizon*-based (e.g., [19], [20]) online multi-robot centralized coverage planner `GAMRCPP` [23] demonstrates this capability, thereby outperforming the state-of-the-art online multi-robot distributed coverage planner `BoB` [25].

A major bottleneck of `GAMRCPP` is that it generates the paths for *all* the robots *synchronously* (i.e., at the same time) in each *horizon* (like [20]), which prevents it from scaling for hundreds of robots in large workspaces. Furthermore, `GAMRCPP` decides the *horizon length* based on the *minimum* path length and *discards* the remaining paths for the robots with longer paths. Finding a better goal assignment for those robots in the next horizon is possible, but discarding the already generated paths leads to considerable computational wastage. In this paper, we propose an alternative approach where, like `GAMRCPP`, the CP decides the horizon length based on the minimum path length but *keeps* the remaining paths for the robots with longer paths for traversal in the subsequent horizons. Thus, in a horizon, our proposed CP has to synchronously generate the paths for only those robots for which no remaining path is available (called the *participant robots*), hence *on-demand*. However, this new on-demand approach brings the additional challenge of computing the new paths under the constraint of the remaining paths for some robots. Our CP tackles this challenge soundly. Though the proposed approach misses the opportunity to find more optimal paths for the *non-participant robots* in a horizon, the computation load in each horizon decreases significantly, which in turn leads to a faster coverage completion of large workspaces with hundreds of robots, promising *scalability*.

We formally prove that the proposed CP can achieve *complete coverage*. To evaluate its performance, we consider eight large 2D grid-based benchmark workspaces of varying size and obstacle density, and two types of robots, TurtleBot, which is a ground robot, and a Quadcopter, which is an aerial robot, for their coverage. We vary the number of robots from 128 to 512 and choose *mission time* as the comparison metric, which is the time required to attain complete coverage.

979-8-3503-8457-4/24/$31.00 ©2024 IEEE

We compare our proposed CP with GAMRCPP and show that it outperforms GAMRCPP consistently in large workspaces involving hundreds of robots. We further demonstrate the practical feasibility of our algorithm through ROS+Gazebo simulations and real experiments.

II. Problem

A. Preliminaries

Let $\mathbb{R}$ and $\mathbb{N}$ denote the set of real numbers and natural numbers, respectively, and $\mathbb{N}_0$ denote the set $\mathbb{N} \cup \{0\}$. Also, for $m \in \mathbb{N}$, we write $[m]$ to denote the set $\{n \in \mathbb{N} \mid n \leq m\}$, and $[m]_0$ to denote the set $[m] \cup \{0\}$. The size of the countable set $\mathcal{S}$ is denoted by $|\mathcal{S}| \in \mathbb{N}_0$. Furthermore, we denote the set $\{0, 1\}$ of Boolean values by $\mathbb{B}$.

1) Workspace: We consider an unknown 2D workspace W represented as a grid of size $X \times Y$, where $X, Y \in \mathbb{N}$. Thus, W is represented as a set of *non-overlapping square-shaped grid cells* $\{(x, y) \mid x \in [X] \wedge y \in [Y]\}$, some of which are *obstacle-free* (denoted by W_{free}) and *traversable* by the robots, while the rest are *static obstacle-occupied* (denoted by W_{obs}), which are not. Note that W_{free} and W_{obs} are not known initially. We assume that W_{free} is *strongly connected* and W_{obs} is *fully* occupied with obstacles.

2) Robots and their States: We employ a team of $R \in \mathbb{N}$ *failure-free homogeneous mobile* robots, where each robot fits entirely within a cell. We denote the $i(\in [R])$-th robot by r^i and assume that r^i is *location-aware*. Let the state of r^i at the $j(\in \mathbb{N}_0)$-th *discrete* time step be s^i_j, which is a tuple of its *location* and possibly *orientation* in the workspace. We define a function $\mathcal{L}$ that takes a state s^i_j as input and returns the corresponding location as a tuple. Initially, the robots get deployed at different obstacle-free cells, comprising *the set of start states* $S = \{s^i_0 \mid i \in [R] \wedge \mathcal{L}(s^i_0) \in W_{free} \wedge \forall j \in [R] \setminus \{i\}.\ \mathcal{L}(s^j_0) \neq \mathcal{L}(s^i_0)\}$. We also assume that each r^i is equipped with four *rangefinders* on all four sides to detect obstacles in the four neighboring cells they are facing.

3) Motions and Paths of the robots: The robots have a common *set of motion primitives* M to change their states in the next time step. It also contains a unique motion primitive Halt(H) to keep the state of a robot unchanged in the next step. Each motion primitive $\mu \in M$ is associated with some cost $\texttt{cost}(\mu) \in \mathbb{R}$, e.g., distance traversed, energy consumed, etc. We assume that all the motion primitives take the same $\tau \in \mathbb{R}$ unit time for execution. Initially, the path π^i for robot r^i contains its start state s^i_0. So, the *length* of π^i, denoted by $\Lambda \in \mathbb{N}_0$, is 0. But, when a *finite sequence* of motion primitives $(\mu_j \in M)_{j \in [\Lambda]}$ of length $\Lambda > 0$ gets applied to s^i_0, it results in generating the Λ-length path π^i. So, π^i is a finite sequence $(s^i_j)_{j \in [\Lambda]_0}$ of length $\Lambda + 1$ of the states of r^i:

$$s^i_{j-1} \xrightarrow{\mu_j} s^i_j,\ \forall j \in [\Lambda].$$

Thus, π^i has the cost $\texttt{cost}(\pi^i) = \sum_{j \in [\Lambda]} \texttt{cost}(\mu_j)$. Note that we can make *the set of paths* $\Pi = \{\pi^i | i \in [R]\}$ of all the robots *equal-length* Λ by suitably applying H at their ends.

Example 1: A TurtleBot [26] not only *drives* forward to change its location but also *rotates* around its axis to change its orientation. So, the state of a TurtleBot is $s^i_j = (x, y, \theta)$, where $(x, y) \in W$ is its location in the workspace and $\theta \in \{\texttt{East(E)}, \texttt{North(N)}, \texttt{West(W)}, \texttt{South(S)}\}$ is its orientation at that location. The set of motion primitives is given by $M = \{\texttt{Halt(H)}, \texttt{TurnRight(TR)}, \texttt{TurnLeft(TL)}, \texttt{MoveNext(MN)}\}$, where TR turns the TurtleBot 90° *clockwise*, TL turns the TurtleBot 90° *counterclockwise*, and MN *moves* the TurtleBot to the next cell pointed by its orientation θ.

B. Problem Definition

We now formally define the CPP problem below.

Problem 1 (Complete Coverage Path Planning): Given an unknown workspace W, start states S of R robots, and their motion primitives M, find paths Π of equal-length Λ for the robots such that the following two conditions hold:

Cond. i: Each path π^i must satisfy the following:

1) $\forall j \in [\Lambda]_0\ \mathcal{L}(s^i_j) \in W_{free}$, [*Avoid obstacles*]
2) $\forall j \in [\Lambda]_0\ \forall k \in [R] \setminus \{i\}$ $\mathcal{L}(s^i_j) \neq \mathcal{L}(s^k_j)$, [*Avert same cell collisions*]
3) $\forall j \in [\Lambda]\ \forall k \in [R] \setminus \{i\}$ $\neg((\mathcal{L}(s^i_{j-1}) = \mathcal{L}(s^k_j)) \wedge (\mathcal{L}(s^i_j) = \mathcal{L}(s^k_{j-1})))$. [*Avert head-on collisions*]

Cond. ii: Each obstacle-free cell must get visited by at least one robot, i.e., $\bigcup_{i \in [R]} \bigcup_{j \in [\Lambda]_0} \{\mathcal{L}(s^i_j)\} = W_{free}$.

III. On-Demand CPP Framework

This section presents the proposed centralized horizon-based online multi-robot on-demand CPP approach for complete coverage of an unknown workspace whose size and boundary are only known to the CP and the mobile robots.

Due to the limited range of the fitted rangefinders, each robot gets a *partial* view of the unknown workspace, called the *local view*. Initially, all the robots share their initial local views with the CP by sending *requests* for paths. The CP then fuses these local views to get the *global view* of the workspace. Based on the global view, it attempts to generate collision-free paths for the robots, possibly of different lengths. The robots with *non-zero-length* paths are said to be *active*, while the rest are *inactive*. Next, the CP determines the horizon length as the minimum path length of the active robots. Subsequently, it makes the paths of the active robots of length equal to that horizon length. If any active robot's path length exceeds the horizon length, the CP stores the remaining part for future horizons. Finally, the CP provides these equal-length paths to respective active robots by sending *responses*. As the CP has failed to find paths for the inactive robots in the current horizon, it considers them again in the next horizon. However, the active robots follow their received paths synchronously and update their local views accordingly. Upon finishing the execution of their current paths, they share their updated local views with the CP again. In the next horizon, after updating the global view, the CP attempts to generate paths for the previous horizon's inactive robots and active robots who have completed following their last planned paths. In other words, the CP generates paths for only those robots with no remaining path, called the *participants*. Note that the robots with remaining paths computed in the previous horizon become the *non-participants*

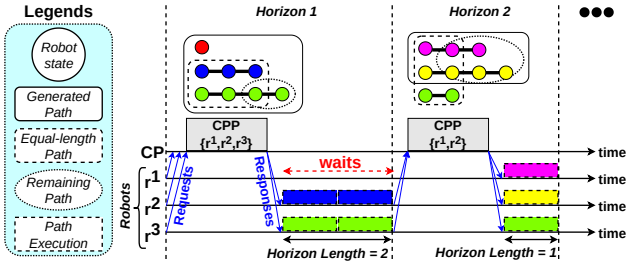


Fig. 1: Overview of on-demand CPP

(inherently active) in the current horizon. Moreover, the CP does not alter the non-participants' remaining paths while generating the participants' paths in the current horizon.

Example 2: In Figure 1, we show a schematic diagram of the on-demand CPP approach for three robots in an arbitrary workspace. In horizon 1, the CP finds paths of lengths $0, 2$, and 3 for the participants r^1, r^2, and r^3, respectively. Observe that only r^2 and r^3 are active. As the horizon length turns out to be 2, the CP stores the remaining part of r^3's path. Now, r^2 and r^3 follow their equal-length paths while r^1 waits. In horizon 2, however, the CP finds paths only for r^1 and r^2 (notice the new colors) as non-participant r^3 already has its remaining path (notice the same color) found in horizon 1.

A. Data structures at CP

Before delving deep into the processes that run at each robot r^i (Algorithm 1) and the CP (Algorithm 2), we define the data structures that the CP uses for computation.

Let Σ_{rem} denote the set $\{\sigma^i_{rem} \mid i \in [R]\}$ of the remaining paths of all the robots for the next horizon, where σ^i_{rem} denotes the remaining path for r^i. Initially, none of the robots has any remaining path, i.e., σ^i_{rem} is NULL for each r^i. We denote the set of indices of the robots participating in the current horizon by $I_{par} \subseteq [R]$ and the number of requests received in the current horizon by $N_{req} \in [R]_0$. Also, $N_{act} \in [R]_0$ denotes the number of active robots in the current horizon. By I^{inact}_{par}, we denote the set of indices of the inactive robots of the previous horizon, and by S^{inact}, their start states. So, the CP must incorporate the robots I^{inact}_{par} into the planning of the current horizon. Finally, $\widetilde{W_g}$ denotes the goals at which the remaining paths end. So, $\widetilde{W_g}$ remains reserved for the corresponding robots, and the CP cannot assign $\widetilde{W_g}$ to any other robot hereafter.

B. Overall On-Demand Coverage Path Planning

In this section, we present the overall on-demand CPP approach in detail. First, we explain Algorithm 1, which runs at the robots' end. Each robot r^i initializes its local view $W^i = \langle W^i_u, W^i_o, W^i_g, W^i_c \rangle$ (line 1), where W^i_u, W^i_o, W^i_g, and W^i_c are the set of unexplored, obstacle, goal, and covered cells, respectively, due to the partial visibility of the workspace. Subsequently, r^i creates a request message $\texttt{M}_{\texttt{req}}[\texttt{id}, \texttt{state}, \texttt{view}]$ (line 2) and sends that to the CP (line 4) to inform about its local view. Then, r^i waits for a response message $\texttt{M}_{\texttt{res}}[\texttt{path}]$ (line 5) from the CP. Upon receiving, r^i extracts its path into σ^i and starts to follow it (line 6). While following, r^i explores newly visible cells and accordingly updates W^i. Finally, once it reaches the last

Algorithm 1: $\texttt{Robot}(i, s^i_0, X, Y)$

```
1 W^i ← init_localview(s^i_0, X, Y)
2 M_req[id, state, view] ← [i, s^i_0, W^i]        // Create M_req
3 while true do
4     send_localview(M_req)
5     σ^i ← receive_path(M_res)
6     W^i ← follow_path(σ^i)
7     M_req[state, view] ← [s^i_|σ^i|, W^i]       // Update M_req
```

state $s^i_{|\sigma^i|}$ of σ^i, it updates its $\texttt{M}_{\texttt{req}}$ (line 7) to inform the CP about its updated W^i. The same communication cycle continues between r^i and the CP in a **while** loop (lines 3-7). Here, we assume that the *communication medium* is *reliable* and each communication takes negligible time.

We now describe Algorithm 2 in detail, which runs at the CP's end. After initializing the required data structures (lines 1-4), the CP starts a *service* (lines 5-18) to receive requests from the robots in a mutually exclusive manner. Upon receiving a $\texttt{M}_{\texttt{req}}$ from r^i (line 6), the CP checks whether r^i has any remaining path σ^i_{rem} or not (line 7). If r^i has, it does not participate in the planning of the current horizon. Otherwise (lines 7-9), r^i participates, and so the CP adds r^i's ID $\{i\}$ into I_{par}, and current state into S. In either case, the CP updates the global view of the workspace $W = \langle W_u, W_o, W_g, W_c \rangle$ *incrementally* (line 10). In the longer version of this paper [27], please refer to lines 19-32 of Algorithm 2 for the definition of the function `update_globalview`, the third paragraph of Section III-B for its description, and see Example 3 for an illustration.

Once the CP receives all the requests from the active robots of the previous horizon (lines 11-12), it makes the inactive robots of the previous horizon I^{inact}_{par} participants (lines 13-14). Then, it invokes `OnDemCPP_Hor` (line 15), which does the path planning for the current horizon (we describe in Section III-C). `OnDemCPP_Hor` updates I^{inact}_{par}, which now contains the inactive robots of the current horizon, and returns the equal-length paths Σ for the active robots $[R] \setminus I^{inact}_{par}$ of the current horizon. So, the CP simultaneously sends paths to those active robots through response messages $\texttt{M}_{\texttt{res}}$ (lines 16 and 19-22). Finally, it reinitializes some data structures for the next horizon (lines 17-18).

C. Coverage Path Planning in the Current Horizon

Here, we describe the rest of Algorithm 2. Recall that the CP has already assigned the goals $\widetilde{W_g}$ to the non-participants in some past horizons. In `OnDemCPP_Hor` (lines 23-36), first, the CP checks whether there are *unassigned* goals, i.e., $W_g \setminus \widetilde{W_g}$ left in the workspace (line 25). If so, it attempts to generate the collision-free paths $\Sigma' = \{\sigma'^i \mid i \in I_{par}\}$ for the participants I_{par} to visit some unassigned goals while keeping the remaining paths Σ_{rem} of the non-participants $[R] \setminus I_{par}$ (hereafter denoted by $\overline{I_{par}}$) **intact** (line 26). We defer the description of `CPPForPar` to Section III-D. Otherwise, the CP cannot plan for the participants. Thereby, it *skips* replanning the participants in the current horizon. Still, if there are non-participants (line 27), they can proceed toward their assigned goals in the current horizon while the participants remain inactive. To signify this, the CP

Algorithm 2: OnDemCPP(R, X, Y)

1 $\Sigma_{rem} \leftarrow \{\text{NULL} \mid i \in [R]\}$ // Init.
2 $I_{par} \leftarrow \emptyset$, $S \leftarrow \emptyset$, $I^{inact}_{par} \leftarrow \emptyset$, $S^{inact} \leftarrow \emptyset$, $W \leftarrow \emptyset$, $\widetilde{W}_g \leftarrow \emptyset$, $\Pi \leftarrow \emptyset$
3 $N_{req} \leftarrow 0$ // Number of requests
4 $N_{act} \leftarrow R$ // Number of active robots
5 **Service** receive_localview($\mathtt{M_{req}}$):
6 $i \leftarrow \mathtt{M_{req}.id}$ // Robot ID
7 **if** $\sigma^i_{rem} = \text{NULL}$ **then**
8 $I_{par} \leftarrow I_{par} \cup \{i\}$ // r^i is a participant
9 $S \leftarrow S \cup \{\mathtt{M_{req}.state}\}$
10 update_globalview($\mathtt{M_{req}.view}$)
11 $N_{req} \leftarrow N_{req} + 1$
12 **if** $(N_{req} = N_{act})$ **then**
13 $I_{par} \leftarrow I_{par} \cup I^{inact}_{par}$ // Formerly inactive
14 $S \leftarrow S \cup S^{inact}$
15 $\Sigma \leftarrow$ OnDemCPP_Hor () // Eq.-length paths
16 send_paths_to_active_robots (Σ)
17 $I_{par} \leftarrow \emptyset$, $S \leftarrow \emptyset$ // Reinit.
18 $N_{req} \leftarrow 0$
19 **Function** send_paths_to_active_robots (Σ):
20 **parallel for** $i \in [R] \setminus I^{inact}_{par}$ **do**
21 $\mathtt{M_{res}[path]} \leftarrow \sigma^i$ // Create $\mathtt{M_{res}}$
22 send_path ($\mathtt{M_{res}}$, i)
23 **Function** OnDemCPP_Hor ():
24 $\Sigma' \leftarrow \emptyset$ // Participants' coll.-free paths
25 **if** $(W_g \setminus \widetilde{W}_g) \neq \emptyset$ **then**
26 $\Sigma' \leftarrow \text{CPPForPar}(W, \widetilde{W}_g, I_{par}, S, M, R, \Sigma_{rem})$
27 **else if** $I_{par} \neq [R]$ **then**
28 **for** $i \in I_{par}$ **do**
29 $\sigma'^i \leftarrow s^i_0$
30 $\Sigma' \leftarrow \Sigma' \cup \{\sigma'^i\}$
31 **else**
32 **exit** () // Coverage Complete !!!
33 $\Sigma'_{all} \leftarrow$ combine_paths$(I_{par}, \Sigma', R, \Sigma_{rem})$
34 $\lambda \leftarrow$ determine_horizon_length(Σ'_{all})
35 $\Sigma \leftarrow$ get_equal_length_paths (Σ'_{all}, λ)
36 **return** Σ
37 **Function** get_equal_length_paths (Σ'_{all}, λ):
38 $\Sigma \leftarrow \emptyset$, $\Sigma_{rem} \leftarrow \emptyset$, $\widetilde{W}_g \leftarrow \emptyset$, $I^{inact}_{par} \leftarrow \emptyset$, $S^{inact} \leftarrow \emptyset$
39 $N_{act} \leftarrow 0$ // Reinit.
40 **for** $i \in [R]$ **do**
41 **if** $|\sigma'^i_{all}| > 0$ **then**
42 $N_{act} \leftarrow N_{act} + 1$ // r^i is active
43 $\langle \sigma^i, \sigma^i_{rem} \rangle \leftarrow$ split_path$(\sigma'^i_{all}, \lambda)$
44 $\Sigma \leftarrow \Sigma \cup \{\sigma^i\}$
45 **if** $|\sigma^i_{rem}| > 0$ **then**
46 $\widetilde{W}_g \leftarrow \widetilde{W}_g \cup \{\mathcal{L}(s^i_{|\sigma'^i_{all}|})\}$ // Reserve
47 **else**
48 $\sigma^i_{rem} \leftarrow \text{NULL}$
49 $\pi^i \leftarrow \pi^i : \sigma^i$ // Concat.
50 **else**
51 $I^{inact}_{par} \leftarrow I^{inact}_{par} \cup \{i\}$ // r^i is inactive
52 $S^{inact} \leftarrow S^{inact} \cup \{s^i_0\}$
53 $\sigma^i_{rem} \leftarrow \text{NULL}$
54 $\pi^i \leftarrow \pi^i :$ dummy_path(s^i_0, λ) // Concat.
55 $\Sigma_{rem} \leftarrow \Sigma_{rem} \cup \{\sigma^i_{rem}\}$
56 **return** Σ

assigns the current states of the participants to their paths Σ' (lines 28-30). Please see Example 4 in [27], where we show such a scenario. Note that when both the criteria, viz all the robots are participants, and there are no unassigned goals get fulfilled (lines 31-32), it means complete coverage (established in Theorem 1 in Section IV). Next, the CP combines Σ' with Σ_{rem} into $\Sigma'_{all} = \{\sigma'^i_{all} | i \in [R]\}$, where

$$\sigma'^i_{all} = \begin{cases} \sigma'^i, & \text{if } i \in I_{par}, \\ \sigma^i_{rem}, & \text{otherwise.} \end{cases}$$

Thus, σ'^i_{all} contains the collision-free path σ'^i if r^i is a participant, otherwise contains the remaining path σ^i_{rem} of the non-participant r^i. Notice that Σ'_{all} contains the collision-free paths for all the robots (line 33). A robot r^i is said to be active in the current horizon if its path length is non-zero, i.e., $|\sigma'^i_{all}| > 0$. Otherwise, r^i is said to be inactive. In the penultimate step, the CP determines the length of the current horizon (line 34) as the minimum path length of the active robots, i.e., $\lambda = \min_{i \in [R]} \{|\sigma'^i_{all}| > 0\}$. Finally, it returns λ-length paths Σ only for the active robots (lines 35-36). This way, the CP ensures that at least one active robot (participant or non-participant) reaches its goal in the current horizon.

In get_equal_length_paths (lines 37-56), after re-initialization of some of the data structures (lines 38-39), the CP examines the paths Σ'_{all} in a **for** loop (lines 40-55) to determine which robots are active (lines 41-49) and which are not (lines 50-54). If r^i is active, first, the CP increments N_{act} by 1 (line 42). Then, the CP splits its path σ'^i_{all} into two parts σ^i and σ^i_{rem} of lengths λ and $|\sigma'^i_{all}| - \lambda$, respectively (line 43). Active r^i traverses the former part σ^i, containing $s^i_0 \ldots s^i_\lambda$, in the current horizon (line 44) and the later part (if any), containing $s^i_\lambda \ldots s^i_{|\sigma'^i_{all}|}$, in the future horizons (line 55). If it has the remaining path σ^i_{rem}, its goal location $\mathcal{L}(s^i_{|\sigma'^i_{all}|})$ gets added to $\widetilde{W}_g$ (lines 45-46), signifying that this goal remains reserved for r^i. Otherwise, σ^i_{rem} is set to NULL (lines 47-48 and 55), indicating the absence of the remaining path for r^i, thereby enabling r^i to become a participant in the next horizon. In contrast, if r^i is inactive (lines 50-54), the CP adds r^i's ID $\{i\}$ and its start state s^i_0 into I^{inact}_{par} and S^{inact}, respectively (lines 51-52), as it would reattempt to find r^i's path in the next horizon. The CP also sets σ^i_{rem} to NULL (lines 53 and 55). Note that the CP also generates the full path for the active robots (line 49) and the inactive robots (line 54), where dummy_paths generates a path of length λ containing only the current state. The CP sends no path to an inactive robot in the current horizon (lines 20-22). So, the active robots of the current horizon send requests in the next horizon (lines 42 and 11-12).

D. Coverage Path Planning for the Participants

At last, we give the outline of CPPForPar (Algorithm 3), which attempts to generate collision-free paths only for the participants without modifying the remaining paths of the non-participants. The in-depth explanation is in [27]. It is a modified version of Algorithms 2 and 3 of [23] combined, which uses the idea of *Goal Assignment-based Prioritized Planning* to generate collision-free paths for all the robots in each horizon. Put differently, all the robots participate in each horizon of [23].

Algorithm 3: `CPPForPar`$(W, \widetilde{W_g}, I_{par}, S, M, R, \Sigma_{rem})$

Result: Collision-free paths for the participants (Σ')

1 $\langle \Gamma,\ \Phi \rangle \leftarrow$ `COPForPar`$(W, \widetilde{W_g}, I_{par}, S, M)$

2 $\Sigma' \leftarrow$ `CFPForPar`$(\Gamma, \Phi, I_{par}, R, \Sigma_{rem})$

1) Sum-of-Costs-optimal goal assignment and corresponding paths: Let R^* and G^* be the number of participants and unassigned goals in the current horizon, respectively. So, $R^* = |I_{par}| = |S|$ and $G^* = |W_g \setminus \widetilde{W_g}|$. The CP uses the Hungarian Algorithm [28] to assign the participants I_{par} to the unassigned goals having IDs $[G^*]$, providing which participant would go to which goal (line 1). Formally, the assignment array is $\Gamma : I_{par} \rightarrow [G^*] \cup \{\mathsf{NULL}\}$. For a participant r^i, where $i \in I_{par}$, its assigned goal $\Gamma[i]$ contains NULL if the CP fails to assign any goal to r^i, e.g., when $R^* > G^*$. Such a participant is said to be *inactive*, whose corresponding optimal path φ^i only contains its current state s_0^i. Otherwise, the participant is said to be *active*, whose corresponding optimal path φ^i leads r^i from its current state s_0^i to the assigned goal $\Gamma[i]$ using motions M. Such an optimal path φ^i passes through the cells $W_g \cup W_c$ but avoids the cells $W_u \cup W_o$. We denote the resultant optimal paths by $\Phi = \{\varphi^i \mid i \in I_{par}\}$. Keep in mind that the non-participants are inherently active because $\sigma_{rem}^j \neq \mathsf{NULL},\ \forall j \in \overline{I_{par}}$.

2) Collision-free paths: The participants' optimal paths Φ are not necessarily *inter-robot collision-free*. A participant r^i may collide with either another participant r^j or a non-participant r^k having remaining path σ_{rem}^k. Though the CP cannot change the remaining paths Σ_{rem} of the non-participants, it can change the paths Φ for the participants. To get rid of any collisions, the CP employs a two-step procedure. First, it *prioritizes* the participants *dynamically* based on their movement constraints in Φ. For example, if a participant r^i's start location is on the path φ^j of another participant r^j, then r^i must leave its start location before r^j reaches there. Similarly, if r^i's goal is on the path φ^j of r^j, then r^j must get past that goal before r^i reaches there. The non-participants have higher priorities than the participants because the CP cannot change their remaining paths. Finally, the CP *offsets* the participants' paths Φ in the order of their priority to ensure that a participant does not collide with its higher priority robots (some other participants and the non-participants). However, if a collision between a participant and a non-participant becomes inevitable (Example 5 in [27] shows such a scenario), the CP inactivates the participant and returns to the first step to recompute the priorities. Otherwise, the CP returns the participants' collision-free paths Σ'. Note that all the participants may get inactivated in the worst case during offsetting due to the non-participants.

IV. Theoretical Analysis

First, we formally prove that `OnDemCPP` covers W completely and then analyze its time complexity.

Theorem 1: `OnDemCPP` eventually terminates, and when it does, it ensures complete coverage of W.

Proof: Please refer to [27] for the proof. ■

Theorem 2: `OnDemCPP`'s time complexity is $\mathcal{O}(|W|^4)$.

Proof: Please refer to [27] for the proof. ■

V. Evaluation

A. Implementation and Experimental Setup

We implement `Robot` (Algorithm 1) and `OnDemCPP` (Algorithm 2) in a common ROS package[1] and run it in a computer having Intel® Core™ i7-4770 CPU @ 3.40 GHz and 16 GB of RAM. For experimentation, we consider eight large 2D grid benchmark workspaces from [29] of varying size and obstacle density. In each workspace, we *incrementally* deploy $R \in \{128, 256, 512\}$ robots and repeat each experiment 10 times with different initial deployments of the robots to report their mean in the performance metrics (standard deviations are available in [27]).

1) Robots for evaluation: We consider both TurtleBot (introduced before) and Quadcopter in the experimentation. A Quadcopter's state $s_j^i = (x, y) \in W$ is its location in the workspace. Its set of motion primitives M = {`Halt`(`H`), `MoveEast`(`ME`), `MoveNorth`(`MN`), `MoveWest`(`MW`), `MoveSouth`(`MS`)}, where `ME`, `MN`, `MW`, and `MS` move it to the immediate *east, north, west*, and *south* cell, respectively.

2) Evaluation metrics: We compare `OnDemCPP` with `GAMRCPP` [23] in terms of *the mission time* (T_m), which is the sum of *the total computation time* (T_c) and *the total path execution time* (T_p) while ignoring *the total communication time* between the CP and the robots. *Total computation time* (T_c) is the sum of the computation times spent on `OnDemCPP_Hor` across all horizons. *Total path execution time* (T_p) can be expressed as $T_p = \Lambda \times \tau$, where Λ is *the total horizon length* (i.e., the sum of the horizon lengths). We can also express T_p as the sum of T_{Halt} and $T_{non-Halt}$, where T_{Halt} is the duration for which the robots remain stationary while following respective paths, i.e., when they execute `Halt`(`H`) moves, and $T_{non-Halt}$ is the duration for which the robots move, i.e., when they do not.

In our evaluation, we take the path cost as the number of moves the corresponding robot performs, where each move takes $\tau = 1\text{s}$. This is realistic because the maximum translational velocity of a TurtleBot2 is 0.65m/s [26], and that of a Quadcopter is $\sim 16\text{m/s}$ [30]. Thus, to ensure $\tau = 1\text{s}$, we can keep the grid cell size for a TurtleBot2 up to 0.65m, which is sufficient considering its size. Similarly, a grid cell size of up to 16m is sufficient for most practical applications involving Quadcopters. We demonstrate this in the real experiments described in Section V-C.

B. Results and Analysis

We show the experimental results in Table I, where we list workspaces in increasing order of their obstacle-free cell count $|W_{free}|$ for each type of robot. In the table, R^* denotes the average number of participants over all horizons.

1) Comparison of Mission time: The total computation time T_c increases with the number of robots R because more robots become participants in a horizon. So, assigning R^* participants to G^* unassigned goals and getting their collision-free paths Σ' becomes computationally intensive. Notice that T_c also increases with the workspace size, specifically with $|W_{free}|$ as the *state-space* increases. Unlike

[1] https://github.com/iitkcpslab/OnDemCPP

TABLE I: Experimental results

M	*Workspace*		R	R^*	T_c (s)		T_p (s)		T_{Halt} (s)		$T_{non-Halt}$ (s)		T_m (s)		**Mission**
					GAMRCPP	OnDemCPP	GAMRCPP	OnDemCPP	GAMRCPP	OnDemCPP	GAMRCPP	OnDemCPP	GAMRCPP	OnDemCPP	**speed up**
Quadcopter	w_woundedcoast 578 × 642 (34,020)		128	80.4	619.5	362.2	696.2	1042.0	230.8	487.4	465.4	554.5	1315.7	1404.2	0.9
			256	170.7	1091.3	521.7	416.8	804.2	182.3	512.2	234.5	291.9	1508.1	1325.9	**1.1**
			512	361.5	1815.2	569.1	231.8	557.2	115.7	404.1	116.1	153.0	2047.0	1126.3	**1.8**
	Paris_1_256 256 × 256 (47,240)		128	81.3	513.5	270.4	815.2	962.6	214.9	298.2	600.3	664.3	1328.7	1233.0	**1.1**
			256	157.2	1461.9	537.2	508.0	656.2	184.2	273.0	323.8	383.1	1969.9	1193.4	**1.7**
			512	335.6	3338.8	894.0	301.1	455.7	137.6	247.1	163.5	208.5	3639.9	1349.7	**2.7**
	Berlin_1_256 256 × 256 (47,540)		128	83.2	489.3	238.4	752.7	898.2	169.9	272.9	582.8	625.2	1242.0	1136.6	**1.1**
			256	167.4	1323.9	515.2	564.6	669.6	258.0	301.0	306.6	368.5	1888.5	1184.8	**1.6**
			512	365.2	3196.5	856.0	434.5	524.9	275.8	324.8	158.7	200.0	3631.0	1380.9	**2.6**
	Boston_0_256 256 × 256 (47,768)		128	79.6	509.5	250.6	773.6	928.0	169.4	262.6	604.2	665.3	1283.1	1178.6	**1.1**
			256	157.8	1220.1	455.7	460.3	625.9	142.5	252.0	317.8	373.8	1680.4	1081.6	**1.6**
			512	345.3	2454.7	628.7	252.9	428.7	95.6	236.7	157.3	191.9	2707.6	1057.4	**2.6**
TurtleBot	maze-128-128-2 128 × 128 (10,858)		128	75.5	90.8	49.5	451.9	791.5	217.1	509.8	234.8	281.6	542.7	841.0	0.6
			256	174.7	193.6	68.1	218.6	417.6	115.7	294.8	102.9	122.7	412.2	485.7	0.8
			512	378.4	411.1	124.1	125.9	224.3	79.4	165.9	46.5	58.3	537.0	348.4	**1.5**
	den520d 257 × 256 (28,178)		128	71.9	341.8	147.6	556.4	773.1	151.6	286.1	404.8	486.9	898.2	920.7	**1.0**
			256	153.8	685.4	253.9	324.2	506.4	118.5	247.1	205.7	259.3	1009.6	760.3	**1.3**
			512	331.1	1292.4	316.0	188.7	398.6	82.8	252.9	105.9	145.6	1481.1	714.6	**2.1**
	warehouse-20-40-10-2-2 164 × 340 (38,756)		128	81.2	467.1	211.3	595.6	707.6	115.2	187.2	480.4	520.3	1062.7	918.9	**1.2**
			256	149.3	1009.5	336.0	361.6	502.5	102.2	182.8	259.4	319.6	1371.1	838.5	**1.6**
			512	335.5	1646.2	394.7	208.9	375.2	75.5	204.8	133.4	170.3	1855.1	769.9	**2.4**
	brc202d 481 × 530 (43,151)		128	65.1	1019.0	395.2	915.9	1444.9	242.5	587.4	673.4	857.4	1934.9	1840.1	**1.1**
			256	142.0	1880.7	697.7	512.5	972.5	177.9	510.9	334.6	461.5	2393.2	1670.2	**1.4**
			512	309.6	3026.8	705.6	302.6	780.9	133.9	514.6	168.7	266.2	3329.4	1486.5	**2.2**

GAMRCPP, where all R robots participate in replanning, OnDemCPP replans with $R^* \leq R$ robots. It results in substantially smaller T_c in OnDemCPP compared to GAMRCPP.

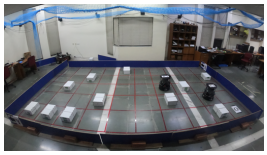
Next, the total horizon length Λ decreases with R as the deployment of more robots expedites the coverage. But, it increases with the workspace size since more W_{free} must be covered. During replanning, GAMRCPP finds paths for R robots without constraint. But, OnDemCPP replans for R^* participants while respecting the non-participants' constraint Σ_{rem}. As a result, there is an increase in the participants' collision-free path lengths, increasing individual horizon length λ and so Λ. Thus, GAMRCPP yields better T_p compared to OnDemCPP. For more insights on T_p, please refer to Section V-B.1 in [27].

In summary, OnDemCPP outperforms GAMRCPP in terms of T_c but underperforms in terms of T_p. As R or the workspace size increases, OnDemCPP's gain in terms of T_c surpasses its loss in terms of T_p by a noticeable margin. So, OnDemCPP beats GAMRCPP w.r.t. the mission time T_m.

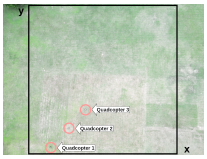
2) Implication on Energy Consumption: A ground robot like TurtleBot consumes energy only for $T_{non-Halt}$ duration. Table I shows that $T_{non-Halt}$ is $7\% - 57\%$ more for OnDemCPP than for GAMRCPP. Thus, the energy consumption for the ground robots for OnDemCPP is also proportionately more than that for GAMRCPP.

The situation is quite different for aerial robots like quadcopters. Once the mission starts, a quadcopter either keeps on *hovering* (during T_c and T_{Halt}) or makes translational moves (during $T_{non-Halt}$). Thus, a quadcopter consumes energy throughout the mission, i.e., during T_m. As OnDemCPP significantly reduces T_m for hundreds of robots compared to GAMRCPP, it also helps reduce power consumption in the quadcopters during a mission.

Given that reducing energy consumption during a mission is more crucial for aerial robots than for ground robots and that OnDemCPP significantly reduces T_m for both types of robots, OnDemCPP establishes itself to be superior to GAMRCPP for hundreds of robots.



(a) 5×10 indoor workspace [cell size = 0.61m]



(b) 10×10 outdoor workspace [cell size = 5m]

Fig. 2: Workspaces for the real experiments

3) Limitation of OnDemCPP: The results obtained from the *maze-128-128-2* workspace for $R \in \{128,\ 256\}$ show the limitation of OnDemCPP. In a *cluttered* workspace with narrow passageways, the flexibility of GAMRCPP allows it to find a more efficient Σ', thereby outmatching OnDemCPP in instances with a smaller R.

C. Simulations and Real Experiments

For validation, we perform Gazebo simulations in five 2D grid benchmark workspaces from [29] with 10 Quadcopters and 10 TurtleBots, respectively. We also perform two real experiments - one indoor with two TurtleBot2s, each fitted with four HC SR04 Ultrasonic Sound Sensors for obstacle detection and using Vicon[31] for localization, and one outdoor with three Quadcopters, each fitted with one Cube Orange autopilot, one Herelink Air Unit for communication with the remote controller, and one Here3 GPS for localization. Figure 2 shows the workspaces for these real experiments. The video containing the real experiments is available at https://youtu.be/5nhysTTp2Fw.

VI. Conclusion

We have proposed a centralized online on-demand CPP algorithm that uses a goal assignment-based prioritized planning method at its core. Our CP guarantees complete coverage of an unknown workspace with multiple homogeneous failure-free robots. Experimental results demonstrate its superiority over its counterpart by decreasing the mission time significantly in large workspaces with hundreds of robots. In the future, we will extend our work to an approach dealing with simultaneous path planning and plan execution.

REFERENCES

[1] R. Bormann, F. Jordan, J. Hampp, and M. Hägele, "Indoor coverage path planning: Survey, implementation, analysis," in *ICRA*, 2018, pp. 1718–1725.

[2] I. Vandermeulen, R. Groß, and A. Kolling, "Turn-minimizing multi-robot coverage," in *ICRA*, 2019, pp. 1014–1020.

[3] W. Jing, J. Polden, C. F. Goh, M. Rajaraman, W. Lin, and K. Shimada, "Sampling-based coverage motion planning for industrial inspection application with redundant robotic system," in *IROS*, 2017, pp. 5211–5218.

[4] A. Barrientos, J. Colorado, J. del Cerro, A. Martinez, C. Rossi, D. Sanz, and J. Valente, "Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots," *J. Field Robotics*, vol. 28, no. 5, pp. 667–689, 2011.

[5] N. Karapetyan, J. Moulton, J. S. Lewis, A. Q. Li, J. M. O'Kane, and I. M. Rekleitis, "Multi-robot dubins coverage with autonomous surface vehicles," in *ICRA*, 2018, pp. 2373–2379.

[6] T. M. Cabreira, L. B. Brisolara, and P. R. Ferreira Jr, "Survey on coverage path planning with unmanned aerial vehicles," *Drones*, vol. 3, no. 1, p. 4, 2019.

[7] Y. Bouzid, Y. Bestaoui, and H. Siguerdidjane, "Quadrotor-uav optimal coverage path planning in cluttered environment with a limited onboard energy," in *IROS*, 2017, pp. 979–984.

[8] A. Kleiner, R. Baravalle, A. Kolling, P. Pilotti, and M. Munich, "A solution to room-by-room coverage for autonomous cleaning robots," in *IROS*, 2017, pp. 5346–5352.

[9] G. Sharma, A. Dutta, and J. Kim, "Optimal online coverage path planning with energy constraints," in *AAMAS*, 2019, pp. 1189–1197.

[10] X. Chen, T. M. Tucker, T. R. Kurfess, and R. W. Vuduc, "Adaptive deep path: Efficient coverage of a known environment under various configurations," in *IROS*, 2019, pp. 3549–3556.

[11] M. Dharmadhikari, T. Dang, L. Solanka, J. Loje, H. Nguyen, N. Khedekar, and K. Alexis, "Motion primitives-based path planning for fast and agile exploration using aerial robots," in *ICRA*, 2020, pp. 179–185.

[12] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics Auton. Syst.*, vol. 61, no. 12, pp. 1258–1276, 2013.

[13] J. Modares, F. Ghanei, N. Mastronarde, and K. Dantu, "UB-ANC planner: Energy efficient coverage path planning with multiple drones," in *ICRA*, 2017, pp. 6182–6189.

[14] N. Karapetyan, K. Benson, C. McKinney, P. Taslakian, and I. M. Rekleitis, "Efficient multi-robot coverage of a known environment," in *IROS*, 2017, pp. 1846–1852.

[15] G. Hardouin, J. Moras, F. Morbidi, J. Marzat, and E. M. Mouaddib, "Next-best-view planning for surface reconstruction of large-scale 3D environments with multiple uavs," in *IROS*, 2020, pp. 1567–1574.

[16] J. Tang, C. Sun, and X. Zhang, "Mstc*: Multi-robot coverage path planning under physical constrain," in *ICRA*, 2021, pp. 2518–2524.

[17] L. Collins, P. Ghassemi, E. T. Esfahani, D. S. Doermann, K. Dantu, and S. Chowdhury, "Scalable coverage path planning of multi-robot teams for monitoring non-convex areas," in *ICRA*, 2021, pp. 7393–7399.

[18] B. Yamauchi, "Frontier-based exploration using multiple robots," in *AGENTS*, K. P. Sycara and M. J. Wooldridge, Eds., 1998, pp. 47–53.

[19] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding horizon "next-best-view" planner for 3D exploration," in *ICRA*, 2016, pp. 1462–1468.

[20] S. N. Das and I. Saha, "Rhocop: receding horizon multi-robot coverage," in *ICCPS*, 2018, pp. 174–185.

[21] A. Özdemir, M. Gauci, A. Kolling, M. D. Hall, and R. Groß, "Spatial coverage without computation," in *ICRA*, 2019, pp. 9674–9680.

[22] Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot," in *ICRA*, 2001, pp. 1927–1933.

[23] R. Mitra and I. Saha, "Scalable online coverage path planning for multi-robot systems," in *IROS*, 2022, pp. 10 102–10 109.

[24] N. Hazon, F. Mieli, and G. A. Kaminka, "Towards robust on-line multi-robot coverage," in *ICRA*, 2006, pp. 1710–1715.

[25] H. H. Viet, V. Dang, S. Y. Choi, and T. Chung, "BoB: an online coverage approach for multi-robot systems," *Appl. Intell.*, vol. 42, no. 2, pp. 157–173, 2015.

[26] TurtleBot 2. [Online]. Available: https://clearpathrobotics.com/turtlebot-2-open-source-robot/

[27] R. Mitra and I. Saha, "Online on-demand multi-robot coverage path planning," *CoRR*, vol. abs/2303.00047, 2023. [Online]. Available: https://arxiv.org/abs/2303.00047

[28] H. W. Kuhn, "The hungarian method for the assignment problem," *Nav. Res. Logist.*, vol. 2, no. 1-2, pp. 83–97, 1955.

[29] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, R. Barták, and E. Boyarski, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *SOCS*, 2019, pp. 151–159.

[30] DJI Consumer Drones Comparison. [Online]. Available: https://www.dji.com/global/products/comparison-consumer-drones

[31] Vicon Motion Capture Systems. [Online]. Available: https://www.vicon.com/