Dec-PPCPP: A Decentralized Predator–Prey-based Approach to Adaptive Coverage Path Planning Amid Moving Obstacles

Mahdi Hassan, Daut Mustafic, Dikai Liu

***Abstract*— Enabling multiple robots to collaboratively perform coverage path planning on complex surfaces embedded in $\mathbb{R}^3$ with the presence of moving obstacles is a challenging problem that has not received much attention from researchers. As robots start to be practically deployed, it is becoming important to address this problem. A novel decentralized multi-robot coverage path planning approach is proposed that is adaptive to unexpected stationary and moving obstacles while aiming to achieve complete coverage with minimal cost. The approach is inspired by the predator-prey relation. For a robot (a prey), a virtual stationary predator enforces spatial ordering on the prey, and dynamic predators (other robots) cause the prey to be repelled resulting in better task allocation and collision-avoidance. The approach makes the best use of both worlds: offline global planning for tuning of model parameters based on a prior map of the surface, and real-time local planning for adaptive and swift decision making amid moving obstacles and other robots while preserving global behavior. Comparisons with other approaches and extensive testing and validation using different number of robots, different surfaces and obstacles, and various scenarios are conducted.**

I. INTRODUCTION

Coverage Path Planning (CPP) is essential for robotic tasks such as surface cleaning, painting and abrasive blasting [1]. Research work has largely focused on planar surfaces (e.g., for floor cleaning robots) [2], [3], [4]. There are numerous works on multi-robot coverage [5], [6], [7]; however, very few of them consider actual moving obstacles (not just considering other robots as moving obstacles) [8]. There have also been many CPP algorithms applicable to 3D coverage [9], [10], [11], few of which consider multiple robots [12], but moving obstacles are typically not considered. To the best of the authors' knowledge, no multi-robot CPP algorithm is validated or shown to be applicable to both planar surfaces and surfaces embedded in $\mathbb{R}^3$ with adaptive capability relative to unexpected moving or stationary obstacles.

As an example scenario, consider an inspection crawler robot [13] that is capable of traversing complex and three-dimensional (3D) structures (e.g. ship-hulls, reservoir tanks and buildings). Suppose that the robot is tasked with a coverage task such as the full surface cleaning. Moving obstacles, e.g. humans performing various operations on the surfaces or gondola-like platforms transporting operators and equipment, may unexpectedly become present in the environment. If multiple robots are to operate on the surfaces, then each robot is to collaborate with other robots, minimize the cost of its path, and prevent collision with moving or stationary obstacles while maintaining a global behavior.

Authors are from the Robotics Institute at the University of Technology Sydney (UTS:RI), 15 Broadway, Ultimo NSW 2007, Australia Mahdi.Hassan@uts.edu.au

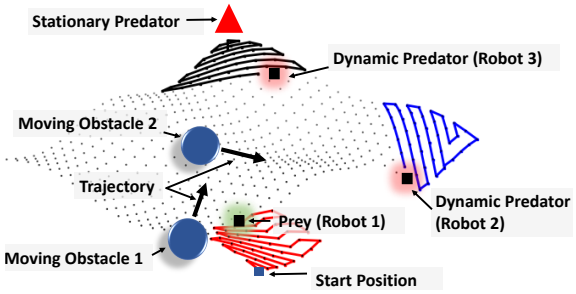


Fig. 1: An illustration of Dec-PPCPP from the perspective of robot 1. The prey (robot 1) continues covering the uncovered areas while avoiding moving obstacles. In doing so, it considers both dynamic predators (other robots) and virtual stationary predator.

A decentralized and adaptive CPP for multi-robot collaborative coverage is presented in this paper. The robot team can be heterogeneous in their speed, coverage size, etc. A decentralized approach is beneficial [14] in providing the desired computational efficiency for scalability with respect to the number of robots as well as robustness to failures (late starts or failure of certain robots does not halt the coverage task). The adaptive behavior of the approach enables it to adapt with respect to unexpected stationary or moving obstacles that may become present. To adapt to such unexpected changes in real-time, each robot needs to make swift local planning since global re-planning may be too expensive for quick responses. However, local real-time planning may produce poor overall paths. Thus, the proposed planner is designed to make the best use of both worlds: global offline planning based on an available map of the surface (to optimize model parameters), and real-time local planning for swift adaptation to local changes while preserving global behavior. It is shown using several case studies, that the real-time planner is able to retain the global behavior. The approach is simple to implement despite the use of both local and global planning.

The approach is inspired by the predator-prey behavior. It is termed Dec-PPCPP, short for Decentralized Predator-Prey CPP. The idea behind the approach is illustrated in Fig. 1. Each robot considers itself to be a prey and perceives other robots as dynamic predators. Each robot also considers a virtual predator that remains stationary at a particular location. Thus, the prey aims to maximize its distance to these two types of predators while searching for food by visiting all unvisited target points in an optimized manner. The

effect of the stationary predator remains active throughout the entire motion of the prey; thus, enforcing a spatial order (an overall direction of motion) on the prey even when the prey encounters unexpected moving obstacles. In other words, it prevents the back-and-forth motion of the prey from one region to another especially when the prey has to temporarily alter its motion strategy due to unexpected obstacles. On the other hand, the effect of dynamic predators is intermittent and becomes active only when any number of dynamic predators (other robots) get within a certain proximity to the prey. This results in fewer motion disruptions between robots and better partitioning of the target surface. It is important to note the distinction between moving obstacles and dynamic predators.

The concept of PPCPP has been investigated for single robot coverage planning problem [15]. This paper aims at addressing the decentralized multi-robot coverage planning problem by developing a decentralized PPCPP approach (Dec-PPCPP). In particular,

- The mathematical modeling and the algorithm are modified to enable decentralized coverage by a robot team that may have different capabilities (e.g. speed);
- A new reward function related to dynamic predators is designed to enable effective interaction between robots for better collision-avoidance and task allocation;
- Analysis for decentralized coverage is included (computational complexity and complete coverage);
- Comparisons and validations are conducted using several case studies and comparative scenarios.

II. PROBLEM DEFINITION

Let a set of target points $\boldsymbol{O} = \{\boldsymbol{o}_j : j = 1,2,\ldots,n^{\boldsymbol{O}}\}$ represent an object's surface which can be planar or embedded in $\mathbb{R}^3$. It is assumed that the set $\boldsymbol{O}$ is given or can be constructed from a known map of the surface. Target points are henceforth simply referred to as *targets*.

Suppose that n robots, $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n$, which can have different capabilities (such as speed and coverage size), are to collectively go through all $\boldsymbol{o}_j \in \boldsymbol{O}$ to perform a coverage task. Let $X = \{X^s \cup X^d\} = \{x_l : l = 1,2,\ldots,n^l\}$ denote all the stationary obstacles X^s and moving obstacles X^d that may become present in the environment. For a robot $\mathcal{R}_i$ and at each step, only a subset $X_i \subseteq X$ may be encountered or relevant for consideration.

The multi-robot decentralized coverage problem is therefore defined as follows: Given the set of targets $\boldsymbol{O}$, how to devise a computationally tractable real-time planner that enables each robot to traverse a path with the aim of achieving: 1) a minimal cost path, 2) swift adaptation to a nearby detected obstacle $x_j \in X_i$ for collision avoidance without drastic impact on the overall performance, and 3) coverage of all targets $\boldsymbol{O}$ in collaboration with other robots. For efficient collaborative coverage by the robots, the decentralized planner should inherently result in appropriate task allocation, avoidance of robots disrupting each others' motion, avoidance of repeated coverage, and complete coverage with minimal cost. The cost of the path could consider the path execution time, path length, smoothness, etc.

In this work, it is assumed that actions and observations are deterministic, the environment is fully observable, and that there are no communication dropouts between robots.

III. THE DEC-PPCPP APPROACH

In brief, using Dec-PPCPP during the real-time deployment, each robot iteratively determines the next best neighboring target that will be visited based on the local environment and the information received from other robots. This one-step planning based on local information makes the approach efficient for real-time coverage and swift for adaptation with respect to obstacles. Four reward functions are designed to act as heuristics for evaluating the performance of uncovered and obstacle-free neighboring targets, and the neighbor with maximum reward is selected for visiting next. In the rare case of a robot reaching a dead-end (no uncovered and obstacle-free neighbor to select), then a point-to-point planner will guide the robot to the nearest uncovered target. If a robot encounters a moving obstacle that is within a certain proximity to itself, it will temporarily stop coverage to avoid collision and resume coverage when the obstacle is no longer a risk. It will then swiftly maintain the global optimized behavior to prevent it from switching back-and-forth between regions. The pseudo-code in Algorithm 1 shows the details of the real-time Dec-PPCPP from ith robot's perspective.

The ith robot will be calculating its next best target while it is moving from its current *position* $\boldsymbol{p}_i$ to its *destination* target $\boldsymbol{o}_i^d$. The first destination target is the *start* target, i.e. $\boldsymbol{o}_i^d = \boldsymbol{o}_i^s$ (Alg. 1: line 2). The coverage task continues so long as the set of *uncovered* targets $\boldsymbol{O}_i^u$ is not empty or until the current coverage *time* t_i is below the *maximum time* t^{max} allowed for the coverage task (Alg. 1: line 3).

For the ith robot to calculate its next best target, it will first communicate with other robot $\mathcal{R}_j, j : \{1,2,\ldots,n\} \setminus i$ to receive updates (Alg. 1: lines 4 to 8). Accordingly, the ith robot will be aware of other robots position $\boldsymbol{p}_j$ and their destination target $\boldsymbol{o}_j^d$, and it will update the set of covered targets $\boldsymbol{O}_i^c$ (Alg. 1: line 6), the set of *uncovered* targets $\boldsymbol{O}_i^u$ (Alg. 1: line 9), and the set of *boundary* targets $\boldsymbol{O}_i^b$ (Alg. 1: lines 7 and 10). The boundary targets are the uncovered targets closest to the covered targets. They are used during the dead-end condition, as will be explained later.

The ith robot then scans the environment (Alg. 1: line 11) and updates the set of targets $\boldsymbol{O}_i^o$ *occupied* by obstacles. It also updates the information of the set of moving obstacles X_i^d that can be observed. Since the robot is only concerned with the neighboring targets (henceforth simply *neighbors*) at each step, then $\boldsymbol{O}_i^o$ can be updated with the status of neighbors only for increased efficiency.

If a moving obstacle in X_i^d is closer than a distance α_i to the ith robot, then the robot's priority is to move away from the obstacle (Alg. 1: lines 12 to 19); otherwise, the robot continues with the coverage task (Alg. 1: lines 20 to 22). From all the moving obstacles, X_i^d detected by the robot, a subset $X_i^\delta \in X_i^d$ that are within δ-proximity to the robot are considered (Alg. 1: line 12 where the function $d(\boldsymbol{p}_i, x_l)$ calculates the closest distance between the

Algorithm 1 Dec-PPCPP (from the perspective of ith robot)

1: **Initialize:** $\boldsymbol{O}_i^u \leftarrow \boldsymbol{O} \setminus \boldsymbol{o}_i^s$; $\boldsymbol{O}_i^c \leftarrow \boldsymbol{o}_i^s$; $\boldsymbol{O}_i^o \leftarrow \emptyset$; $\boldsymbol{o}_i^d \leftarrow \boldsymbol{o}_i^s$;
▷ Move to the first target
2: StartMoving($\boldsymbol{p}_i, \boldsymbol{o}_i^d$) ▷ $\boldsymbol{p}_i$ is current position, $\boldsymbol{o}_i^d$ is destination
▷ Continue covering targets until a stopping criterion is met
3: **while** $\boldsymbol{O}_i^u \neq \emptyset$ **or** $t_i \leq t^{max}$ **do**
▷ Receive updates from other robots
4: **for all** $j \in \{1,2,\ldots,n\} \setminus i$ **do**
5: $(\boldsymbol{O}_j^c, \boldsymbol{O}_i^b, \boldsymbol{p}_j, \boldsymbol{o}_j^d) \leftarrow$ ReceiveUpdate($\mathcal{R}_j$)
6: $\boldsymbol{O}_i^c \leftarrow \boldsymbol{O}_i^c \cup \boldsymbol{O}_j^c \cup \boldsymbol{o}_j^d$ ▷ update *covered* targets
7: $\boldsymbol{O}_i^b \leftarrow \boldsymbol{O}_i^b \cup \boldsymbol{O}_j^b$ ▷ update *boundary* targets
8: **end for**
9: $\boldsymbol{O}_i^u \leftarrow \boldsymbol{O}_i^u \setminus \boldsymbol{O}_i^c$ ▷ update *uncovered* targets
10: $\boldsymbol{O}_i^b \leftarrow \boldsymbol{O}_i^b \setminus \boldsymbol{O}_i^c$ ▷ update *boundary* targets
▷ Scan the environment and update
11: $(\boldsymbol{O}_i^o, X_i^d) \leftarrow$ Scan&Update($\boldsymbol{O}, \boldsymbol{O}_i^o$)
▷ Move away from a nearby moving obstacle
12: $X_i^\delta \leftarrow \{x_l \in X_i^d | d(\boldsymbol{p}_i, x_l) \leq \delta,\ l = 1,2,\ldots,n^l\}$ ▷ set of obstacles in δ-proximity of robot
13: $\alpha_i \leftarrow f(X_i^\delta)$ ▷ virtual sphere radius based on fastest obs.
14: $m^* \leftarrow \arg\min_m d(\boldsymbol{p}_i, x_m \in X_i^\delta)$ ▷ index of closest obstacle
15: **if** $d(\boldsymbol{p}_i, x_{m^*}) \leq \alpha_i$ **then** ▷ if obstacle inside virtual sphere
16: $\boldsymbol{N} \leftarrow$ NearestNeighbors($\boldsymbol{O}, \boldsymbol{o}_i^d, r$)
17: $\boldsymbol{N}^f \leftarrow \boldsymbol{N} \setminus \boldsymbol{O}_i^o$ ▷ Obstacle *free* targets
18: $k^* \leftarrow \arg\max_k \left(d(\boldsymbol{o}_k \in \boldsymbol{N}^f,\ x_{m^*})\right)$ ▷ farthest target index
19: $\boldsymbol{o}_i^n \leftarrow \boldsymbol{o}_{k^*}$ ▷ *next* target to move to
20: **else**
21: $\boldsymbol{o}_i^n \leftarrow$ NextBestNeighbor$\big(i, \boldsymbol{O}_i^c, \boldsymbol{O}_i^o, \boldsymbol{o}_i^p, \boldsymbol{o}_i^d, \cdots$
22: $\Psi_i^s, r, \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n\} \setminus \boldsymbol{p}_i\big)$ ▷ Algorithm 2
23: **end if**
▷ Move to the next target and update states
24: **if** $\boldsymbol{p}_i = \boldsymbol{o}_i^d$ **then** ▷ wait until robot arrives at destination
25: StartMoving($\boldsymbol{p}_i, \boldsymbol{o}_i^n$) ▷ start moving to next target
26: $\boldsymbol{O}_i^c \leftarrow \boldsymbol{O}_i^c \cup \boldsymbol{o}_i^n$; $\boldsymbol{O}_i^u \leftarrow \boldsymbol{O}_i^u \setminus \boldsymbol{o}_i^n$; $\boldsymbol{O}_i^b \leftarrow \boldsymbol{O}_i^b \setminus \boldsymbol{o}_i^n$
27: $\boldsymbol{o}_i^p \leftarrow \boldsymbol{o}_i^d$; $\boldsymbol{o}_i^d \leftarrow \boldsymbol{o}_i^n$
28: **end if**
29: **end while**

Algorithm 2 NextBestNeighbor

1: $\boldsymbol{N}^u \leftarrow$ NearestNeighbors($\boldsymbol{O}_i^u, \boldsymbol{o}_i^d, r$) ▷ *uncovered* neighbors
2: $\boldsymbol{N}^{u,f} \leftarrow \boldsymbol{N}^u \setminus \boldsymbol{O}_i^o$ ▷ *uncovered* and obstacle *free* neighbors
3: $\boldsymbol{O}_i^b \leftarrow \{\boldsymbol{O}_i^b \cup \boldsymbol{N}^{u,f}\} \setminus \boldsymbol{O}_i^o$ ▷ update *boundary* targets
▷ Perform dead-end recovery if all neighbors are covered
4: **if** $\boldsymbol{N}^{u,f} = \emptyset$ **then**
5: $\boldsymbol{o}_i^b \leftarrow$ TempGoal($\boldsymbol{o}_i^d, \boldsymbol{O}_i^b, \boldsymbol{O}$) ▷ temporary goal target
6: $\boldsymbol{o}_i^n \leftarrow$ Pt2PtPlanner($\boldsymbol{o}_i^d, \boldsymbol{o}_i^b, \boldsymbol{O}$)
▷ Find the neighbor with maximum reward
7: **else**
8: **for** $j = 1$ to $|\boldsymbol{N}^{u,f}|$ **do**
9: $\boldsymbol{N}_j^u \leftarrow$ NearestNeighbors($\boldsymbol{O}_i^u, \boldsymbol{o}_j \in \boldsymbol{N}^{u,f}, r$)
10: $\boldsymbol{N}_j^{u,f} \leftarrow \boldsymbol{N}_j^u \setminus \boldsymbol{O}_i^o$
11: $R_j \leftarrow$ TotalReward$\big(\boldsymbol{N}_j^{u,f}, \boldsymbol{o}_j, \boldsymbol{o}_i^p, \boldsymbol{o}_i^d, \Omega_i, \Psi_i^s, \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n\} \setminus \boldsymbol{p}_i\big)$
12: **end for**
13: $j^* \leftarrow \arg\max_j (R_j)$ ▷ Equations (6) and (7)
14: $\boldsymbol{o}_i^n \leftarrow \boldsymbol{o}_{j^*}$
15: **end if**

ith robot's position $\boldsymbol{p}_i$ and the lth obstacle $x_l \in X_i^\delta$). A virtual sphere with radius α_i protects the robot from collisions. The value of α_i can be chosen based on an appropriate function that considers the fastest obstacle in X_i^δ (Alg. 1: line 13). If the closest obstacle is within the virtual sphere (Alg. 1: lines 14 and 15), then the robot first finds the obstacle-free neighbors $\boldsymbol{N}^f$ which are within a radius r (Alg. 1: lines 16 and 17), and moves to the farthest target in $\boldsymbol{N}^f$ (Alg. 1: lines 18 and 19). Other methods for collision avoidance that are suitable to the intended application may be used.

If no moving obstacle is close enough to the robot, then the robot will determine its next best neighbor $\boldsymbol{o}_i^n$ for coverage (Alg. 1: line 21). To do so, it will follow the process shown in Algorithm 2. At first, it will find the nearest uncovered neighbors, $\boldsymbol{O}_i^u$ that are within a neighborhood radius r (Alg. 2: line 1). The obstacle-free neighbors, $\boldsymbol{N}^{u,f}$ are then determined (Alg. 2 line 2), and the boundary targets, $\boldsymbol{O}_i^b$ are updated (Alg. 2: line 3).

Due to obstacles and interaction with other robots, it may happen that the robot ends up in a location where all neighbors are already covered or occupied, i.e. $\boldsymbol{N}^{u,f} = \emptyset$. This state of the robot is referred to as *dead-end* in this paper since the robot has no uncovered neighbors to choose from. If the robot ends up in a dead-end (Alg. 2: line 4), only then it is allowed to *repeat* coverage of certain targets to reach an uncovered target. To do so, it first finds the nearest uncovered boundary target, $\boldsymbol{o}_i^b \in \boldsymbol{O}_i^b$ which it considers as the temporary target to reach (Alg. 2: line 5). To reach $\boldsymbol{o}_i^b$, the robot will utilize a point-to-point planner (Alg. 2: line 6) that is suitable for the application (e.g., A* or Dijkstra). However, as the robot moves towards $\boldsymbol{o}_i^b$ and every-time it reaches a target along the planned path, the robot may switch to another path or boundary target since the coverage status and environment can change due to obstacles and other robots. Thus, only the next target, $\boldsymbol{o}_i^n$ is of interest (Alg. 2: line 6). The coverage task resumes once the robot is out of dead-end.

If the robot is *not* in a dead-end, then it will determine the neighbor with the maximum reward (Alg. 2: lines 7 to 14). In brief, it will loop through the neighbors in $\boldsymbol{N}^{u,f}$ (Alg. 2: line 8), and for each neighbor $\boldsymbol{o}_j, j \in \{1,2,\ldots,|\boldsymbol{N}^{u,f}|\}$, it will calculate the total reward R_j (Alg. 2: line 11). Then, it will consider the neighbor with maximum reward as the next best neighbor $\boldsymbol{o}_i^n$ (Alg. 2: lines 13 and 14). Four reward functions are considered for calculating the total reward R_j for the jth neighboring uncovered target $\boldsymbol{o}_j$: 1) stationary predator avoidance reward, 2) dynamic predator avoidance reward, 3) path smoothness reward, and 4) boundary reward. The formulations of these rewards are explained in the next Section. The inputs (Alg. 2: line 11) to these reward functions include the jth neighbor $\boldsymbol{o}_j$ and its uncovered and obstacle-free neighbors $\boldsymbol{N}_j^{u,f}$, the previous target $\mathbf{o}_i^p$ covered by the ith robot, the destination target $\boldsymbol{o}_i^d$, the weighting factors for the reward functions Ω_i (obtained through offline optimization - explained in the next section), the stationary predator location Ψ_i^s, and all other robots' locations $\{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n\} \setminus \boldsymbol{p}_i$ (act as

dynamic predators on the ith robot).

The ith robot, after arriving at the destination target $\boldsymbol{o}_i^d$ (Alg. 1: line 24), will start moving towards the next best neighbor $\boldsymbol{o}_i^n$ (Alg. 1: line 25) and updates all states (Alg. 1: lines 26 and 27). Note that $\boldsymbol{o}_i^d$ becomes $\boldsymbol{o}_i^n$ (Alg. 1: line 27), and the process is repeated to find the next best neighbor $\boldsymbol{o}_i^n$.

IV. MATHEMATICAL MODELING

A. Stationary Predator Avoidance Reward

In Dec-PPCPP, each robot considers itself a prey that needs to avoid predation from two types of predators: 1) a stationary virtual predator, and 2) dynamic predators (other robots). In this subsection, the stationary predator avoidance reward is formulated, which is similar to the work in [15]. A prey, while searching the target area for food (to achieve coverage), aims to continually maximize it distance to a stationary predator, denoted as Ψ_i^s where i is the robot index. Thus, naturally it will cover the regions farthest from the predator and gradually moves closer and closer to the predator as it covers more of the target area. This behavior results in an orderly overall motion for the prey which prevents it from moving back-and-forth between regions. This behavior is particularly helpful when the prey encounters a moving obstacle since after a temporary change of motion to avoid a collision, the prey will be forced (due to predator avoidance) to resume coverage from the region where it was covering last. This results in shorter and more efficient paths.

The function for calculating the stationary predator avoidance reward for moving to the jth neighbor, $\boldsymbol{o}_j$ is:

$$R^{sp}(\boldsymbol{o}_j) = \frac{D(\boldsymbol{o}_j) - D_{min}(\boldsymbol{o}_i)}{D_{max}(\boldsymbol{o}_i) - D_{min}(\boldsymbol{o}_i)} \tag{1}$$

where $D(\boldsymbol{o}_j) = \|\boldsymbol{o}_j - \Psi_i^s\|$ is the distance from $\boldsymbol{o}_j$ to the predator Ψ_i^s, $D_{max}(\boldsymbol{o}_i) = \max_j \|\boldsymbol{o}_j - \Psi_i^s\|$ is the maximum distance to the predator from one of the neighbors of the prey $\boldsymbol{o}_i$, and similarly, $D_{min}(\boldsymbol{o}_i) = \min_j \|\boldsymbol{o}_j - \Psi_i^s\|$ is the minimum distance. Note that $D_{max}(\boldsymbol{o}_i) - D_{min}(\boldsymbol{o}_i)$ is therefore a constant for a prey location and $R^{sp}(\boldsymbol{o}_j) \in [0,1]$. Thus, the prey will aim to move to the farthest neighbor to achieve the maximum reward $R^{sp}(\boldsymbol{o}_j) = 1$.

B. Dynamic Predator Avoidance Reward

Unlike the stationary predator, the dynamic predators (other robots) do *not* affect the prey's motion for the entire coverage task. Instead, they only cause a local temporary effect on the prey's motion when they are close to the prey. The closer a dynamic predator is to a prey, the higher the reward that the prey obtains by moving to an uncovered neighbor that is farther from the predator. Since the robots see each other as predators, then this behavior results in the robots pushing each other away while performing the coverage task. Thus, two main benefits are obtained: 1) the robots are less likely to collide with each other or block each other's path, and 2) the robots will naturally cover different regions which result in better task allocation.

Let $\Psi_{i,k}^d$ denote the kth dynamic predator ($k \in \{1,2,\ldots,n\} \setminus i$) for the prey representing the ith robot. The function for calculating the dynamic predator avoidance reward for the prey (ith robot) moving to the jth neighbor, $\boldsymbol{o}_j$ is formulated as (due to the kth predator $\Psi_{i,k}^d$):

$$R^{dp}(\boldsymbol{o}_j, \Psi_{i,k}^d) = S(\boldsymbol{o}_i) \frac{D(\boldsymbol{o}_j) - D_{min}(\boldsymbol{o}_i)}{D_{max}(\boldsymbol{o}_i) - D_{min}(\boldsymbol{o}_i)} \tag{2}$$

where $D(\boldsymbol{o}_j)$, $D_{max}(\boldsymbol{o}_i)$, and $D_{min}(\boldsymbol{o}_i)$ are calculated in the same way as in Section IV-A except that the predator is now the location of the kth robot, i.e. $\Psi_{i,k}^d = \boldsymbol{p}_k$. For brevity, the notation $\Psi_{i,k}^d$ is dropped from above functions.

The function $S(\boldsymbol{o}_j)$ is the inverted Sigmoid function:

$$S(\boldsymbol{o}_i, \Psi_{i,k}^d) = \frac{1}{1 + \exp^{\kappa(a - \frac{b}{2})}} \tag{3}$$

where κ determines the slope of the sigmoid function, $a = \|\boldsymbol{o}_i - \Psi_{i,k}^d\|$ is the distance from the prey's current destination target $\boldsymbol{o}_i = \boldsymbol{o}_i^d$ to the predator $\Psi_{i,k}^d$, and b is the effective range. When the kth predator $\Psi_{i,k}^d$ is far away (i.e., when $a > b$), then $S(\boldsymbol{o}_i, \Psi_{i,k}^d) \approx 0$ meaning that this reward becomes almost negligible. This is important since the dynamic predator should only be relevant as it moves closer to the prey, otherwise the prey should continue emphasizing more on maintaining the global behavior of keeping a spatial order (or an overall direction of motion) enforced by the stationary predator (previous reward). The larger the effective range b, the farther away the robots will be from each other when covering the target area. Note that $R^{dp}(\boldsymbol{o}_j, \Psi_{i,k}^d) \in (0,1)$.

C. Smoothness Reward

Another relevant reward for robotic coverage applications is the smoothness of the path. A smoother path that has fewer turns can reduce frequent accelerations and decelerations and can result in a faster and energy-efficient coverage. As such, the prey is given an extra reward for continuing motion in a straight direction. This reward function, which is similar to the work in [15], is formulated as follows:

$$R^s(\boldsymbol{o}_j) = \frac{\Theta(\boldsymbol{o}_j) - \Theta_{min}(\boldsymbol{o}_i)}{\Theta_{max}(\boldsymbol{o}_i) - \Theta_{min}(\boldsymbol{o}_i)} \tag{4}$$

where $R^s(\boldsymbol{o}_j) \in (0,1]$ is the reward associated with the jth neighbor, $\boldsymbol{o}_j$, of the current prey target, $\boldsymbol{o}_i$, due to the angle $\Theta(\boldsymbol{o}_j) = \angle \boldsymbol{o}^p \boldsymbol{o}_i \boldsymbol{o}_j \in (0^\circ, 180^\circ]$ which is the angle between the vectors $(\boldsymbol{o}^p - \boldsymbol{o}_i)$ and $(\boldsymbol{o}_i - \boldsymbol{o}_j)$, and $\boldsymbol{o}^p$ is the previous target covered by the prey. $\Theta_{max}(\boldsymbol{o}_i) = \max_j \angle \boldsymbol{o}^p \boldsymbol{o}_i \boldsymbol{o}_j$ is the maximum possible angle considering all neighbors, and similarly $\Theta_{min}(\boldsymbol{o}_i)$ is the minimum angle.

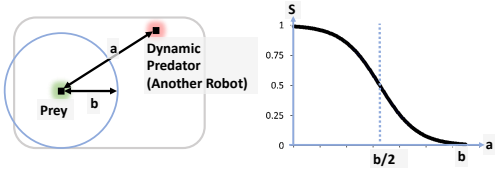


Fig. 2: The concept for the dynamic predator avoidance reward.

D. Boundary Reward

Boundary targets are those that are closest to the boundary of the surface and the *covered* regions (i.e. lie on the boundary of the *uncovered* region). An additional reward is given to the prey for covering the boundary targets. This results in regular coverage since it is more natural to continually cover the targets closest to what has already been covered. Boundary targets have less uncovered neighbors since they are next to the covered targets and surface boundary. The less uncovered neighbors a boundary target has, the stronger it fits the definition of a boundary target. As such, this reward function aims to reward the prey for covering a neighbor that has the least uncovered neighbors.

Similar to [15], this reward function is formulated as:

$$R^b(\boldsymbol{o}_j) = \frac{B_{max}(\boldsymbol{o}_i) - B(\boldsymbol{o}_j)}{B_{max}(\boldsymbol{o}_i) - B_{min}(\boldsymbol{o}_i)} \tag{5}$$

where $R^b(\boldsymbol{o}_j) \in [0,1]$ is the reward associated with the jth neighbor $\boldsymbol{o}_j$ of the current prey target $\boldsymbol{o}_i$, $B(\boldsymbol{o}_j)$ returns the number of *uncovered* neighbors of $\boldsymbol{o}_j$, $B_{max}(\boldsymbol{o}_i) = \max_j B(\boldsymbol{o}_j)$ for $j = \{1,\ldots,|\boldsymbol{N}_j^{u,f}|\}$ where $\boldsymbol{N}_j^{u,f}$ is the set of uncovered and obstacle-free neighbors (Alg. 2: line 10), and similarly $B_{min}(\boldsymbol{o}_i) = \min_j B(\boldsymbol{o}_j)$.

E. Total Reward (Sum of All Rewards)

The total reward for moving to an uncovered neighbor $\boldsymbol{o}_j$ is the sum of all the rewards previously stated, i.e.:

$$\begin{aligned} R(\boldsymbol{o}_j) = R^{sp}(\boldsymbol{o}_j) + \omega^p \Big(R^{dp}(\boldsymbol{o}_j, \Psi_{i,1}^d) + R^{dp}(\boldsymbol{o}_j, \Psi_{i,2}^d) + \cdots \\ + R^{dp}(\boldsymbol{o}_j, \Psi_{i,n}^d) \Big) + \omega^s \Big(R^s(\boldsymbol{o}_j) \Big) + \omega^b \Big(R^b(\boldsymbol{o}_j) \Big) \end{aligned} \tag{6}$$

where ω^p, ω^s and ω^b are the weighting factors associated with the dynamic predator avoidance, the smoothness and the boundary reward functions, respectively.

Thus, for $j = \{1,\ldots,|\boldsymbol{N}_j^{u,f}|\}$, the index j^* of the uncovered neighbor with maximum reward is:

$$j^* = \arg\max_j \Big(R(\boldsymbol{o}_j \in \boldsymbol{N}_j^{u,f}) \Big). \tag{7}$$

Hence, the prey will move to the target $\boldsymbol{o}_{j^*} \in \boldsymbol{N}_j^{u,f}$ next (Alg. 2: lines 13 and 14), and the process is repeated.

F. Mathematical Model for Optimizing Weighting Factors

Various factors such as the location of the stationary predator and the geometric shape of the object will play a role in the importance of each reward. Thus, for a given surface, the weighting factors in (6) need to be optimized by each robot, but only once prior to the real-time deployment. This will enable each robot to make use of the available map to optimize the parameters. The weighting factor ω^p which is related to the dynamic predator avoidance reward can be tuned to suit robots' speed, size of the environment, and the type of robot. Thus, only the weighting factors related to the smoothness and boundary rewards are optimized; hence, the design variables are $\boldsymbol{Z} = (\omega^s, \omega^b)$.

The cost of a path for a robot is defined with respect to the total coverage time. Therefore, the objective function is:

$$\min_{\mathbf{Z}} f(\boldsymbol{Z}) = T(\boldsymbol{P}_{\boldsymbol{Z}}) \tag{8}$$

where $T(\boldsymbol{P}_{\boldsymbol{Z}})$ is the time it takes the corresponding robot to cover the path $\boldsymbol{P}_{\boldsymbol{Z}}$ using the same procedure in Alg. 1 but considering the values in $\mathbf{Z}$ decided by an optimization algorithm. The optimizer will iteratively change the values in $\mathbf{Z}$ to obtain a path with minimal completion time. Other objective functions, such as minimal path length, minimal energy, or minimal coverage repetition rate may also be used.

V. ANALYSIS

A. Computational Complexity

Suppose binary vectors are used to store the status of the targets in the sets $\boldsymbol{O}_i^c$, $\boldsymbol{O}_i^u$, and $\boldsymbol{O}_i^b$. For example, if the kth target is covered, then the kth index of the vector for $\boldsymbol{O}_i^c$ is set to 'true', and vice versa. Thus, the updates from other robots can be done fast (Alg. 1: lines 4 to 10) by sharing the index of the targets that changed status since the last communication. Let $c_1, c_2, \ldots, c_n$ be the number of targets with status change for each robot since the last communication with the ith robot. These values are small and can be considered constant for deriving computation complexity. Thus, the time complexity for the updates is $\mathcal{O}(n)$ where n is the number of robots.

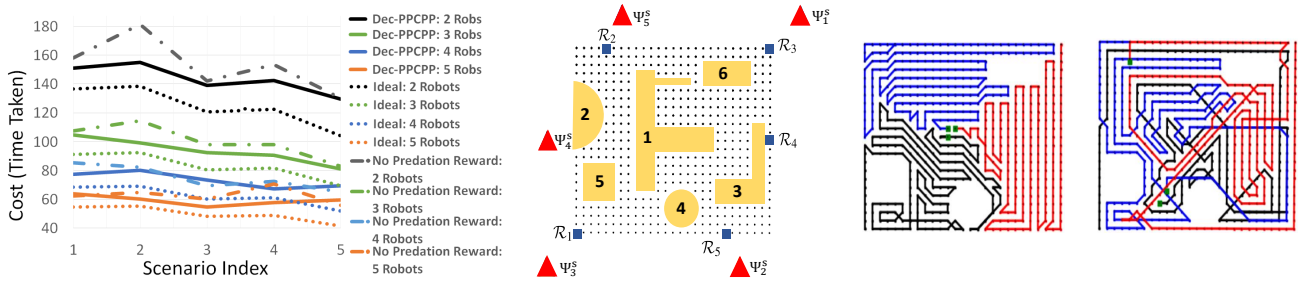
Suppose k-d tree data structure is used for storing the targets in $\boldsymbol{O}$ and finding nearest neighbors. The queried nearest neighbors can then be check with the above binary vectors to find their status. The collision status of these neighbors is also determined by scanning the environment (Alg. 1: line 11). To query, delete or insert a point in the k-d tree, the time complexity is $\mathcal{O}(\log m)$ where m is the number of targets in the set $\boldsymbol{O}$. Therefore, following the same procedure as in the previous work [15], the time complexity for finding the neighbor with maximum reward (Alg. 2: line 7 to 14) is $\mathcal{O}(\log m)$.

When the prey is at a dead-end (Alg. 2: lines 4 to 6), the time complexity depends on the point-to-point planner. Note that the dead-end situation happens only a small number of times during the entire coverage task. The time complexity of the 'scan and update' procedure (Alg. 1: line 11) is needed for all algorithms, and therefore not considered in this analysis. The other components of the algorithms are single operations (no loops) that don't considerably grow in time with respect to the number of robots or the number of targets representing the surface.

The overall time complexity of the algorithm (not including the dead-end recovery or 'scan and update' procedure) is $\mathcal{O}(n) + \mathcal{O}(\log m)$ where n is the number of robots and m is the number of targets in the set $\boldsymbol{O}$. Note that typically $n \ll m$.

B. Complete Coverage

As per Alg. 1, each robot is only allowed to select a neighbor that is uncovered. This restriction stops a robot from revisiting covered targets or getting stuck. Thus, all robots



(a) Results (b) The environment (c) Scen. 2 with 3 robs. (d) $R^{dp} = 0$

Fig. 3: Five scenarios with different number of obstacles are considered (S1: Obs. 1; S2: Obs. 4-6; S3: Obs. 1-3; S4: Obs. 2-6; S5: Obs. 1-6). Results for different number of robots (2-5robots) and without dynamic predator avoidance reward R^{dp} are also shown. Example paths with/without R^{dp} are shown. Robots 1, 2 and 3's paths are in black, blue and red, respectively.

collectively cover the uncovered targets one by one until no more uncovered target is left (Alg. 1 line). A robot is allowed to revisit a covered target only when it is in a dead-end or when it needs to avoid collision with a moving obstacle. In both of these cases, the robot will resume coverage as soon as these two conditions are no longer present. Each robot will always communicate with other robots before making a decision on its next best neighbor. This communication prevents a robot from covering a target that has already been covered by another robot. If another robot malfunctions and stops coverage, there is *no* condition for other robots to stop covering the remaining targets.

VI. CASE STUDIES

Four case studies, each considering various scenarios and conditions, are designed to validate and test Dec-PPCPP. More specifically, the following aspects are considered: different size, shape and number of stationary obstacles; different number of robots; Dec-PPCPP with and without the dynamic predator avoidance reward; different 3D objects; different cost functions; robot team with different speeds; moving obstacles with different speeds and sizes; and certain robots in the team malfunctioning (failing to start on time or breaking down during the coverage task).

Genetic Algorithm (GA) is used to optimize the weighting factors in (8). The aim is to show the performance of Dec-PPCPP assuming that optimized weighting factors are given; thus, although various optimization algorithms may be used, comparing the performance of these algorithms with respect to Dec-PPCPP is beyond the scope of this paper. It takes less than 5 milliseconds for the prey to calculate its next best neighbor at each step, but there is room for improvement.

A. Case Study 1: Coverage amid stationary obstacles

The purpose of this case study is to demonstrate the performance of Dec-PPCPP with respect to different number of robots and to validate the use of the dynamic predator avoidance reward (i.e., R^{dp} in Eq. (2)). Five scenarios are used where in each scenario different number of stationary obstacles from those shown in Fig. 3b are considered. Each scenario is also repeated with different number of robots (2 to 5 robots). The obstacles are not known to the robots prior to the real-time deployment.

The results are shown in Fig. 3a where performance is compared relative to two cases: 1) the ideal case (lower bound on optimum) where the cost is calculated as [number of targets $\times$ time to travel between two non-diagonal neighbors] / number of robots, and 2) the case with no dynamic predator avoidance reward, i.e. $R^{dp} = 0$. Taking the average of all results, Dec-PPCPP performs 16.9% worse compared to the ideal paths. Note that in Dec-PPCPP the locations of the obstacles are not known in advance and the ideal paths may not be achievable (since no diagonal moves, interaction between robots, or repetitions are considered in the ideal path). Similarly, taking the average of all results, Dec-PPCPP performs 5.3% better compared to the case where $R^{dp} = 0$. As can be seen in Fig. 3c and 3d (for scenario 2 with 3 robots), when R^{dp} is set to zero, the paths are not only less efficient but also very chaotic. Note that other reward functions have already been validated in the previous work [15]. Video for the scenario shown in Fig. 3c is provided.

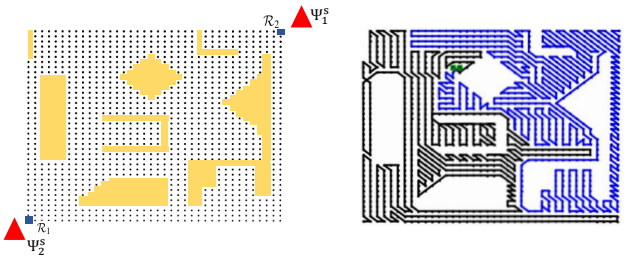
B. Case Study 2: Comparison with other approaches

Dec-PPCPP is compared to BNNB, NB-MSTC, and B-MSTC. A comparison study of these methods was conducted in [8] where BNNB was shown to perform better for the scenario in Fig. 4. The robots are considered as obstacles with respect to each other. The resulting path from Dec-PPCPP for this scenario is shown in Fig. 5b. The comparison results are shown in Table I. A video is also provided.

As shown in Table I, Dec-PPCPP performs almost the same as BNNB. However, unlike BNNB, it is shown in the following case studies that Dec-PPCPP is applicable to 3D coverage with dynamic obstacles. Note that the optimization

TABLE I: Comparison against other approaches.

Approaches	Coverage rate of R_1 (%)	Coverage rate of R_2 (%)	Repetition rate (%)
Dec-PPCPP	51.5	48.5	1.87
BNNB	50.1	49.9	1.9
NB-MSTC	12.6	87.4	0
B-MSTC	32.3	67.7	7



(a) The scenario (b) The paths

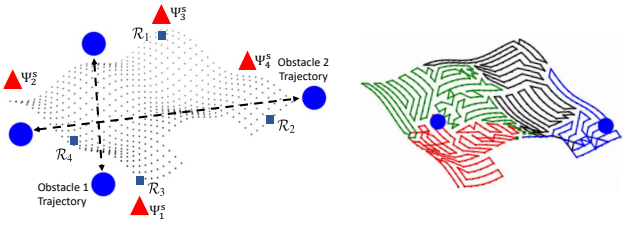
Fig. 4: The scenario and the resulting paths are shown.

was carried to minimize the repetition rate so as to compare with the above methods. Path execution time and other objectives could be added for better results.

C. Case Study 3: 3D Coverage amid dynamic obstacles

The aim of this case study is to show the performance of four robots operating in a 3D environment with dynamic obstacles, as illustrated in Fig. 5a. To show the robustness of Dec-PPCPP to failures, robot 3 fails to start on-time and starts at t=20 and robot 2 malfunctions and stops at t=50. To make the scenario harder, the obstacles are designed to continuously go back-and-forth on the trajectories shown in Fig. 5a. For comparison's sake, three cases are considered: 1) with no dynamic obstacles, 2) with dynamic obstacles having half the speed of the robots, and 3) with dynamic obstacles having the same speed as the robots. Figure 5b is related to case 2 and a video is provided.

The results are shown in Table II. It can be seen that the performance of Dec-PPCPP, amid dynamic obstacles continuously running through the surface, is *not* substantially worse than when there are no obstacles. For further validation of the dynamic predator avoidance reward R^{dp}, the three cases are repeated with $R^{dp} = 0$. It is clear from Table II that when $R^{dp} = 0$ the results are significantly worse.



(a) The scenario (b) The paths for case 2

Fig. 5: The scenario and the resulting paths are shown.

TABLE II: Results for different cases.

	No obstacles	Obs. speed = 0.5*Robs' speed	Obs. speed = Robs' speed
Dec-PPCPP	99.8	110.8	112.1
Dec-PPCPP with $R^{dp} = 0$	130.9	172.6	160.5

TABLE III: Results for case study 4.

	Dec-PPCPP	No obstacles	$R^{dp} = 0$
Ute Scenario	8.02	7.51	8.51
Complex Object	8.99	8.89	9.58

D. Case Study 4: 3D Coverage of complex surfaces

In this case study, the performance of Dec-PPCPP is shown for coverage of complex surfaces amid dynamic obstacles by robots that have different speeds. Two different surfaces are considered (Fig. 6). To increase the difficulty, the dynamic obstacles are designed to continually go back-and-forth on the shown trajectories (Figs. 6c and 6e). Note that the targets are not uniformly distributed, nonetheless Dec-PPCPP is able to generate a path for these targets.

The real-time performance of Dec-PPCPP is compared to the case where there are no obstacles and the case where dynamic predator avoidance reward is not considered ($R^{dp} = 0$). The result are shown in Table III. The resulting paths are also shown in Figs. 6c and 6e. Videos of both scenarios are also provided. Note that for scenario 1 (the ute), robot 2 has twice the speed fo robot 1. For scenario 2 (complex object), robots 2 and 3 have $1.5\times$ and $2\times$ the speed of robot 1.

VII. CONCLUSION

The Dec-PPCPP approach was shown to be applicable to a robot team performing decentralized coverage tasks on planar surfaces as well as surfaces embedded in $\mathbb{R}^3$. Particularly, when dynamic obstacles are present in the environment, it was shown to adapt and generate good results relative to the case where there are no obstacles. Due to the stationary virtual predator, each robot maintains a direction of overall motion causing the overall path to be reasonably organized (less back-and-forth between regions) despite dynamic obstacles being present. Each robot perceives other robots as dynamic predators, which results in the robots repelling each other causing better task allocation and collision-avoidance. However, the effect of dynamic predators is only relevant when robots are close to each other since, otherwise, they should emphasize more on the global behavior resulting from the stationary obstacles. Case studies were presented to validate and show the effectiveness of the approach with respect to aspects such as different obstacles, different number of robots, different 3D objects, and failure of robots.

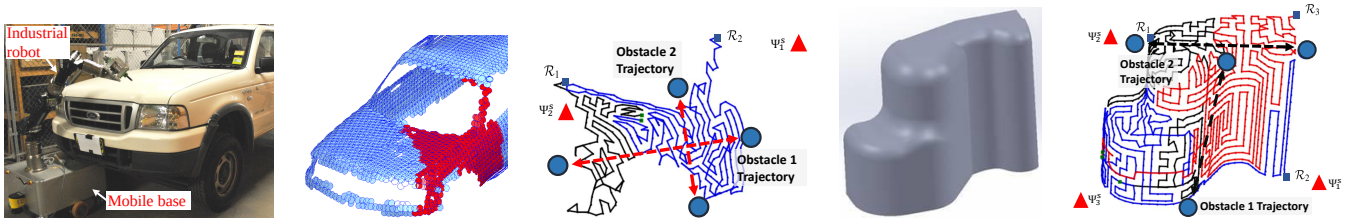
Future work includes extending Dec-PPCPP to be robust for intermittent communication and applicable to unknown environments.

REFERENCES

[1] R. Almadhoun, T. Taha, L. Seneviratne, and Y. Zweiri, "A survey on multi-robot coverage path planning for model reconstruction and mapping," *SN Applied Sciences*, vol. 1, no. 8, p. 847, 2019.

[2] R. Bormann, F. Jordan, J. Hampp, and M. Hägele, "Indoor coverage path planning: Survey, implementation, analysis," in *International Conference on Robotics and Automation*, May 2018, pp. 1718–1725.

[3] J. Song and S. Gupta, "$\varepsilon^\star$: An online coverage path planning algorithm," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 526–533, April 2018.



(a) S1: the ute (b) Area to be covered (c) The resulting path (d) S2: Complex object (e) The resulting path

Fig. 6: Two scenarios where Dec-PPCPP is used for covering complex surface with different robots' speed.

[4] M. Hassan and D. Liu, "A deformable spiral based algorithm to smooth coverage path planning for marine growth removal," in *International Conference on Intelligent Robots and Systems*, 2018, pp. 1913–1918.

[5] A. Mellone, G. Franzini, L. Pollini, and M. Innocenti, "Persistent coverage control for teams of heterogeneous agents," in *Conference on Decision and Control (CDC)*, Dec 2018, pp. 2114–2119.

[6] A. C. Kapoutsis, S. A. Chatzichristofis, and E. B. Kosmatopoulos, "Darp: divide areas algorithm for optimal multi-robot coverage path planning," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 3-4, pp. 663–680, 2017.

[7] C. Gao, Y. Kou, Z. Li, A. Xu, Y. Li, and Y. Chang, "Optimal multirobot coverage path planning: ideal-shaped spanning tree," *Mathematical Problems in Engineering*, vol. 2018, 2018.

[8] H. Liu, J. Ma, and W. Huang, "Sensor-based complete coverage path planning in dynamic environment for cleaning robot," *CAAI Transactions on Intelligence Technology*, vol. 3, no. 1, pp. 65–72, 2018.

[9] A. Bircher, M. Kamel, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, and R. Siegwart, "Three-dimensional coverage path planning via viewpoint resampling and tour optimization for aerial robots," *Autonomous Robots*, vol. 40, no. 6, pp. 1059–1078, 2016.

[10] I. A. Hameed, A. la Cour-Harbo, and O. L. Osen, "Side-to-side 3d coverage path planning approach for agricultural robots to minimize skip/overlap areas between swaths," *Robotics and Autonomous Systems*, vol. 76, pp. 36–45, 2016.

[11] C. Wu, C. Dai, X. Gong, Y.-J. Liu, J. Wang, X. D. Gu, and C. C. Wang, "Energy-efficient coverage path planning for general terrain surfaces," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2584–2591, 2019.

[12] H. I. A. Perez-imaz, P. A. F. Rezeck, D. G. Macharet, and M. F. M. Campos, "Multi-robot 3d coverage path planning for first responders teams," in *International Conference on Automation Science and Engineering*, Aug 2016, pp. 1374–1379.

[13] M. Eich, F. Bonnin-Pascual, E. Garcia-Fidalgo, A. Ortiz, G. Bruzzone, Y. Koveos, and F. Kirchner, "A robot application for marine vessel inspection," *Journal of Field Robotics*, vol. 31, no. 2, pp. 319–341, 2014.

[14] G. Best, O. M. Cliff, T. Patten, R. R. Mettu, and R. Fitch, "Dec-mcts: Decentralized planning for multi-robot active perception," *The International Journal of Robotics Research*, vol. 38, no. 2-3, pp. 316–337, 2019.

[15] M. Hassan and D. Liu, "PPCPP: A predator–prey-based approach to adaptive coverage path planning," *IEEE Transactions on Robotics*, vol. 36, no. 1, pp. 284–301, Feb 2020.