PPCPP: A Predator-Prey Based Approach to Adaptive Coverage Path Planning

Mahdi Hassan and Dikai Liu

***Abstract*—Most of the existing Coverage Path Planning (CPP) algorithms do not have the capability of enabling a robot to handle unexpected changes in the coverage area of interest. Examples of unexpected changes include the sudden introduction of stationary or dynamic obstacles in the environment and change in the reachable area for coverage (e.g. due to imperfect base localization by an industrial robot). Thus, a novel adaptive CPP approach is developed that is efficient to respond to changes in real-time while aiming to achieve complete coverage with minimal cost. As part of the approach, a total reward function that incorporates three rewards is designed where the first reward is inspired by the predator-prey relation, the second reward is related to continuing motion in a straight direction, and the third reward is related to covering the boundary. The total reward function acts as a heuristic to guide the robot at each step. For a given map of an environment, model parameters are first tuned offline to minimize the path length while assuming no obstacles. It is shown that applying these learned parameters during real-time adaptive planning in the presence of obstacles will still result in a coverage path with a length close to the optimized path length. Many case studies with various scenarios are presented to validate the approach and to perform numerous comparisons.**

***Index Terms*—Adaptive coverage path planning, complete coverage, 3D coverage of complex surfaces, dynamic obstacles**

I. Introduction

ROBOTS CAN be used to perform tasks such as surface cleaning, high-pressure blasting, and harvesting, which are tasks requiring complete coverage; that is, all surface areas of interest need to be covered (operated on). Appropriate paths must be generated on the surfaces to achieve complete coverage. The robot's end-effector tool, which is specific to the intended task (e.g. buffing pad or pressure cleaning nozzle), follows these paths. The task of generating these paths on the surfaces is called Coverage Path Planning (CPP) [1].

Enabling a robot to be autonomous and perform CPP in complex environments where unexpected changes can occur, and possibly on objects with complex geometric shapes, is a challenging problem. As autonomous robots start to be practically deployed in such environments, addressing this problem becomes increasingly important. Example applications where the aforementioned problem applies include autonomous robots operating on underwater structures to clean surfaces while avoiding obstacles (e.g. marine life), or a floor cleaning robot avoiding dynamic obstacles (e.g. people) while still obtaining complete coverage of the surface with minimal cost (e.g. minimal travel time or path length).

Manuscript received March 27, 2019; accepted September 8, 2019. This paper was recommended for publication by Editor I.-M. Chen upon evaluation of the reviewers comments. *(Corresponding author: Mahdi Hassan)*

The authors are with the Centre for Autonomous Systems (CAS) at the University of Technology Sydney (UTS), Ultimo, 2007, Australia (e-mail: Mahdi.Hassan@uts.edu.au; Dikai.Liu@uts.edu.au).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the author. The material consists of four videos in WMV format related to case studies 3, 5, 6, and 7 where dynamic obstacles are present.

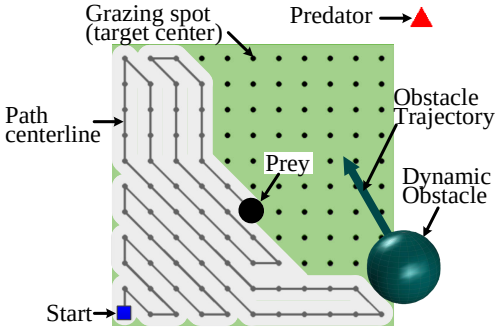


Fig. 1: A grazing path of a prey while avoiding a dynamic obstacle and simultaneously maximizing its distance to a perceived ambushing predator. Note that obstacles are not considered as predators.

A novel adaptive CPP approach is presented in this paper, which is based on the concepts of *foraging* and *risk of predation* in predator-prey relation. The *prey* represents the coverage spot of the robot's end-effector tool, e.g. the cleaning unit's coverage spot of a floor cleaning robot. The *predator* is a virtual stationary point that the prey continually maximizes its distance to while *grazing* on (covering) the target area and avoiding obstacles. The predator is not an obstacle; it is a virtual point enforcing a spatial order (or a direction of the overall traversal) on the movement of the prey. It is shown using several case studies that this spatial ordering causes the path to be shorter as it avoids the back and forth motion of the prey from one region to another. This concept is shown using a simple example in Fig. 1. The use of the predator is a unique feature of the presented approach, which sets it apart from other CPP approaches. Additionally, the approach accounts for improving the path length and smoothness by rewarding the prey to continue its motion in a straight direction and covering the boundary as much as possible. The approach enables learning from prior environmental information and applying the learned parameters to perform adaptive local planning in real-time to handle obstacles and changes in the environment.

Using the developed CPP approach, the path can quickly adapt to unforeseen changes in real-time, e.g. when stationary or dynamic obstacles (with initially unknown size and trajectory) are unexpectedly introduced in the environment or when an incorrect base placement of an industrial robot

slightly changes the coverage area of interest. Furthermore, the approach can be applied to surfaces embedded in $\mathbb{R}^3$, i.e. it is applicable to arbitrarily bent 2D manifolds and not just piecewise planar surfaces. The approach is computationally tractable for real-time deployment amid unexpected dynamic obstacles, and has only two parameters to tune for a given environment. Aiming to achieve optimal coverage, i.e. a path with minimal cost, is also taken into account. The results show that the difference between the length of the path generated in real-time and the offline optimized path length is small. The advantages of the developed CPP approach relative to existing approaches are discussed in the next section.

The rest of the paper is organized as follows. Section II presents an overview of the research work related to CPP. Section III provides a more detailed explanation of the problem. In Section IV, the concept of predator-prey is utilized to explain the intuition behind the approach. Section V presents the approach followed by the mathematical modeling in Section VI. Many case studies are then presented in Section VII to validate the approach. A brief discussion on the limitations and advantages of the approach, followed by an outline of the future work, is presented in Section VIII. Concluding remarks are stated in Section IX.

II. Related Work

A. Coverage of Planar Surfaces (2D Coverage)

1) Offline Coverage: Most of the existing CPP algorithms are designed for mobile robots, e.g. floor cleaning, lawn-mowing, and harvesting robots. The areas that these robots cover can be approximated as planar surfaces (i.e. 2D). The work in [1] groups the algorithms for 2D coverage into categories such as cellular decomposition, grid-based, and graph-based coverage. Many of such algorithms are appropriate for offline coverage, i.e. for applications where the environment is known and the planning can be performed prior to executing the coverage task by the robot.

The work in [2] presents an approximation algorithm for the basic forms of the "lawn mowing" type problems (e.g. optimal inspection and spray painting) and "milling" type problems where the tool is constrained to only cut within the allowed region. In another similar work [3], both lawn mowing and milling are considered when solving the combined problem of finding a minimal cost closed path for complete scanning of a polygonal environment while minimizing the number of scan points. Interestingly, the works in [2, 3] prove that even for simple cases, these problems are NP-hard.

2) Online Coverage: The problems become more challenging when considering online CPP, i.e. when the robot is expected to appropriately plan and simultaneously execute the coverage task, and there may not be full knowledge of the environment prior to the execution. Boustrophedon decomposition is a popular algorithm where the area is appropriately divided into regions, and then each region is covered using boustrophedon motion (lawnmower-like motion). An algorithm named BA* [4] is based on the boustrophedon motion and the A* search algorithm: it determines an appropriate backtracking point to an uncovered region when the boustrophedon path reaches a deadlock, then it performs an A* search to reach the backtracking point and repeats the boustrophedon motion for the newly uncovered region. Another online algorithm is the ε^* [5] which has the desirable property of producing the boustrophedon-like back and forth motion without relying on critical point detection. When developing online CPP algorithms, there may be additional application-specific constraints that need to be satisfied. For example, in mobile robotics, a family of problems is concerned with the limited resources of the robots [6, 7, 8]. In [6], an algorithm is developed that takes into account the battery capacity of a mobile robot, and in [7] the cable length of a tethered mobile robot is considered, and the avoidance of cable crossing by the robot is part of the planning process.

3) Online and Adaptive Coverage: A group of coverage approaches is based on the neural network [1, 9] (note that this is different from the neural network used in deep learning). In neural network-based approaches, the environment is represented as a uniform grid map where each grid is associated with a neuron. Dynamics of each neuron is characterized using a shunting equation. The neural activity of the neurons that are in collision with obstacles is negative, whereas the neural activity of the neurons that are in the uncovered areas of the environment is positive. The neural activity of all neurons at a given instant of time is referred to as the activity landscape. This activity landscape globally attracts the robot towards the uncovered areas (due to positive neural activities) and repulses the robot from obstacles (due to negative neural activities). At each step, the robot selects its next position based on the activity of its neighboring neurons and its previous position. Neural network-based approaches have the advantage of being adaptive [1] in that they can re-plan the path online when dynamic obstacles are present in the environment. However, for large environments such as the seabed, neural network approach can be computationally intractable [1]. The work in [9] tries to reduce the computational burden of the approach by adding rolling path planning and heuristic search to the neural network-based approach. The work shows that using the combined method, shorter path length with fewer turns and repeated coverage can be obtained as opposed to using the neural network-based approach alone.

B. Coverage of Complex Surfaces (3D Coverage)

Algorithms designed for coverage of planar surfaces, such as those previously discussed, typically hold assumptions on the type of the environment [1]; e.g. polygonal, differential boundaries for obstacles, rectilinear, grid-discretized, and projectively planar (2.5D). These assumptions are application-specific and simplify the problem by enabling various operations such as slicing to decompose the surface into simpler shapes and finding critical points on the boundary of obstacles for partitioning. Generalizing many of these algorithms to 3D coverage may not be appropriate since some of these assumptions do not apply to 3D coverage or applying such algorithms to 3D coverage may not be efficient. Thus, many researchers had developed algorithms that are more tailored to 3D coverage and applications.

1) Offline Coverage: Much research related to offline 3D coverage is focused on industrial robots performing CPP for the application of spray painting, and in particular in the automobile industry [10, 11, 12]. Due to the nature of this task,the planning time prior to the spray-painting process may not be critical as the preplanned path is repeated on a large number of vehicles. In such applications, precise dimensions of the object are usually available [11], e.g. through CAD models. Having accurate dimensions of the objects can simplify many aspects of the CPP problem. For example, segmentation can be done based on topology or surface normals [11, 13], i.e. complex surfaces are segmented into simple patches, which in turn enable certain CPP algorithms to be implemented. Another example of CPP for industrial spray painting is presented in [14] where regular surfaces are considered (e.g. cylindrical, conical or spherical surfaces). A similar application to robot spray-painting is robotic grit-blasting. In [15], trajectory planning and path planning are combined, and a Genetic Algorithm based method is employed to optimize the planning for a grit-blasting robot.

2) Online and Adaptive Coverage: Online or adaptive coverage is useful for Unmanned Aerial Vehicles (UAVs) [16] and Autonomous Underwater Vehicles (AUVs) [17] performing various tasks. The work in [16] modifies a simple back and forth coverage path (lawnmower-like motion) to reduce time and energy consumption for a UAV while considering photogrammetric constraints. An adaptive CPP algorithm for coverage of underwater structures is proposed by [18] where periodically part of the path is reshaped online using the STOMP algorithm by taking into account the new information obtained from the range-sensing sonars. The work in [19] considers closed and orientable surfaces that are embedded in $\mathbb{R}^3$ to be covered by industrial robots. This work modifies the Morse decomposition method to be applicable to non-planar surfaces.

C. Exploration and Surface Representation

Exploration of an environment by robots' sensors and CPP are two separate problems in robotics that have some similarities. For example, in both problems, the goal is usually to achieve coverage of the entire environment (or areas of interest) through optimal paths. A sampling-based receding horizon approach to the exploration problem is presented in [20], where the approach utilizes two nested steps: the first step aims to generate a finite-step path to maximize the amount of space to be explored; and the second step takes the first viewpoint on the path as a goal point for a path that ensures tracking of the good landmarks and maintaining low-uncertainty belief. In another similar approach [21], sampling-based receding horizon approach is used to explore the environment and to inspect surfaces within the environment.

Although a coverage path generated through a CPP algorithm may be used to explore an environment; it may be a poor-quality path for exploration. This is because the fundamental difference between exploration and CPP problem is that the path generated for exploration may not need to visit every single point in the environment in order for the robot's sensor to observe or map the entire environment; whereas in CPP the coverage path is typically designed to visit every single point on the area of interest. Consider a robot deployed in an office environment as an example. If the robot's mission is to explore, then a straight path through a corridor may be sufficient to explore and map the corridor; whereas if the robot's mission is to perform CPP (e.g. to clean the floor) then another methodical and longer path may be needed to visit every point on the corridor floor.

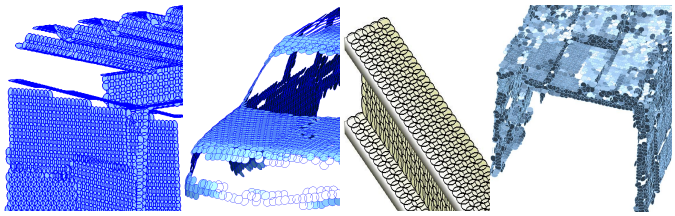


Fig. 2: Representation of various objects' surfaces using targets generated from point cloud data (left to right: part of a bridge, a vehicle, an I-beam, and a box-like structure).

In this paper, it is assumed that a map of the environment is given to the robot. If this map is not available, then an exploration of the environment [22] may be needed which can result in a point cloud representation of the environment. The point cloud is then used to generate target representation [23] of the surfaces, as shown in Fig. 2. Unlike many existing methods, such as those that rely on uniform grid cells to represent surfaces [1], the proposed approach can plan over targets which may *not* be arranged perfectly in a regular or systematic pattern, such as those illustrated in Fig. 2.

D. Advantages of the Proposed Approach over Existing Work

The proposed approach has the benefit of being generic in that it can be applied to mobile robots performing coverage on planar surfaces or industrial robots operating on objects with complex geometric shapes. Thus, the approach is capable of handling arbitrary bent 2D manifolds (and not just piecewise planar surfaces). It has the additional benefit of being adaptive such that it can be used for environments where prior knowledge of the stationary or dynamic obstacles is not available, and changes to the environment (e.g. coverage area) can occur. To the best of authors' knowledge, no approach has both of the aforementioned benefits combined. Most online CPP algorithms cannot handle dynamic obstacles, or cannot be applied to objects with complex shapes. For example, in the aforementioned research works [6, 7, 4], the suitability of the approaches for objects with complex geometric shapes and dynamic obstacles is not investigated. Although neural network-based approaches, such as the approach presented in [9], can handle dynamic obstacles; they rely on uniform grid-discretized surfaces, require iterative update of the map which can be computationally intractable for large environments, and model strongly depends on parameter setting (6 parameters to tune). It is shown later in the paper that the proposed approach outperforms neural network-based approaches. Regarding CPP algorithms developed for AUVs and UAVs, adaptability with respect to dynamic obstacles is not of main concern; e.g. in

[18] the adaptability is mainly to minimize the distance from the executed path to the nominal path (created offline) by taking into account the uncertainty in the vehicle's state. Some approaches developed for industrial robots can handle objects with complex geometric shapes; however, these approaches are not adaptive to sudden changes and cannot swiftly deal with environments where dynamic or stationary obstacles can unexpectedly become present, and typically limitations apply (e.g. closed orientable surfaces) or segmentation is needed. Various algorithms to multirobot coverage exist, e.g. multirobot forest coverage algorithm [24] and an algorithm for multirobot persistent coverage [25]. These approaches are not shown to be adaptive to dynamic obstacles, or cannot be applied to objects with complex shapes.

An additional advantage of the proposed approach is that, unlike many of the existing methods, it is able to manage the imperfect representation of the surface due to errors associated with sensing and generating the targets from the point cloud, as shown in Fig. 2. Thus, the approach can plan over targets that are irregularly arranged. Many existing algorithms have a simple implementation, and the aforementioned benefits of the presented approach don't come at the expense of implementation simplicity. The approach has only two parameters to optimize for a given environment (learned offline from prior info). Furthermore, the approach is computationally tractable for real-time adaptability (asymptotic time complexity of $\mathcal{O}(\log n)$) since the robot, during the real-time deployment and at each step, takes decisions only based on limited information and does not need to perform iterative update of the entire map for planning.

III. Problem Description

The CPP problem is to plan a path such that the robot covers all points representing a target area of a surface. This target area is henceforth referred to as the *coverage area*. The robot achieves complete coverage by utilizing its end-effector tool (e.g. buffing pad, pressure cleaning nozzle, cleaning head of a robotic vacuum cleaner) to operate on all points representing the coverage area.

The CPP problem addressed in this paper involves unexpected changes occurring during the real-time deployment of the robot. For example, the coverage area may change in size/shape or unknown dynamic/stationary obstacles may suddenly become present in the environment. Unexpected stationary obstacles make the coverage problem challenging since the robot needs to avoid the obstacles and efficiently replan the path while still aiming to achieve complete coverage with minimal cost. Unexpected dynamic obstacles further complicate the problem since unlike stationary obstacles the occupied regions of the coverage areas vary with time.

Thus, solving the CPP with minimal cost problem for environments that are subject to unexpected changes is challenging because: (1) the robot is initially unaware of the obstacles (i.e. their size, location, and trajectory) or the changes to the coverage area, and (2) the robot is not only expected to achieve complete coverage amid unexpected changes, but it needs to do so with minimal cost, e.g. with minimal path length. Complete coverage of objects with complex geometric shapes and non-uniform decomposition of the surface (e.g. when targets are not distributed uniformly due to unavailable CAD model and reliance on sensing data) add to the complexity of the problem.

As part of this complex problem, when the robot is deployed for real-time coverage, it needs to make quick decisions at each step in terms of its next best direction of motion, especially as it needs to avoid unexpected dynamic obstacles. The goal is therefore to develop a CPP planner that is fast, adaptive and that aims to achieve a near-optimal path in real-time.

Note that even though the accuracy in sensing and tracking obstacles affect the robot's motion and its decision at each step, sensing and obstacle detection are outside the scope of the paper. It is assumed that the obstacle can be detected and its trajectory can be predicted when the obstacle enters the sensing range of the robot.

IV. Predator-Prey Inspiration for CPP

The proposed approach is inspired by the concepts of *foraging* and *risk of predation* in predator-prey relation. Although animals prefer foraging in areas with a lower risk of predation, if food is abundant in an area, they lower their vigilance and select that particular area for foraging. This is because when food is abundant in an area, animals worry that they may not find such an abundance of food elsewhere and are more likely to stay in that area for foraging [26]. However, when foraging in an area "prey must select their optimal level of vigilance in response to their perceptions of a predator's whereabouts"[27]. This means that while foraging, animals continue to assess the risk of predation. The ambushing predator may not be at the exact location where the prey has perceived it to be; however, the perception on the whereabouts of the predator is based on the prey's experience, sensed noise, awareness of the habitat, etc. It is important to note that although these concepts inspired the approach and are used as a nice explanation of the intuition behind the approach, the coverage path resulting from the approach may not fully represent the behavior of a particular animal or animals in general.

Figure 1 showed a simple example where a prey assumes a square-shaped area to be a satisfying area for grazing due to the abundance of food and its faraway distance from a perceived ambushing predator. Since animals have to accomplish more than just avoiding predation (e.g. search for food), they need to continuously assess their current risk level of predation [28]. This means that while foraging, the prey will aim at visiting all regions of the target area in search of food (achieving complete coverage), but it will do so by first consuming the food in regions farthest away from the predator and gradually taking greater risks to move closer and closer to the predator. As shown in Fig. 1, this influence of the predator on the prey's overall direction of traversal causes a coverage path that is orderly which in turns helps with achieving a shorter path particularly when unexpected obstacles are introduced to the environment. The empirical verification of this finding is shown using several case studies later in the paper.

V. THE PPCPP APPROACH

The proposed approach is named Predator-Prey Coverage Path Planning (PPCPP).

Definition 1. The *prey* is the coverage spot with a size equivalent to the coverage size of the robot's end-effector tool. For example, for the floor cleaning robot, the prey is the coverage spot of the cleaning unit. Similarly, for the spray painting or grit-blasting industrial robots, the prey is the spray painting or the grit-blasting spot on the surface. The prey's location at step k is denoted as $\boldsymbol{o}_k$.

Definition 2. The *predator*, denoted as $\boldsymbol{\Psi}$, is a virtual point outside the coverage area of the robot and it is inputted to the PPCPP algorithm. The predator is considered to be stationary (e.g. an ambushing predator). The predator is not an obstacle, although it can also be placed inside one of the known obstacles. The predator enforces a spatial order for the prey, and as a result, a shorter path can be obtained in real-time as will be shown later in the paper. The predator may be placed closest to a region of the coverage area where it is preferred for the coverage path to end, as shown in Fig. 1.

Algorithm 1 illustrates a pseudo-code of the real-time implementation of PPCPP. Table I provides a list of symbols used and their definitions. Sets and vectors are written using bold uppercase and bold lowercase letters, respectively; whereas functions and constants (or single variables) are written using non-bold uppercase and non-bold lowercase letters, respectively. The only exception is the vector $\boldsymbol{\Psi}$ denoting the location of a predator which is written in uppercase.

Definition 3. Let the surface of interest be represented as circular disks, henceforth referred to as *targets*. Target representation of several surfaces was shown in Fig. 2. The size and the density of the targets can be determined based on the intended application and the properties of the end-effector tool (e.g. the nozzle coverage size in a grit-blasting robot). In Fig. 1, the targets are the grazing spots of the prey. As part of the input to Alg. 1 are the targets that are *reachable* by the robot, i.e. $\boldsymbol{O}^r \subseteq \boldsymbol{O}$ where $\boldsymbol{O}$ is a set containing all the targets that represent the surface. Other inputs are shown at the top of Alg. 1.

Definition 4. The ith *neighbor* of $\boldsymbol{o}_k$, i.e. $\boldsymbol{o}_i \in \boldsymbol{N}(\boldsymbol{o}_k)$, is defined as a neighboring target adjacent to $\boldsymbol{o}_k$. A target $\boldsymbol{o} \in \boldsymbol{O}^r$ belongs to the neighboring set $\boldsymbol{N}(\boldsymbol{o}_k)$ only if $\|\boldsymbol{o}-\boldsymbol{o}_k\| \leq r$. For a uniform decomposition of a surface, r is the distance to the diagonal neighbor (i.e. $r=\sqrt{a^2+a^2}$ where a is the Euclidean distance from $\boldsymbol{o}_k$ to the nearest neighbor). For a uniform grid, this definition is equivalent to the 8-connectivity used in the image processing field. Other distance measures may be used.

A total reward function, R, is designed as a heuristic for the prey to select its next best move at each step k ($k=1,2,\cdots,n^k$ where ideally n^k is equal to n^{O^r} which is the total number of targets in the set $\boldsymbol{O}^r$). Thus, at step k, the prey evaluates R for all neighbors and moves to the neighbor with maximal reward. The function R comprises of three rewards: rewards for maximizing the distance to a predator (predation avoidance reward), continuing motion in a straight direction (smoothness reward), and covering the boundary (boundary reward). More specifically, the total reward function is:

$$R(\boldsymbol{o}_j) = R^d(\boldsymbol{o}_j) + \omega^s\left(R^s(\boldsymbol{o}_j)\right) + \omega^b\left(R^b(\boldsymbol{o}_j)\right) \qquad (1)$$

where $R^d(\boldsymbol{o}_j)$, $R^s(\boldsymbol{o}_j)$ and $R^b(\boldsymbol{o}_j)$ are the predation avoidance reward, smoothness reward, and boundary reward, respectively; ω^s and ω^b (input to Alg. 1) are the weighting factors as-

TABLE I: Nomenclature

Symbol	Definition
$\boldsymbol{O}$	All *targets* representing the surface
$\boldsymbol{O}^r \subseteq \boldsymbol{O}$	*Reachable* targets representing reachable areas
$\boldsymbol{O}_k^c \subseteq \boldsymbol{O}^r$	All *covered* targets by the prey up-to step k
$\boldsymbol{O}_k^u \subseteq \boldsymbol{O}^r$	All *uncovered* targets at step k
$\boldsymbol{O}_k^o \subseteq \boldsymbol{O}^r$	All targets predicted to be *occupied* by obstacles at step k
$\boldsymbol{N}(\boldsymbol{o}_k) \subset \boldsymbol{O}^r$	All *neighbors* of the prey target $\boldsymbol{o}_k$
$\boldsymbol{N}^{u^f}(\boldsymbol{o}_k) \subseteq \boldsymbol{O}_k^u$	*Neighboring uncovered* and obstacle-*free* targets of the prey target $\boldsymbol{o}_k$
$\boldsymbol{N}^u(\boldsymbol{o}_j) \subseteq \boldsymbol{O}_k^u$	*Neighboring uncovered* targets of the jth neighbor of the prey at step k
$\boldsymbol{o} \in \boldsymbol{O}^r$	A *target* from the reachable areas of the surface
$\boldsymbol{o}^s$	*Start* target of the prey
$\boldsymbol{o}_k$	Prey location at step k
$\boldsymbol{o}_{k-1}$	Target covered by the prey at previous step, $k-1$
$\boldsymbol{o}_i \in \boldsymbol{N}(\boldsymbol{o}_k)$	ith *neighbor* of the prey at step k
$\boldsymbol{o}_j \in \boldsymbol{N}^{u^f}(\boldsymbol{o}_k)$	jth *uncovered* and obstacle-free *neighbor* of the prey at step k
$\boldsymbol{o}_{j_k^*} \in \boldsymbol{N}^{u^f}(\boldsymbol{o}_k)$	*neighbor* with maximal reward at step k
k	Index representing the current step number
i	Index of a neighbor
j	Index of an uncovered and unoccupied neighbor
j_k^*	Index of a neighbor with max. reward at step k
$\boldsymbol{\Psi}$	*Predator* location
$n^{N_{max}}$	*Max.* possible number of *neighbors* of a target
n^O	No. of *targets* that represent the surface
n^{O^r}	No. of *reachable* targets, i.e. size of the set $\boldsymbol{O}^r$
$n^N(\boldsymbol{o}_j)$	No. of *neighbors* of the jth prey's neighbor
n^k	No. of steps associated with a prey's path
t_{max}	Maximum time allocated to the coverage task
t	Current execution time at step k
r	The *radius* of a sphere within which targets are considered to be neighbors of a target/prey
$D_{max}(\boldsymbol{o}_k)$	$\max_j \|\boldsymbol{o}_j - \boldsymbol{\Psi}\|$
$D_{min}(\boldsymbol{o}_k)$	$\min_j \|\boldsymbol{o}_j - \boldsymbol{\Psi}\|$
$R^d(\boldsymbol{o}_j)$	*Predation avoidance* reward associated with $\boldsymbol{o}_j$
$R^s(\boldsymbol{o}_j)$	*Smoothness* reward associated with $\boldsymbol{o}_j$
$R^b(\boldsymbol{o}_j)$	*Boundary* reward associated with $\boldsymbol{o}_j$
$R(\boldsymbol{o}_j)$	Total *reward* associated with $\boldsymbol{o}_j$
ω^s	Weighting factor for the *smoothness* reward
ω^b	Weighting factor for the *boundary* reward
$\boldsymbol{Z}$	Design variables, which are (ω^s, ω^b)
$\boldsymbol{P_Z}$	A *path* generated based on the values of $\boldsymbol{Z}$
$L(\boldsymbol{P_Z})$	*Length* of the path $\boldsymbol{P_Z}$

Algorithm 1 Real-time PPCPP

Input: reachable targets $\boldsymbol{O}^r$; start target $\boldsymbol{o}^s$; predator $\boldsymbol{\Psi}$; weighting factors (ω^s, ω^b); No. of nearest neighbors $n^{N_{max}}$; neighborhood radius r; max. allocated time t_{max}

1: **Initialize:** $k \leftarrow 1$; $\boldsymbol{O}^u_k \leftarrow \boldsymbol{O}^r$; $\boldsymbol{O}^c_k \leftarrow \boldsymbol{o}^s$; $\boldsymbol{o}_k \leftarrow \boldsymbol{o}^s$; $\boldsymbol{O}^o_k \leftarrow \emptyset$
2: **while** $\boldsymbol{O}^u_k \neq \emptyset$ **or** $t < t_{max}$ **do**
▷ Scan the environment and update the map
3: $\boldsymbol{O}^o_k \leftarrow$ Scan&Update$(\boldsymbol{O}^r, \boldsymbol{O}^o_k)$ ▷ Remark 1
▷ Search for $n^{N_{max}}$ nearest neighbors (e.g. using k-d tree) and keep those that are within a distance r from $\boldsymbol{o}_k$
4: $\boldsymbol{N}(\boldsymbol{o}_k) \leftarrow$ NearestNeighbors$(\boldsymbol{O}^u_k, \boldsymbol{o}_k, n^{N_{max}}, r)$
▷ Determine the obstacle-free neighbors of the prey
5: $\boldsymbol{N}^{u^f}(\boldsymbol{o}_k) \leftarrow \boldsymbol{N}(\boldsymbol{o}_k) \setminus \{\boldsymbol{N}(\boldsymbol{o}_k) \cap \boldsymbol{O}^o_k\}$
▷ Determine the index of the neighbor with max. reward
6: **for** $j = 1$ to Size$\big(\boldsymbol{N}^{u^f}(\boldsymbol{o}_k)\big)$ **do**
7: $\boldsymbol{N}(\boldsymbol{o}_j) \leftarrow$ NearestNeighbors$(\boldsymbol{O}^u_k, \boldsymbol{o}_j, n^{N_{max}}, r)$ ▷ where $\boldsymbol{o}_j \in \boldsymbol{N}^{u^f}(\boldsymbol{o}_k)$
8: $\boldsymbol{N}^u(\boldsymbol{o}_j) \leftarrow \big\{\boldsymbol{o}_m \notin \boldsymbol{O}^o_k$ where $\boldsymbol{o}_m \in \boldsymbol{N}(\boldsymbol{o}_j);\ \forall m : m = 1, 2, \ldots,$Size$\big(\boldsymbol{N}(\boldsymbol{o}_j)\big)\big\}$
▷ Calculate reward for $\boldsymbol{o}_j$ using Eqs. (2) to (5)
9: $R_j \leftarrow$ Reward$(\omega^s, \omega^b, \boldsymbol{o}_j, \boldsymbol{N}^u(\boldsymbol{o}_j), \boldsymbol{o}_k, \boldsymbol{o}_{k-1}, n^{N_{max}}, \boldsymbol{\Psi})$
10: **end for**
11: $j^*_k \leftarrow \arg\max_j(R_j)$
▷ Update states
12: $\boldsymbol{o}_{k+1} \leftarrow \boldsymbol{o}_{j^*_k}$ ▷ prey moves to the best neighbor
13: $k \leftarrow k+1$
14: $\boldsymbol{O}^c_k \leftarrow \boldsymbol{O}^c_{k-1} \cup \boldsymbol{o}_{j^*_k}$; $\boldsymbol{O}^u_k \leftarrow \boldsymbol{O}^u_{k-1} \setminus \boldsymbol{o}_{j^*_k}$; $\boldsymbol{O}^o_k \leftarrow \boldsymbol{O}^o_{k-1}$
15: **end while**

sociated with the smoothness reward and the boundary reward, respectively. The values of the weighting factors govern the extent to which each reward is emphasized by the prey when deciding on the next movement. The weighting factors need to be optimized only once for a given environment prior to the real-time deployment and based on the initial knowledge of the environment. It is shown in Section VII that when the approach is used in real-time, the same optimized weighting factors are sufficient to manage changes in the environment effectively and achieve a complete coverage path with a length close to the optimized length (length of the path optimized offline). The reward functions and the weighting factors are explained in more details in Section VI (Mathematical Modeling).

Remark 1 (Scan and Update)**.** At first, by scanning the environment (line 3 of Alg. 1), obstacles are detected. Then, the map of the environment is updated accordingly and the set $\boldsymbol{O}^o_k$ which contains the targets that are predicted to be *occupied* by obstacles at step k is modified. That is, by performing the Scan&Update procedure in line 3, the aim is to remove the targets that are predicted to have become obstacle-free at step k from the set $\boldsymbol{O}^o_k$, and conversely, to add the targets that are predicted to have become occupied by obstacles to the set $\boldsymbol{O}^o_k$.

At each step k, prey's neighbors, $\boldsymbol{N}(\boldsymbol{o}_k) \subseteq \boldsymbol{O}^u_k$, are determined (line 4 of Alg. 1). A subset of the targets in $\boldsymbol{O}^u_k$ may become occupied by stationary or dynamic obstacles, hence, each target $\boldsymbol{o}_i \in \boldsymbol{N}(\boldsymbol{o}_k)$ is checked and if it is not occupied by any obstacle then it is added to the list of *uncovered* and *obstacle-free* neighbors, $\boldsymbol{N}^{u^f}(\boldsymbol{o}_k)$ (line 5). Note that $\boldsymbol{O}^u_k$ contains both occupied and obstacle-free targets that are uncovered; whereas $\boldsymbol{N}^{u^f}(\boldsymbol{o}_k)$ only contains the neighboring obstacle-*free* targets of the prey that are *uncovered*. The prey continues to cover all targets until the set of uncovered targets, $\boldsymbol{O}^u_k$, becomes empty or until the current execution time, t, exceeds the maximum time, t_{max}, allocated to the coverage task (line 2).

From $\boldsymbol{N}^{u^f}(\boldsymbol{o}_k)$, the index j^*_k (line 11) of the neighbor that results in maximum reward at step k is calculated (lines 6 to 11 of Alg. 1) and the corresponding best neighbor, $\boldsymbol{o}_{j^*_k}$, is selected for the prey (line 12). R_j in line 9 is the total reward for $\boldsymbol{o}_j \in \boldsymbol{N}^{u^f}(\boldsymbol{o}_k)$. The states are then updated (lines 12 to 14), and the process is repeated for $k+1$th step (next loop).

Definition 5. A *dead-end* is when the prey arrives at a target where all neighbors are already covered. In this case, the prey needs to repeat coverage of a certain number of targets in order to reach an uncovered target. The coverage task (PPCPP) resumes when the prey reaches an uncovered target.

Remark 2 (Dead-end Recovery)**.** If the prey arrives at a dead-end, PPCPP is temporarily stopped. Then, by switching to a suitable point-to-point path planner, such as A^* or Dijkstra's algorithm, the shortest path from the dead-end target to the nearest uncovered target is found. PPCPP resumes when the prey reaches an uncovered target. The uncovered target to reach is an intermediate goal target and is a target closest to both the already covered targets and the prey. The intermediate goal target can change or may become irrelevant since the environment may change at each step of the prey's movement, e.g. an obstacle may move and the dead-end situation may no longer exist.

Remark 3 (Computational Complexity)**.** Using a k-d tree structure for finding the m-nearest neighbors (line 4 of Alg. 1) of a point (target) in a tree, the time complexity is $\mathcal{O}(m \log n)$ where n is the total number of points in the tree which is the same as n^{O^r} (number of targets in $\boldsymbol{O}^r$). To check for a point in the tree, the complexity is $\mathcal{O}(\log n)$. Thus, the time complexity of the procedure in line 5 for obtaining the obstacle-free neighbors is $\mathcal{O}(m \log n)$ where m is the number of nearest neighbors of the prey. For the 'for-loop' (lines 6 to 10), the time complexity is $\mathcal{O}(2m^2 \log n)$, since within the loop (i.e. for each j) the m-nearest neighbors for the target $\boldsymbol{o}_j$ needs to be found (line 7) and each neighbor needs to be checked to determine whether or not it is in $\boldsymbol{O}^o_k$ (line 8). To insert and delete a point from the k-d tree (line 14), the time complexity is $\mathcal{O}(\log n)$. Therefore, the overall time complexity is $\mathcal{O}\big((2m^2 + 2m + 2)\log n\big)$, and the dominant term is $\mathcal{O}(m^2 \log n)$. Note that m (number of neighbors of any target) is small since CPP is concerned with the surface of an object. For uniform decomposition of the surface, it is reasonable to consider a maximum number of $m = 8$ neighbors as can be seen in Fig. 1. If surface decomposition is not uniform, the number of neighbors can be slightly more than 8.

Thus, the asymptotic time complexity for the robot to make a decision for moving to the next best neighbor during the real-time deployment is $\mathcal{O}(\log n)$. Note that the above analysis doesn't consider the time complexity of the algorithm used for scanning the environment (line 3 and Remark 1).

Remark 4 (Complete Coverage)**.** Suppose an environment is not subject to any changes and stationary or dynamic obstacles do not appear unexpectedly. Under such conditions, the PPCPP algorithm combined with the dead-end recovery (Remark 2) can achieve complete coverage of the surface because the main stopping criterion for the coverage task of the prey is when all the targets in the set $\boldsymbol{O}_k^u$ are covered (line 2 of Alg. 1). At each step k, the target covered by the prey is added to the set $\boldsymbol{O}_k^c$ and removed from the set $\boldsymbol{O}_k^u$ (line 14). This prevents the prey from getting stuck in revisiting targets since it is restricted in selecting a target from the set $\boldsymbol{O}_k^u$ only. The prey is allowed to repeat coverage only when PPCPP is temporarily stopped as a result of a dead-end situation. PPCPP is resumed only after reaching an uncovered target (Remark 2). Hence, when the set $\boldsymbol{O}_k^u$ becomes empty, then the entire environment is covered and the coverage task is stopped. If dynamic obstacles are present in the environment, complete coverage of the surface is still possible since the prey will continue covering the uncovered targets from the set $\boldsymbol{O}_k^u$ until it becomes empty.

There are however two scenarios in which the prey may not achieve complete coverage. The first scenario is related to a stationary obstacle unexpectedly occupying part of the surface that has not yet been covered by the prey. In this circumstance, after covering all other obstacle-free targets, PPCPP will continue to attempt covering the occupied targets in the hope that they become unoccupied, until the current execution time, t, exceeds the maximum time, t_{max}, allocated to the coverage task. The second scenario is concerned with the rare event that the prey, after covering numerous unoccupied targets, fails at covering the remaining targets that are being periodically occupied by dynamic obstacles and ends up being stuck in a cyclic behavior. This cyclic behavior causes the prey to repeatedly move between regions that are periodically occupied by obstacles, and fails to cover any of these regions due to the unfortunate timing or sequence of the prey moving between these regions relative to the motion of the obstacles. A strategy to resolve this issue is to add randomness to the decision making when the prey fails to cover these periodically occupied regions after n attempts. For example, the intermediate goal target (defined in Remark 2) can be selected randomly from the uncovered targets, or the prey selects its next best move randomly for a small number of steps. This randomness would aim to break the cycle after which normal coverage will continue using the PPCPP algorithm. Although these scenarios are very rare (did not occur in any of the case studies presented in Section VII), the strategies for addressing these issues can be further investigated as future work for specific applications that may encounter such scenarios.

VI. Mathematical Modeling of Reward Functions and Optimization of Weighting Factors

In this section, the mathematical modeling related to the reward functions and the weighting factors is presented.

A. Reward Functions

*1) Reward for Moving Away from the Predator (**Predation Avoidance Reward**):* The farther away the prey is from the predator, the lower the risk of predation. Thus, at each step, the prey maximizes its reward by moving towards a neighbor that is uncovered (not yet covered) and that has the farthest distance from the predator. The function for calculating the predation avoidance reward for the prey moving to the jth neighbor, $\boldsymbol{o}_j$ is formulated as:

$$R^d\left(\boldsymbol{o}_j\right) = \frac{D(\boldsymbol{o}_j) - D_{min}(\boldsymbol{o}_k)}{D_{max}(\boldsymbol{o}_k) - D_{min}(\boldsymbol{o}_k)} \tag{2}$$

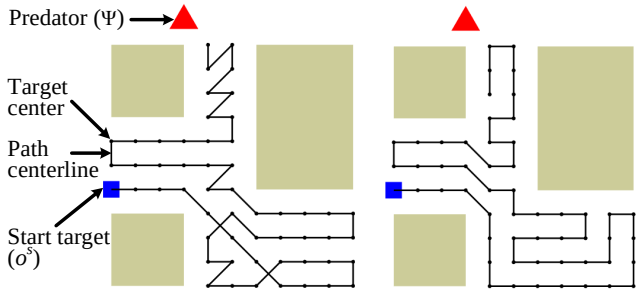
where $D(\boldsymbol{o}_j) = \|\boldsymbol{o}_j - \boldsymbol{\Psi}\|$ gives the distance from $\boldsymbol{o}_j$ to the predator $\boldsymbol{\Psi}$, $D_{max}(\boldsymbol{o}_k) = \max_j \|\boldsymbol{o}_j - \boldsymbol{\Psi}\|$ gives the maximum distance from one of the neighbors of the current prey target to the predator, and similarly, $D_{min}(\boldsymbol{o}_k) = \min_j \|\boldsymbol{o}_j - \boldsymbol{\Psi}\|$ gives the minimum distance. Note that $D_{max}(\boldsymbol{o}_k) - D_{min}(\boldsymbol{o}_k)$ is therefore a constant for a prey location. Based on Eq. (2), the prey obtains a maximum possible reward of 1, i.e. $R^d\left(\boldsymbol{o}_j\right) = 1$, for moving to an uncovered neighbor that is farthest away from the predator, and conversely it obtains zero reward, i.e. $R^d\left(\boldsymbol{o}_j\right) = 0$, for moving towards an uncovered neighbor closest to the predator. It gets a reward between 0 and 1 for moving to any other neighbor.

This reward provides a spatial order for the prey, i.e. the foraging behavior of the prey due to this reward will at first steer the prey towards the region farthest away from the predator, but eventually the prey will have no choice but to gradually move closer and closer to the predator in search of new food, as a result searching the entire environment leading to complete coverage. Thus, the predator enforces a direction of overall motion for the prey, as was shown in Fig. 1. This behavior prevents the prey from moving back and forth from one region to another simply to cover what it has missed in the previous visit, thus resulting in a shorter path as will be demonstrated later in the paper using several case studies (Section VII).

*2) Reward for Continuing Motion in a Straight Direction (**Smoothness Reward**):* For many applications, it is desirable to have the prey move in a more regular pattern than the pattern shown in Fig. 1; i.e., to move in a straight direction, make fewer turns, and try to cover the boundary. Thus, two additional rewards are considered for the prey, one of which is related to the prey continuing in the direction of its motion and only turning when it encounters a boundary or a dead-end. Having a path that has more straight lines (fewer turns) can be beneficial for certain robots and applications, e.g. mobile robots that consume more energy or time due to frequent turns. This reward function can also help with obtaining a shorter path, as will be explained later.

The second reward function is formulated as follows:

$$R^s\left(\boldsymbol{o}_j\right) = \frac{\angle\, \boldsymbol{o}_{k-1}\boldsymbol{o}_k\boldsymbol{o}_j}{180^\circ} \tag{3}$$



(a) Without boundary reward. (b) With boundary reward.

Fig. 3: Resulting path with and without the boundary reward.

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, $\boldsymbol{o}_k$, due to the angle $\angle \boldsymbol{o}_{k-1}\boldsymbol{o}_k\boldsymbol{o}_j \in (0^\circ, 180^\circ]$ which is the angle between the vectors $(\boldsymbol{o}_{k-1} - \boldsymbol{o}_k)$ and $(\boldsymbol{o}_k - \boldsymbol{o}_j)$, and $\boldsymbol{o}_{k-1}$ is the target covered by the prey at the previous step $(k-1)$.

*3) Reward for Covering Boundary Targets (**Boundary Reward**):* Another reward for the prey is related to covering the boundary targets.

Definition 6. Let the *boundary targets* be the targets that represent the boundary of the surface as well as the targets that are on the boundary of the uncovered regions, i.e. the uncovered targets closest to the already covered region of the surface (targets closest to $\boldsymbol{O}_k^c$).

At each step, the prey will be given extra reward for covering a boundary target. Figure 3a shows an example where a prey traverses a surface *without* utilizing the boundary reward. In Fig. 3b, the prey traverses the same surface but utilizing the boundary reward resulting in a shorter path length with fewer turns and interlacing. For a fair comparison, the relevant weighting factors in Eq. (1) were optimized for both cases (also explained in the following subsection).

The third reward function is formulated as follows:

$$R^b(\boldsymbol{o}_j) = \frac{n^{N_{max}} - n^N(\boldsymbol{o}_j)}{n^{N_{max}}} \tag{4}$$

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, $\boldsymbol{o}_k$, and $n^N(\boldsymbol{o}_j)$ calculates the number of uncovered neighbors of the target $\boldsymbol{o}_j$. $n^{N_{max}}$ is the maximum possible number of neighbors for a target, and can be determined based on the target decomposition of the surface. If the aim is to achieve a uniform decomposition of the surface, then a value of 8 for $n^{N_{max}}$ is reasonable since CPP is concerned with the surface of an object. For this reward function, the smaller the number of uncovered neighbors for the target $\boldsymbol{o}_j$, the higher the reward.

4) 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.:

$$R(\boldsymbol{o}_j) = R^d(\boldsymbol{o}_j) + \omega^s\left(R^s(\boldsymbol{o}_j)\right) + \omega^b\left(R^b(\boldsymbol{o}_j)\right) \tag{5}$$

where ω^s and ω^b are the weighting factors associated with the smoothness and the boundary reward functions, respectively. Note that factors such as the geometric shape of the object, the location of the predator, and the decomposition of the surface (arrangement of the targets) can influence the significance of each reward for a new environment. Hence, for a given environment, the weighting factors are optimized first, but only once prior to the real-time deployment of the robot.

5) Maximum Reward at Step k: At step $k (k = 1, 2, \ldots, n^k)$, the index of the neighbor that gives the maximum reward can be found:

$$j_k^* = \arg\max_j \left(R(\boldsymbol{o}_j \in \boldsymbol{N}^{uf}(\boldsymbol{o}_k))\right) \tag{6}$$

where $R(\boldsymbol{o}_j)$ is calculated based on Eq. (5). Thus at step k, the prey will move to the target $\boldsymbol{o}_{j_k^*} \in \boldsymbol{N}^{uf}(\boldsymbol{o}_k)$, the current prey target $\boldsymbol{o}_k$ will become $\boldsymbol{o}_{j_k^*}$, and the process is repeated until all targets are covered. If $\boldsymbol{N}^{uf}(\boldsymbol{o}_k)$ is empty at any step, i.e. the path is in a dead-end (Definition 5 and Remark 2), then the index j_k^* is the index of the already covered neighbor that the point-to-point path planner decides to go to next.

B. Mathematical Modeling for Optimizing Weighting Factors

Prior to the real-time deployment of the robot for executing the coverage task, the weighting factors within the total reward function (Eq. (5)) need to be optimized. It is shown empirically (Section VII) that using the same offline learned weighting factors, the prey can adapt to changes in real-time while achieving a path length close to the optimized path length. The offline optimization iteratively changes the values of the design variables (weighting factors) with the aim of improving the path length (cost). In each iteration, a complete path that covers the entire surface is generated using the same process used for real-time deployment (Alg. 1) combined with the dead-end recovery procedure (Remark 2); however, only utilizing the available knowledge of the environment. Note that optimizing the weighting factors may not be the only necessary condition for obtaining the globally optimal path. The following is the mathematical modeling for optimizing the weighting factors. Various optimization algorithms may be employed to optimize the weighting factors; detailed study and comparison of the algorithms for optimizing the weighting factors are outside the scope of the paper.

1) Design Variables: The design variables, $\boldsymbol{Z}$, for the optimization problem are the weighing factors, i.e.:

$$\boldsymbol{Z} = (\omega^s, \omega^b). \tag{7}$$

2) Design Objective:

Remark 5 (Path Cost)**.** The cost of a path is defined with respect to its length. Thus, a path that covers all targets representing the surfaces of interest is considered optimal if its length is minimal. Other cost functions, such as minimal coverage time, may also be used.

Minimizing the total length of the path has the added benefit of reducing the number of instances where the path crosses itself or ends up in a dead-end (Definition 5). This is because, for a complete coverage path to be minimal in length, the prey needs to prevent repeated coverage that is caused by the

prey crossing its path or ending up in a dead-end. Recall that in a dead-end situation, the prey needs to repeat coverage of a certain number of targets in order to reach the nearest uncovered target and then resume the coverage task.

The objective function for the optimization problem is:

$$\min_{\boldsymbol{Z}} f(\boldsymbol{Z}) = L(\boldsymbol{P_Z}) \tag{8}$$

where $L(\boldsymbol{P_Z})$ is the length of the path $\boldsymbol{P_Z}$ which is generated using the function π_1 which concatenates the targets in the proper sequence (iteratively based on Eq. (6)) with the current values of the design variables taken into account:

$$\pi_1 : \boldsymbol{Z} \mapsto \boldsymbol{P_Z} = \left\{ \boldsymbol{o}^s, \boldsymbol{o}_{j_1^*}, \boldsymbol{o}_{j_2^*}, \ldots, \boldsymbol{o}_{j_{n^k}^*} \right\}, \tag{9}$$

and a target in the generated path is determined by:

$$\pi_2 : j_k^* \mapsto \boldsymbol{o}_{j_k^*}. \tag{10}$$

where π_2 is the function that derives the corresponding target from the index j_k^* (in Eq. (6)).

Therefore, the length of the path is

$$L(\boldsymbol{P_Z}) = \left\| \boldsymbol{o}^s - \boldsymbol{o}_{j_1^*} \right\| + \sum_{k=1}^{n^k-1} \left\| \boldsymbol{o}_{j_k^*} - \boldsymbol{o}_{j_{k+1}^*} \right\|, \tag{11}$$

and the aim is to obtain a path with minimal length through appropriate values of the design variables $\boldsymbol{Z}$.

VII. Case Studies

Seven case studies are used to validate the approach. The first case study investigates the practicality and benefits of the reward functions and the effect of the predator location on the overall path. In the second case study, eight scenarios are used with stationary obstacles that are arbitrarily populated in the environment to perform various comparative studies such as comparing to: (1) an ideal path (lower bound on the optimum), (2) a path optimized offline using a method to the traveling salesman problem (TSP), (3) a path optimized offline using the optimized weighting factors, (4) a boustrophedon-based path using the online BA* algorithm [4], (5) random neighbor selection, and (6) paths generated by considering different variations of PPCPP (setting different rewards to zero). Case Study 3 is designed to compare PPCPP to approaches that can handle dynamic obstacles on planar surfaces. Many scenarios are used in Case Studies 5 to 7 for robust empirical validation of PPCPP for surfaces embedded in $\mathbb{R}^3$ with and without dynamic obstacles. In Case Study 4, the target surface changes unexpectedly but no dynamic obstacles are considered. In Case Study 5, various scenarios with a single dynamic obstacle are considered. In Case Study 6, various scenarios with multiple dynamic obstacles (having different sizes and speeds) are considered. In Case Study 7, a different surface with a varying speed obstacle is used.

The robot has no prior knowledge of the stationary or dynamic obstacles that may become present in the environment and is unaware of their size, location, and trajectory. During the real-time deployment and at each step, the robot only needs to know which of the neighboring targets are obstacle-free and selects the neighbor that results in the maximum total reward. For the case studies, it takes less than 1 millisecond on average for a robot to compute the best next neighbor. This efficiency is necessary for the robot to quickly decide on its next best move amid dynamic obstacles. The optimization time for finding the optimized weighting factors is included in each case study.

The code for testing the approach is written in MATLAB R2013a. For a given environment, optimization (as per the explanation in Section VI-B) is performed offline using Genetic Algorithm (GA) through MATLAB optimization toolbox. As the first paper to introduce this new approach, the aim has been to introduce the main concept clearly and to showcase the performance of PPCPP algorithm assuming optimized weighting factors can be obtained through an appropriate optimization algorithm. Thus, other optimization algorithms can be utilized; comparisons between optimization algorithms and selection of the most appropriate one are outside the scope of this paper. Parallel computation is enabled in MATLAB. The default settings of 'ga' from MATLAB optimization toolbox were found to be appropriate for the case studies. As per the default settings, population size is 50, crossover fraction is 0.8, maximum generations is 200, and the elite count is 3. For all case studies, GA is terminated as a result of the average relative change in the best fitness function value over 50 generations (maximum stall generations) being less than or equal to function tolerance (1e-6). All simulations were carried out using Intel Core i5-2400 CPU @ 3.10 GHz with four cores.

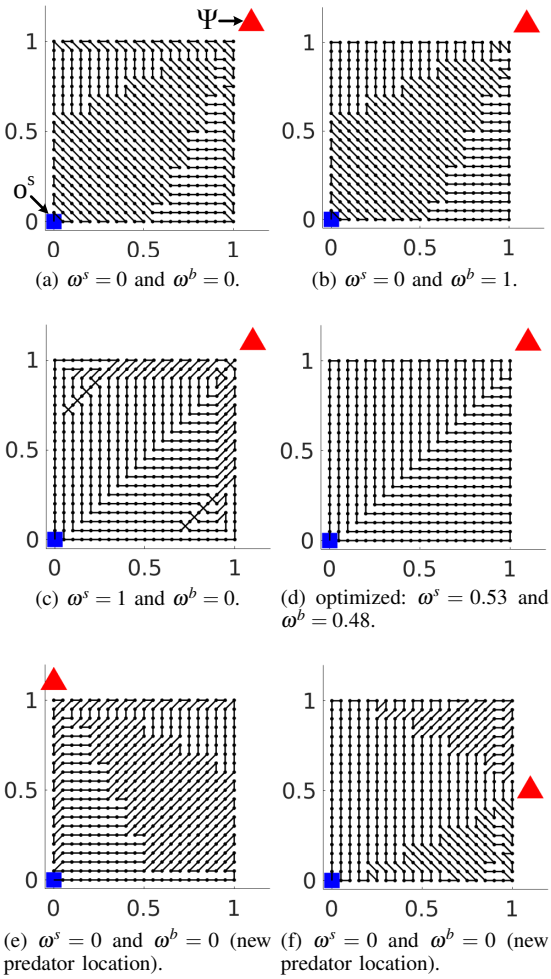
A. Case Study 1: Verifying the Reward Functions

A case study is presented in this section to validate the use of each reward function and to demonstrate how the overall path is affected by the location of a predator. A 1 m by 1 m surface is used which is represented as 441 targets with a spacing of 0.05 m between any two non-diagonal neighbors.

PPCPP is first applied with predation avoidance reward only (Fig. 4a). This path is 27.50 m in length. The path can be slightly improved (26.80 m in length) by making the boundary reward have the same weighting as the predation avoidance reward ($\omega^b = 1$), as shown in Fig. 4b where there are less diagonal motions near the boundary of the surface. Similarly, the path can be improved (23.57 m in length) by making the smoothness reward have the same weighting as the predation avoidance reward ($\omega^s = 1$), as shown in Fig. 4c where there are fewer turns and diagonal moves. However, the best performance is obtained when both weighting factors are optimized ($\omega^s = 0.53$ and $\omega^b = 0.48$), as shown in Fig. 4d. In this case, the path is optimal in that it is the shortest possible path (22 m in length) and doesn't cross itself. Running the optimization to find optimal weighting factors took 46 s (average of 20 runs).

Figures 4e and 4f are added to show how the location of the predator dictates the direction of the overall traversal for the prey. Hence, the predator can be placed closest to a target where it is preferred for the coverage path to end.

Optimization was repeated 20 times to investigate convergence and consistency of the solutions provided by GA. An optimal solution with minimal path length (22 m) was found



(a) $\omega^s = 0$ and $\omega^b = 0$. (b) $\omega^s = 0$ and $\omega^b = 1$.

(c) $\omega^s = 1$ and $\omega^b = 0$. (d) optimized: $\omega^s = 0.53$ and $\omega^b = 0.48$.

(e) $\omega^s = 0$ and $\omega^b = 0$ (new predator location). (f) $\omega^s = 0$ and $\omega^b = 0$ (new predator location).

Fig. 4: Paths generated on the surface using different values of weighting factors or different locations for the predator.

by each of the 20 optimization runs. To gain an insight into the function space and to ensure GA performs well for the problem under consideration, a grid search was performed as shown in Fig. 5. Values of 0 to 1 (in steps of 0.01) is considered for both weighting factors to conduct the grid search (total of 10,000 grids). As shown in Fig. 5, the function space is reasonably convex. Although a maximum of 200 generations is considered for GA, convergence occurred earlier than 51 generations in all 20 optimization runs. More interestingly, in each optimization run, an optimal solution was found within the first three GA generations. This quick rate of finding optimal solutions and the fast convergence can be due to the convex nature of this scenario. However, this scenario is rather simple and as will be shown in later case studies, the function space can be highly non-convex. Nonetheless, GA converged with much less number of evaluations (less than 2,550 pairs of weighting factors are evaluated) as compared to grid search (10,000 evaluations). A detailed investigation into various optimization algorithms and parameters will be conducted as future work.

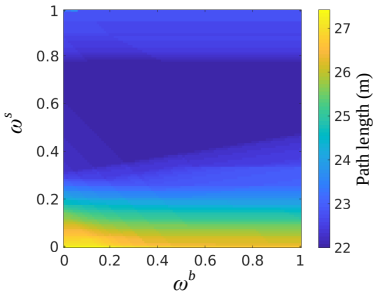


Fig. 5: Grid search of the weighting factors for the simple scenario shown in Fig. 4a.

B. Case Study 2: Coverage Amid Various Unexpected Stationary Obstacles

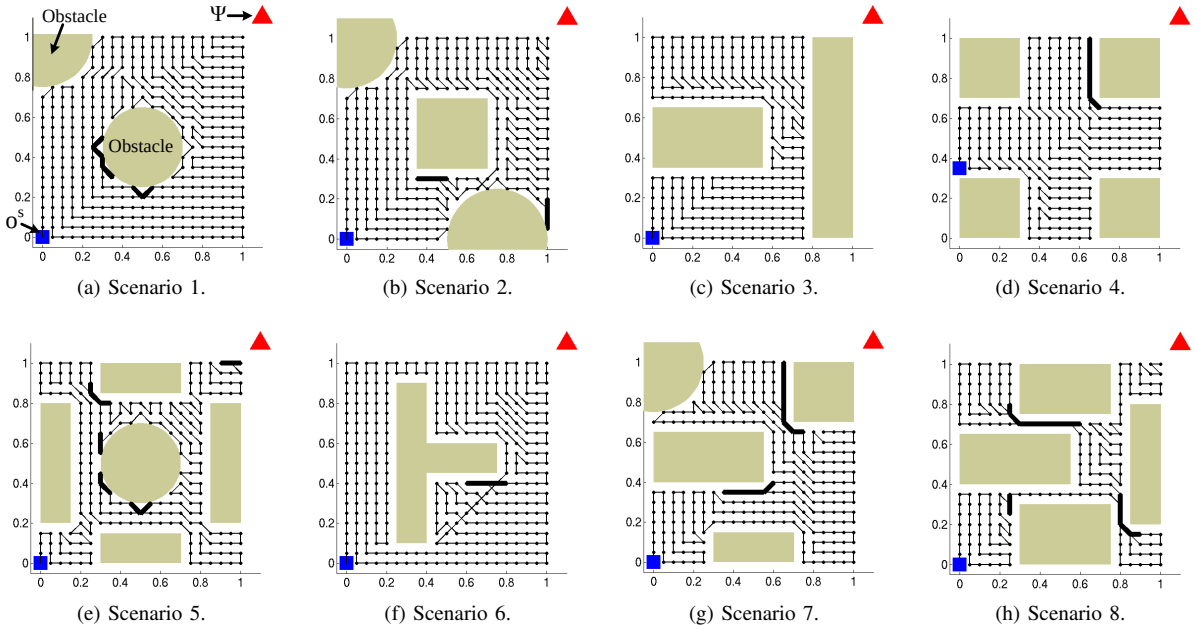
The purpose of this case study is to perform comparative studies and investigate the adaptability of the PPCPP approach with respect to various unexpected stationary obstacles. Eight scenarios are considered, and in each scenario, a number of obstacles with different shapes and sizes are arbitrarily placed in the environment, as shown in Fig. 6. The same surface as in Case Study 1 (Section VII-A), with the same optimized weighting factors ($\omega^s = 0.53$ and $\omega^b = 0.48$), is used.

For each scenario, the path that the prey travels in real-time (referred to as real-time PPCPP) is compared to the following:

- ideal path (a lower bound on optimum)
- offline TSP-based path
- offline PPCPP (optimized path assuming obstacles are known)
- real-time PPCPP with no smoothness reward ($R^s = 0$)
- real-time PPCPP with no boundary reward ($R^b = 0$)
- real-time PPCPP with $R^s = 0$ and $R^b = 0$
- real-time PPCPP with no predation avoidance reward ($R^d = 0$)
- online BA* algorithm [4]
- random neighbor selection

The results are shown in Fig. 7. It can be seen that the performance of real-time PPCPP is very close to that of offline PPCPP (1.6 % difference when averaging the 8 solutions form the 8 scenarios). Note that in offline PPCPP, the path is optimized assuming obstacles are known; whereas in real-time PPCPP, the robot is not aware of the obstacles and their sizes/locations in advance. At each step, it takes less than 1 millisecond for the prey to decide on its next best neighbor. Repeated coverage is shown as dark black lines in Fig. 6.

Obtaining an optimal path for the sake of comparison is challenging. Even for a known environment without dynamic obstacles, the problem reduces to the well known Traveling Salesman Problem (TSP) which is an NP-hard problem. Thus, to gain an insight into how far the length of a real-time PPCPP path can be from an optimal path length, two measures are considered for analysis: (1) ideal path length (a lower bound on optimum) where path length is $l = \big((\text{number of targets} - 1) \times \text{distance between two non-diagonal neighbors}\big)$; and (2) a path length through solving the TSP using GA, and improving the length by repeating the optimization many times and making manual modifications to achieve a path length as close to the



(a) Scenario 1. (b) Scenario 2. (c) Scenario 3. (d) Scenario 4.

(e) Scenario 5. (f) Scenario 6. (g) Scenario 7. (h) Scenario 8.

Fig. 6: Eight different scenarios, and a path created for each scenario in real-time using PPCPP.

ideal path length as possible. Note that the TSP-based path is feasible whereas no feasible path may exist that can achieve the ideal path length (since ideal length assumes no diagonal moves, dead-ends, or overlaps). Nevertheless, as shown in Fig. 7, the TSP-based path lengths are very close to the ideal path lengths (1.3 % difference by averaging the 8 solutions). Unlike PPCPP, smoothness of the path was not considered for the TSP-based path and an open-end TSP was implemented (i.e. the path is allowed to end at any target). Taking the average of the solutions from the 8 scenarios, the path lengths using real-time PPCPP are worse by 8.2 % relative to the ideal path lengths, and worse by 6.8 % relative to the TSP-based path.

The paths generated using real-time PPCPP are also compared to the online BA* algorithm [4] which utilizes boustrophedon motion and A* search. Taking the average of the solutions from the 8 scenarios, the real-time PPCPP outperforms BA* by 0.33 %. This difference is small; however, there are additional benefits to using PPCPP that is not shown to be present in algorithms that utilize boustrophedon-like motion. Using a boustrophedon approach, the path may not end close to a region of interest, whereas using PPCPP the path does end in the region closest to the predator as shown in Fig. 6. This behavior can be beneficial for some applications. Other advantages of PPCPP is that it can handle dynamic obstacles, unexpected changes, and surfaces embedded in $\mathbb{R}^3$ as will be illustrated in the following case studies.

Figure 7 also shows how the real-time PPCPP performs when any of the three rewards is not considered (i.e. when $R^s = 0$, or $R^b = 0$, or $R^d = 0$ in Eq. (5)). Not incorporating any of these rewards causes a longer path (relative to real-time PPCPP with all rewards). For this case study, not incorporating the smoothness reward R^s has the largest negative impact on the path length, followed by not incorporating the predation avoidance reward R^d, and finally the boundary reward R^b. Not incorporating any two of these rewards causes the path to be even longer, as shown in Fig. 7 for the case with $R^s = R^b = 0$. Taking a random neighbor decision at each step causes the path to be significantly longer and highly non-smooth.

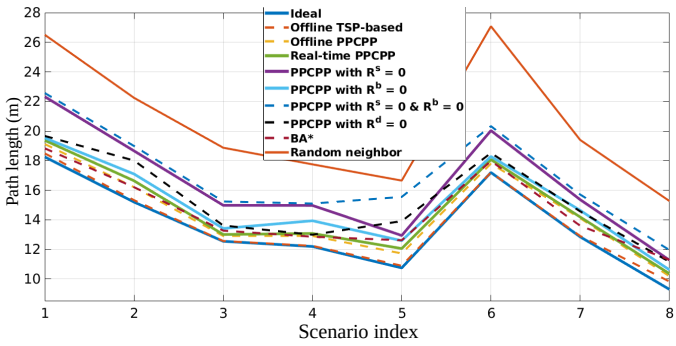
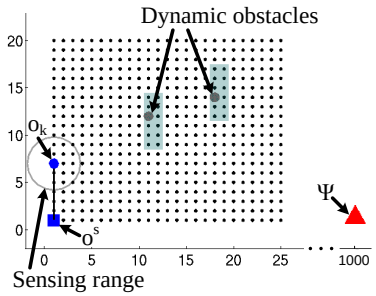


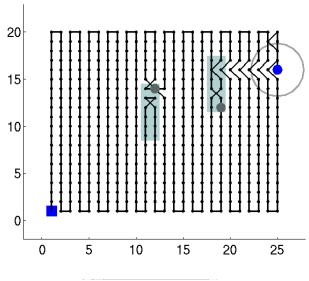
Fig. 7: Comparison and results for the 8 scenarios.

C. Case Study 3: Comparison Against Other Adaptive Approaches

This case study is carried out to illustrate that the proposed PPCPP approach can achieve better results when compared with a neural network-based approach, and an approach that adds rolling path planning and heuristic search to neural network [9]. The test scenario is shown in Fig. 8 where the obstacle on the left continuously moves clockwise within the highlighted region and the obstacle on the right moves counterclockwise. The scenario has been used by Qui et al. [9] to compare their combined approach to the neural network only based approach. Although PPCPP only considers minimizing the path length, the results shown in Table II demonstrate that PPCPP also achieves better results in terms of number of turns and rate of repeated coverage.



(a) A scenario with two dynamic obstacles.



(b) The path when obstacles' speed = 0.25 × robot's speed.

Fig. 8: A scenario where two dynamic obstacles continuously move within the highlighted rectangular regions, and an example path where the robot covers the whole surface.

The neural network-based approaches were explained in Section II-A3. The combined approach presented in [9] outperforms the neural network only based approach mainly because in a dead-end situation it can find the shortest path from the dead-end point to an intermediate goal point, and it only considers local information. The proposed PPCPP approach outperforms both of the aforementioned approaches since it can immediately cover the free areas recently visited by the obstacles; whereas in the neural network-based approaches the robot can cover the areas recently visited by an obstacle only after all other areas are covered, because the neural activity of the neurons that have just been visited by an obstacle is lower than the uncovered neurons.

In the scenario shown in Fig. 8a, 500 targets (25 × 20) represent the surface. The environment is initially unknown to the robot, and at each step the robot can only scan a circular region around itself. It is assumed that the robot can detect obstacles within a radius of 2.9 units as shown in Fig. 8a, meaning that at each step and for the given surface, the robot can update the status of 2 neighboring targets in any direction. Note that the total reward function is calculated only for the directly adjacent neighbors of the prey (maximum 8 neighbors). The weighting factors are optimized only once using the first scan of the robot when the prey is at its start location $\boldsymbol{o}^s$. Optimization takes less than 5 s using the 9 targets that fall within the relevant sensing range of the robot at $\boldsymbol{o}^s$. The values for the optimized weighting factors are $\omega^s = 0.52$ and $\omega^b = 0.20$, and the same weighting factors are used for the prey to cover the entire surface in real-time. Since the environment is initially unknown, the predator is placed at a very far away distance (1000 units) to the right of the prey's start point. Although this case study shows that the PPCPP approach may be able to handle unknown environments, a more thorough study of this potential advantage needs to be carried out as future work for validation.

TABLE II: Comparison against other adaptive approaches.

	Path length	No. of turns	Repeated coverage (%)
Neural network	513	75	2.6
Approach presented in [9]	511	70	2.2
Proposed PPCPP approach	503	70	0.5

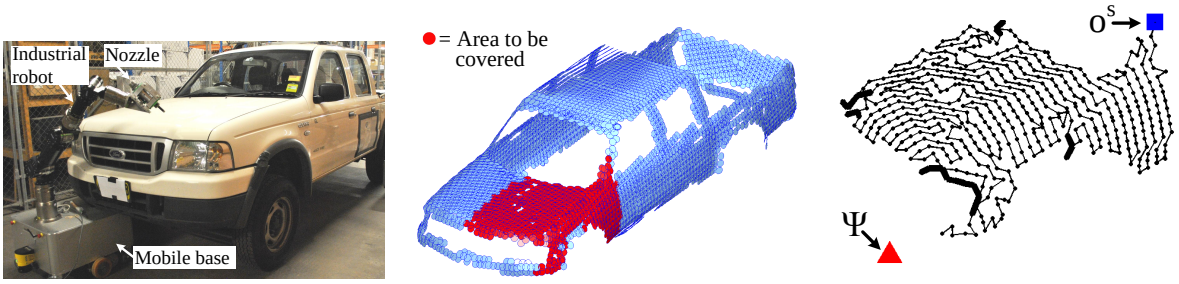
Note that in [9], the speed of the robot relative to the obstacles is not provided. Thus for a fair comparison, the simulation is repeated three times, each time with a different speed, i.e. with obstacles' speed being quarter, half, and equal the speed of the robot. The result shown in Table II is based on the average of the solutions obtained from the three simulations. A supplementary video in WMV format is available[1] at http://ieeexplore.ieee.org. which shows the motion of the prey and the path for each of the three simulations. Figure 8b shows the path corresponding to the scenario where obstacles' speed is quarter the speed of the robot. Note that unlike in [9], the robot is not expected to return to its start point. Thus, all the results shown in Table II consider coverage path planning only, i.e. the prey stops when the whole surface is covered. Point-to-point path planning, such as heuristic search used in [9], can be added to the PPCPP if the robot is required to return to its start point after it has finished covering the whole surface. In Table II, path length, the number of turns, and the rate of repeated coverage is determined in the same way as in [9]. Neural network-based approaches are not shown to handle surfaces embedded in $\mathbb{R}^3$ and conditions where the target representation of the surface is not uniform. However, the following case studies will demonstrate that the PPCPP approach is capable of handling such conditions.

D. Case Study 4: Coverage of a Complex object Amid Changes to the Coverage Area

In this case study and the following case studies, the PPCPP approach is tested using surfaces embedded in $\mathbb{R}^3$. The same surface and optimized weighting factors in this case study are used for the next two case studies so as to provide an insight into how PPCPP performs in real-time when various changes occur within the same environment.

The purpose of this case study is to demonstrate that an unexpected change in the reachable surface area (by an industrial robot) can be managed using the adaptive behavior of the PPCPP. Figure. 9a shows a vehicle and an autonomous industrial robot (with a manipulator) to perform a one-off task of high-pressure cleaning of the vehicle. Before the cleaning process, the robot calculates a set of strategic base positions (e.g. using the approach presented in [29]) from which it will operate on the vehicle and clean all metallic surfaces of the

[1]A video for Case Study 3 can be viewed through https://youtu.be/-zsoTqfM9IM.



(a) An autonomous industrial robot to high-pressure clean a vehicle.

(b) Targets representing the vehicle and the areas expected to be covered.

(c) 3D view of the path corresponding to the areas highlighted in Fig. 9b.

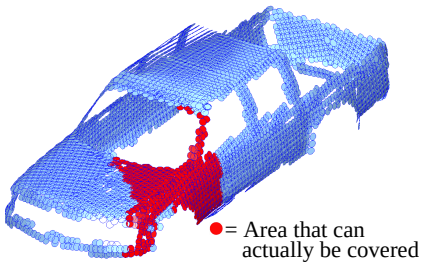
Fig. 9: The areas expected to be covered by the robot are shown, then optimization is performed to obtain appropriate weighting factors ($\omega^s = 0.45$ and $\omega^b = 0.96$) based on which the shown path is generated.

vehicle. From one of these base positions, the robot anticipates covering the 483 targets highlighted in Fig. 9b. For these targets, offline optimization is performed and the optimized weighting factors, $\omega^s = 0.45$ and $\omega^b = 0.96$, are obtained. The corresponding optimized path is shown in Fig. 9c. Running the optimization took 78 s (average of 5 runs). The paths due to dead-end recovery are shown as thicker black lines in Fig. 9c. All targets representing the vehicle are generated by considering a 0.0563 m distance between neighbors; however, since a point cloud is used to generate the targets, the distance between targets are not consistent. Nevertheless, it is shown that PPCPP is capable of handling such inconsistency.

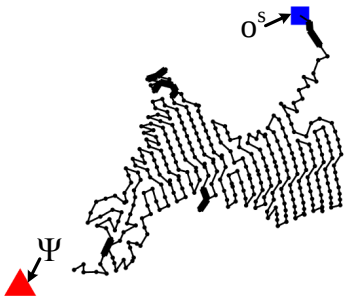
Note that manipulator motion planning with a fixed or mobile base is not the focus of the paper, but rather the focus is on planning the end-effector path for achieving complete coverage amid changes in the environment. However, as future work, it will be interesting to investigate and incorporate constraints related to the industrial robot's motion when following the end-effector path on the surface.

Once the robot starts cleaning the vehicle's surfaces, the robot may become stationary at a position slightly different to the desired base position that is calculated offline since there are uncertainties associated with the robot's base positioning (e.g. due to localization error and sensor noise). Suppose that these uncertainties cause a mispositioning of robot's base, which leads to the robot being able to reach the targets highlighted in Fig. 10a instead of the targets highlighted in Fig. 9b. Thus, during the real-time operation, the robot has to cover the updated set of targets using the same weighting factors, i.e. without repeating the optimization. The path of the robot's end-effector (prey's path) to cover the updated set of targets is shown in Fig. 10b, which is 21.69 m in length. During the real-time deployment and at each step, it takes less than 1 millisecond to compute the next best neighbor.

GA optimization is repeated for the changed environment to compare the prey's path with an optimized path. Furthermore, to ensure that GA optimization performs well and generates consistent solutions, it was repeated 20 times and compared to a grid search. The best solution from the 20 GA runs gives a path length of 21.22 m. This solution is only 0.47 m (2.2%) shorter than the path the prey traveled in real-time. From the 20 GA runs, 16 runs provided this solution (21.22 m), and



(a) Areas that can actually be covered from the current base position of the robot.



(b) 3D view of the path corresponding to the areas highlighted in Fig. 10a.

Fig. 10: The areas that can actually be covered by the robot at its current base position are shown. The same weightings ($\omega^s = 0.45$ and $\omega^b = 0.96$) are used to generate the path.

the rest provided solutions that are at most 1.4% worse than the best solution. From one of the GA runs, $\omega^s = 0.19$ and $\omega^b = 1.41$. A grid search is conducted where values of 0 to 2, in steps of 0.02 (total of 10,000 grids), is considered for the weighting factors. The best solution from the grid search results in a path length of 21.22 m which is exactly the same as the solution returned by GA in majority of times even though the function space is highly non-convex for this scenario. GA convergence happened in less than 55 generations, i.e. less than 2,550 pairs of weighting factors are evaluated in total which is about a quarter of the number of evaluation done for the grid search. GA optimization took 60 s (average of 20 runs).

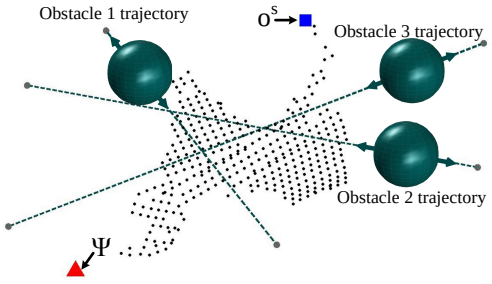
To analyze the effect of each reward functions on the performance of the approach, GA optimization is repeated while considering the following conditions: (1) without incorporating the smoothness reward which results in $\omega^b = 0.88$ with a path length of 21.67 m with (worse by 2.1% compared to the best solution found above with path length of 21.22 m); (2) without incorporating the boundary reward which results in $\omega^s = 0.53$ with a path length of 23.38 m (worse by 9.2% compared to the best solution found above); and (3) without incorporating the predation avoidance reward which results in $\omega^b = 1.56$ and $\omega^s = 0.55$ with a path length of 22.70 m (worse by 6.6% compared to the best solution found above).

E. Case Study 5: Coverage of a Complex Object in the Presence of a Dynamic Obstacle

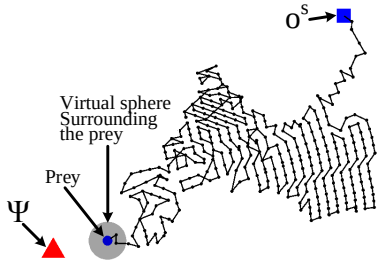
Nine scenarios are used in this case study (Table III) to demonstrate that PPCPP can achieve complete coverage of a complex object while being adaptable to unexpected dynamic obstacles that can be faster or slower than the robot's end-effector (Fig. 11a). The same surface as in the previous case study (Fig. 10a), with the same optimized weighting factors ($\omega^s = 0.19$, $\omega^b = 1.41$), is used.

Suppose that multiple robots are operating in a decentralized manner to cover the surface, then each robot may consider other robots as obstacles. Thus, the dynamic obstacles in this case study may represent other robots' end-effector tool, which may have unexpectedly entered the workspace of a robot. A sphere may be used as a conservative approximation of the volume that the obstacle occupies. The scenarios in this case study, as well as subsequent case studies, are purposefully made more difficult (e.g. by considering multiple dynamic obstacles and having the obstacles continuously move back and forth) for rigorous empirical testing of the approach.

Obstacles continuously move back and forth along the trajectories shown in Fig. 11a. All obstacles are represented by a sphere having a 0.2 m radius. The end-effector of the robot needs to move at an approximately constant speed for uniform coverage. To prevent the prey (robot's end-effector) from colliding with an obstacle, a virtual field (sphere in this case) is made to surround the prey, as shown in Fig. 11b. The size of the virtual sphere can be made bigger proportionally to the speed of the obstacle. For speed ratios of 2:1, 1:1, and 2:3 (robot:obstacle), the radius size of the virtual sphere is set to 0.11 m, 0.17 m, and 0.23 m, respectively. As future work, a variable size virtual sphere can be investigated such that the virtual sphere becomes bigger when uncertainties in predicting the motion or speed of the obstacle are larger, and vice versa; however investigating uncertainties is outside the scope of this paper. It needs to be noted that the PPCPP approach is not limited to the use of virtual field and it can be discarded if there are no dynamic obstacles or if the obstacles are slow or can be predicted reasonably accurately. Alternative methods may also be incorporated.



(a) The trajectories of three spherical obstacles.



(b) An example path corresponding to scenario 7 where the first obstacle is 1.5 times faster than the robot's end-effector.

Fig. 11: The trajectory of each obstacle and an example path are shown.

TABLE III: Results for the 9 scenarios where in each scenario a single dynamic obstacle is present in the environment.

	Obst. (index)	Speed ratio (robot: obst.)	Path length resulting from real-time PPCPP while considering the following rewards: All (m)	R^d & R^s (m)	R^d & R^b (m)	R^s & R^b (m)	Increase (PPCPP wrt optimized path) (%)
1	1	2:1	22.73	23.65	22.83	23.33	7.12
2	2	2:1	21.78	23.49	22.90	23.83	2.64
3	3	2:1	21.48	23.87	22.60	23.01	1.23
4	1	1:1	22.03	24.54	21.65	23.48	3.82
5	2	1:1	21.52	23.59	22.42	24.65	1.41
6	3	1:1	22.75	24.42	22.45	23.98	7.21
7	1	2:3	23.86	27.69	24.82	26.93	12.44
8	2	2:3	22.52	27.22	25.28	25.33	6.13
9	3	2:3	22.75	26.41	24.57	25.12	6.73

The results in Table III tend to show that when an obstacle is faster than the robot, the path length is longer. This is because when the robot detects an obstacle its top priority is to avoid collisions by moving away in a direction that maximizes its distance to the obstacle; as a result, repeated coverage can occur more often causing a longer path. In Case Study 4, GA optimization was repeated three times where each time one of the three reward functions in Eq. (5) was omitted (i.e. first considering R^d & R^s only; then R^d & R^b only, and finally R^s & R^b only). Each of these three models is then applied to the 9 scenarios in this case study and the results are shown in Table III. In almost all 9 scenarios, not incorporating any of the three rewards causes the path to be longer. It is possible for the obstacle to penetrate the virtual sphere surrounding the prey. In such a situation, the prey's priority is to maximize its distance from the obstacle until the obstacle is no longer inside the virtual sphere. Due to this behavior, the prey may be temporarily pushed away from the region it is covering to another region. If spatial order is not enforced on the motion of the prey through the predation avoidance reward, then the

prey may not immediately come back to the region it was covering and instead it can move back and forth between regions more often causing longer paths as illustrated in Table III. The difference between the lengths of the paths traveled by the prey in real-time and the optimized path (21.22 m) are also shown in Table III which is 5.33% by taking the average of the results from the 9 scenarios.

As an example, the path corresponding to scenario 7 is shown in Fig. 11b. A supplementary video in WMV format is available[2] at http://ieeexplore.ieee.org which shows how the paths are generated for each of the nine scenarios.

F. Case Study 6: Coverage in the Presence of Multiple Dynamic Obstacles having Different Speed and Size

The purpose of this case study is to show that PPCPP is effective when applying even more complicated changes (multiple dynamic obstacles with different sizes and speeds) to the same surface used in the previous two case studies.

As shown in Table IV, 4 scenarios are considered wherein each scenario, 2 or 3 obstacles are used. The difference between the lengths of the paths traveled by the prey in real-time and the optimized path (21.22 m) are shown in the table which is 3.31% by taking the average of the results from the 4 scenarios. The same environment as in Fig. 11a is used; however, obstacles' size and end-effector coverage speed are changed as per the values shown in Table V. The obstacles continuously move back and forth with the trajectories that were shown in Fig. 11a. The same weighting factors, $\omega^s = 0.19$ and $\omega^b = 1.41$ as in Case Study 4 (Section VII-D) are used.

Each of the 4 scenarios is repeated while considering no boundary reward (i.e. only R^d & R^s), no smoothness reward (i.e. only R^d & R^b), and no predation avoidance reward (i.e. only R^s & R^b). Results are shown in Table IV. The same optimized weighting factors as in Case Study 4 are used for these model variations. In all 4 scenarios, not incorporating any of the three rewards causes the path to be longer. As discussed in the previous case study, not incorporating the predation avoidance reward causes the prey to move back and forth between regions and in turn results in a longer path with more overlaps (as shown in Fig. 12).

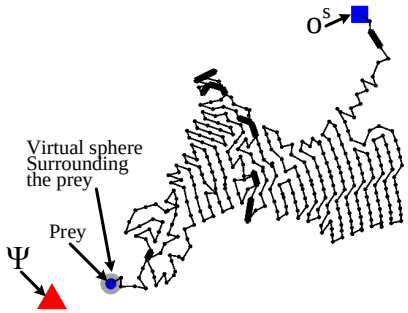
[2]A video for Case Study 5 can be viewed through https://youtu.be/c5sNqjYua6E.

TABLE IV: Results for the 4 scenarios where in each scenario multiple obstacles with different speed and size are present in the environment.

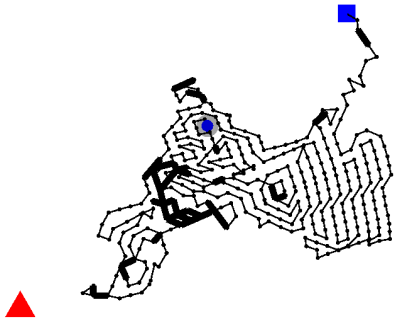
	Obstacles used (indices)	Path length resulting from real-time PPCPP while considering the following rewards: All (m)	R^d & R^s (m)	R^d & R^b (m)	R^s & R^b (m)	Increase (PPCPP wrt optimized path) (%)
1	1 & 2	22.76	24.12	23.12	24.23	7.26
2	1 & 3	21.76	24.04	21.94	25.46	2.54
3	2 & 3	21.42	24.84	23.15	23.67	0.94
4	1, 2 & 3	21.75	24.21	22.62	23.59	2.50

TABLE V: Speed and size of each obstacle.

obstacle (index)	Speed ratio (robot : obstacle)	Obstacles' radius size (m)
1	10:3	0.2
2	10:5	0.15
3	10:7	0.1



(a) Path with all rewards incorporated.



(b) Path with no predation avoidance reward.

Fig. 12: Example path (scenario 2 of Table IV) with and without predation avoidance reward where path overlaps are shown as thick black lines.

Note that a virtual sphere of size 0.11 m surrounds the prey (end-effector point) and stops the prey from colliding with the obstacles, as explained in Case Study 4 (Section VII-E). A supplementary video in WMV format is available[3] at http://ieeexplore.ieee.org which shows how the path is generated for each of the four scenarios.

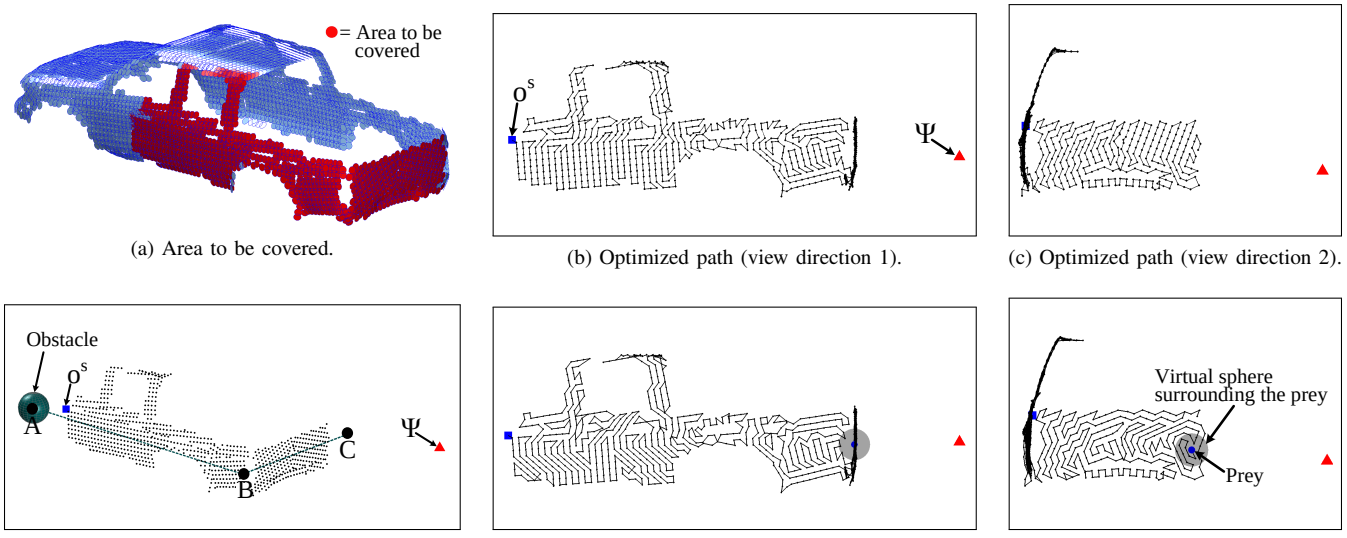
G. Case Study 7: Coverage of a Complex Object in the Presence of a Varying Speed Obstacle

Case Studies 4 to 6 considered the same surface; whereas this case study aims at testing PPCPP using another complex surface with varying speed obstacle, so as to have more test variations.

The speed of the obstacle varies and can exceed the robot's end-effector speed. The surface to be covered is highlighted in Fig. 13a where 886 targets represent the surface. The distance between neighboring targets is not consistent; nevertheless, it is shown that PPCPP is capable of planning a path over targets that are not uniformly arranged.

The trajectory of the varying speed obstacle is shown in Fig. 13d. The obstacle moves through the points indicated on

[3]A video for Case Study 6 can be viewed through https://youtu.be/A6H99lqLHMk.



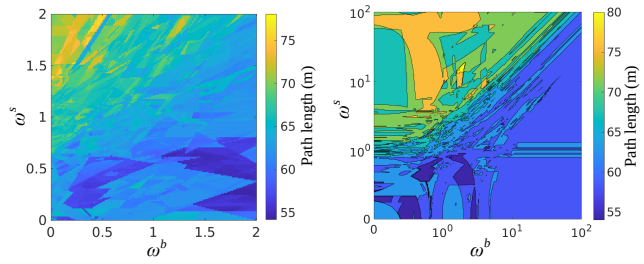
(a) Area to be covered. (b) Optimized path (view direction 1). (c) Optimized path (view direction 2).

(d) Obstacle's trajectory. (e) Path generated in real-time (view direction 1). (f) Path generated in real-time (view direction 2).

Fig. 13: A scenario where a varying speed obstacle continuously moves through the area that needs to be covered.

the trajectory as follows: point A to B, then to C, then back to B, and then back to A, with an end-effector to obstacle speed ratio of 1:2, 1:1.5, 1:1.5, and 1:1, respectively. The obstacle continuously repeats this trajectory. Since the obstacle can move faster than the robot's end-effector (or the prey), a virtual sphere of size 0.28 m surrounds the prey and stops the obstacle from colliding with the prey, as explained in Case Study 5 (Section VII-E).

At first, GA optimization is performed offline to obtain the optimized weighting factors ($\omega^s = 0.63$ and $\omega^b = 1.55$) for the condition where it is assumed that there are no obstacles. The corresponding optimized path is shown in Figs. 13b and 13c. This path is 54.89 m in length. To check that GA optimization performs well, a grid search is conducted where values of 0 to 2, in steps of 0.01 (total of 40,000 grids), is considered for the weighting factors (Fig. 14a). A logarithmic scale grid search for values of 0 to 100 for the weighting factors is also constructed with total of 90,000 grids (Fig. 14b using filled contour map). As can be seen, the function space is highly non-convex (many local minima), which may cause GA not to find a global solution. The best solution from the grid searches gives a path length of 54 m. This solution is 1.6% better than that found by GA. However, GA converges in less than 60 generations using the default setting, meaning that less than 3,000 pairs of weighting factors are evaluated in total, whereas for the grid search minimum of 40,000 grids were used. Increasing the population size of GA slightly to 55 can achieve a solution with the same path length as the grid search and yet with much fewer function evaluations (less than 3300).



(a) linear scale. (b) logarithmic scale.

Fig. 14: Grid search of the weighting factors for the scenario shown in Fig. 13.

The real-time path of the robot's end-effector (prey) is shown in Figs. 13e and 13f. A supplementary video in WMV format is available[4] at http://ieeexplore.ieee.org which shows how the path is generated. The path that is generated in real-time is 58.76 m long which is 7.1 % longer than the optimized path. This increase in the length of the path is mainly due to two reasons: (1) the prey moving away from the obstacle in many instances to avoid collisions, and (2) collision avoidance causing dead-ends and repeated coverage. Generating the end-effector path in real-time without incorporating the smoothness reward, but using an optimized weighting factor $\omega^b = 0.25$, results in a path length of 62.60 m which is worse by 6.54%. Similarly, the real-time end-effector path without incorporating the boundary reward, but using an optimized weighting factor $\omega^s = 0.12$, results in path length of 68.63 m which is worse by 16.80%.

VIII. Discussion

The case studies demonstrated that PPCPP is capable of achieving complete coverage and adapting to changes in an environment. The goal has been to construct a planner that uses the initial knowledge of the environment to optimize its parameters so that when applied in real-time it can quickly adapt to the unexpected changes while aiming to minimize the

[4]A video for Case Study 7 can be viewed through https://youtu.be/9veGRBIhZGQ.

coverage cost of the surfaces. It was shown that the approach is efficient for real-time deployment because the robot plans one step at a time by utilizing sensor information to determine which neighboring targets are obstacle-free and selects the neighbor that results in maximum total reward, hence updating the entire map is not necessary.

A limitation of the approach is that, depending on the complexity of the changes that occur in the environment, the generated path may not be optimal in terms of length. This issue also causes overlaps with previously covered regions which is not acceptable for some applications. However, guaranteeing an optimal path in real-time where various changes can unexpectedly occur in the environment may be unrealistic and computationally intractable. A potential limitation of the approach is related to the kinodynamic constraints of the robot and the motion planner utilized for the robot to follow the changing coverage path. Thus, these aspects need to be investigated when applying the proposed CPP approach to a particular application and a particular robot.

There are very few CPP approaches that can handle dynamic obstacles and unexpected changes in the coverage area. As the first paper presenting this novel approach (PPCPP), the main focus has been to formulate the approach and validate its adaptive capability using various case studies. The approach opens up many interesting opportunities for potential future research, some of which include:

- Modifying the approach to be capable of handling the aforementioned limitations (e.g. unknown environments, various uncertainties, and kinodynamic constraints).
- Studying multi-robot coordination by optimizing the start target and the predator location for each robot.
- Considering multiple predators where both ambushing and dynamic predators (e.g. dynamic obstacles) can be considered for the prey to assess the risk of predation.
- Investigating and comparing methodologies for optimizing the weighting factors both offline and in real-time.
- Investigating the applicability of the same optimized weighting factors to various surfaces.

Some of the above future works were briefly investigated in this paper. For example, an unknown environment was used for Case Study 3 in Section VII-C; and virtual field was used in the case studies (Sections VII-D to VII-G) to prevent the prey colliding with an obstacle that is faster than the robot's end-effector, but can also be used for handling uncertainties in predicting obstacles' motion.

IX. Conclusion

A predator-prey based approach, named PPCPP, for adaptive coverage path planning was presented in this paper. The approach is mainly designed for environments where unexpected changes can occur, and predicting such changes prior to the real-time deployment of the robot is impractical. Using many case studies, the approach was proven to enable a robot to obtain complete coverage of target surfaces in real-time amid various unforeseen changes in the environment. The approach minimizes the path length and has only two parameters to optimize for a given environment before real-time deployment. Importantly, the approach was shown to be adaptable to unexpected changes in the environment even if a 3D object with complex geometric shape is to be operated on. It is also shown that the approach is computationally tractable such that the robot can quickly respond and adapt in the case of unexpected stationary or dynamic obstacles being present.

Acknowledgment

This research is supported by the Centre for Autonomous Systems at the University of Technology Sydney, Australia. Authors thank Dr. Gavin Paul, Dr. Teng Zhang, and Dr. Raphael Falque for their valuable suggestions and discussions.

References

[1] E. Galceran and M. Carreras, "A Survey on Coverage Path Planning for Robotics," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258 – 1276, 2013.

[2] E. M. Arkin, S. P. Fekete, and J. S. Mitchell, "Approximation algorithms for lawn mowing and milling," *Computational Geometry*, vol. 17, no. 12, pp. 25 – 50, 2000.

[3] S. P. Fekete, J. S. B. Mitchell, and C. Schmidt, "Minimum covering with travel cost," *Journal of Combinatorial Optimization*, vol. 24, pp. 32–51, Jul 2012.

[4] H. H. Viet, V. H. Dang, M. N. U. Laskar, and T. Chung, "BA*: an online complete coverage algorithm for cleaning robots," *Applied Intelligence*, vol. 39, pp. 217–235, Sep 2013.

[5] J. Song and S. Gupta, "ε^*: An online coverage path planning algorithm," *IEEE Transactions on Robotics*, vol. 34, pp. 526–533, April 2018.

[6] I. Shnaps and E. Rimon, "Online coverage of planar environments by a battery powered autonomous mobile robot," *IEEE Transactions on Automation Science and Engineering*, vol. 13, pp. 425–436, Apr 2016.

[7] I. Shnaps and E. Rimon, "Online coverage by a tethered autonomous mobile robot in planar unknown environments," *IEEE Transactions on Robotics*, vol. 30, pp. 966–974, Aug 2014.

[8] T. M. Cabreira, C. D. Franco, P. R. Ferreira, and G. C. Buttazzo, "Energy-aware spiral coverage path planning for UAV photogrammetric applications," *IEEE Robotics and Automation Letters*, vol. 3, pp. 3662–3668, Oct 2018.

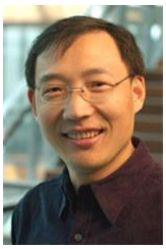
[9] X. Qiu, J. Song, X. Zhang, and S. Liu, "A Complete Coverage Path Planning Method for Mobile Robot in Uncertain Environments," in *The Sixth World Congress on Intelligent Control and Automation*, vol. 2, pp. 8892–8896, 2006.

[10] H. Chen, T. Fuhlbrigge, and X. Li, "Automated Industrial Robot Path Planning for Spray Painting Process: A review," in *2008 IEEE International Conference on Automation Science and Engineering*, pp. 522–527, Aug 2008.

[11] P. Atkar, A. Greenfield, D. Conner, H. Choset, and A. Rizzi, "Uniform Coverage of Automotive Surface Patches," *The International Journal of Robotics Research*, vol. 24, no. 11, pp. 883–898, 2005.

[12] W. Sheng, H. Chen, N. Xi, and Y. Chen, "Tool path planning for compound surfaces in spray forming processes," *IEEE Transactions on Automation Science and Engineering*, vol. 2, pp. 240–249, Jul 2005.
[13] P. Olivieri, L. Birglen, X. Maldague, and I. Mantegh, "Coverage Path Planning for Eddy Current Inspection on Complex Aeronautical Parts," *Robotics and Computer-Integrated Manufacturing*, vol. 30, no. 3, pp. 305 – 314, 2014.
[14] Z. Bo, F. Fang, S. Zhenhua, M. Zhengda, and D. Xianzhong, "Fast and Templatable Path Planning of Spray Painting Robots for Regular Surfaces," in *34th Chinese Control Conference (CCC)*, pp. 5925–5930, Jul 2015.
[15] W. To, G. Paul, N. Kwok, and D. Liu, "An Efficient Trajectory Planning Approach for Autonomous Robots in Complex Bridge Environments," *International Journal of Computer Aided Engineering and Technology*, vol. 1, no. 2, pp. 185–208, 2009.
[16] H. Wang, H. Li, C. Zhang, S. He, and J. Liu, "A 3D coverage path planning approach for flying cameras in nature environment under photogrammetric constraints," in *2017 36th Chinese Control Conference (CCC)*, pp. 6761–6766, July 2017.
[17] L. Paull, M. Seto, and H. Li, "Area Coverage Planning that Accounts for Pose Uncertainty with an AUV Seabed Surveying Application," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6592–6599, May 2014.
[18] E. Galceran, R. Campos, N. Palomeras, D. Ribas, M. Carreras, and P. Ridao, "Coverage Path Planning with Real-time Replanning and Surface Reconstruction for Inspection of Three-dimensional Underwater Structures using Autonomous Underwater Vehicles," *Journal of Field Robotics*, vol. 32, no. 7, pp. 952–983, 2015.
[19] P. Atkar, H. Choset, A. Rizzi, and E. Acar, "Exact Cellular Decomposition of Closed Orientable Surfaces Embedded in R3," in *IEEE International Conference on Robotics and Automation*, vol. 1, pp. 699–704, 2001.
[20] C. Papachristos, S. Khattak, and K. Alexis, "Uncertainty-aware receding horizon exploration and mapping using aerial robots," in *International Conference on Robotics and Automation (ICRA)*, pp. 4568–4575, May 2017.
[21] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding horizon path planning for 3D exploration and surface inspection," *Autonomous Robots*, vol. 42, pp. 291–306, Feb 2018.
[22] G. Paul, S. Webb, D. Liu, and G. Dissanayake, "Autonomous Robot Manipulator-Based Exploration and Mapping System for Bridge Maintenance," *Robotics and Autonomous Systems*, vol. 59, no. 78, pp. 543–554, 2011.
[23] M. Hassan and D. Liu, "Simultaneous area partitioning and allocation for complete coverage by multiple autonomous industrial robots," in *Autonomous Robots*, pp. 1–20, 2017.
[24] X. Zheng, S. Koenig, D. Kempe, and S. Jain, "Multirobot forest coverage for weighted and unweighted terrain," *IEEE Transactions on Robotics*, vol. 26, pp. 1018–1031, Dec 2010.
[25] J. M. Palacios-Gass, E. Montijano, C. Sags, and S. Llorente, "Distributed coverage estimation and control for multirobot persistent tasks," *IEEE Transactions on Robotics*, vol. 32, pp. 1444–1460, Dec 2016.
[26] T. Stankowich and D. Blumstein, "Fear in Animals: a Meta-Analysis and Review of Risk Assessment," *Proceedings of the Royal Society of London B: Biological Sciences*, vol. 272, no. 1581, pp. 2627–2634, 2005.
[27] J. Brown, J. Laundré, and M. Gurung, "The Ecology of Fear: Optimal Foraging, Game Theory, and Trophic Interactions," *Journal of Mammalogy*, vol. 80, no. 2, pp. 385–399, 1999.
[28] S. Creel, J. Winnie, B. Maxwell, K. Hamlin, and M. Creel, "Elk Alter Habitat Selection as an Antipredator Response to Wolves," *Ecology*, vol. 86, no. 12, pp. 3387–3397, 2005.
[29] M. Hassan, D. Liu, and G. Paul, "Modeling and stochastic optimization of complete coverage under uncertainties in multi-robot base placements," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2978–2984, Oct 2016.

Mahdi Hassan received his Mechanical Engineering degree with first class honours in 2013 from the University of Technology Sydney (UTS). He undertook his Ph.D. studies at UTS Centre for Autonomous Systems (CAS) and received his Ph.D. early 2018. Dr. Mahdi Hassan has been working as a research fellow at CAS since early 2017. His research interests include adaptive and cooperative complete coverage in complex 3D environments; autonomous industrial robots; path, motion and task planning; multi-robot collaboration and coordination; and optimization-based algorithms.



Dikai Liu received his Ph.D. degree in 1997. His main research interest is robotics including exploration, motion planning, robot teams, and physical human-robot interaction. Professor Liu has been developing novel methods and algorithms that enable robots to operate in unstructured and complex 3D environments autonomously or collaboratively with human users. Example robotic systems he developed and practically deployed include autonomous grit-blasting robots for steel bridge maintenance, assistive robots for human strength augmentation in industrial applications and bio-inspired climbing robots for steel structure inspection.