Search-based Planning for Sensor-based Coverage

Master's Thesis

Tushar Kusnur
December, 2022
CMU-RI-TR-22-76

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania

Committee:
Maxim Likhachev, *co-chair*
Oliver Kroemer, *co-chair*
Zachary Manchester
Cherie Ho

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Abstract.

Robots are excellent candidates for the dull, dirty, and dangerous jobs we do not want humans to perform. Today, these include inspection of large areas or structures, post-disaster assessment, and surveillance. Assessing the aftermath of the recent Fern Hollow bridge collapse in Pittsburgh is one such example. Many such real-world inspection or surveillance tasks can be cast as robot coverage problems. Coverage path planning (CPP) is the problem of determining a collision-free path for a robot that observes all reachable points of interest in an environment. A variant of this problem is persistent coverage, wherein coverage levels decay over time and the robot must ensure to persistently observe all points of interest over time. Further, systems performing such tasks can be extended to have actively controllable sensors or multiple robots. This thesis proposes novel motion planning methods for robots performing such tasks. Planning via graph search on state lattices has the advantages of resolution completeness and bounded suboptimality, and this is the overarching approach to motion planning we use in this thesis.

We first look at a system of multiple unmanned aerial vehicles (UAVs) with fixed, downward-facing cameras persistently covering a large environment. This requires stitching together solutions to subproblems such as goal assignment, multi-agent planning, and handling kinematic and dynamic constraints of the robots. We demonstrate the effectiveness of our method on real-world field tests. Next, we look at the case of having an actively controllable onboard sensor and zoom into the problem of a single UAV actively sensing its environment while navigating towards a goal. We develop two novel search-based planners for simultaneous robot and sensor trajectory generation. The first decouples robot and sensor planning to compute solutions quickly, and the second (optionally) iteratively refines this solution in the joint robot and sensor search space.

We then look at the standard CPP problem, where most popular approaches are decomposition based and require extensive pre-processing of the environment. These methods come with strong properties of completeness and optimality with respect to the corresponding objective. Decomposition-free methods are few, simpler, but heuristic or random and offer weak performance guarantees. We bridge this methodological gap with a novel, online, decomposition-free coverage path planner that provides completeness guarantees with minimal environment processing. Our planner leverages precomputed coverage behaviors and automatically determines where to execute them online. And finally, we tackle an important limitation of this plannerthat it exhaustively evaluates all coverage behaviors online, which is prohibitively expensive with large numbers of behaviors or higher-fidelity sensor models such as ray casting. We propose a method that uses supervised learning to predict how useful a behavior will be in a given local environment, and use this to reduce the online computational burden on the planner.

To everyone not living up to their potential.

CONTENTS

List of Tables

List of Figures

1 Introduction and Background

1.1 Robots for Sensing

Robots are excellent candidates to perform the dull, dirty, and dangerous tasks that we do not want humans to perform. Today, these include tasks such as cave exploration [90], inspection of complex structures [27], and scientific data sample collection in extreme environments[1]. Much of this involves a robot navigating while avoiding collisions, but this navigation is also deliberative—the robot must plan its actions in order to *actively sense* its environment. This concept has been studied in various flavors, and is perhaps most well-known in the robotics and computer vision communities through Bajcsy's pioneering work on active perception [10]. In this thesis, we look at three forms of robotic sensing:

1. **Coverage path planning with fixed sensors**: Coverage path planning is the problem of determining a collision free path for a robot, that observes within some sensor footprint all reachable points of interest in an environment.
2. **Persistent coverage with fixed sensors**: Persistent coverage is a variant of standard coverage where the robot must continuously maintain a desired visitation frequency over points of interest in an environment.
3. **Planning for active sensing with an actuated sensor**: Sensor planning involves controlling the degrees of freedom of a sensor system to achieve a certain operational objective. In this thesis, we look at this within the context of a robot with an actively controllable onboard sensor.

1.2 Scope and contributions

This thesis focuses on the process of planning feasible paths for single or multiple robots over short time horizons (e.g. seconds with a focus on sensing tasks arising from sensor-based standard and persistent coverage path planning. We focus on *planning* methods, namely yielding kinematically and dynamically feasible paths for robots in a hierarchical navigation system, assuming these paths are fed to a lowel level controller for execution. While a subset of the approaches in this thesis are demonstrated on unmanned aerial vehicles (UAVs), they are general and can be applied to any robotic navigation problem given the kinematic and dynamic constraints of the robot. Coverage is generally defined as computing paths to observe all points of interest in a *known* environment [37]. In all the works in this thesis, we assume that we are given a binary occupancy grid representation

[1] https://www.nasa.gov/press-release/nasa-s-perseverance-rover-collects-first-mars-rock-sample

of the environment. The research challenges arise from the computational complexity of computing search-based plans for high-dimensional robotic systems.

The contributions of this thesis are as follows:

1. **Planning for Persistent, Multi-UAV Coverage:** Chapter 2 proposes a planning framework that enables multiple kinodynamically constrained UAVs with fixed, downward-facing sensors (cameras) to persistently cover a large environment. We solve the goal assignment problem, i.e., computing goal locations for each UAV, the multi-agent planning problem gien these goal locations, and a kinodynamic path planning problem per UAV. Most importantly, we detail how to stitch together these solutions in a systematic manner that maintains global deconfliction between UAVs. This system provides a foundation for the subsequent contributions in this thesis.
2. **Planning for Active Sensing in Goal-Directed Coverage Tasks:** In Chapter 3, we look at the single-agent planning problem wherein the UAVs from the previous work are modified to have a forward-facing camera with a single controllable degree of freedom (the camera's yaw angle). The addition of this degree of freedom along with constraints on the angular speed of the camera lead to the problem of computing simultaneous robot and sensor trajectories for efficient coverage. We propose two search-based approaches to compute such trajectories. The first one decouples solving for the robot and sensor trajectories, giving us fast but suboptimal solutions. The second method iteratively refines the solution obtained from the first by searching for better solutions in the joint search space of robot and sensor trajectories.
3. **Complete, Decomposition-Free Coverage Path Planning:** In Chapter 4, we look at the standard coverage path planning problem (again with a fixed, downward-facing sensor) and propose a new coverage path planner that balances complexity of the solution and the strength of properties it yields, filling an important gap in coverage path planning literature. The approach presented can be related to the work in Chapter 3 by thinking of it as a way to combine the goal assignment and path planning steps for a single robot into one search routine. The search has access to precomputed space-filling coverage behaviors that we know from experience to be effective, and automatically determines where to apply them online. By repeatedly invoking this search routine and executing plans, we can guarantee complete coverage without requiring any geometric environmental decomposition like several popoular coverage path planners do.
4. **Speeding Up Decomposition-Free Coverage by Learning Behavior Footprints:** In Chapter 5, we address a significant limitation of the approach proposed in Chapter 4: it requires exhaustive searches that forward-simulate the aformentioned space-filling coverage behaviors. The more realistic the sensor model we use for the robot

gets, the more expensive it is to forward-simulate such behaviors online, making the search prohibitively slow. We look at how we can adapt this approach when we use a ray-casting sensor model in particular. We propose learning to predict behavior footprints that these coverage behaviors will produce, so that foward-simulating them online is a fast call to a neural network model. We present preliminary results, limitations, and ideas for future work on this approach.

1.3 Background: Search-based Planning

At a fundamental level, all our approaches are extensions of running A* search [42] on a graph. Let a graph $G = (V, E)$, a start node s_{start}, and a goal node s_{goal} together represent a motion planning problem. The vertices or nodes V represent robot configurations, and edges E represent actions the robot can take to transition from the configurations connected by the edge. Briefly, A* search begins with s_{start} and selects the next best state, s_{best}, using the known cost from s_{start} to s_{best}, plus an estimated cost from s_{best} to s_{goal} known as the heuristic estimate of the "cost to go". Dijkstra's search [25] can be thought of as A* search with a zero heuristic. We present pseudocode for A* search in Alg. 1.

Algorithm 1 A* search

Input: s_{start}, s_{goal}
Output: π

procedure A*
  $g(s_{\text{start}}) \leftarrow 0$
  OPEN $\leftarrow$ OPEN $\cup \{s_{\text{start}}, g(s_{\text{start}})\}$
  while OPEN not empty **do**
    $x \leftarrow$ OPEN.MIN()
    SUCCS $\leftarrow \emptyset$ // List of successor nodes
    COSTS $\leftarrow \emptyset$ // List of corresponding edge costs
    if $x == s_{\text{goal}}$ **then**
      $\pi \leftarrow$ Backtrack from $G_{\text{imaginary}}$ to s_{start}
      return π // Path found
    else
      EXPAND $(x,$ SUCCS, COSTS$)$ // Expands x and populates SUCCS, COSTS
    for successor $s' \in$ SUCCS and cost $c \in$ COSTS **do**
      if $g(s') > g(s) + c$ **then**
        $g(s') \leftarrow g(s) + c$
        $bp(s') \leftarrow s$
        OPEN $\leftarrow$ OPEN $\cup \{s', g(s)\}$
  return π

2 Persistent, Multi-UAV Coverage

2.1 Introduction

Traditional robot-coverage is the problem of determining a collision-free path for a robot that covers all points of interest in an environment [37]. *Persistent* coverage seeks to continuously maintain a desired *coverage-level* over an environment [31, 62, 85]. In our case, coverage-levels degrade over time, thereby increasing the urgency with which points must be revisited. For example, consider a floor-cleaning robot—once a part of the floor is cleaned, more dust will eventually collect over it and thereby decrease its coverage-level. There has been a rise in the use of robots to perform tasks cast as persistent-coverage problems, such as environmental monitoring, exploration, inspection, post-disaster assessment, monitoring traffic congestion over a city, etc. [65, 85, 87, 91].

In general, coverage path-planning is closely related to the intractable *covering-salesman problem*, where an agent must visit neighborhoods of multiple cities while minimizing travel-distance [5]. Extending this to multiple robots requires collision avoidance (with both static and dynamic obstacles), which becomes computationally expensive as the number of robots increase. For persistent coverage, additional algorithmic challenges emerge since robots need to continuously revisit parts of the environment to maintain coverage-levels. For a single robot tasked with persistent coverage, there are broadly two decisions to be made:

Q1 Where should the robot go next?
Q2 How should it get there?

We refer to **Q1** and **Q2** as *goal assignment* and *goal planning* respectively. Additional questions arise when dealing with multiple robots:

Q3 How do we plan for multiple robots?
Q4 How do we avoid inter-robot collisions?

Assuming a centralized planner, we address **Q3** by sequential, decoupled planning instead of planning in the joint state-space of all robots for tractability. Specifically, we use *prioritized planning* [28].

We use the notion of *committed plans* to tie our answers to these questions together. These are defined as parts of plans that robots commit to execute, in contrast to the rest of their plans which may be modified after reassessing the environment. We maintain the invariant of *global deconfliction*—no new committed plan intersects any other existing committed plan in space or time, thereby answering **Q4**.

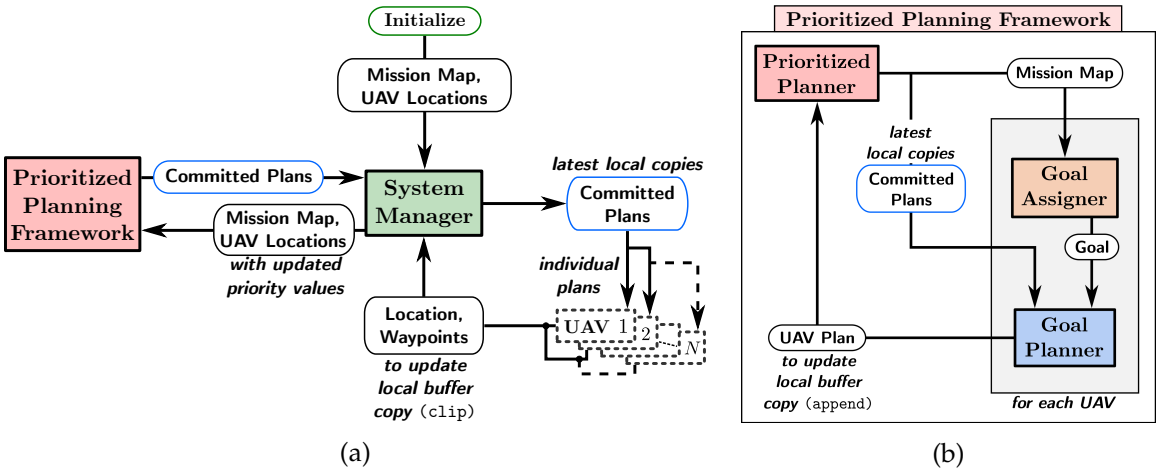


Figure 2.1: (a) The System Manager (SM) communicates with all UAVs; it sends up-to-date copies of committed plans to corresponding UAVs and updates them using information received from the Prioritized Planner (PP) (b) Prioritized planning framework: For each UAV, the Goal Assigner (GA) selects the next goal using the up-to-date map from the PP and the Goal Planner (GP) then plans a feasible path to this goal, which is appended to its committed plan

In this chapter, we present a planning framework to address all the above questions. A block diagram of the complete framework is presented in Fig. 2.1. To the best of our knowledge, this work is the first to answer all of these questions in unison for real-world, persistent coverage with multiple robots. For our application, these robots are Unmanned Aerial Vehicles (UAVs). The *System Manager* (SM, Sec. 2.4.1) is our communication interface between the centralized planner and individual UAVs. It is responsible for back-and-forth communication of up-to-date information between the two. The *Prioritized Planner* (PP, Sec. 2.4.2) is responsible for answering **Q3**. Thereafter, the *Goal Assigner* (GA, Sec. 2.4.3) answers **Q1** and specifies the location the next UAV should fly to. This is fed to the *Goal Planner* (GP, Sec. 2.4.4), which plans a kinodynamically feasible path for the UAV to the goal, answering **Q2**. Part of this plan is appended to the existing committed plan of the UAV. The PP, which operates continuously in a loop, then begins the next planning cycle, answering **Q3** again.

We only append a part of computed plans to committed plans of UAVs (further detailed in Sec. 2.4). This is done so that we maintain committed plans of a pre-specified maximum duration (denoted by $t_{\max}$)[1]. There is a trade-off involved in making $t_{\max}$ too short or too long. Sec. 2.4 provides more insight into how and why this arises. We rely on globally deconflicting paths for UAVs since other, more local collision-checking mechanisms have a higher risk of causing *deadlocks*[2]. Some other approaches utilise conflict

[1]This duration depends on the type and capability of the robots, the number of robots, and the coverage map.

[2]Two or more robots are in a *deadlock* if they are not in collision, but executing any valid action for any one

detection and resolution systems to repair conflicting paths [12]. Our approach solves for deconflicting paths by construction and maintains the invariant of global deconfliction.

In Sec. 2.2, we contextualize our work among the Orienteering Problem (OP) [98] and the Vehicle Routing Problem (VRP) [94], coverage path-planning [37], and frontier-based exploration [101]. Sec. 2.3 formalizes the persistent-coverage problem we aim to solve and introduces the notion of priorities over coverage zones. Sec. 2.4 provides details about our planning framework and each of its constituent parts. In Sec. 2.5, we present results of the performance of our framework in both simulated and real-world environments. Finally, Sec. 2.6 discusses the consequences of some design-related decisions and proposed extensions.

2.2 Related Work

A majority of existing literature casts the persistent coverage problem as an instantiation of the OP or the VRP. An OP seeks to determine a path for an agent limited by a time- or distance-budget that visits a subset of all surveillance sites in an environment and maximizes the sum of associated scores collected. An extension of this to multiple agents is known as the Team Orienteering Problem (TOP) [98]. There are a number of works that apply solutions of the OP to surveillance with aerial vehicles over a small number of surveillance sites [50]. The formulation by Leahy et al. comes closest to our work because of its capability of assigning time windows to each coverage-zone as temporal logic constraints [56].

Given a number of agents at a depot and distances among all surveillance sites and the depot, the VRP seeks to find a minimum-distance tour for each agent such that it visits each site at least once. Michael et al. address persistent surveillance with aerial vehicles as a VRP with modifications to model continuous site visits [40, 88].

The OP and VRP are both variants of the Traveling Salesman Problem (TSP) and thus NP-hard. Moreover, in our case, formulating the problem as a TOP or VRP is not enough since our problem also has an element of 2D coverage. One could consider every discrete part of the environment as a surveillance site in a TOP or VRP to enable 2D coverage, but this is impractical due to increasing computational complexity with the number of surveillance sites.

Smith et al. tackle a very similar problem by assuming pre-planned paths, decoupling path-planning and speed-control for deconfliction [86]. Planning paths for 2D coverage has been extensively studied in robotics [37]. Typical solutions to static coverage involve environmental partitioning via Voroni tessellations and feedback control laws with local

of the robots would cause them to collide with another. Hence, they remain stationary.

collision-avoidance schemes [82]. Environmental partitioning is also a recurring strategy for dynamic and persistent coverage [49]. However, such partitioning restricts individual robots to specific parts of the environment.

For real-world settings, robot path-planners must adhere to kinodynamic motion-constraints. Thakur et al. solve the TOP and path-planning simultaneously to generate 3D (x, y, θ) space-parametrized trajectories, but without temporal deconfliction [92]. Many also formulate planning for persistent surveillance as a mixed-integer linear program (MILP) [13]. However, such MILP formulations introduce limitations that are typically managed by planning for constant-speed paths [2].

A vital part of our framework is to ensure UAVs fly to appropriate goals, so that they simultaneously cover different parts of the environment. We adapt frontier-based exploration to assign such goals to UAVs. Yamauchi et al. first proposed this for single and multiple robots based on an occupancy-grid representation of the environment [101]. Many extensions have been proposed since then, which combine the information gain or expected benefit of the goal and the distance to it, also called next-best-view approaches [3]. Zhu et al. also apply this to exploration and coverage with micro aerial vehicles (MAVs) [103]. We use frontier-based exploration to trade off between cell-proximity and cell-criticality (defined formally in Sec. 2.3). To be able to do this online, we use a search-based approach similar to Butzke et al. [20].

2.3 Problem Formulation

Our problem definition is characterized by a mission-map $\mathcal{M}$, and a set of N UAVs $\{U_1, \ldots, U_N\}$, tasked with covering the area represented by $\mathcal{M}$. Each UAV U_k is a kinodynamically constrained system, with an associated coverage- or sensor-radius r_k. Let the cell at row i and column j of $\mathcal{M}$ be $c_{i,j}$. A UAV U_k is at cell $c_{i,j}$ if the projection of its reference point onto the x, y plane (denoted by U_k^{loc}) lies in cell $c_{i,j}$. A cell is said to be covered by a UAV flying over it if any point on the cell is at at most an r_k distance from U_k^{loc}. This effectively assumes a circular sensor-footprint of radius r_k centered around U_k^{loc}. Each cell $c_{i,j}$ is associated with two temporal properties—its *lifetime* $\ell(c_{i,j})$ and its *age* $a(c_{i,j})$. The age of a cell is the time passed since the cell was last covered by a UAV, while its lifetime is a desired bound on its age (as shown in Fig. 2.2).

There are two types of cells in our problem specification: (i) standard cells (that UAVs need to cover), and (ii) obstacle cells (that UAVs cannot fly over). Based on this, we define the following maps:

1. Priority Map, $\mathcal{M}^{\text{P}} = \{c_{i,j} : c_{i,j} \text{ is a standard cell} \wedge \ell(c_{i,j}) < \infty\}$. In other words, $\mathcal{M}^{\text{P}}$ is the set of all standard cells that a UAV will need to cover in finite time since the

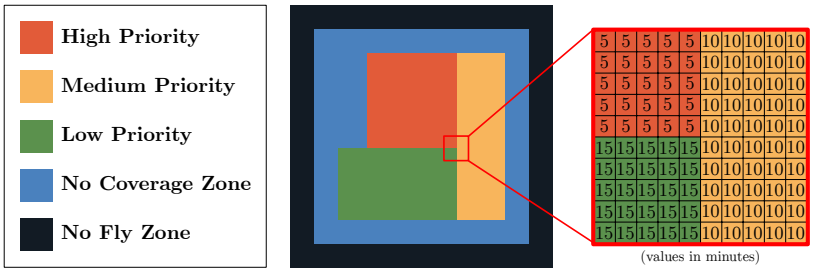


Figure 2.2: An example of a mission-map $\mathcal{M}$, where each cell is colored according to its lifetime. For example, a green cell with a lifetime of 15 minutes implies that no more than 15 minutes should pass between two consecutive times a UAV covers it.

start of the mission.

2. No-coverage Map, $\mathcal{M}^{\text{NC}} = \{c_{i,j} : c_{i,j} \text{ is a standard cell} \wedge \ell(c_{i,j}) = \infty\}$. In other words, $\mathcal{M}^{\text{NC}}$ is the set of all standard cells in the mission-map that a UAV can fly over, but does not need to cover.
3. Obstacle Map, $\mathcal{M}^{\text{O}} = \{c_{i,j} : c_{i,j} \text{ is an obstacle cell}\}$.

The mission-map can then be defined as $\mathcal{M} = \mathcal{M}^{\text{P}} \cup \mathcal{M}^{\text{NC}} \cup \mathcal{M}^{\text{O}}$. Fig. 2.2 contains an example of a mission-map with three different priority-levels. Given a mission-map and a set of UAVs, our problem is to compute UAV-plans such that the following properties hold:

P1 Feasibility—the motion of each UAV adheres to its kinodynamic constraints.

P2 Deconfliction—no two UAVs collide. In our specific setting, all UAVs are constrained to fly at the same altitude, so we say that two UAVs U_p and U_q collide if they come within some predefined distance d_{min} of each other.

P3 Persistence—the age of each cell is smaller or equal to its lifetime.

P4 Flexibility—UAVs can be dynamically added or removed to the system.

P1 and **P2** are hard constraints that must always be satisfied. **P3** depends on the mission specification—the number of UAVs deployed, their capabilities (speed, sensor radii etc.) and the mission-map (size and lifetime of each cell). Our framework supports **P4** without affecting **P1** and **P2**, however it will affect (help or hinder) our ability to satisfy **P3**.

2.4 Approach

Our approach, depicted in Fig. 2.1, consists of a *System Manager* (SM) which serves as an interface between the centralized planning framework and individual UAVs. It maintains the priority map $\mathcal{M}^{\text{P}}$ by updating locations of individual UAVs[3]. Every planning cycle,

[3]Our framework also allows for static obstacles and no-coverage zones to change during a mission, but this is beyond the scope of this thesis.

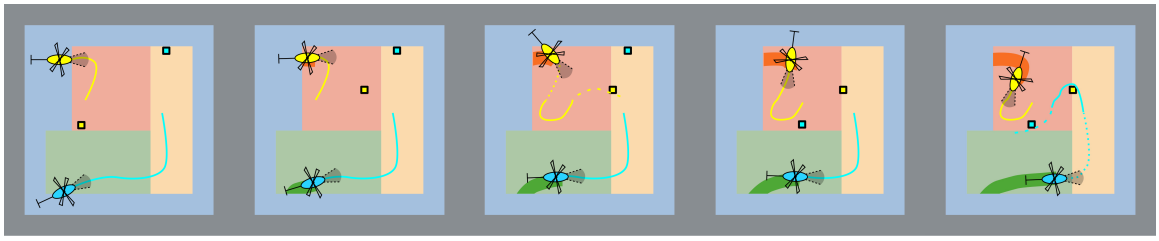
the SM passes the three maps to the *Prioritized Planner* (PP). The PP then iterates over all the UAVs in a round-robin fashion. For each UAV U_k, it generates a local copy of $\mathcal{M}_k^{\mathrm{P}}$ that accounts for the committed plans of the other UAVs. This priority map $\mathcal{M}_k^{\mathrm{P}}$ is used by the *Goal Assigner* (GA) to assign a goal (in terms of x, y location) for the UAV U_k. A kinodynamically feasible path to reach this goal is then computed for U_k using the *Goal Planner* (GP).

A key notion used within our framework is that of *committed plans*, which are collision-free plans each UAV is committed to execute. While the GP might compute plans of long durations (if the goal is far away or plan execution requires long, feasible maneuvers), only a portion of this plan will be added to the committed plan, such that it is at most $t_{\max}$ long in time. The value of $t_{\max}$ must be decided based on the mission since the following effects introduce a trade-off:

1. Effects of a high value for $t_{\max}$:
   - The longer the committed plans are, the less reactive UAVs are to any map updates including updates by an operator.
   - The path computed by the GP might take U_k through areas coinciding with parts of the committed plans of $U_{j\neq k}$ before $U_{j\neq k}$ covers them. This results in redundant coverage. The chances of this happening increase as $t_{\max}$ increases.
2. Effects of a low value for $t_{\max}$:
   - The shorter the committed plan, the lesser the chance of deviating from it during execution due to imperfect controllers. However, this increases the chances of UAV U_k executing the committed plan before the next planning cycle ends, leaving the UAV's controller with no plan to execute. We handle this by ensuring that the UAV can execute a stopping-maneuver in this situation, but repeated execution of stopping-maneuvers is undesirable.
   - Having short committed plans increases the frequency of goal assignment for a UAV. This causes two adverse effects: (i) A UAV may be assigned a new goal that is different from its previous one, which would cause it to abandon trying to reach the previous goal. (ii) Even if the same goal is assigned, it might lead to "plan thrashing". A UAV's complex dynamics may restrict short plans from letting it make significant progress towards a goal if complicated maneuvers are required.

2.4.1 System Manager

The System Manager (SM) is the communication interface between the centralized planner and the individual UAVs, shown in Fig. 2.1 (left): The SM sends the mission-map $\mathcal{M}$ to the PP, and it receives the updated committed plans from the PP and sends them out



(a) (b) (c) (d) (e)

Figure 2.3: Two successive executions of the GA-GP loop (the map $\mathcal{M}$ is colored with 50% opacity). (a) The solid yellow and blue lines show committed plans for both UAVs. (b) A new goal is assigned to the yellow UAV. (c) A path planned is for the yellow UAV (the old committed plan is a dotted line, the new committed plan is a solid line, and the discarded part of the new plan is a dashed line). (d) The same as (b) but for the blue UAV. (e) The same as (b) but for the blue UAV.

to the respective UAVs. It maintains the priority map $\mathcal{M}^{\mathrm{P}}$ by: (i) resetting the age of any standard cells that have been covered since the last update to zero, and (ii) incrementing the age of all other standard cells.

2.4.2 Prioritized Planner

The Prioritized Planner (PP) is a state-machine that controls our centralized planning framework by continuously executing RUNPLANNER from Alg. 2. During each execution of RUNPLANNER, which we call a *planning cycle*, the PP is responsible for the following tasks:

- **T1** Update its local copy of the committed plans and $\mathcal{M}^{\mathrm{P}}$ to reflect the most recent information received from the SM.
- **T2** Determine which UAV will be planned for during this cycle.
- **T3** Perform a new goal assginment for the UAV and plan a path to it[4].
- **T4** Communicate the result of **T3** throughout the framework.

T1 is performed in Lines 4–6 in Alg. 2. The PP clips the part of its local copy of committed plans that have been executed by the UAVs since the last planning cycle, and accounts for the part yet to be executed by resetting the age of all the cells in $\mathcal{M}^{\mathrm{P}}$ that will be covered by the committed plans to zero.

We do not prioritize any one UAV over the others, which corresponds to the PP selecting UAVs in a round-robin fashion, thus achieving **T2**. While the PP supports a prioritization over UAVs, in our experience, assigning equal priorities in this way showed satisfactory performance.

[4]Task **T3** is only done if the selected UAV's committed plan is shorter than $t_{\max}$.

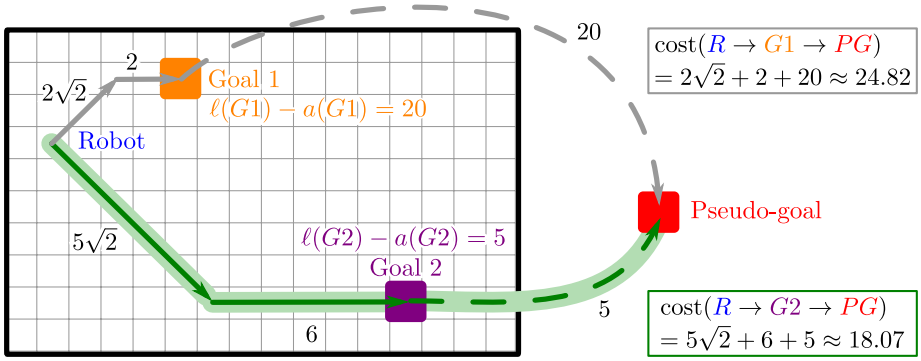


Figure 2.4: Multi-goal Dijkstra search for goal assignment finds the least-cost path to a pseudo-goal connected to all potential goal locations for a robot. It considers the cost of reaching a potential goal from the robot's location, and the priority of a goal location reflected in the cost of the edge connecting it to the pseudogoal. In the example above, goal G2 is assigned to the robot.

T3 is achieved through one iteration of the GA-GP loop shown in Fig. 2.1b. For ease of understanding, we show two successive executions of this loop in Fig. 2.3. Fig. 2.3b and 2.3c correspond to Lines 9 and 10 for the yellow UAV. Fig. 2.3d and 2.3e correspond to the same lines for the blue UAV.

T4 is performed by Lines 11–13. First, we append to a UAV's committed plan to bring it up to $t_{\max}$ in length. Second, $\mathcal{M}^{\mathrm{P}}$ is updated to account for this new plan (same as in Line 6) by resetting the age of all the cells that will be covered. Finally, the newly committed plan is communicated to the UAV via the SM.

Algorithm 2 Prioritized Planner Loop

```
1:  procedure RunPlanner
2:      N ← number of UAVs
3:      while true do
4:          Get latest M = {M^P, M^NC, M^O} and U^loc_{1:N} from SM
5:          Update committed plans (local copies) by using U^loc_{1:N}        ▷ Clip
6:          Update M^P with committed plans
7:          for k ← 1 : N do
8:              if committed plan for U_k is short then
9:                  G_k ← GetGoal(M^P)                                  ▷ Goal assignment (GA)
10:                 π_k ← GetPlan(M, G_k)                               ▷ Goal planning (GP)
11:                 Update committed plan for U_k with π_k              ▷ Append a portion of π_k
12:                 Update M^P with committed plan for U_k
13:                 Send committed plan for U_k to SM
```

2.4.3 Goal Assigner

Recall that the Goal Assigner (GA) selects a goal for each UAV to fly to. The grid $\mathcal{M}^{\mathrm{P}}_k$ is the local copy of the Priority Map ($\mathcal{M}^{\mathrm{P}}$) that belongs to UAV U_k. Each of the cells in

the eight-connected grid that represents $\mathcal{M}_k^{\mathrm{P}}$ is a potential goal for U_k. The GA computes a goal-location G_k for each UAV, accounting for how urgent every cell in $\mathcal{M}_k^{\mathrm{P}}$ is and the location of the UAV U_k. To simultaneously reason about the urgency of the goal and its distance from U_k, we build a graph by connecting a *pseudo-goal* to every cell of the grid $\mathcal{M}_k^{\mathrm{P}}$. The pseudo-goal is an "imaginary" state connected by "imaginary" *pseudo-edges* to all cells in the grid as shown in Fig. 2.4. The cost of these pseudo-edges is proportional to $(\ell(i,j) - a(i,j))$[5]. The costs of the rest of the edges between adjacent cells in the grid are equal to the Euclidean distance between them.

To find an optimal goal G_k, we first find an optimal path from U_k^{loc} to the pseudo-goal on this graph using Dijkstra's search [25]. The optimal path minimizes the sum of the costs of the real-edges (which reflects the cost to reach the goal) and the cost of the pseudo-edge (which reflects the priority of the cell that the pseudo-edge connects). The final goal G_k for UAV U_k is the parent of the pseudo-goal on this optimal path.

2.4.4 Goal Planner

Once a goal is assigned to U_k, the GP computes a kinodynamically feasible path to it via search-based planning on a state-lattice [57, 75]. The *state-space* for each UAV includes its two-dimensional pose (x, y, θ), linear velocity v, and time t. Assuming double-integrator dynamics for each UAV, we generate an action-space consisting of feasible motion primitives. These primitives use cells of size 1m $\times$ 1m. This discretization is independent of the mission-map discretization. We implicitly construct a graph using the action space during a weighted-A* search to the goal [42, 77]. The search prunes away all transitions that correspond to trajectories that either intersect no-fly zones or collide with the committed plans of other vehicles in space or time. We terminate the search as soon as a state is expanded whose incoming edge (from the predecessor on the found path) covers the goal cell (specifically, the trajectory corresponding to this edge contains a point whose distance to the goal cell is less than r_k). We assign the time taken to execute an action as its edge-cost in the graph. While our aim is to plan for minimum-time paths, it is also desirable for the UAVs to fly at high speeds whenever possible and avoid stopping. For this reason, we incentivize actions with increasing velocities, penalize actions with decreasing velocities, and heavily penalize hovering.

[5]We lower-limit these costs by a constant to ensure they are strictly positive. Moreover, since the UAVs' sensors cover multiple cells, the cost of an edge between cell $c_{i,j}$ and the pseudo-goal is the average of $\ell(u,v) - a(u,v)$ for all cells $c_{u,v}$ covered when $U_k^{\mathrm{loc}} = c_{i,j}$.

2.5 Experimental Setup and Results

We evaluate our framework in both simulation, where we assume perfect state estimation and control, and on real UAVs, which can deviate from their planned paths. The UAVs can withstand winds of speeds up to 30 m/s. Thus, the effect of wind is negligible under normal conditions. Accounting for large deviations from planned paths requires replanning and is part of future work. Our framework uses an identical set of parameters in both cases. We generate motion primitives with a maximum speed of 6 m/s and a maximum turning rate of 6 °/s. We avoid adding angular velocity to the state-space by ensuring that these primitives start and end at zero angular velocity. Two UAVs are deemed to have *collided* if at any point in time the distance between these 2D locations is less than $d_{\text{min}} = 10$ m. The parameter $r_{\text{k}} = 15$ m defines the circular area directly underneath a UAV that is deemed *covered* for any 2D location of the UAV. We impose a planning timeout of 4 s on the Goal Planner, which is how long it is given to compute a plan. We use an Euclidean-distance heuristic in the Weighted-A* search [77] with an inflation of 5. Fig. 2.7d and 2.7e shows the two coverage-maps used for the experiments presented in this chapter.

2.5.1 Evaluation Metrics

We look at timing statistics for Goal Assignment (t_{GA}) and Goal Planning (t_{GP}), number of state expansions, and number of expansions per second[6]. Since we desire continued movement from each UAV, we also look at the amount of time a UAV is stationary on average across the simulation run (t_{stopped}). Additionally, we evaluate our framework's performance with respect to the *criticality* of a cell, defined for any cell (i, j) at timestamp t as $C_t(i, j) = \frac{a_t(i,j)}{\ell(i,j)}$, where $C_t : \mathbb{E}(2) \to \mathbb{R}$ is a time-varying measure of criticality. $C_t(i, j)$ starts with a value of zero and as the cell *ages* over time, $C_t(i, j)$ starts to increase. Once the age of a cell reaches its lifetime—namely, $C_t(i, j)$ equals one—we say that the cell has *expired*. If we average this value across all cells in the map to be covered, we obtain an estimate idea of how well a team of UAVs is covering the areas they have been assigned:

$$\bar{C}_t = \frac{1}{|\mathcal{M}^{\text{P}}|} \sum_{(i,j) \in \mathcal{M}^{\text{P}}} C_t(i, j) \tag{2.1}$$

Given this measure of criticality, ideal behavior would be to keep the value of $\bar{C}_t$ constant over the course of a mission, a lower value being better.

[6]These expansions refer to graph-node expansions in the Weighted-A* search.

Map	Number of UAVs	Timing (ms)			Path Planning		
		t_{GA}	t_{GP}	t_{Total}	Number of Expansions	Expansions per second	t_{stopped} (%)
Fig. 2.7d	1	90	223	323	413	215	0
	2	90	796	898	187	226	2
	3	91	1326	1430	286	209	2
Fig. 2.7e	1	92	172	275	28	204	0
	2	91	1163	1265	200	188	< 1
	3	91	1433	1536	248	185	3

Table 2.1: Results of simulation experiments (values rounded down to the closest integer) and t_{stopped} expressed as a percentage of total mission time.

2.5.2 Simulation Experiments

Hardware Platform Simulation experiments were performed on a desktop computer running Ubuntu 16.04 and equipped with an Intel Core i7-4790K processor and 20 GB of RAM.

Planner Evaluation We present simulation results on both maps from Fig. 2.7 for 10 minutes in Table 2.1. We observe that by increasing the number of UAVs, Goal Assignment times are not affected, whereas Goal Planning times increase significantly since GP must now compute deconflicting plans for a larger number of UAVs. Despite this, the total amount of stationary time spent by a UAV was at most 19.90 s in a 10-minute period. Fig. 2.5 shows a plot of $\bar{C}_t$ (Eq. (2.5.1)) over time during a 30-minute mission. In all experiments, our framework is able to keep $\bar{C}_t$ below a value of one except where the number of UAVs was too small given the size of the map (experiment with one UAV and map from Fig. 2.7). This, along with the fact that all plots plateau over time, implies that cell-expiration is regulated on average.

Global Deconfliction Satisfactory global deconfliction relies on each UAV always having access to committed plans that are long enough. As discussed in Sec. 2.4, size of committed plans can significantly affect performance. We empirically evaluate the need and effect of global deconfliction by varying the lengths of committed plans for different collision-checking schemes. These schemes are:

1. Proposed Framework: Our approach with global deconfliction.
2. No Collision Checking: The GP performs no collision-checking between UAVs.
3. Current Position (static): The GP considers the other UAVs' current positions as static

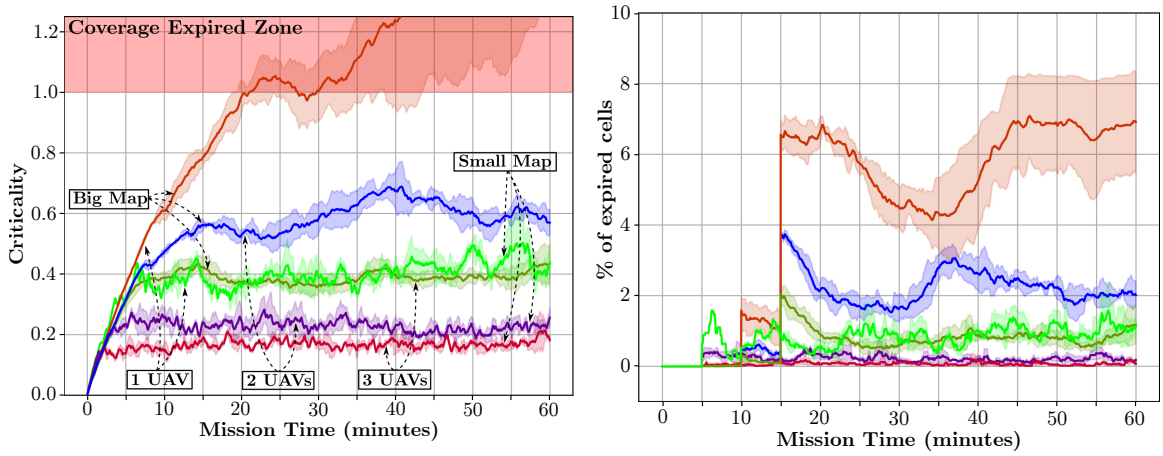


Figure 2.5: Left: Average criticality of cells in $\mathcal{M}^{\mathrm{P}}$ during simulated experiments. Right: Spikes occur at 5, 10, 15 minutes since these are the three different cell lifetimes in the initial $\mathcal{M}^{\mathrm{P}}$. Colors are consistent across the two plots.

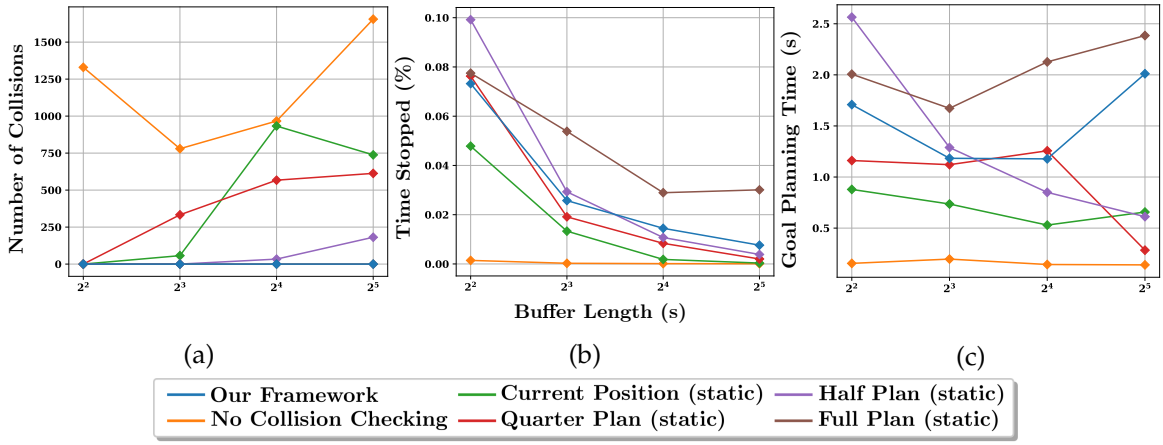
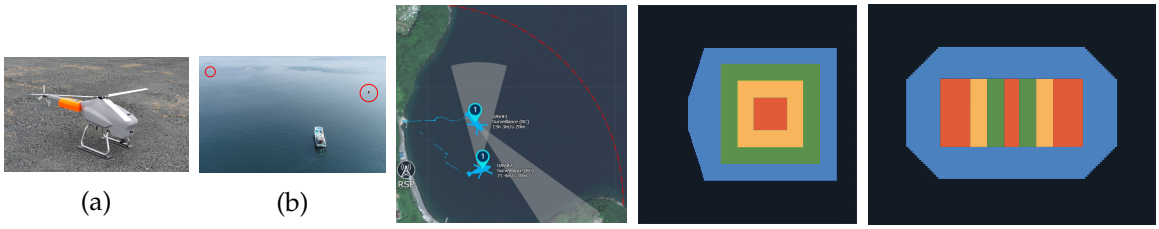


Figure 2.6: Effect of changing the committed plan length for different collision checking mechanisms. Since each curve represents a single mission execution, we do not show confidence intervals. The lines for *Full Plan (static)* and *Our Framework* coincide in (a).

obstacles.

4. Quarter, Half and Full Plan (static): The GP assumes the first quarter, half and whole of the committed plans of other UAVs to be static obstacles respectively.

We plot the number of collisions between UAVs in Fig. 2.6a, the average time a UAV is stationary in Fig. 2.6b, and the time taken by the Goal Planner in Fig. 2.6c, all against varying lengths of committed plans. Fig. 2.6 shows that our deconfliction scheme results in competitive planning times and also guarantees collision-free UAV movement with negligible



(a) (b) (c) (d) (e)

Figure 2.7: (a) UAV used in the experiments. (b) The two UAVs (circled in red) executing a mission. (c) GUI showing a satellite image of the mission site along with UAV data overlay. (d) Map used for experiments (big; coverage zone roughly 400m $\times$ 400m). (e) Map used for experiments (small; coverage zone roughly 200m $\times$ 100m)

Map	**Number of UAVs**	**Timing** (ms)			**Path Planning**		
		t_{GA}	t_{GP}	t_{total}	Number of Expansions	Expansions per second	t_{stopped} (%)
Fig. 2.7e	1	105	240	363	37	186	0.00
	2	117	1181	1325	162	152	7.12

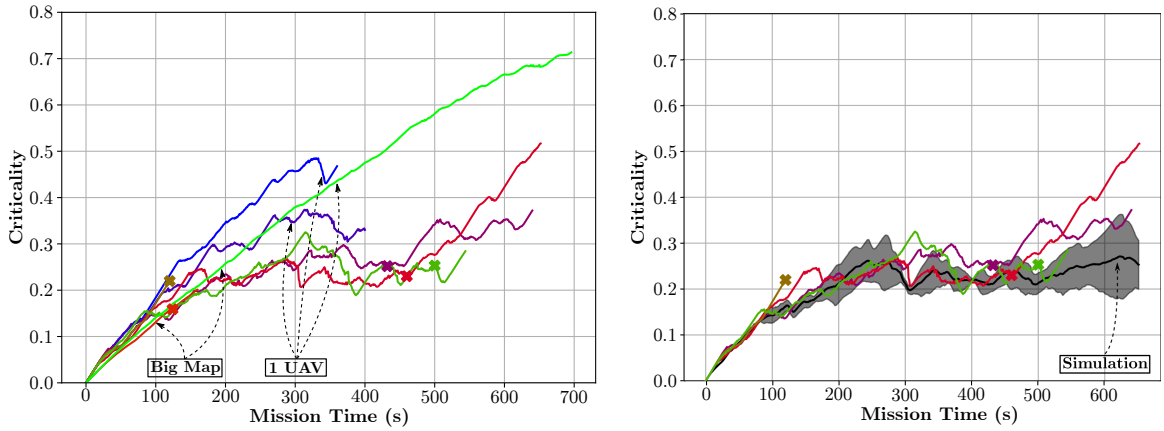
Table 2.2: Results of real-world experiments averaged over 8 runs (rounded down to the nearest integer) and t_{stopped} expressed as a percentage of total mission time (maximum over all runs).

stoppage. Other collision-checking schemes are either overly conservative and compute convoluted, long-winding paths, or simply fail to avoid collisions.

2.5.3 Real-World Experiments

Hardware Setup For the real-world experiments, our planning framework was run on a Lenovo T470s laptop running Ubuntu 16.04 and equipped with an Intel Core i7-7600U processor and 20 GB of RAM.

Planner Evaluation We present results from real-world experiments in Table 2.2. Fig. 2.8a plots $\bar{C}_t$ for eight real-world runs. Additionally, dynamic removal of UAVs from the mission was tested in five of these runs, and the remaining UAVs were left undisturbed. Beyond this point, we are only covering the area with one UAV. Hence, it is expected to see criticality increase. By iteratively planning and executing computed paths, we isolate our planner from the stochasticity of controller-based execution in the real world. Our planner quantitatively performs just as well in the real-world as it does in simulation in spite of this stochasticity. This is because our framework seeks latest information about UAV and map statuses from the real world and constantly uses it to repeatedly solve our myopic version of the full problem (as explained in Sec. 2.4).



(a) (b)

Figure 2.8: (a) Coverage criticality over time in the real-world: unless explicitly pointed to by arrows, the curves represent an experiment with two UAVs run on the smaller map. A cross (×) on a curve indicates the point in time when one of the two UAVs was removed from the mission. (b) Comparison of all simulation and real-world missions executed in the small map from Fig. 2.7e by two UAVs. The black curve with confidence intervals corresponds to the simulated experiments.

2.6 Conclusion and Future Work

We present and evaluate a planning framework for real-world, persistent coverage with multiple UAVs. Our framework continuously decides where UAVs should fly and computes kinodynamically feasible, globally deconflicting plans. We evaluate our framework in both simulated and real-world settings. We also motivate and compare global deconfliction with weaker, more local collision-avoidance schemes.

In many practical settings like ours, state spaces are high-dimensional and time for deliberation is limited. Planning times can be a bottleneck in these cases and cause delays. While our stopping maneuvers handle such situations, a natural extension is to incorporate anytime planning [89].

In the current framework, the goal assignment is based on priorities and is decoupled from goal planning. This is greedy and not optimal. Better strategies to repeatedly cover previously observed coverage-zones can be learned from data and added as macro-actions in the planner.

3 Planning for Active Sensing

3.1 Introduction

Path planning for traditional robotic coverage is the task of determining a collision-free robot trajectory that observes all points of interest in a given environment [37]. Numerous real-world tasks including environmental exploration, traffic monitoring, and post-disaster assessment can be cast as robotic coverage problems [65, 85, 87, 91]. Robots employed for such coverage tasks are often equipped with limited-range sensors to observe the environment and can exercise active control over them. An important problem is to plan robot and sensor trajectories that maximize coverage or information gain in these tasks, while also respecting kinodynamic constraints and arriving at a goal location. This is akin to an *informative path planning* problem. Real-time kinodynamic planning is a computationally expensive problem in itself due to the many degrees of freedom in a kinodynamic robot state. Additionally planning trajectories for sensors onboard these robots further increases the computational complexity of this task.

In this chapter, we consider the specific problem of planning trajectories for an unmanned aerial vehicle (UAV) and its onboard sensor covering cells of a discrete map—representing a known, deterministic environment—to achieve efficient 2D area coverage while navigating to an assigned goal. We assume that the UAV flies at a fixed altitude, and that the yaw angle of a pan-only camera onboard the UAV can be controlled, thereby controlling the camera's projected footprint on the ground. In turn, we include the robot's x and y coordinates, heading θ, velocity v, timestamp t, and sensor angle ψ in our state space—making it *at least* a 6-degree-of-freedom (6-DoF) planning problem (we describe in Sec. 3.3 how this can amount to planning in more than 6 DoFs). We tackle this problem using a search-based approach, which comes with the advantage of guarantees on solution quality up to the discretization of the graph representing the problem.

However, naïvely applying search-based approaches to such a problem results in high computational cost. A key challenge in planning non-myopic sensor trajectories that maximize coverage is that in general, for a given robot trajectory, the optimal sensor configuration at a given point in the trajectory depends on all previous sensor measurements (the full *sensor history*). One can appreciate this in 2D environments by thinking of sensor footprint overlaps: to compute an optimal sensor configuration at a given location, a planner must take into account all overlaps with previously planned sensor footprints in the plan being considered. However, including the full sensor history in a state makes the search computationally intractable. Our first approach, **S**ensor **Pla**nning with **S**ensor **H**istory (SPLASH), first computes a robot trajectory. It then searches for sensor plans for this fixed robot trajectory while maintaining a *partial* sensor history during the search—this

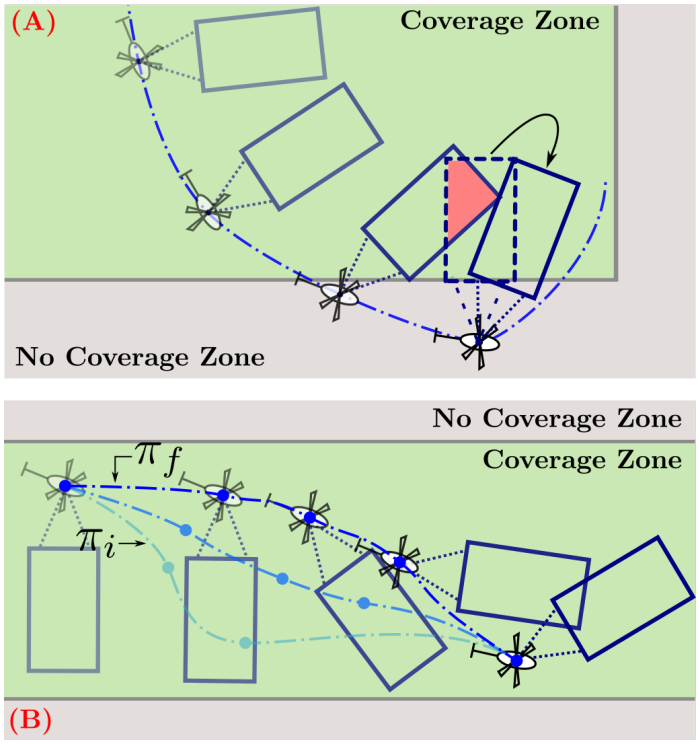


Figure 3.1: We present two algorithms for search-based planning for active sensing. (A) SPLASH tries to minimize sensor footprint overlaps along the robot trajectory. For the last state in the figure, it would prefer the solid blue sensor footprint over the dashed blue so as to avoid the overlap denoted by the red area. (B) SPLIT iteratively refines an initial trajectory π_i to maximize the area covered by the sensor to come up with the final trajectory π_f.

prevents covering areas already sensed before in the trajectory, up to the maintained sensor history horizon. Even for a fixed robot trajectory, the space of sensor trajectories exponentially increases in dimensionality as we account for longer sensor histories (this might not be trivial to see, and so we detail how this occurs in Sec. 3.4). Our results show that in most planning problems considering 2D sensor footprint overlaps, only a partial history of sensor footprints actually affect computing the next optimal sensor configuration.

The basis of our second approach is the empirical observation that approximate search algorithms like Weighted A* (WA*) overlook better solutions that are actually "close" to the computed solution in the space of solution paths [32, 77]. In our second approach, **S**ensor **P**lanning with **L**ocal **I**terative **T**unneling (SPLIT), we *split* the process into two steps: (1) We first quickly compute suboptimal robot and sensor trajectories in *decoupled* robot and sensor state spaces using SPLASH (initialization). (2) We then use this solution as initialization to a local-search routine that iteratively improves this solution in the *joint* robot and sensor state space until time runs out (refinement). We adapt Iterative Tunneling

Search with A* (ITSA*) [32] to our problem for this refinement step. This is detailed in Sec. 3.5.

The illustation in Fig. 3.1 depicts both of our approaches on a toy example. Our approaches can be contextualized within several related works on sensor management and informative path planning, as we do in Sec. 3.2. In Sec. 3.3, we describe our problem and notation in detail. In sections 3.4 and 3.5, we describe and provide pseudocode for both of our approaches. In Sec. 3.6, we evaluate our approaches and show their benefits in the context of a previously established planning framework for persistent coverage with multiple UAVs [52], where individual UAVs are tasked with generating collision-free trajectories that maximize coverage while navigating to continuously assigned goals. Note that our contribution lies in planning robot and sensor trajectories for a single UAV navigating to a goal (we do not attempt to solve the problem of coordinating UAV plans for coverage). To the best of our knowledge, no previous work applies search-based planning to this problem.

3.2 Related Work

Hero and Cochran present an extensive survey on sensor management [43]. Gutpta et al. state general challenges and computational complexity of optimal sensor selection in detail in [41]—this is on similar lines as the computational challenge of maintaining a sensor history (see Fig 3.4, explained later). In general, optimal coverage has been addressed in various settings including mobile sensors and autonomous robots. Robotic sensing systems have used with both fixed sensors [33, 45] as well as sensors that execute precomputed patterns [80]. The problem of optimal mobile sensor location with unbounded ranges has been tackled as Voronoi space partitioning in [26]. Many approaches have also been targeted to specific applications, such as active perception work [24, 63]. The measurement control problem, also essentially a sensor scheduling problem, was shown to be solved by tree-search in general [61]. To deal with computational intractability, several greedy solutions have been proposed [23, 41, 47, 64, 67]. Further, Finite-horizon model predictive control provide improvement over myopic techniques but suffer from high runtimes in large state spaces and provide no performance guarantees beyond the horizon depth [16, 79]. Arora et al. propose a data-driven approaches to sensor trajectory generation that map calculated features to sensory actions [6].

Several recent works that fall under informative path planning are closely related to our work. Perhaps the most closely related is a recent line of work on active information acquisition, although with *fixed* sensors onboard robots: Atanasov et al. propose a non-greedy, value-iteration based offline solution with applications to gas distribution mapping and

target localization [7]. Schlotfeldt et al. then reformulate the problem as a deterministic planning problem and apply A* search with the first consistent heuristic for information acquisition, with applications to active mapping [81]. Kantaros et al. then propose a probabilistically complete and asymptotically optimal sampling-based approach to this problem, along with strategies to bias exploration toward informative regions [48]. Lu et al. propose a potential-function based method for integrated planning and control of robotic sensors deployed to classify multiple targets in an obstacle-populated environment [58].

There are also lines of work that formulate information gathering as an Orienteering Problem. Of particular interest are [30, 72, 73, 97] because of a similarity in their approaches with our initialize-and-refine approach in SPLIT (detailed further in Sec. 3.5). However these approaches focus on computing informative *tours*—unlike our goal-directed setting, and perform local refinement over heading angles constrained by Dubin's-car dynamics—unlike our approach that refines sensor angles that observe the environment.

3.3 Problem Formulation and Notation

3.3.1 Persistent coverage framework

We contextualize and evaluate our approaches within the persistent-coverage framework established in previous work [52]. This is a centralized framework that continuously computes goal locations to which UAVs should fly and kinodynamically feasible, globally deconflicting plans for them to do so, in a *prioritized planning* setting [28]. While it is a multi-UAV system, we plan for UAVs independently—plans between UAVs are not explictly coordinated in [52] and is out of the scope of this chapter. Our specific contribution lies in planning robot and sensor trajectories for a single UAV navigating to a goal. The framework in [52] assumes a circular sensor footprint directly underneath the UAV. In this chapter, we extend the system to incorporate a rectangular footprint offset from the UAV—consequently, different sensor headings correspond to the UAV observing different areas of the environment around it.

Map. The environment map $\mathcal{M}$ consists of a priority map $\mathcal{M}^{\mathrm{P}}$ and a no-coverage map $\mathcal{M}^{\mathrm{NC}}$. The UAV must attempt to cover each cell $c_{i,j}$ at row i and column j of $\mathcal{M}^{\mathrm{P}}$. Such a cell is associated with two values: its lifetime $l(i,j)$ and age $a(i,j)$—the age of a cell is the time passed since the cell was last covered by a UAV, and its lifetime is a desired bound on its age. At any point of time t, $\mathcal{M}^{\mathrm{P}}$ holds the quantity $p(i,j) = l(i,j) - a(i,j)$ for each cell $c_{i,j}$. $\mathcal{M}^{\mathrm{P}}$ *decays* with time, meaning $p(i,j)$ for each cell $c_{i,j}$ reduces by one every second, thus making $c_{i,j}$ more *urgent*. Cells part of $\mathcal{M}^{\mathrm{NC}}$ do not need to be covered.

Sensor. In our setting, the sensor is a pan-only camera with one controllable DoF (yaw), which controls a downward-looking rectangular footprint of fixed and limited field-of-

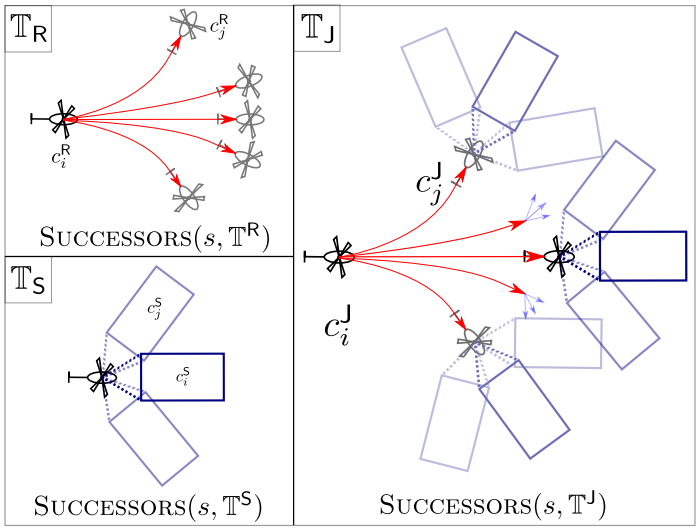


Figure 3.2: An example of the successor-generation functions for the three search spaces described in Sec. 3.3.

view. The area of the footprint is discretized into cells on the map according to an underlying resolution. We assume no noise in the footprint observed by the sensor.

Robot. The UAV is a kinodynamically constrained system, accounting for the robot's x and y coordinates, heading angle θ, velocity v, and timestamp t. The UAV is said to be at a cell $c_{i,j}$ if the projection of its reference point onto the xy-plane lies in cell $c_{i,j}$. A cell is said to be covered by the UAV if any point on the cell is contained in the rectangular projection of the sensor footprint on the xy-plane.

3.3.2 Problem formulation and definitions

We represent this planning problem as a search over a finite, discrete search space. Here, we define the configuration spaces of the robot (UAV) and sensor, and three state spaces that are relevant to our approaches. Each state space is associated with a set of transitions, and they together define three separate search spaces.

Robot state space

A feasible robot configuration is represented by $c^R = (x, y, \theta, v, t)$ where x and y are the robot's $2D$ coordinates, θ is the UAV's heading, and t is the global timestamp at which this configuration is achieved (the timestamp t is part of c^R as we plan spatiotemporally collision-free trajectories for multiple robots in this framework). These five degrees of freedom together define the $5D$ robot state space $\mathbb{E}_R$. A set of kinodynamically feasible

motion primitives computed offline define a state lattice [75] via a set of transitions

$$\mathbb{T}_{\mathsf{R}} = \{(c_i^{\mathsf{R}}, c_j^{\mathsf{R}}) \mid c_i^{\mathsf{R}}, c_j^{\mathsf{R}} \in \mathbb{E}_{\mathsf{R}}\}$$

This defines a search space represented by a graph $\mathbb{G}_{\mathsf{R}}$ with nodes $\mathbb{E}_{\mathsf{R}}$ and edges $\mathbb{T}_{\mathsf{R}}$. A robot trajectory π_{R} is a sequence of feasible robot configurations.

Sensor state space

A sensor configuration is defined with respect to a corresponding robot configuration c^{R} as a tuple $c^{\mathsf{S}} = (t, \psi, H^{\psi})$, where t is the timestamp in c^{R}, ψ is the sensor's heading angle in the global frame, and H^{ψ} is a list denoting the history of sensor angles assigned at all timestamps earlier than t. These state variables collectively define the sensor state space $\mathbb{E}_{\mathsf{S}}$, with dimensionality $(1 + |H^{\psi}|)$[3]. The set of feasible sensor motions define a set of transitions

$$\mathbb{T}_{\mathsf{S}} = \{(c_i^{\mathsf{S}}, c_j^{\mathsf{S}}) \mid c_i^{\mathsf{S}}, c_j^{\mathsf{S}} \in \mathbb{E}_{\mathsf{S}}\}$$

This defines a search space represented by a graph $\mathbb{G}_{\mathsf{S}}$ with nodes $\mathbb{E}_{\mathsf{S}}$ and edges $\mathbb{T}_{\mathsf{S}}$. A sensor trajectory π_{S} is a sequence of sensor configurations.

Joint state space

A feasible *joint-state* configuration c^{J} is a concatenation of a feasible robot configuration and sensor configuration $\langle c^{\mathsf{R}}, c^{\mathsf{S}} \rangle$. These state variables collectively define the joint state space $\mathbb{E}_{\mathsf{J}}$ of dimensionality $(6 + |H^{\psi}|)$. The set of feasible transitions in $\mathbb{E}_{\mathsf{J}}$ is a combination of feasible transitions in $\mathbb{E}_{\mathsf{R}}$ and $\mathbb{E}_{\mathsf{S}}$

$$\mathbb{T}_{\mathsf{J}} = \{(c_i^{\mathsf{J}}, c_j^{\mathsf{J}}) \mid c_i^{\mathsf{J}}, c_j^{\mathsf{J}} \in \mathbb{E}_{\mathsf{J}}\}$$

This defines a search space represented by a graph $\mathbb{G}_{\mathsf{J}}$ with nodes $\mathbb{E}_{\mathsf{J}}$ and edges $\mathbb{T}_{\mathsf{J}}$. Note that since the state-lattice discretization in $\mathbb{E}_{\mathsf{R}}$ can be different from that in $\mathbb{E}_{\mathsf{S}}$, the transition set $\mathbb{T}_{\mathsf{J}}$ consists of actions that change robot and sensor states at their respective state discretizations.

Given these search spaces, we define the routine $\textsc{Successors}(s, \mathbb{T}_{\mathsf{R}})$ to be the successor-generation routine for a state s that returns successor states in $\mathbb{E}_{\mathsf{R}}$. Similarly, we have the routines $\textsc{Successors}(s, \mathbb{T}_{\mathsf{S}})$ and $\textsc{Successors}(s, \mathbb{T}_{\mathsf{J}})$. Fig. 3.2 illustrates these three types of successors, although with much smaller branching factors for $\mathbb{T}_{\mathsf{R}}$ and $\mathbb{T}_{\mathsf{J}}$. Specifically, in $\mathbb{T}_{\mathsf{J}}$ for example, we generate 3 sensor-space successors for points at every

[3]Note that t is a known variable and no search is performed over it, and thus it does not contribute to the dimensionality of c^{S}.

1s of a 4-second long motion primitive. We have 12 motion primitives per robot state on average, making the branching factor in the joint space $12 \times 4 \times 3 = 144$ on average. We also denote running algorithm X searching for a path from s_{start} to s_{goal} in search space $\mathbb{G}$ by X(s_{start}, $s_{\mathsf{goal}} \mid \mathbb{G}$). For example, A*($s_{\mathsf{start}}$, $s_{\mathsf{goal}} \mid \mathbb{G}_{\mathsf{J}}$) denotes running A* search in the search space determined by $\mathbb{G}_{\mathsf{J}}$ (meaning state transitions are determined by $\mathbb{T}_{\mathsf{J}}$).

3.3.3 Cost Function

We now define the cost function associated with a transition from state s to s'. We use two costs—one associated with sensor coverage at s' where $s' \in \mathbb{E}_{\mathsf{S}}$ or $\mathbb{E}_{\mathsf{J}}$, and the other associated with the UAV's motion primitive from s to s' where $s, s' \in \mathbb{E}_{\mathsf{R}}$ or $\mathbb{E}_{\mathsf{J}}$.

Motion primitive cost. Each motion primitive is a sequence of states forward-simulated from the corresponding robot state at s, $s_{robot} = (x, y, \theta, v, t)$, following double-integrator dynamics. The cost of the primitive is equal to the time taken for the UAV to execute it. More details about the motion primitives can be found in [52].

Sensor coverage cost. For the corresponding sensor state at s', $s'_{sensor} = (t, \psi, H^{\psi})$, the variables x, y, θ, ψ together define a 2D specific footprint of cells $\mathcal{F}$. Let a given footprint $\mathcal{F}$ cover $|\mathcal{F}|$ discrete cells in the map $\mathcal{M}$. Let the number of these cells lying in a coverage zone be given by $N_{\mathcal{C}}$ and those lying in a no-coverage zone be $N_{\mathcal{NC}}$:

$$N_{\mathcal{C}} = \sum_{i \in \mathcal{F}} \mathbb{1}\left[i \in \mathcal{M}^{\mathrm{P}}\right] \text{ and } N_{\mathcal{NC}} = \sum_{i \in \mathcal{F}} \mathbb{1}\left[i \in \mathcal{M}^{\mathrm{NC}}\right]$$

No sensor history

If we ignore the sensor history, the cost of a footprint is given by the sum of priorities of all coverage cells in $\mathcal{F}$, and an additive penalty λ scaled by the fraction of no-coverage cells in $\mathcal{F}$:

$$\mathsf{cost}_0(\mathcal{F}) = \overbrace{\sum_{i \in \mathcal{F} \wedge \mathcal{C}} p_i}^{\text{criticality measure}} + \overbrace{\lambda \times \frac{N_{\mathcal{NC}}}{|\mathcal{F}|}}^{\substack{\text{penalize} \\ \text{no-coverage cells}}} \tag{3.1}$$

With sensor history

We define the following sensor coverage cost for this footprint $\mathcal{F}$ (where '$\mathbb{1}$' represents the indicator function):

$$\mathsf{cost}_H(\mathcal{F}) = \overbrace{\sum_{i \in \mathcal{F} \wedge \mathcal{C}} \underbrace{\mathbb{1}[i \notin \mathcal{H}^{\psi}] \times p_i}_{\text{not in history}} + \underbrace{\mathbb{1}[i \in \mathcal{H}^{\psi}] \times l_i}_{\text{in history}}}^{\text{criticality measure}} + \overbrace{\lambda \times \frac{N_{\mathcal{NC}}}{|\mathcal{F}|}}^{\substack{\text{penalize} \\ \text{no-coverage cells}}} \tag{3.2}$$

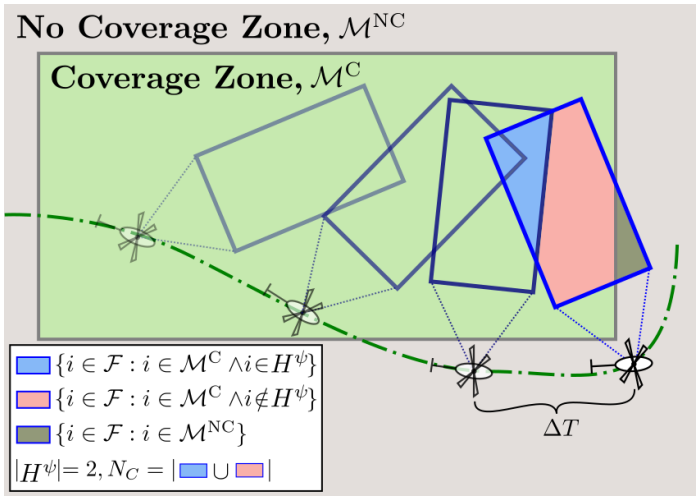


Figure 3.3: Pictorial explanation of our cost function $\text{cost}_H(\mathcal{F})$ from Eq. 3.2. We consider history size, $H = 2$ in this example. For the last UAV state on the green trajectory π_{R}, the sensor footprint is shaded in three colours. The blue area is the overlap with previous footprints in π_{R}, while the red and dark green areas do not overlap. For Eq. 3.2, the blue area is *in history*, the red area is *not in history*, and the dark green area is *penalize no-coverage cells*. Note that for footprints too far in the past, even if there was an overlap, it has no effect.

This cost function is illustrated in Fig. 3.3. Note that Eq. 3.2 reduces to Eq. 3.1 when no history is considered.

3.4 Sensor Planning with Sensor History (SPLASH)

In this section, we describe out first approach, SPLASH. SPLASH first quickly computes a suboptimal robot trajectory using Multi-Heuristic A* (MHA*) search in $\mathbb{G}_{\mathsf{R}}$. This search is performed with the motion primitive cost function (Sec. 3.3.3). Then, it computes a sensor trajectory using (uninformed-)A* search in $\mathbb{G}_{\mathsf{S}}$ for a given history H. This search is performed with the sensor coverage cost function $\text{cost}_H(\mathcal{F})$ (Sec. 3.3.3). Using A* here guarantees that the sensor trajectory will be optimal for the computed robot trajectory up to the discretization and history used.

MHA* is a variant of A* that can use multiple arbitrarily inadmissible heuristics. We omit details for brevity and refer the reader to the paper for details [4]. We use (1) a Euclidean distance heuristic, (2) a Dubin's path length heuristic, and (3) a Dijkstra's shortest path length heuristic.

Note that here and henceforth in this chapter, when we mention A*, we are talking about *uninformed* A* (without a heuristic). We set aside formulating with a consistent heuristic for sensor coverage in our setting for future work. However, both SPLASH

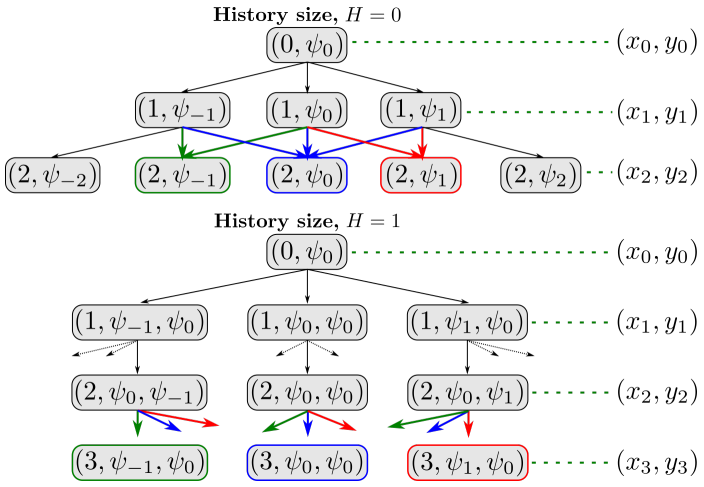


Figure 3.4: Graph representation for sensor planning for history size $H = 0$ (*above*) and $H = 1$ (*below*). Each level l in the graph corresponds to a state in the UAV trajectory π_{R}. The search space size increases with increasing history sizes. Thus, duplicates (highlighted by coloured arrows and nodes) appear less frequently with increasing history sizes making the search for an optimal π_{S} more expensive.

and SPLIT both work unchanged with the addition of a heuristic. The pseudocode for SPLASH can be found in Alg. 3.

Algorithm 3 **S**ensor **Pla**nning with **S**ensor **H**istory (SPLASH)

Input: s_{start}, s_{goal}, H
Output: π_f (final trajectory in joint space)

1: **procedure** MAIN()
2: $\pi_{robot} \leftarrow$ MHA*($s_{\text{start}}, s_{\text{goal}} \mid \mathsf{G}_{\mathsf{R}}$)
▷ using motion primitive cost as in Sec. 3.3.3
3: $\pi_{sensor} \leftarrow$ A*($s_{\text{start}}, s_{\text{goal}} \mid \mathsf{G}_{\mathsf{S}}$) with H states in sensor history
▷ using sensor coverage cost, $\text{cost}_H(\mathcal{F})$ as in Sec. 3.3.3
4: $\pi_{joint} \leftarrow$ concatenate π_{robot} and π_{sensor}
▷ creates a joint-space plan as in Sec. 3.3.2
5: **return** π_{joint}

The most important aspect of SPLASH is accounting for sensor history in Line 3. Fig. 3.4 illustrates the effect of history values $H = 0$ and $H = 1$ on the search graph G_{S} for a given initial sensor heading ψ_0. Each level in Fig. 3.4 corresponds to a waypoint along the robot trajectory π_{R}. The figure denotes state $c^{\mathsf{S}} = (t, \psi, H^{\psi})$ as a tuple where H^{ψ} is the last H elements in the tuple. For any state c^{S}, the sensor angle can either be changed by one step (increment or decrement), or it may remain the same.

The effect of H values is illustrated by the colored arrows and vertices in the graph—two arrows of the same color end up at a unique state in the graph. The key idea is as follows: For $H = 0$, ending up at ψ_0 on level 2 is considered the same state,

whether you come from ψ_0 or ψ_{-1} or ψ_1—we only care about the *current* sensor angle. But for $H = 1$, ending up at ψ_0 on level 2 is considered a different state in all these three cases because we maintain 1 historical sensor angle. This can be incorporated by defining the state as $c^{\mathsf{S}} = (t, \psi, H^{\psi})$ where $|H^{\psi}| = 1$. Observe that states are replicated in this way a lot more frequently for $H = 0$ than for $H = 1$, meaning the graph for $H = 0$ has much (in fact, exponentially) lesser states than that for $H = 1$.

3.5 Sensor Planning with Local Iterative Tunneling (SPLIT)

SPLASH takes into account sensor history and incentivizes the search to compute plans where overlaps are minimized. However, it operates with a fixed, suboptimal robot trajectory that optimizes only motion primitive cost. Recall that it is empirically observed that approximate search algorithms tend to overlook better solutions that are actually "close" to the computed solution in the space of solution paths [32]. The final solution that SPLASH gives us—say π—is most likely suboptimal with respect to the *coverage* cost in the space of joint-space solutions. **S**ensor **P**lanning with **L**ocal **I**terative **T**unneling (SPLIT) locally refines π by performing searches in small search spaces around π in the joint space using the sensor coverage cost function $\text{cost}_H(\mathcal{F})$ (Sec. 3.3.3), increasing in size with each iteration. We call these search spaces *tunnels*, and this is an application of the ITSA* algorithm [32].

We provide pseudocode for SPLIT in Alg. 4. Lines in blue indicate the differences from standard A* search. Line 2 obtains the initial solution from SPLASH. Then, LOCALITERATIVETUNNELING refines this solution locally by performing A* searches in iterative tunnels. The "level" of a state s corresponds to the distance from the initial path π_i to s computed as the smallest number of edges on a path from any state on π_i to s [32]. In the beginning, every state on the initial plan π_i is stored in memory with level 0.

The refinement process is essentially A* being performed repeatedly, with the addition of lines 25–27. The level of any newly generated state is set to one more than the level of its parent. Only a state whose level is lesser than the current iteration number is inserted into the OPEN list. This is what creates tunnels increasing in size per iteration. LOCALITERATIVETUNNELING—and consequently, SPLIT—terminates when the time available for local refinement runs out.

3.6 Experimental Results

We evaluate our approaches by running them over 10 randomly generated start-goal pairs per map for 20 maps. We pick maps as seen in the persistent coverage framework described in [52] (Sec. 3.3). The maps are generated by letting the map in the framework

Algorithm 4 **S**ensor **P**lanning with **L**ocal **I**terative **T**unneling (SPLIT)

Input: s_{start}, s_{goal}, $\mathsf{T}_{\text{overall}}$ (time limit)
Output: π_f (final trajectory in joint space)

procedure MAIN()
  $\pi_i \leftarrow \text{SPLASH}(s_{\text{start}}, s_{\text{goal}}, \mathsf{H} = 0)$ ▷ **Initialization step**
  $\mathsf{t}_{\text{SPLASH}} \leftarrow$ time taken for SPLASH to terminate
  $\pi_f \leftarrow \text{LOCALITERATIVETUNNELING}(\pi, \mathsf{T}_{\text{overall}} - \mathsf{t}_{\text{SPLASH}})$ ▷ **Refinement step**
  return π_f

procedure LOCALITERATIVETUNNELING(π_i, t)
  iteration $\leftarrow 1$
  Create and store all states $s_i \in \pi_i$ in memory with $level(s_i) = 0$
  while time t remains **do** ▷ Iterative Tunneling loop
    $s_{\text{start}} \leftarrow$ first state in π
    $s_{\text{goal}} \leftarrow$ last state in π
    $g(s_{\text{goal}}) = \infty$; $g(s_{\text{start}}) = 0$
    $bp(s_{\text{start}}) = bp(s_{\text{goal}}) =$ `NULL`
    Insert s_{start} into `OPEN` with $\text{KEY}(s_{\text{start}})$
    while `OPEN` not empty **do** ▷ Modified A* loop
      $s \leftarrow$ `OPEN`.MIN() ▷ where `OPEN` is a min-heap
      if s is goal **then**
        Backtrack from s to obtain solution π_f
        break
      for successor s' in $\text{SUCCESSORS}(s, \mathbb{T}_\mathsf{J})$ **do** ▷ successors are computed via transitions in $\mathbb{E}_\mathsf{J}$
        if s' not closed **then**
          if $g(s') > g(s) + c(s, s')$ **then**
            $g(s') = g(s) + c(s, s')$ ▷ where $c(s, s')$ is the sensor coverage cost
            $bp(s') = s$
            $level(s') = level(s) + 1$
          if $level(s') \leq$ iteration **then**
            Insert s' into `OPEN` with $\text{KEY}(s')$
    iteration $\leftarrow$ iteration + 1
    t $\leftarrow$ t $-$ time elapsed in current iteration
  return π_f

procedure KEY(s)
  return $g(s) + h(s)$ ▷ where $h(.)$ is a consistent heuristic

decay for several minutes while one UAV with a fixed sensor covers it persistently and pick snapshots of the map at different points in time, giving us maps with complex coverage zones.

Evaluation. Let a given trajectory in the joint space cover N cells that lie in coverage zones. Let the quantity $\sum_i p_i$ denote the sum of priorities of all such cells. We value two things: covering a large number of cells, and covering important cells (those with a low priority value). Thus, a large value of N and low values of $\sum_i p_i$ are desirable for a given plan. The quantity $\bar{P} = \frac{\sum_i p_i}{N}$ denotes the average of the priority values of all such cells.

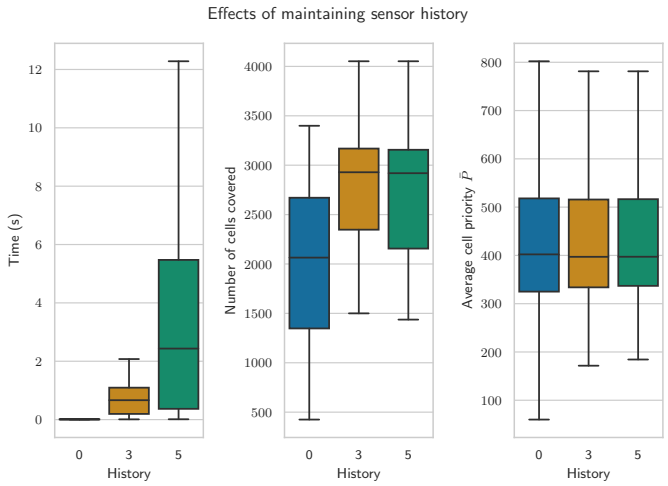


Figure 3.5: Results of running SPLASH for sensor histories of size $0, 3, 5$.

A low value of $\bar{P}$ is not necessarily an indicator of a desirable plan due to various cell priorities encountered in a complex map. For any two plans having the same value of $\bar{P}$, it depends on whether the user prefers covering more cells or important cells. As an illustration, consider plan A that covers only one cell with priority $\{10\}$, and plan B that covers 6 cells with priorities $\{5, 5, 5, 5, 20, 20\}$. The average priority of cells in both plans is the same $(= 10)$, and it is up to the user to decide which plan is considered "better".

Since SPLASH penalizes footprint overlaps, we see an increase in the number of cells covered as a larger sensor history is maintained (see Fig. 3.5). We also see that maintaining a sensor history of size 5 gives us no more value than size 3 in practical settings. Also notice that $\bar{P}$ stays fairly unchanged over several trajectories. This can be attributed to sufficiently complex maps have many different priority values for cells and no single, contiguous coverage zone—maintaining sensor history would indeed lead to covering more cells, but the average priority over these cells would be approximately the same.

Since SPLIT refines the trajectory locally to optimize coverage cost, we see an increase in information gain, or a decrease in $\sum_i p_i$, with each iteration (see Fig. 3.6). We also see decreasing path costs (g-value of the goal) with each iteration. Note that its performance largely depends on the immediate area around the initial plan which will be explored in the iterative tunnels. If this immediate area has only a few more important cells to cover, the refined plan will largely stay the same.

A natural baseline is to search directly in the joint space of robot and sensor variables. This requires a cost function that is a linear combination of the motion primitive and sensor coverage cost. Running MHA* on these start-goal pairs with such a cost function yielded an average planning time of 8.44 ± 6.40s—significantly larger than running SPLASH with

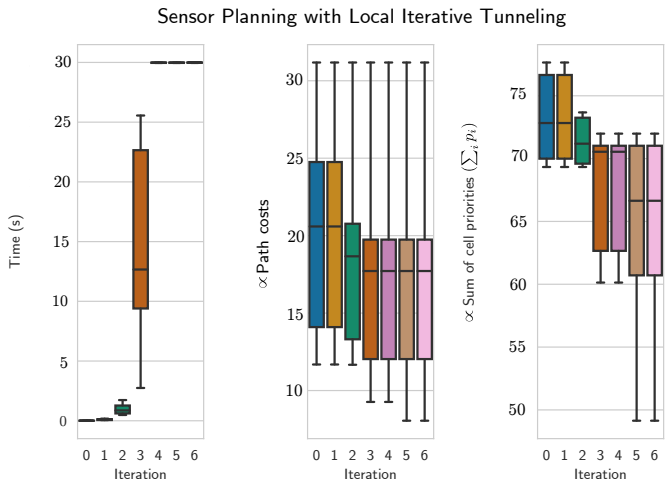


Figure 3.6: Results of running SPLIT timed out at 30s.

a sensor history of size 3 or SPLIT for 2 or 3 iterations. (We set a timeout of 20s while obtaining this value, so this is a conservative estimate and true value is in fact larger.)

Note that iteration 4 onward, SPLIT takes a long time to terminate. This is useful for real-world planning problems only if the planner is given sufficient time. For example, in the framework that we have built upon [52], it is possible to tune the duration of a committed UAV trajectory. For a longer duration, we have more time available to plan to the next goal assigned to the UAV. Moreover, these results show that the solution can still improve after the third iteration and that a local minimum is not encountered by then.

3.7 Conclusion and Future Work

We present two search-based approaches for generating robot and sensor trajectories in goal-directed 2D coverage tasks, namely **S**ensor **Pla**nning with **S**ensor **H**istory (SPLASH) and **S**ensor **P**lanning with **L**ocal **I**terative **T**unneling (SPLIT). SPLASH solves for robot and sensor trajectories independently in state spaces while maintaining a history of sensor headings. SPLIT is a two-step approach that refines this solution by searching its local neighborhood in the joint space for a better solution. We show that SPLASH is a practical alternative to to running standard search-based planning in the full joint space of robot and sensor state variables, and that SPLIT can be used to further refine the solution computed by SPLASH given enough time.

A limitation of ITSA*, and consequently of SPLIT, is that the A* searches do not reuse any search efforts between subsequent iterations, and so each iteration takes longer than the last. Reusing search efforts between iterations will lead to considerable speed-ups,

leading to faster refinement of the trajectory. Further, we do not look into maintaining sensor histories within SPLIT. It can be useful to adaptively increase the size of the sensor history maintained with increasing iterations in SPLIT. We set aside these two limitations as opportunities for future work.

4 Complete, Decomposition-Free Coverage

4.1 Introduction

Coverage Path Planning (CPP) is the problem of computing a feasible, collision-free path for a robot that passes within some sensor footprint of all reachable points of interest in an environment [21, 37]. Several real-world tasks can be cast as 2D coverage problems, including aerial surveys [52, 83], post-disaster assessment [65], and agricultural operations [59, 66]. The CPP problem is computationally expensive—it is related to the covering salesman problem, where an agent must visit a neighborhood of each city [5], which is in turn related to the NP-Complete Traveling Salesman Problem (TSP) [71]. Further, real robots like unmanned aerial vehicles (UAVs) often have various constraints on movement, such as limits on turning radius, translational and rotational speed, and sensor footprint field-of-view [6, 53].

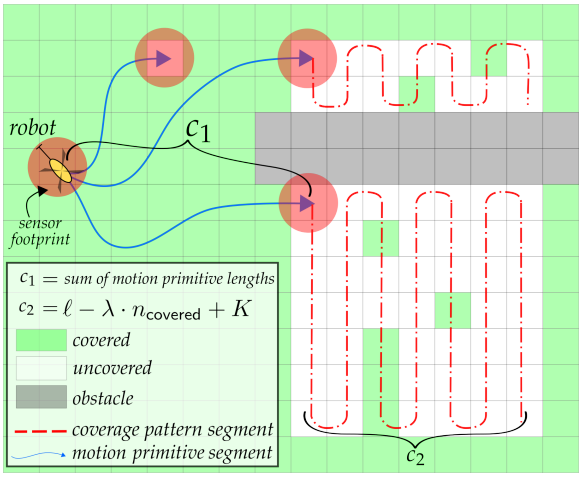


Figure 4.1: A figure illustrating three out of several possible paths our planner might choose in a toy environment, reasoning over combinations of frontier-seeking motions and space-filling patterns. Full paths include both the motion primitive segment *and* the coverage pattern segment if present.

Prior work (Sec. 4.2) shows that most popular CPP solutions are hierarchical, and typically involve three steps: (1) Decomposition: This divides the environment into mutually exclusive and exhaustive sub-regions, wherein it is simple to generate space-filling or sweeping coverage paths. These paths do not necessarily account for obstacles and are composed of simple motions—such as straight lines or circles—to provide simplicity in control. (2) Sub-region traversal: This involves determining a visitation order over these sub-regions, typically via a TSP solver. (3) Generation of space-filling paths: This involves generating sweeping coverage patterns in each sub-region, such as back and forth Bous-

trophedon[1] motions, spiral patterns, and others.

Decomposition strategies are often tailored to specific objectives and come with guarantees of satisfying them, but at the cost of significant effort spent in processing the environment. However due to the geometric nature of decomposition strategies, they can fail in among certain types of environment geometries. They can also lead to over-decomposition in non-convex environments [19, 21, 39], amended by post-processing step to merge adjacent sub-regions. Furthermore, decomposition almost always follows solving a TSP or similar [8, 19, 60, 78].

Few approaches require no environment decomposition (or even representation) which Choset categorized as "heuristic and randomized approaches" [21]. These arguably require the least development effort, but can be inefficient—they do not search for paths but randomly select local behaviors or templates, relying on probabilistic completeness [44, 69]. However they do not suggest methods to combine these templates to achieve full coverage of a target region [21].

Our key insight is that we can fold the decomposition and subregion-traversal steps into a single search routine that can be called repeatedly till coverage is complete (Sec. 4.3). To that end, our contribution is an online, **decomposition-free, resolution-complete coverage path planner**. Our planner uses a grid-based representation of the environment and provides continuous coverage paths. We fill a gap in previous work on CPP by providing completeness guarantees with little environment processing effort (see Fig. 4.2) and applicability to a variety of environments. There is of course no free lunch—to achieve this, our planner leverages a precomputed set of space-filling patterns. The planner reasons over these patterns and determines suitable parts of the environment where they can be applied. We draw inspiration from node selection strategies in frontier-based exploration [100] to enable this behavior.

In Sec. 4.4, we demonstrate our method on a variety of environments and compare quantitatively with a simpler and complete decomposition-free baseline that employs frontier-based exploration. We also qualitatively compare against [19] to show that our approach yields similar, intuitive paths without the decomposition and sub-region traversal steps. However our method does not similarly guarantee optimality with respect to coverage objectives such as turn- or distance-minimization. We summarize our work and discuss future research problems in Sec. 4.5.

[1] Boustrophedon means back-and-forth, like the motion of an ox ploughing a field.

4.2 Related Work

We categorize prior work into decomposition-based and decomposition-free methods and briefly touch upon frontier-based exploration planning. We point the reader to [21, 37] for extensive surveys on coverage path planning.

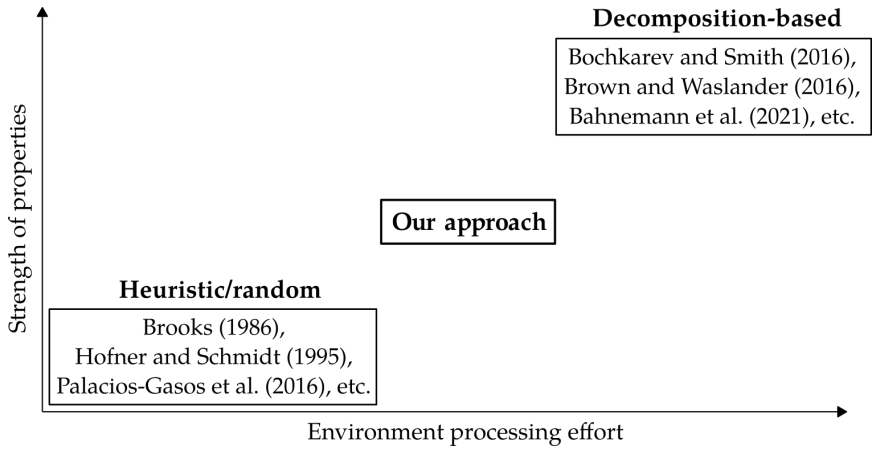


Figure 4.2: Our approach contextualized within representative prior work roughly clustered by amount of explicit environmental processing effort involved and strength of properties provided.

Decomposition-based approaches. These can be categorized into exact (continuous) and inexact (grid-based) cellular decomposition (CD). Exact CD involves partitioning the free space of the environment into simple, non-overlapping regions called cells. Early exact CD approaches include Trapezoidal, Boustropheon, and Morse-based decomposition. More recent approaches exist that optimize for specific objectives. These include minimizing turns in coverage paths [14], minimizing coverage path length [60], and creating intuitive, human-like sub-regions [19]. While typically separate from CPP literature, there also exist closely related "room segmentation" strategies that operate specifically on indoor floor plans, of which Bormann et al. provide a thorough survey [15]. Among exact decomposition strategies, the simple trapezoidal decomposition can fail in non-polygonal environments [22, 54], and the more general Morse-based decomposition fails in rectilinear environments [1].

Inexact or grid-based approaches discretize the environment into uniform grid cells, typically as large as the robot's coverage footprint, and employ various space-filling methods with backtracking strategies such as spanning trees and spiral patterns [34, 35]. However these approaches do not take advantage of smooth sweeping motions such as the Boustrophedon pattern, except for a recent paper utilizes it with turn-minimizing cellular decomposition [78].

Decomposition-free approaches. Early approaches like these are heuristic algorithms where the robot is equipped with a set of behaviors such as wall-following, obstacle-avoidance, etc. [18, 36, 38, 44]. There are also commercial floor-cleaning robots based on this approach (the RC3000 by Karcher, Trilobite by Electrolux, and early versions of the Roomba by iRobot) [37, 70]. These approaches do not guarantee completeness but have advantages from a cost/benefit standpoint—they are easy to implement and do not require costly localization sensors [11]. Our work can be thought of as a method to stitch together such local behaviors to enable complete coverage.

Frontier-based exploration planning In frontier-based exploration, a robot repeatedly moves toward the closest point on the frontier—which is the set of cells at the border between explored and unexplored parts of the environment (covered and uncovered in our case) [20, 29, 52, 95, 99, 100, 102]. We employ a frontier-based search to determine where space-filling coverage patterns should be applied (details in Sec. 4.3).

4.3 Method

We assume we are given an occupancy grid map of an uncovered 2D environment, the start configuration of the robot, and parameters relating to constraints on the robot's motion and sensor footprint. We also assume access to a library of precomputed space-filling coverage patterns. We require the robot to observe all reachable points of interest in the environment. Typical desirable properties of coverage paths include simple motions and minimal overlaps and number of turns. The precomputed coverage patterns can individually be optimal with respect to such objectives, but our approach does not provide any guarantees on optimality of the overall path.

4.3.1 Framework

Control flow. Our framework relies on repeated queries to a search-based planner which returns a feasible, collision-free coverage path for the robot. In Alg. 5, we present pseudocode for our overall framework We begin coverage by calling PATHPLANNER(s_{start}, $\mathcal{M}$) to obtain and execute a path π and then repeatedly do this until the planner does not return a path (Lines 2–8) At this point (line 4), coverage is complete because the planner is guaranteed to return a path as long as reachable uncovered cells exist in the environment.

Search-based planning on lattice graphs. As is common in search-based planners operating over grids, we discretize the environment into grid cells with a uniform resolution. The planner is resolution-complete with respect to this underlying grid. Each cell in the grid falls into one of three categories: free & uncovered, free & covered, and blocked. The planner searches for a path over a finite, discrete lattice graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

Algorithm 5 Framework overview—MAIN procedure

Inputs: s_{start} (robot start configuration), $\mathcal{M}$ (grid map)

```
1: procedure MAIN(s_start, M)
2:     repeat
3:         π ← PATHPLANNER(s_start, M)                ▷ See Alg. 6.
4:         if π is NULL then
5:             done                                   ▷ Coverage complete; terminate
6:         Execute π and update M
7:         s_start ← last state in π                  ▷ Assuming perfect execution
8:     until done
```

where vertices/nodes $\mathcal{V}$ represent robot configurations connected by edges $\mathcal{E}$ representing continuous motions feasible by the robot [57, 76]. These paths are always start and end on a lattice state, and can either be motion primitives or space-filling coverage paths (details follow).

We require a set of motion primitives $\mathcal{A}$, which are simply short, continuous, feasible motions, that connect pairs of lattice states. This is standard practice [57, 76] and allows our approach to operate on a variety of robots given their kinematic and dynamic constraints.

We also require access to a set of precomputed space-filling coverage patterns $\mathcal{P}$ which completely cover areas of various simple shapes, each of which is a continuous path centered at the origin. In this chapter, we run experiments only using Boustrophedon patterns that cover rectangular areas of various sizes, but any set of patterns that are feasible for the robot can be used in our framework.

4.3.2 Details—Procedure PATHPLANNER

A frontier cell $f \in \mathcal{F}$ is an uncovered cell in $\mathcal{M}$ that has at least one neighboring cell that is covered. Standard frontier-based exploration [100] is an iterative approach, where in each iteration the robot moves to the closest frontier cell, covers it, and repeats this until coverage is complete. Our approach is similar, except that in each iteration, the planner reasons about all combinations of frontier-seeking motions and the several coverage patterns available to it at each frontier cell.

Multi-Goal search. At any arbitrary point in time during coverage, let s_{start} be the node corresponding to the robot's current location. We also define a *frontier node* as a node that corresponds to a robot configuration whose position (x_R, y_R) is a frontier cell in $\mathcal{M}$. The graph representing the environment contains a set of frontier nodes $f_i \in \mathcal{F}$, $i = 1, \ldots, |\mathcal{F}|$[2]. To make progress in coverage, the robot must visit (explore) a frontier node and possibly execute a coverage pattern. Since there are multiple frontier nodes, this can be viewed as a multi-goal motion planning problem where each frontier node is a potential goal.

[2]For brevity, we represent the sets of both frontier cells and frontier nodes as $\mathcal{F}$.

Algorithm 6 Method details—Multi-goal Dijkstra's search

```
1: procedure PATHPLANNER(s_start, M)
2:     g(s_start) ← 0
3:     OPEN ← OPEN ∪ {s_start, g(s_start)}
4:     while OPEN not empty do
5:         x ← OPEN.MIN()
6:         SUCCS ← ∅ // List of successor nodes
7:         COSTS ← ∅ // List of corresponding edge costs
8:         if x == G_imaginary then
9:             π ← Backtrack from G_imaginary to s_start
10:            return π // Path found
11:        else
12:            EXPAND (x, M, SUCCS, COSTS)
13:        for successor s' ∈ SUCCS and cost c ∈ COSTS do
14:            if g(s') > g(s) + c then
15:                g(s') ← g(s) + c
16:                bp(s') ← s
17:                OPEN ← OPEN ∪ {s', g(s)}
18:    return NULL // No path found

19: procedure EXPAND(x, M, SUCCS, COSTS)
20:    if x is a frontier node then
21:        // Apply coverage patterns at frontier cell
22:        for collision-free pattern p in P do
23:            APPLYPATTERN (x, p, M, SUCCS, COSTS)
24:        // Only cover frontier cell (no pattern applied)
25:        SUCCS ← SUCCS ∪ G_imaginary
26:        COSTS ← COSTS ∪ K − λ · 1
27:    else
28:        // Apply motion primitives at standard cell
29:        for collision-free motion primitive a in A do
30:            APPLYMPRIM (x, a, M, SUCCS, COSTS)

31: procedure APPLYPATTERN(x, p, M, SUCCS, COSTS)
32:    SUCCS ← SUCCS ∪ G_imaginary
33:    COSTS ← COSTS ∪ {ℓ^p − λ * n_c^p + K}

34: procedure APPLYMPRIM(x, a, M, SUCCS, COSTS)
35:    SUCCS ← SUCCS ∪ {node representing state after a applied at x}
36:    COSTS ← COSTS ∪ {a.ℓ}
```

We introduce an *imaginary goal node* $G_{\text{imaginary}} \in \mathcal{V}$ in $\mathcal{G}$ —this node does not correspond to any real location, but is only present to better formulate our search. Multi-goal Dijkstra's search is equivalent to standard Dijkstra's search [25] from s_{start} to $G_{\text{imaginary}}$. As in standard graph search terminology [77], (1) the g-value of a node is its cost-to-come from node s_{start}, and (2) OPEN is a priority queue of nodes (in our case, implemented as a min-heap), where priorities associated with each node are their g-values in the case of Dijkstra's search.

In Alg. 6, we initialize the search by inserting s_{start} into OPEN with a g-value of 0 (Lines

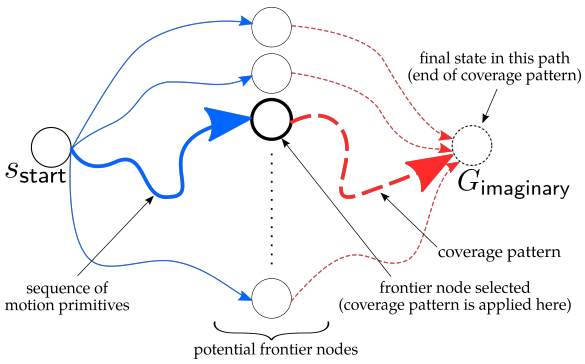


Figure 4.3: Multi-goal Dijkstra search tree. For illustration, we show only a single coverage pattern at each frontier node, but there are often multiple in reality.

2–3). We then proceed as in standard Dijkstra's search, expanding nodes from `OPEN` in a best-first fashion. The steps involved in expanding a node are key to our approach. Suppose at a point during the search, node x is obtained (popped) from `OPEN`. We perform one of the following three steps:

- If x is not a frontier node, we apply motion primitives $\mathcal{A}$ applicable at x to generate successors `SUCCS` and costs `COSTS` (Lines 17–20).
- If x is a frontier node, we apply coverage patterns $\mathcal{P}$ (Lines 10–13). The successor at the end of each applied pattern is $G_{\text{imaginary}}$ and the costs incorporate distance traversed by the pattern as well as the number of cells covered. (We elaborate on this below.) We also add a successor corresponding to not executing any pattern at the frontier node (Lines 14–16). This provides the option of simply executing a frontier-seeking motion.
- The third and last option for x is that it is node $G_{\text{imaginary}}$, in which case we terminate the search, backtrack from $G_{\text{imaginary}}$ to s_{start}, and return the resulting path (Lines 7–9).

Edges outgoing from a frontier node incorporate the cost of the robot traveling and covering cells using a particular pattern applied at that frontier node in their each of its successor nodes. Edges incoming to a frontier node incorporate the cost of the robot traveling to that frontier node. In this way, for the given set of patterns and motion primitives, the search computes a coverage path for the robot to execute in a query. A visual of this is presented in Fig. 4.3 for clearer understanding. The rest of the algorithm (Lines 21–25) is identical to Dijkstra's search, where g-values of successor nodes are updated or inserted into `OPEN` whenever better paths to them are found.

4.3.3 Details—Procedure ApplyPattern

We now detail the cost function used when applying a coverage pattern. Say executing a pattern p results in traveling a distance ℓ^p and covering n_c^p number of cells in $\mathcal{M}$ that

were uncovered before executing the pattern. The cost function we use for executing this pattern is:

$$\mathsf{cost}(x, p, \mathcal{M}) = \ell^p - \lambda \cdot n_c^p + K \tag{4.1}$$

Here, $\lambda > 1$ is a user-defined parameter that controls how much to weigh covering n_c^p cells at the expense of traveling a distance ℓ^p. The larger λ is, the more likely the planner is to apply coverage patterns, but this might increase non-working distance as it might result in traveling more often over cells that are already covered. The constant K ensures that this cost is always non-negative. Note that this does not distort the solution by incentivizing shorter paths, as this constant is not added to every edge in the graph. As mentioned before, the successor node of applying a pattern is always $G_{\mathsf{imaginary}}$ (Line 37).

4.3.4 Details—Procedure ApplyMPrim

Finally, in ApplyMPrim we use the length of the primitive as the cost, to be consistent with ℓ^p above. For UAV-like robots, motion primitives practical for planning are simple, short motions starting and ending at lattice states. From a coverage perspective, frontier-seeking motions will contain such short primitives to approach a frontier cell, but with sweep-coverage patterns we can more easily have long, smooth paths.

4.4 Evaluation

4.4.1 Experiment details

Sweep-coverage patterns. We precompute and store boustrophedon coverage patterns that each cover a rectangular area of some width and height. Each pattern consists of long, straight-line segments and circular arc segments connecting them. We discretize our map into cells of dimension D and we choose $D = 2R$ where R is the radius of turns in the boustrophedon patterns we use in experiments. This is done so that each pattern exhaustively covers said rectangular areas without any overlapping paths. Different coverage patterns for different settings of R, D can be precomputed and just as easily be used in our approach, provided we have access to ℓ^p and n_c^p (Sec. 4.3.3).

Environments. We evaluate our approach in simulation across multiple environments, namely, rooms and walls with random gaps (E1), corridors and rooms (E2), and city-like maps from the MovingAI benchmark[3] (E3). Several prior works focus on indoor scenarios due to heavy presence of non-convex and cluttered shapes and use special methods to decompose these environments [15, 19, 20, 78]. While we show the completeness of our decomposition-free approach on such environments, our approach is applicable to any

[3] https://movingai.com/benchmarks/grids.html

environment that can be represented as a uniformly discretized grid map. Dimensions of the environments we consider are 100 times the sensor footprint diameter D (small), and 250 times D (large). From previous work on aerial coverage [52], we know that a typical sensor footprint diameter for a helicopter-like UAV is $\sim$ 30m. Thus our largest environments may represent areas up to $\sim$ 7500m $\times$ 7500m. Fig. 4.4 shows representative examples from these environment categories.

For experiments on environments of $100D \times 100D$ units, we use use coverage patterns covering rectangles of sizes up to $30D \times 30D$ units in increments of D units—a dense set of space-filling patterns. We include separate patterns beginning execution at each of the four corners of the corresponding rectangular area, which makes over 3000 patterns. For environments of $250D \times 250D$ units, we uniformly sample up to 25 patterns of sizes up to $160D \times 160D$ units to save on computational effort.

Finally, as seen in Fig. 4.6, we also use two handcrafted environments to compare with decomposition-based baselines[4].

Robot. While our approach is general and works with any type of robot, for experiments we model a planar UAV with a fixed, circular sensor footprint, the configuration of which consists of its position and yaw: $(x, y, \gamma) \in SE(2)$ [52]. We use simple unicycle dynamics [55] to generate motion primitives: $\dot{x} = v\cos\theta$, $\dot{y} = v\sin\theta$, $\dot{\theta} = \omega$. Further, we have upper bounds on the robot's translational speed $v \leq v_{\max} \coloneqq 8\text{m/s}$ and rotational speed $\omega \leq \omega_{\max} \coloneqq 0.14/\text{s}$ (these values are borrowed from previous work [52]). Lastly, the robot's sensor has a circular footprint on the ground of radius $\frac{D}{2}$ centered at the robot's position, meaning the robot's footprint covers one cell at a time. We measure performance in terms of the distance traveled while covering the environment, total planning times, and total path execution times.

4.4.2 Decomposition-free comparison.

Few, early decomposition-free approaches exist, which are are heuristic or random. Running random coverage planning as in [69] is highly inefficient in the environments we consider, and so we do not include it in our experiments. For example, for rooms in E1 with very narrow gaps, the robot takes a very long time to move to another room when using random actions. We implement a simple, *complete* decomposition-free baseline: a frontier-based approach (akin to vanilla frontier-based exploration), which we denote this by "frontier-based coverage" (FBC). This is equivalent to our approach *without* any pre-ocmputed space-filling patterns (i.e., Alg. 6 without lines 21–23). In this approach, the robot repeatedly covers the frontier cell closest to it until coverage is complete.

[4]The maps and open-source code for these baselines were not available at the time of working on the paper corresponding to this chapter. So we manually replicated two environments for a qualitative comparison.

We present results for two measures of efficiency: (1) The amount of coverage versus distance traveled by the robot, and (2) total planning times and execution times. Experiments were run on an Intel Core i7-6700 CPU @ 3.40GHz 8 and approaches implemented in C++ and compiled using g++ 9.4 with the -O3 optimization flag. For maps of size $100D \times 100D$, for both environment types E1 and E2, and we run both our framework and the baseline until complete coverage is achieved for 20 trials. For maps of size $250D \times 250D$ of environment types E1 and E2 and that of size $256D \times 256D$ of environment type E3, we terminate our framework when 90% coverage is achieved by it. At this point of time, FBC has covered a significantly smaller area of the environment, and we terminate it in the interest saving computational effort. We would require FBC to run to completion to compute *total* planning and execution times, and so we do not include them for this set of environments. We run 5 trials for maps of size $250D \times 250D$ of environment types E1 and E2. Note that we use axis-aligned boustrophedon (back-and-forth) patterns as sweep coverage patterns available to the planner. These patterns are highly suitable for environment types E1 and E2, but not particularly for E3, as the obstacles are less structured. We run experiments on maps of type E3 (4 trials) to illustrate that even if coverage patterns are not specifically tailored to the environment, our framework still shows significant benefits over FBC.

Our framework (plots in red) significantly outperforms FBC (plots in blue) in all cases with respect to the distance traveled to complete coverage, as seen in subfigures A, B, and C in Fig. 4.5. We also demonstrate much smaller planning and execution times as the planner is queried significantly fewer times in our approach (Tables **I** and **II**). We post-process plans to compute total execution times assuming constant velocities for the motion primitives and coverage patterns ranging between 2m/s and 8m/s (same across both approaches).

In maps of size $250D \times 250D$ for environment types E1 and E2 we see steep increases in coverage (vertical red increases in the plots) whenever a coverage pattern is executed. The more gentle increases in coverage either correspond to coverage patterns overlapping with previous paths or several consecutive frontier-seeking motions. This occurs because, as mentioned earlier, we uniformly sample coverage patterns to keep planning times reasonable, and as a result, there does not exist a coverage pattern that exhaustively covers every possible rectangular area the planner might encounter.

4.4.3 Decomposition-based comparison.

In Fig. 4.6 we present a qualitative comparison of coverage paths computed by our approach with representative examples from Brown and Waslander's Constriction Decomposition method [19] and the linear-programming based turn-minimizing approach by Ramesh et al. [78]. Similar to us, Brown and Waslander do not claim any overall optimal-

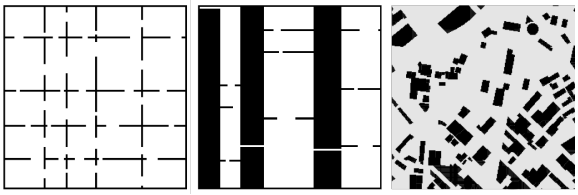


Figure 4.4: Representative examples of environments used in experiments. Starting from top-left moving clockwise: E1: walls + gaps, E2: corridors + rooms, and E3: city maps (Boston_1_256 map shown here) from the MovingAI benchmark.

TABLE I: Planning and execution time results $100D \times 100D$

Metrics	E1		E2	
	Ours	FBC	Ours	FBC
Total planning times (s)	$\mathbf{222 \pm 37}$	2931 ± 449	$\mathbf{57 \pm 12}$	551 ± 132
Total execution times (s)	$\mathbf{12530 \pm 1144}$	29825 ± 658	$\mathbf{4842 \pm 424}$	17946 ± 943

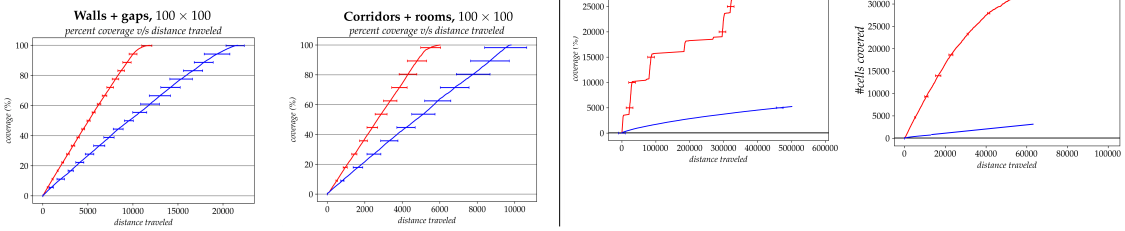


Figure 4.5: All results including plots of coverage versus distance traveled for all environments (**A**, **B**, **C**) and total planning and execution times for environments of size $100D \times 100D$ (Tables I and II), as well as plots for larger environments of size $250D \times 250D$.

ity guarantees. Their approach is based on geometrically computing "constriction points" in the environment, which correspond to areas like openings near doors or corridors in indoor scenarios. We observe that in our approach as well, constriction points naturally emerge as frontier nodes near such areas. Unlike the approach by Ramesh et al., we do not claim any overall optimality guarantees, which is where the environment partitioning effort of that baseline pays off. However our approach is complete without needing any environment partitioning/decomposition and yields intuitive coverage paths.

4.5 Conclusions

We present a decomposition-free approach to complete coverage path planning. The search reasons over paths consisting of motion primitives and precomputed space-filling coverage patterns and automatically determines where in the environment a space-filling pattern should be applied. We evaluate our approach in a variety of environments and compare it qualitatively and quantatively with appropriate baselines. In large environments, we sample patterns to prevent slow performance and show that it is still effective and does not require decomposition. For the future, in order to improve scalability, we

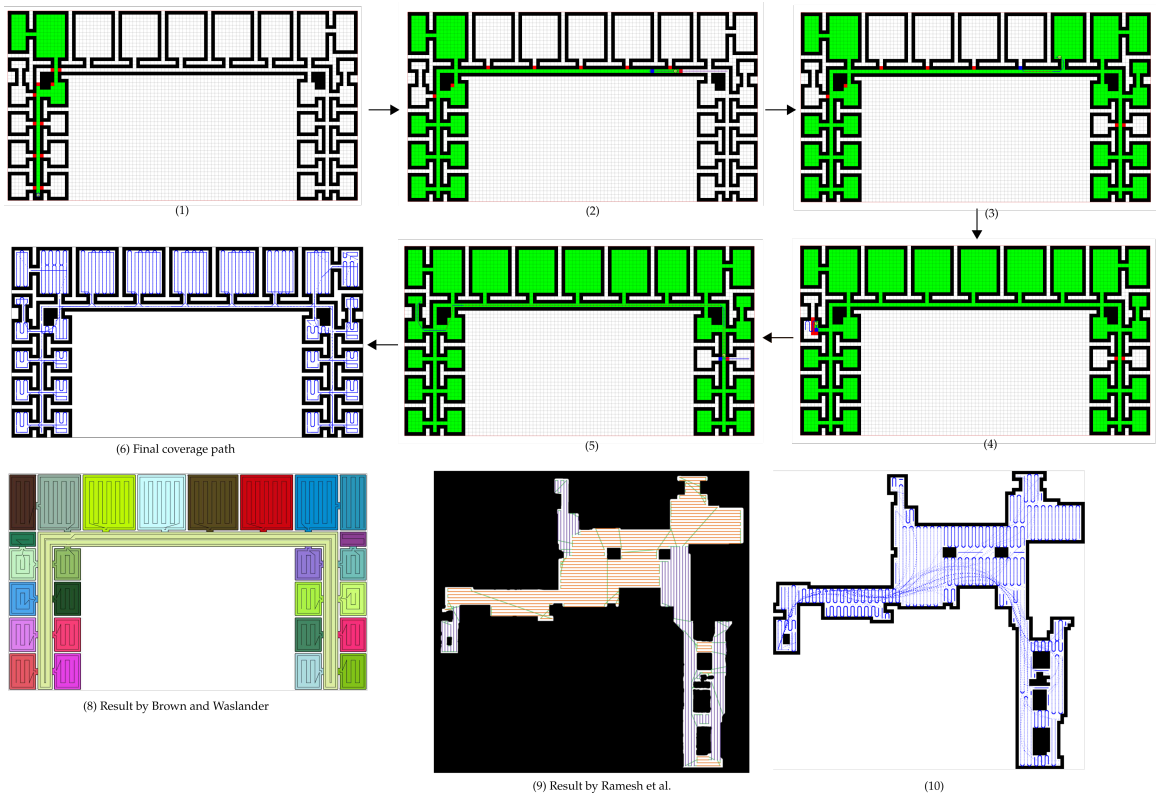


Figure 4.6: Comparison with Brown and Waslander's Constriction Decomposition method [19] on and the turn-minimizing approach by Ramesh et al. [78] on representative environments. Subfigures 1–6 show our method on an indoor lab environment and subfigure (7) shows Brown and Waslander's result on a similar environment. Subfigure 9 shows our method on another indoor environment and subfigure (8) shows the turn-minimal solution by Ramesh at al. on a similar environment. (We handcrafted these environments in an occupancy grid format and so they are not fully identical to those used in the baselines.)

wish to look into dynamically generating these space-filling patterns online by geometric or data-driven controllers.

5 Speeding Up Decomposition-Free Coverage by Learning Behavior Footprints

5.1 Introduction and Related Work

We continue to look at Coverage Path Planning (CPP), which, as a reminder, is the problem of computing a feasible, collision-free path for a robot that passes within some sensor footprint of all reachable points of interest in an environment [21, 37]. Existing CPP solutions can be broadly categorized into decomposition-based and decomposition-free methods [37]. Decomposition-based methods are by far the most popular and typically involve three steps: (1) decomposing the environment into sub-regions (2) determining a visitation order over these sub-regions via graph search or a Traveling Salesman Problem (TSP) solver, and (3) generation of space-filling paths in each sub-region. This requires significant pre-processing of the environment and the availability of suitable TSP solvers [8, 19, 60, 78]. Furthermore, step (1) can sometimes fail in non-convex environments or lead to "over-decomposition" in cluttered environments [19, 21, 39]. In this chapter, we extend the complete, decomposition-free coverage path planner, hereafter called CDF [51] described in the previous chapter. CDF effectively folds steps (1) and (2) above into a single online search routine. CDF is resolution-complete, requires no pre-processing of the environment except discretizing it into a uniform grid, and is easier to integrate into existing robot navigation architectures.

However CDF suffers from a significant limitation, which is what we address here. Many real-world planning problems involve expensive-to-evaluate actions. In coverage, this can occur when determining the utility of a behavior with a sensor model such as ray casting to forward-simulate its effects on the environment. CDF's search routine is exhaustive, and as it stands, requires evaluating *all* coverage behaviors that it can access at multiple points in the search—this can get prohibitively expensive.

Our key idea is to reduce the computational burden on the search by predicting the utility of coverage behaviors online. Through a simple modification, we present our extension CDF++, that utilizes these predictions in the search. To that end, our approach has two phases: The first is a learning phase wherein we train a convolutional autoencoder model to predict the footprint of a behavior given a behavior map and a local occupancy grid. Next, we present CDF++ wherein we augment CDF's search routine this learnt model to replace online behavior evaluation. We demonstrate in our experiments how CDF++ yields comparable coverage performance to CDF and simpler extensions of CDF with much faster planning times.

There have been several methods that learn map predictions in autonomous exploration in partially known or unknown envirionments. Our work differs from existing mainly because (1) we learn coverage map predictions that depend on a specific cover-

age behavior, as opposed to predicting obstacle maps beyond exploration frontiers in unknown environments, and (2) we feed our learnt model to a search-based planner more sophisticated than vanilla frontier-based exploration. Shrestha et al. learn occupancy grid map predictions for frontier-based exploration in unknown environments. They train a variational autoencoder to predict unknown regions of the environment with the aim of identifying navigation goals to maximize map coverage [84]. Oswald et al. speed up frontier-based exploration by exploiting background information in environments where a rough structure, such as a user-defined topological map, is known beforehand [68]. They employ a global TSP solver over this structure, in combination with local frontier-based exploration. Our method is related but separate, in that CDF or CDF++ can be used on top of this method: given access to precomputed converage behaviors, CDF or CDF++ can be adapted to replace their local frontier-based exploration planner, while simultaneously taking advantge of their global TSP solver that exploits background information.

While these works are closely related to ours, there are also other methods to estimate information gain, which is the ultimate aim of these map prediction approaches. Pimental et al. estimate expected information gain by assuming that wall segments at frontiers extend into unknown space [74]. Cost-utility exploration methods count unknown cells within a local region around frontier cells to estimate information gain [17, 96]. Finally, a set of methods employed in exploration that are separate from frontier-based exploration methods in literature include information-theoretic methods. These methods predict information gain using on Bayesian inference with Gaussian processes or Gaussian Mixture Models in the context of one-stop lookahead measurements [9, 46, 93].

5.2 Method

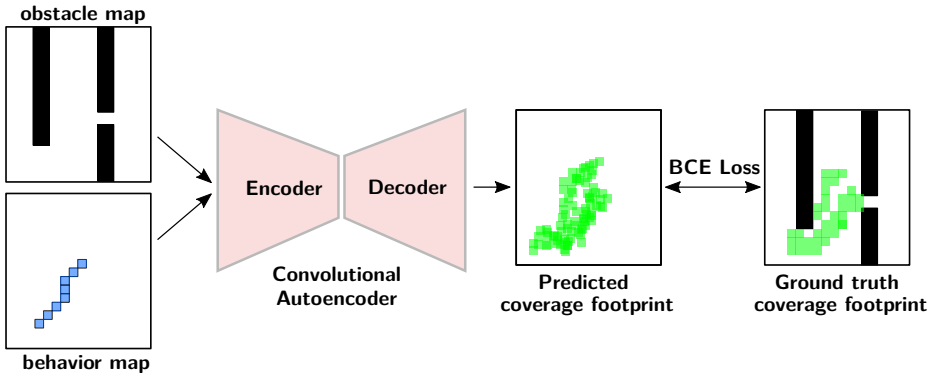


Figure 5.1: Method used for learning coverage behavior footprints in the learning phase.

5.2.1 Learning phase

Our goal in the learning phase is to learn a behavior footprint for each behavior given the behavior itself and a local occupancy grid map or obstacle map. We then give the planner access to this footprint model in order to reduce the computational burden of evaluating expensive behaviors online.

Consider a distribution of environments E (for example, indoor floor plans). We first generate training examples of coverage behaviors executed at random locations in maps drawn from E. We maintain windows of the obstacle map around these random locations of fixed dimensions $H \times W$. We assume that behaviors are executed at the center of this window. Further, we represent each behavior by a binary map of the same dimensions where the set of discrete cells it traverses are marked with 1, and the rest 0. Fig. 5.1 shows these two maps on the left. These are then fed as a two channels of an image into a convolutional autoencoder, which predicts a single-channel binary coverage map where cells expected to be covered are marked with 1s. We train this model using a Binary Cross-Entropy loss with respect to the ground truth coverage map that we have from the generated data. The ground truth is obtained during data generation by fully forward-simulating the behavior. In this way, we learn a model to predict online a set of 2D locations in the map that we expect to be covered given a behavior, a $2D$ location, and an $H \times W$ window of the obstacle map centered at this location.

5.2.2 Planning phase

CDF is a complete, decomposition-free coverage planner. It leverages precomputed space-filling coverage behaviors, such as long straight-line motions or back-and-forth lawnmower behaviors, and automatically determines where to execute them online. It achieves this formulating the problem as a multi-goal Dijkstra's search which is invoked repeatedly until coverage is complete. We now briefly describe CDF's search routine and detail how we modify it. The search is essentially a Dijkstra's search extended to multiple goals, where each goal is a frontier cell in the map. Alg. 7 depicts the search routine, which is repeatedly called until no frontier cells remain in the environment (meaning coverage is complete). Line 11 in blue indicates the modified step.

The g-value of a node is its cost-to-come from node s_{start}, which represents the starting location of the robot. `OPEN` is a priority queue of nodes, where priorities associated with each node are their g-values in the case of Dijkstra's search. The search is initialized in lines 2–3. Then nodes are expanded according to their priority values (we detail the expansion step next). After the successor nodes and their corresponding edge costs are generated, the nodes' g-values and back-pointers are updated (lines 13–15). Finally successor nodes

are inserted into OPEN (line 16), continuing the search.

CDF++::EXPAND. When a node is popped from OPEN (line 5), CDF expands it (Alg. 8) by either (1) applying motion primitives if it is not a frontier cell (lines 10–11), or (2) applying coverage behaviors at a frontier cell. It is in this latter step that we introduce the learnt coverage prediction model for each behavior that is applied at node x that is being expanded.

We provide the model with a local obstacle map, $\mathcal{M}_{\text{local}}$, which is a window of $\mathcal{M}_o$ around the cell represented by node x. We also provide the model with the discrete behavior map of the behavior in question, $\mathcal{M}_{\text{b}}$. The model predicts a footprint of cells $\mathcal{F}_{\text{pred}}$ that the behavior will cover (line 4). This means that the search does not need to evaluate the full behavior online. Next, in line 5, we compute the cells in $\mathcal{F}_{\text{pred}}$ that are currently uncovered to yield the number of newly covered cells, C. This is linearly combined with the length of the behavior, ℓ, in line 6 (exactly as in CDF), to give us the final edge cost of the behavior. The constant K is added to ensure that edge costs are always non-negative.

Algorithm 7 Multi-goal Dijkstra's Search

```
1:  procedure CDF::PATHPLANNER(s_start, M)
2:      g(s_start) ← 0
3:      OPEN ← {s_start, g(s_start)}
4:      while OPEN not empty do
5:          x ← OPEN.MIN()
6:          SUCCS ← ∅, COSTS ← ∅
7:          if x == G_imaginary then
8:              π ← Backtrack from G_imaginary to s_start
9:              return π // Path found
10:         else
11:             CDF++::EXPAND (x, M, SUCCS, COSTS)
12:         for successor s' ∈ SUCCS and cost c ∈ COSTS do
13:             if g(s') > g(s) + c then
14:                 g(s') ← g(s) + c
15:                 bp(s') ← s
16:                 OPEN ← OPEN ∪ {s', g(s)}
17:     return NULL // No path found
```

5.3 Preliminary Results

We run experiments on the first distribution of environments shown in Fig. 4.4, "walls + gaps (E1)", with 32 precomputed coverage behaviors. These behaviors span long, straight-line behaviors traversing up to 80 units, and back-and-forth lawnmower-style behaviors covering an areas up to 80×80 units. The sensor model used by these behaviors is a ray-casting model covering an angular range of $[\frac{-pi}{4}, \frac{\pi}{4}]$ with respect to the heading of the robot, and a distance of up to 12 units. We synthesize 12 environments, and train our

Algorithm 8 Augmenting CDF with learnt behavior footprints

procedure CDF++::EXPAND $(x, \mathcal{M}, \texttt{SUCCS}, \texttt{COSTS})$
  if x is a frontier node **then**
    for collision-free behavior b in $\mathcal{B}$ **do**
      $\mathcal{F}_{\text{pred}} \leftarrow \text{model}(\mathcal{M}_{\text{local}}, \mathcal{M}_{\text{b}})$
      $C \leftarrow \text{covered}(\mathcal{M}_{\text{local}}, \mathcal{F}_{\text{pred}})$
      $\texttt{COSTS} \leftarrow \texttt{COSTS} \cup \{\ell - \lambda * C + K\}$
      $\texttt{SUCCS} \leftarrow \texttt{SUCCS} \cup \{G_{\text{imaginary}}\}$
  else
    for collision-free motion primitive a in $\mathcal{A}$ **do**
      $\texttt{COSTS} \leftarrow \texttt{COSTS} \cup \{\text{length of } a\}$
      $\texttt{SUCCS} \leftarrow \texttt{SUCCS} \cup \{\text{node after } a \text{ applied at } x\}$

model on 7 of them. In each of these 7 environments, we randomly select 50 locations and execute all behaviors to generate a total of 11,200 datapoints with a local 200×200 window of the obstacle map, a behavior, and a ground truth output coverage map. We test on the remaining environments and randomize the start location of the robot 3 times for each environment. This gives us 15 trials of performing coverage on entire maps.

We present preliminary results in Table 5.1 consisting the total planning time summed across all calls to the planner, and total time spent in forward-simulating behaviors during planning, averaged across all trials. For CDF, forward-simulating the behaviors amounts to performing ray-casting at each waypoint along the behavior. For CDF++, this amounts to preparing inputs to the autoencoder, one forward pass through the model to obtain a predicted coverage map, and counting how many new cells will be covered given the current status of coverage in the environment.

Method	Planning time	Behavior evaluation time
CDF++	$\mathbf{122 \pm 82}$	$\mathbf{26 \pm 18}$
CDF	176 ± 118	154 ± 112

Table 5.1: Planning times in seconds: average planning time and average time spend in evaluating coverage behaviors across complete coverage in the indoor maps environment distribution.

5.4 Limitations and Future Work

We observe that while the above results show promise, they currently rely on the baseline (CDF) forward-simulating each behavior by performing ray-casting at *each* waypoint in the behavior. However we observe that if CDF forward-simulates each behavior by performing ray-casting at every K^{th} waypoint in the behavior, CDF yields faster planning times compared to CDF++ while still computing the same coverage path for values of

$K \geq 10$. In other words, using a slightly lower-fidelity ray-casting model in CDF is sufficient to defeat the purpose of learning behavior footprints in CDF++ for *this* set of behaviors and environments on which we performed perliminary experiments.

In the future, we plan to:

1. Run more extensive experiments to identify the set of experiment settings where CDF++ does not outperform CDF
2. Do away with learning coverage footprints for each behavior and instead learn to identify the most promising m behaviors online, for small values of m. This will further increase the speed-up that CDF++ provides over CDF.

6 CONCLUSIONS

In this thesis, we presented search-based algorithms to realize various forms of sensor-based robotic coverage. In Chapter 2, we looked at a system of multiple unmanned aerial vehicles (UAVs) with fixed, downward-facing cameras performing persistent coverage over a large environment. We presented a framework to combine solutions to the sub-problems of goal assignment for each UAV, multi-agent planning, and maintaining the invariant of spatiotemporal deconfliction for every committed plan (i.e., plans that UAVs are committed to execute). We showed the benefits of our planning framework in simulation and in the real world in maintaining persistent coverage with sufficient resources, and exhibiting graceful degradation in persistent coverage when resources fall short (for example, when there are too few UAVs for an environment of a given size).

We then zoomed into the single-UAV planning problem in Chapter 3 and modified the problem to incorporate an actively controllable camera that looks forward. This added to the computational complexity of the problem via an additional degree of freedom (the pan angle of the camera). Moreover, the computing optimal solution through a search in this case also requires maintaining the sensor state history in the planning state, making this even more computationally expensive. We presented two search-based methods to simultaneously compute these robot and sensor trajectories. The first method, **S**ensor **Pla**nning with **S**ensor **H**istory (SPLASH), decoupled the robot and sensor planning problems, making it feasible to maintain a partial sensor state history during sensor planning. The second method, **S**ensor **P**lanning with **L**ocal **I**terative **T**unneling (SPLIT), iteratively refined the solution computed by SPLASH by computing better solutions in the joint robot and sensor state space (but without maintaining a sensor state history). We showed the benefits of both these methods in simulation.

Next, we switched gears from persistent coverage to the problem of standard coverage path planning in Chapter 4. The literature in coverage path planning is largely split between (1) methods with strong properties that require extensive pre-processing and decomposition of the environment, and (2) random or heuristic strategies that are much more simple but provide weak properties. We bridge this gap with a complete, decomposition-free coverage path planner, CDF, that does not require any geometric decomposition of the environment as in existing decomposition-based approaches, and also is also provably resolution-complete. CDF leverages a library of precomputed coverage behaviors known through experience to be effective in certain local environments, and determines where to apply them online. We show quantitative and qualitative benefits of CDF over decomposition-based and decomposition-free methods.

Finally, in Chapter 5, we took a step towards addressing a major limitation of CDF—that it performs an exhaustive searches that involve forward-simulation of coverage behaviors at frontier cells. This becomes prohibitively slow with more complex sensor

models such as ray casting. We propose an extension, CDF++, that uses learnt coverage footprints for behaviors, doing away with the need to forward simulate entire behaviors during the search. We show the benefits of CDF++ over CDF in a certain set of problems, and mention further improvements and analysis in progress.

BIBLIOGRAPHY

[1] ACAR, E. U., CHOSET, H., RIZZI, A. A., ATKAR, P. N., AND HULL, D. Morse decompositions for coverage tasks. *The international journal of robotics research 21*, 4 (2002), 331–344.

[2] ADEMOYE, T. A., AND DAVARI, A. Trajectory planning for multiple autonomous systems using mixed integer linear programming. In *Proceedings of the Thirty-Eighth Southeastern Symposium on System Theory* (2006), IEEE, pp. 175–179.

[3] ADLER, B., XIAO, J., AND ZHANG, J. Autonomous exploration of urban environments using unmanned aerial vehicles. *Journal of Field Robotics 31*, 6 (2014), 912–939.

[4] AINE, S., SWAMINATHAN, S., NARAYANAN, V., HWANG, V., AND LIKHACHEV, M. Multi-heuristic A*. *The International Journal of Robotics Research 35*, 1-3 (2016), 224–243.

[5] ARKIN, E. M., AND HASSIN, R. Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics 55*, 3 (1994), 197–218.

[6] ARORA, S., AND SCHERER, S. Pasp: Policy based approach for sensor planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (2015), IEEE, pp. 3479–3486.

[7] ATANASOV, N., LE NY, J., DANIILIDIS, K., AND PAPPAS, G. J. Information acquisition with sensing robots: Algorithms and error bounds. In *2014 IEEE International Conference on Robotics and Automation (ICRA)* (2014), IEEE, pp. 6447–6454.

[8] BÄHNEMANN, R., LAWRANCE, N., CHUNG, J. J., PANTIC, M., SIEGWART, R., AND NIETO, J. Revisiting boustrophedon coverage path planning as a generalized traveling salesman problem. In *Field and Service Robotics* (2021), Springer, pp. 277–290.

[9] BAI, S., WANG, J., CHEN, F., AND ENGLOT, B. Information-theoretic exploration with bayesian optimization. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2016), IEEE, pp. 1816–1822.

[10] BAJCSY, R., ALOIMONOS, Y., AND TSOTSOS, J. K. Revisiting active perception. *Autonomous Robots 42*, 2 (2018), 177–196.

[11] BALCH, T. The case for randomized search. In *Workshop on Sensors and Motion, IEEE International Conference on Robotics and Automation, San Francisco, CA* (2000), pp. 213–215.

[12] BARNIER, N., AND ALLIGNOL, C. Trajectory deconfliction with constraint programming. *The Knowledge Engineering Review 27*, 3 (2012), 291–307.

[13] BELLINGHAM, J. S. *Coordination and control of UAV fleets using mixed-integer linear programming*. PhD thesis, Massachusetts Institute of Technology, 2002.

[14] BOCHKAREV, S., AND SMITH, S. L. On minimizing turns in robot coverage path planning. In *2016 IEEE international conference on automation science and engineering (CASE)* (2016), IEEE, pp. 1237–1242.

[15] BORMANN, R., JORDAN, F., LI, W., HAMPP, J., AND HÄGELE, M. Room segmentation: Survey, implementation, and analysis. In *2016 IEEE international conference on robotics and automation (ICRA)* (2016), IEEE, pp. 1019–1026.

[16] BOURGAULT, F., FURUKAWA, T., AND DURRANT-WHYTE, H. F. Coordinated decentralized search for a lost target in a bayesian world. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)* (2003), vol. 1, IEEE, pp. 48–53.

[17] BOURGAULT, F., MAKARENKO, A. A., WILLIAMS, S. B., GROCHOLSKY, B., AND DURRANT-WHYTE, H. F. Information based adaptive robotic exploration. In *IEEE/RSJ international conference on intelligent robots and systems* (2002), vol. 1, IEEE, pp. 540–545.

[18] BROOKS, R. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation 2*, 1 (1986), 14–23.

[19] BROWN, S., AND WASLANDER, S. L. The constriction decomposition method for coverage path planning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2016), IEEE, pp. 3233–3238.

[20] BUTZKE, J., AND LIKHACHEV, M. Planning for multi-robot exploration with multiple objective utility functions. In *IROS* (2011), pp. 3254–3259.

[21] CHOSET, H. Coverage for robotics–a survey of recent results. *Annals of mathematics and artificial intelligence 31*, 1-4 (2001), 113–126.

[22] CHOSET, H., LYNCH, K. M., HUTCHINSON, S., KANTOR, G. A., AND BURGARD, W. *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[23] CHUNG, T. H., GUPTA, V., BURDICK, J. W., AND MURRAY, R. M. On a decentralized active sensing strategy using mobile sensor platforms in a network. In *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)* (2004), vol. 2, IEEE, pp. 1914–1919.

[24] COSTANTE, G., FORSTER, C., DELMERICO, J., VALIGI, P., AND SCARAMUZZA, D. Perception-aware path planning. *arXiv preprint arXiv:1605.04151* (2016).

[25] DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische mathematik 1*, 1 (1959), 269–271.

[26] DU, Q., FABER, V., AND GUNZBURGER, M. Centroidal voronoi tessellations: Applications and algorithms. *SIAM review 41*, 4 (1999), 637–676.

[27] ENGLOT, B., AND HOVER, F. S. Three-dimensional coverage planning for an underwater inspection robot. *The International Journal of Robotics Research 32*, 9-10 (2013), 1048–1073.

[28] ERDMANN, M., AND LOZANO-PEREZ, T. On multiple moving objects. *Algorithmica 2*, 1-4 (1987), 477.

[29] FAIGL, J., AND KULICH, M. On determination of goal candidates in frontier-based multi-robot exploration. In *2013 European Conference on Mobile Robots* (2013), IEEE, pp. 210–215.

[30] FAIGL, J., VÁŇA, P., SASKA, M., BÁČA, T., AND SPURNỲ, V. On solution of the dubins touring problem. In *2017 European Conference on Mobile Robots (ECMR)* (2017), IEEE, pp. 1–6.

[31] FRANCO, C., LÓPEZ-NICOLÁS, G., SAGÜÉS, C., AND LLORENTE, S. Persistent coverage control with variable coverage action in multi-robot environment. In *CDC* (2013), pp. 6055–6060.

[32] FURCY, D. ITSA*: Iterative tunneling search with A*. In *Proceedings of the National Conference on Artificial Intelligence (AAAI), Workshop on Heuristic Search, Memory-Based Heuristics and Their Applications, Boston, Massachusetts* (2006).

[33] FURGALE, P., AND BARFOOT, T. D. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics 27*, 5 (2010), 534–560.

[34] GABRIELY, Y., AND RIMON, E. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of mathematics and artificial intelligence 31*, 1 (2001), 77–98.

[35] GABRIELY, Y., AND RIMON, E. Spiral-stc: An on-line coverage algorithm of grid environments by a mobile robot. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)* (2002), vol. 1, IEEE, pp. 954–960.

[36] GAGE, D. W. Randomized search strategies with imperfect sensors. In *Mobile Robots VIII* (1994), vol. 2058, International Society for Optics and Photonics, pp. 270–279.

[37] GALCERAN, E., AND CARRERAS, M. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems 61*, 12 (2013), 1258–1276.

[38] GAT, E., AND DORAIS, G. Robot navigation by conditional sequencing.

[39] GHOLAMI SHAHBANDI, S., AND MAGNUSSON, M. 2d map alignment with region decomposition. *Autonomous Robots 43*, 5 (2019), 1117–1136.

[40] GOLDEN, B. L., RAGHAVAN, S., AND WASIL, E. A. *The vehicle routing problem: latest advances and new challenges*, vol. 43. Springer Science & Business Media, 2008.

[41] GUPTA, V., CHUNG, T. H., HASSIBI, B., AND MURRAY, R. M. On a stochastic sensor selection algorithm with applications in sensor scheduling and sensor coverage. *Automatica 42*, 2 (2006), 251–260.

[42] HART, P. E., NILSSON, N. J., AND RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics 4*, 2 (1968), 100–107.

[43] HERO, A. O., AND COCHRAN, D. Sensor management: Past, present, and future. *IEEE Sensors Journal 11*, 12 (2011), 3064–3075.

[44] HOFNER, C., AND SCHMIDT, G. Path planning and guidance techniques for an autonomous mobile cleaning robot. *Robotics and autonomous systems 14*, 2-3 (1995), 199–212.

[45] HUANG, A. S., BACHRACH, A., HENRY, P., KRAININ, M., MATURANA, D., FOX, D., AND ROY, N. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Robotics Research*. Springer, 2017, pp. 235–252.

[46] JADIDI, M. G., MIRO, J. V., AND DISSANAYAKE, G. Mutual information-based exploration on continuous occupancy maps. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2015), IEEE, pp. 6086–6092.

[47] KAGAMI, S., AND ISHIKAWA, M. A sensor selection method considering communication delays. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science) 89*, 5 (2006), 21–31.

[48] KANTAROS, Y., SCHLOTFELDT, B., ATANASOV, N., AND PAPPAS, G. J. Asymptotically optimal planning for non-myopic multi-robot information gathering. In *Robotics: Science and Systems* (2019).

[49] KAPOUTSIS, A. C., CHATZICHRISTOFIS, S. A., AND KOSMATOPOULOS, E. B. DARP: divide areas algorithm for optimal multi-robot coverage path planning. *Journal of Intelligent & Robotic Systems 86*, 3-4 (2017), 663–680.

[50] KELLER, J. F. Path planning for persistent surveillance applications using fixed-wing unmanned aerial vehicles. *Computers & Operations Research* (2016).

[51] KUSNUR, T., AND LIKHACHEV, M. Complete, decomposition-free coverage path planning. In *IEEE International Conference on Automation Science and Engineering, Mexico* (2022).

[52] KUSNUR, T., MUKHERJEE, S., SAXENA, D. M., FUKAMI, T., KOYAMA, T., SALZMAN, O., AND LIKHACHEV, M. A planning framework for persistent, multi-uav coverage with global deconfliction. In *2019 International Conference on Field and Service Robotics (FSR)* (2019).

[53] KUSNUR, T., SAXENA, D. M., AND LIKHACHEV, M. Search-based planning for active sensing in goal-directed coverage tasks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), IEEE, pp. 15–21.

[54] LATOMBE, J.-C. *Robot motion planning*, vol. 124. Springer Science & Business Media, 2012.

[55] LAVALLE, S. M. *Planning algorithms*. Cambridge university press, 2006.

[56] LEAHY, K., ZHOU, D., VASILE, C.-I., OIKONOMOPOULOS, K., SCHWAGER, M., AND BELTA, C. Persistent surveillance for unmanned aerial vehicles subject to charging and temporal logic constraints. *Auton. Robots 40*, 8 (2016), 1363–1378.

[57] LIKHACHEV, M., AND FERGUSON, D. Planning long dynamically feasible maneuvers for autonomous vehicles. *IJRR 28*, 8 (2009), 933–945.

[58] LU, W., ZHANG, G., AND FERRARI, S. An information potential approach to integrated sensor path planning and control. *IEEE Transactions on Robotics 30*, 4 (2014), 919–934.

[59] MAINI, P., GONULTAS, B. M., AND ISLER, V. Online coverage planning for an autonomous weed mowing robot with curvature constraints. *IEEE Robotics and Automation Letters* (2022).

[60] MANNADIAR, R., AND REKLEITIS, I. Optimal coverage of a known arbitrary environment. In *2010 IEEE International conference on robotics and automation* (2010), IEEE, pp. 5525–5530.

[61] MEIER, L., PESCHON, J., AND DRESSLER, R. Optimal control of measurement subsystems. *IEEE Transactions on Automatic Control 12*, 5 (1967), 528–536.

[62] MELLONE, A., FRANZINI, G., POLLINI, L., AND INNOCENTI, M. Persistent coverage control for teams of heterogeneous agents. In *CDC* (2018), pp. 2114–2119.

[63] MORI, H. Active sensing in vision-based stereotyped motion. In *EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications* (1990), IEEE, pp. 167–174.

[64] MUKAI, T., AND ISHIKAWA, M. An active sensing method using estimated errors for multisensor fusion systems. *IEEE Transactions on Industrial Electronics 43*, 3 (1996), 380–386.

[65] NEDJATI, A., IZBIRAK, G., VIZVARI, B., AND ARKAT, J. Complete coverage path planning for a multi-UAV response system in post-earthquake assessment. *Robotics 5*, 4 (2016), 26.

[66] OKSANEN, T., AND VISALA, A. Coverage path planning algorithms for agricultural field machines. *Journal of field robotics 26*, 8 (2009), 651–668.

[67] OSHMAN, Y. Optimal sensor selection strategy for discrete-time state estimators. *IEEE Transactions on Aerospace and Electronic Systems 30*, 2 (1994), 307–314.

[68] OSSWALD, S., BENNEWITZ, M., BURGARD, W., AND STACHNISS, C. Speeding-up robot exploration by exploiting background information. *IEEE Robotics and Automation Letters 1*, 2 (2016), 716–723.

[69] PALACIN, J., PALLEJA, T., VALGANÓN, I., PERNIA, R., AND ROCA, J. Measuring coverage performances of a floor cleaning mobile robot using a vision system. In *Proceedings of the 2005 IEEE international conference on robotics and automation* (2005), IEEE, pp. 4236–4241.

[70] PALACIOS-GASÓS, J. M., MONTIJANO, E., SAGUES, C., AND LLORENTE, S. Multi-robot persistent coverage using branch and bound. In *ACC* (2016), pp. 5697–5702.

[71] PAPADIMITRIOU, C. H. The euclidean travelling salesman problem is NP-complete. *Theoretical computer science 4*, 3 (1977), 237–244.

[72] PĚNIČKA, R., FAIGL, J., AND SASKA, M. Physical orienteering problem for unmanned aerial vehicle data collection planning in environments with obstacles. *IEEE Robotics and Automation Letters 4*, 3 (2019), 3005–3012.

[73] PĚNIČKA, R., FAIGL, J., SASKA, M., AND VÁŇA, P. Data collection planning with non-zero sensing distance for a budget and curvature constrained unmanned aerial vehicle. *Autonomous Robots 43*, 8 (2019), 1937–1956.

[74] PIMENTEL, J. M., ALVIM, M. S., CAMPOS, M. F., AND MACHARET, D. G. Information-driven rapidly-exploring random tree for efficient environment exploration. *Journal of Intelligent & Robotic Systems 91*, 2 (2018), 313–331.

[75] PIVTORAIKO, M., AND KELLY, A. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *IROS* (2005), pp. 3231–3237.

[76] PIVTORAIKO, M., KNEPPER, R. A., AND KELLY, A. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics 26*, 3 (2009), 308–333.

[77] POHL, I. Heuristic search viewed as path finding in a graph. *Artificial intelligence 1*, 3-4 (1970), 193–204.

[78] RAMESH, M., IMESON, F., FIDAN, B., AND SMITH, S. L. Optimal partitioning of non-convex environments for minimum turn coverage planning. *arXiv preprint arXiv:2109.08185* (2021).

[79] RYAN, A., AND HEDRICK, J. K. Particle filter based information-theoretic active sensing. *Robotics and Autonomous Systems 58*, 5 (2010), 574–584.

[80] SCHERER, S., CHAMBERLAIN, L., AND SINGH, S. Autonomous landing at unprepared sites by a full-scale helicopter. *Robotics and Autonomous Systems 60*, 12 (2012), 1545–1562.

[81] SCHLOTFELDT, B., ATANASOV, N., AND PAPPAS, G. J. Maximum information bounds for planning active sensing trajectories. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019), IEEE, pp. 4913–4920.

[82] SCHWAGER, M., RUS, D., AND SLOTINE, J.-J. Decentralized, adaptive coverage control for networked robots. *IJRR 28*, 3 (2009), 357–375.

[83] SHAH, K., BALLARD, G., SCHMIDT, A., AND SCHWAGER, M. Multidrone aerial surveys of penguin colonies in antarctica. *Science Robotics 5*, 47 (2020), eabc3000.

[84] SHRESTHA, R., TIAN, F.-P., FENG, W., TAN, P., AND VAUGHAN, R. Learned map prediction for enhanced mobile robot exploration. In *2019 International Conference on Robotics and Automation (ICRA)* (2019), IEEE, pp. 1197–1204.

[85] SMITH, R. N., SCHWAGER, M., SMITH, S. L., JONES, B. H., RUS, D., AND SUKHATME, G. S. Persistent ocean monitoring with underwater gliders: Adapting sampling resolution. *JFR 28*, 5 (2011), 714–741.

[86] SMITH, S. L., SCHWAGER, M., AND RUS, D. Persistent robotic tasks: Monitoring and sweeping in changing environments. *arXiv preprint arXiv:1102.0603* (2011).

[87] SRINIVASAN, S., LATCHMAN, H., SHEA, J., WONG, T., AND MCNAIR, J. Airborne traffic surveillance systems: video surveillance of highway traffic. In *Proceedings of the ACM 2nd international workshop on Video surveillance & sensor networks* (2004), pp. 131–135.

[88] STUMP, E., AND MICHAEL, N. Multi-robot persistent surveillance planning as a vehicle routing problem. In *IEEE International Conference on Automation Science and Engineering* (2011), pp. 569–575.

[89] SUN, X., KOENIG, S., AND YEOH, W. Generalized adaptive A*. In *AAMAS* (2008), International Foundation for Autonomous Agents and Multiagent Systems, pp. 469–476.

[90] TABIB, W., GOEL, K., YAO, J., BOIRUM, C., AND MICHAEL, N. Autonomous cave surveying with an aerial robot. *IEEE Transactions on Robotics 38*, 2 (2021), 1016–1032.

[91] TEIXEIRA, L., ALZUGARAY, I., AND CHLI, M. Autonomous aerial inspection using visual-inertial robust localization and mapping. In *FSR* (2018), Springer, pp. 191–204.

[92] THAKUR, D., LIKHACHEV, M., KELLER, J., KUMAR, V., DOBROKHODOV, V., JONES, K., WURZ, J., AND KAMINER, I. Planning for opportunistic surveillance with multiple robots. In *IROS* (2013), pp. 5750–5757.

[93] THRUN, S. A probabilistic on-line mapping algorithm for teams of mobile robots. *The International Journal of Robotics Research 20*, 5 (2001), 335–363.

[94] TOTH, P., AND VIGO, D. *The vehicle routing problem*. SIAM, 2002.

[95] TOVEY, C., AND KOENIG, S. Improved analysis of greedy mapping. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)* (2003), vol. 4, IEEE, pp. 3251–3257.

[96] UMARI, H., AND MUKHOPADHYAY, S. Autonomous robotic exploration based on multiple rapidly-exploring randomized trees. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017), IEEE, pp. 1396–1402.

[97] VÁŇA, P., AND FAIGL, J. On the dubins traveling salesman problem with neighborhoods. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2015), IEEE, pp. 4029–4034.

[98] VANSTEENWEGEN, P., SOUFFRIAU, W., AND VAN OUDHEUSDEN, D. The orienteering problem: A survey. *European Journal of Operational Research 209*, 1 (2011), 1–10.

[99] VIDAL, E., PALOMERAS, N., ISTENIČ, K., HERNÁNDEZ, J. D., AND CARRERAS, M. Two-dimensional frontier-based viewpoint generation for exploring and mapping underwater environments. *Sensors 19*, 6 (2019), 1460.

[100] YAMAUCHI, B. A frontier-based approach for autonomous exploration. In *cira* (1997), IEEE, p. 146.

[101] YAMAUCHI, B., ET AL. Frontier-based exploration using multiple robots. In *Agents* (1998), vol. 98, pp. 47–53.

[102] ZHOU, B., ZHANG, Y., CHEN, X., AND SHEN, S. Fuel: Fast uav exploration using incremental frontier structure and hierarchical planning. *IEEE Robotics and Automation Letters 6*, 2 (2021), 779–786.

[103] ZHU, C., DING, R., LIN, M., AND WU, Y. A 3D frontier-based exploration tool for MAVs. In *ICTAI* (2015), pp. 348–352.