

Data-driven Spatial Classification using Multi-Arm Bandits for Monitoring with Energy-Constrained Mobile Robots

Xiaoshan Lin, Siddharth Nayak, Stefano Di Cairano, Abraham P. Vinod[†]

Abstract—We consider the spatial classification problem for monitoring using data collected by a coordinated team of mobile robots. Such classification problems arise in several applications including search-and-rescue and precision agriculture. Specifically, we want to classify the regions of a search environment into interesting and uninteresting as quickly as possible using a team of mobile sensors and mobile charging stations. We develop a data-driven strategy that accommodates the noise in sensed data and the limited energy capacity of the sensors, and generates collision-free motion plans for the team. We propose a bi-level approach, where a high-level planner leverages a multi-armed bandit framework to determine the potential regions of interest for the drones to visit next based on the data collected online. Then, a low-level path planner based on integer programming coordinates the paths for the team to visit the target regions subject to the physical constraints. We characterize several theoretical properties of the proposed approach, including anytime guarantees and task completion time. We show the efficacy of our approach in simulation, and further validate these observations in physical experiments using mobile robots.

Index Terms—Environmental monitoring, Multi-arm bandits, Optimization-based planning, Multi-agent motion planning

I. INTRODUCTION

Monitoring extensive areas using autonomous search teams has several applications in infrastructure maintenance, search and rescue operations, and wildlife tracking [1]–[4]. In this paper, we study the spatial classification problem that requires rapid identification of regions in a search environment containing interesting objects or phenomena. We tackle the spatial classification problem using a coordinated team of heterogeneous robots comprising of mobile sensors and mobile charging stations. The mobile sensors (e.g. drones) are used for collecting data from the environment, while the mobile stations (e.g. ground vehicles) address the energy limitations of drones. Apart from the physical constraints on the existing mobile robotic platforms like dynamics, collision-avoidance, and battery, the deployment strategies for the team must also accommodate noisy measurements arising from low-cost, low-weight onboard sensors. This paper, builds on our recent work [5], to propose a *data-driven spatial classification algorithm that explicitly accounts for physical constraints on the robot as well and accommodates noisy data collected by mobile sensors*.

* Corresponding author.

X. Lin is with the Aerospace Engineering and Mechanics department, University of Minnesota, Minneapolis, MN 55455, USA (email: lin00668@umn.edu). S. Nayak is with the Aeronautics and Astronautics department, Massachusetts Institute of Technology, Cambridge, MA 02139, USA (email: sidnayak@mit.edu). S. Di Cairano and A. P. Vinod are with Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA 02139, USA (email: dicairano@ieee.org, abraham.p.vinod@ieee.org). This work was developed during the summer internships of X. Lin and S. Nayak at MERL.

Various strategies have been proposed for multi-agent monitoring [2]–[4], including submodular maximization [6], Voronoi-based search with function approximation [7]–[9], active sensing/perception [10], graph-based search [11], [12], and statistical learning [13], [14]. Despite their effectiveness, many of these approaches may require perfect sensing, may lack finite-time guarantees on search performance, or may ignore/relax the physical constraints on the team.

Multi-arm bandits are a special class of reinforcement learning algorithms, designed for problems where the current actions *do not* impact future reward [15]. These algorithms exhibit non-asymptotic guarantees of performance, and have been recently applied to monitoring tasks [1], [16], [17]. In [16], [17], bandits-based algorithms are used to identify the top- k points of interest with a probabilistic finite-time guarantee, but required prior knowledge on the number of interesting regions. Our prior work [1] used *thresholding bandits* [18]–[20] and relaxed such a requirement to identify *all* interesting regions. However, these approaches consider the physical limitations on the team (dynamics and energy) only as soft constraints.

Energy-aware coordinated control of heterogeneous teams of aerial and ground vehicles has gained increasing attention [2]–[4], [21]–[23]. For example, approaches based on satisfiability modulo theories in [21] plan energy-efficient trajectories for mobile charging stations given pre-computed trajectories of sensors. In contrast, [22] generates offline trajectories for both sensors and mobile charging stations simultaneously via partitioning, with the ability to modify them during online execution to handle unknown obstacles. Additionally, approaches based on traveling salesman problems [23] have also been explored. However, these works may be limited to static goals due to the need for offline planning, or may not adapt to dynamic assignments of sub-teams formed by a ground vehicle and its associated aerial vehicles.

The main contribution of this work is a bi-level approach that uses a combination of multi-arm bandits and optimization to address the data-driven spatial classification problem using a heterogeneous team. We extend our recent work [5] to: 1) incorporation of additional constraints into the low-level planner, and 2) online adjustment of computed paths to guarantee collision avoidance using linear assignment, and 3) an experimental validation of the proposed approach in a search-and-rescue application using drones and ground robots with vision-based sensing. We also provide theoretical guarantees for the proposed approach, including anytime guarantees (the algorithm provides a useful result, even when terminated prematurely), and finite upper bounds on task completion time.

Notation: $|\mathcal{S}|$ is the cardinality of a set \mathcal{S} and $\mathbb{N}_{[a,b]}$ is the set of natural numbers between (and including) $a, b \in \mathbb{N}$, $a \leq b$.

II. PROBLEM STATEMENT

Search environment: We define the search environment as a set \mathcal{G} of grid cells. Within \mathcal{G} , let $\mathcal{O} \subset \mathcal{G}$ be the known set of cells that are occupied by obstacles or no-fly zones, and $\mathcal{I} \subset \mathcal{G} \setminus \mathcal{O}$ be the *a priori* unknown set of *interesting cells* that are occupied by objects/phenomena of interest. The objective of the spatial classification problem is to identify \mathcal{I} as soon as possible using data collected online by a team.

Search team: We define the search team as comprising of $N_d \in \mathbb{N}$ mobile sensors (e.g., drones) and $N_c \in \mathbb{N}$ charging stations (e.g., autonomous trucks). For simplicity, we will refer to the mobile sensors as *sensors* and the charging stations as *stations* throughout the rest of the paper.

- *Sensors.* The sensors are responsible for monitoring the search environment and collecting data. They are energy-limited and must recharge by rendezvous with a station every $T_d \in \mathbb{N}$ moves. At each move, the sensors can move to their neighbouring cells in all cardinal directions and all diagonal directions (like a king piece in chess). For any cell $l \in \mathcal{G} \setminus \mathcal{O}$, the set of neighboring cells that a sensor can move is $\mathcal{N}(l) \subseteq \mathcal{G} \setminus \mathcal{O}$.
- *Stations.* Upon rendezvous with a sensor, a station recharges its battery. We assume that the stations have sufficient energy for the mission and do not require recharging themselves, similarly to [5], [22], [23]. The stations move as the sensors, but their workspace may be more restricted (e.g. stations can only move on the roads in a search-and-rescue scenario). We define *station-admissible set* $\mathcal{R} \subseteq \mathcal{G} \setminus \mathcal{O}$ as the set of grid cells that can be visited by a station. For any cell $l \in \mathcal{R}$, the set of neighboring cells that a station can move is $\mathcal{N}_c(l) \triangleq \mathcal{R} \cap \mathcal{N}(l)$.

We model the difference in agility between sensors and stations by their speeds v_d and v_c respectively, with $v_d > v_c$. We assume that the stations can move at most $T_c < T_d$ cells by the time the sensors covers T_d cells, with $T_c \triangleq T_d \frac{v_c}{v_d} \in \mathbb{N}$.

Sensing: Let $\mathcal{C} = \mathcal{G} \setminus \mathcal{O}$ be the set of candidate cells that may potentially be “interesting”, i.e., it contains the search objective. When a sensor visits a cell in \mathcal{C} , it receives noisy, binary measurements (data) of whether that cell is interesting or not. Formally, every cell $l \in \mathcal{C}$ has a corresponding Bernoulli random variable ν_l with *a priori* unknown mean μ_l , and each cell visit generates $B \in \mathbb{N}$ realizations of ν_l , where B is the sample batch size. The mean μ_l may be influenced by the underlying spatial distribution of the interesting cells as well as the imperfections of the noisy sensors and the perception algorithms used by the team. We assume that ν_{l_1} and ν_{l_2} are independent for any $l_1, l_2 \in \mathcal{C}$.

Labeling error criterion: For a user-defined threshold $\theta \in (0, 1)$, we formally define \mathcal{I} as

$$\mathcal{I} = \mathcal{S}_\theta \triangleq \{l \in \mathcal{C} : \mu_l \geq \theta\}. \quad (1)$$

From the definition of μ_l , \mathcal{S}_θ are the cells with a likelihood of at least θ of being sensed as interesting. Thus, the spatial classification problem of interest is to identify \mathcal{S}_θ using noisy measurements collected by the team.

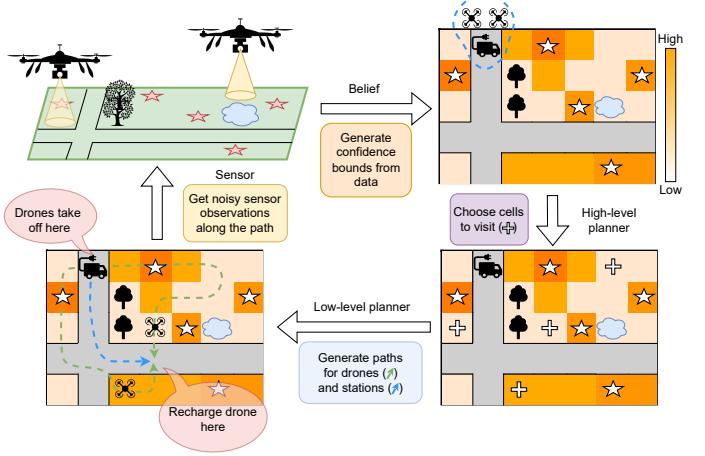


Fig. 1: Data-driven multi-agent search under noisy observations. A high-level planner uses the available measurements of cells to determine the confidences of an unclassified cell to be interesting, and determines the cells to visit next. A low-level planner computes trajectories for the agents subject to various motion constraints arising from dynamics as well as energy limitations. Altogether, the proposed approach guarantees safe, data-driven sensor deployment to address Problem 1.

We assume tolerance to labeling errors for approximating \mathcal{S}_θ using noisy data in finite time, similarly to [1], [5].

Assumption 1. *We ignore the labeling error for all cells $l \in \mathcal{G}$ with $\mu_l \in (\theta - \epsilon, \theta + \epsilon)$ for some small tolerance $\epsilon > 0$.*

Under Assumption 1, \mathcal{S}_θ may be approximated by a (keep) set $\mathcal{K} \subseteq \mathcal{G}$ such that $\mathcal{S}_{\theta+\epsilon} \subseteq \mathcal{K} \subseteq \mathcal{S}_{\theta-\epsilon}$. For a given *labeling error probability* $\delta \in (0, 1)$, the spatial classification problem may be (approximately) solved by identifying \mathcal{K} that satisfies

$$\mathbb{P}[(\mathcal{S}_{\theta+\epsilon} \setminus \mathcal{K}) \cup (\mathcal{K} \setminus \mathcal{S}_{\theta-\epsilon}) = \emptyset] \geq 1 - \delta. \quad (2)$$

We refer to (2) as the *labeling error criterion*, which ensures $\mathcal{S}_{\theta+\epsilon} \subseteq \mathcal{K} \subseteq \mathcal{S}_{\theta-\epsilon}$ by requiring that \mathcal{K} includes almost all interesting cells ($\mu_l \geq \theta + \epsilon$) and excludes almost all uninteresting cells ($\mu_l \leq \theta - \epsilon$) with probability $1 - \delta$.

We now state the two problems tackled by this paper:

Problem 1. *Design a data-driven algorithm for the search team that terminates in finite time, upon termination meets the labeling error criterion (2), and ensures that the motion and energy constraints on the team are satisfied.*

Problem 2. *For the algorithm solving Problem 1, determine upper bounds on the time to terminate the search.*

III. METHODOLOGY

We address Problem 1 by an iterative algorithm that uses a bi-level approach, as illustrated in Figure 1. We use *epoch* $p \in \mathbb{N}$ to denote time in a slower time scale used by a high-level planner, as compared to *time step* $t \in \mathbb{N}$, which denotes time in a faster time scale used by a low-level planner.

The high-level planner identifies a set of *epoch goals* based on available (noisy) data collected online. Epoch goals are

cells that the team must visit in the current decision epoch to collect new measurements. We use the multi-arm bandits framework to design the high-level planner. Given a set of epoch goals, the low-level planner generates motion plans for the team to visit the epoch goals and collect the new measurements. The two-staged algorithm continues until a termination criterion is met. In this section, we detail the planners, and describe the termination criterion.

A. High-level Planner: Bandit-based decision making

We propose a high-level planner based on the bandit monitoring algorithm in our prior work [1]. Specifically, the spatial classification problem is cast as a $|\mathcal{C}|$ -armed bandit problem, where the bandit arms are the cells. We design the high-level planner as a sequential decision maker that processes the collected information to decide which cells must be visited at each epoch. We use *upper confidence bounds* typical of bandit-based algorithms [15], [18]–[20] to sample the unclassified cells that are “most likely” to be interesting.

Let $\mathcal{H}_l(p)$ be the history of measurements taken at cell $l \in \mathcal{C}$ collected by the search team until epoch p , and define $\mathcal{H}(p) = \{\mathcal{H}_l(p)\}_{l \in \mathcal{C}}$. Then, at epoch p , we choose D (typically, $D \geq N_d$) distinct cells that achieve the top- D values of a function $J : \mathcal{C} \times \mathbb{N} \times (0, 1) \rightarrow \mathbb{R}$,

$$J(l, p, \delta) = \hat{\mu}_l(p) + U_l(p, \delta), \quad \hat{\mu}_l(p) = \frac{\sum_{h \in \mathcal{H}_l(p)} h}{|\mathcal{H}_l(p)|}, \quad (3a)$$

$$U_l(p, \delta) = 2\sqrt{\frac{2\log(\log_2(2|\mathcal{H}_l(p)|)) + \log(12|\mathcal{C}|/\delta)}{2|\mathcal{H}_l(p)|}}, \quad (3b)$$

with $\hat{\mu}_l(p) = U_l(p, \delta) = \infty$, whenever $\mathcal{H}_l(p) = \emptyset$. Here, $\delta \in (0, 1)$ is a given (small) labeling error probability. After getting the data for the current epoch, we update the sets $\mathcal{K}(p+1)$ and $\mathcal{R}(p+1)$ as follows,

$$\mathcal{K}(p+1) = \{l \in \mathcal{C} : \hat{\mu}_l(p+1) - U_l(p+1, \delta) \geq \theta - \epsilon\}, \quad (4a)$$

$$\mathcal{R}(p+1) = \{l \in \mathcal{C} : \hat{\mu}_l(p+1) + U_l(p+1, \delta) \leq \theta + \epsilon\}. \quad (4b)$$

Since $U_l(p, \delta)$ is a non-increasing function of $|\mathcal{H}_l|$ and $|\mathcal{H}_l|$ is a non-decreasing function in p , the sets \mathcal{K} and \mathcal{R} are monotonic in p with $|\mathcal{K}|$ and $|\mathcal{R}|$ non-decreasing in p . Here, (3) and (4) are motivated by the desire to obtain *anytime guarantees* (see Section IV) along with ensuring a low classification time.

Algorithm 1 summarizes the proposed approach to address Problem 1, also shown in Figure 1. Step 3 of Algorithm 1 identifies the epoch goals for the search team using (3). Step 4 uses a low-level planner to design a deployment for the search team to visit at all epoch goals, which we describe next.

B. Low-level coordination: Integer Program-based solution

The optimization-based planner determines the deployment plan for the team based on the epoch goals identified by the high-level planner. Since the team may not be able to visit all cells in \mathcal{E}_p using a single charge of their batteries, we divide the deployment into *sensing cycles*. Each sensing cycle is such that the sensors always start and end on a station, the sensing cycles are of length T_d , and the paths of the sensors and the stations in sensing cycles satisfy their respective motion constraints.

Algorithm 1 Bandits-based, autonomous spatial classification

Input: Set of grid cells \mathcal{G} , threshold $\theta \in (0, 1)$, labeling error tolerance $\epsilon > 0$, labelling error probability $\delta \in (0, 1)$, number of epoch goals $D \in \mathbb{N}$, sample batch size $B \in \mathbb{N}$, obstacle set \mathcal{O} , station admissible set \mathcal{R}

Output: $\{\mathcal{K}(p)\}_{p \in \mathbb{N}}$, sequence of (keep) sets

- 1: Initialize $p \leftarrow 0$, $\mathcal{K}(p) \leftarrow \emptyset$, and $\mathcal{R}(p) \leftarrow \emptyset$
 - 2: **while** $\mathcal{R}(p) \cup \mathcal{K}(p) \neq \mathcal{C}$ **do**
 - 3: Define \mathcal{E}_p as the top D elements in the list of unclassified cells $l \in \mathcal{C} \setminus (\mathcal{K}(p) \cup \mathcal{R}(p))$, sorted in descending order based on $J(l, p, \delta)$ (3)
 - 4: Deploy the search team to visit cells in \mathcal{E}_p while avoiding \mathcal{O} by using paths generated by the low-level planner (Algorithm 2), take measurements along the way, and update history $\mathcal{H}(p+1)$
 - 5: Update sets of classified cells $\mathcal{K}(p+1)$ and $\mathcal{R}(p+1)$ based on (4) using $\mathcal{H}(p+1)$, \mathcal{C} , θ , δ , and ϵ
 - 6: Increment epoch $p \leftarrow p + 1$
 - 7: **end while**
-

Subsequently, the optimization-based low-level planner solves the following problem at each epoch p ,

$$\min. \quad \text{Number of sensing cycles}, \quad (5a)$$

$$\text{s. t.} \quad \begin{aligned} &\text{Search team respects motion constraints,} \\ &\text{Search team visits all cells in } \mathcal{E}_p, \\ &\text{Search team never visits } \mathcal{O}, \text{ and stations stay in } \mathcal{R}, \\ &\text{Sensors never run out of battery.} \end{aligned} \quad (5b)$$

We formulate the optimization problem (5) as an integer program (IP), and solve it using off-the-shelf solvers [24]. We use the following set of binary decision variables in the IP formulation: $x_{ijkl}, y_{abkl}, \lambda_k \in \{0, 1\}$ for every

$$\begin{aligned} &\text{(Sensor/station index):} \quad i \in \mathcal{D} \triangleq \mathbb{N}_{[1, N_d]}, \quad a \in \mathcal{C} \triangleq \mathbb{N}_{[1, N_c]}, \\ &\text{(Sensing cycle time):} \quad j \in \mathcal{T}_d \triangleq \mathbb{N}_{[0, T_d-1]}, \quad b \in \mathcal{T}_c \triangleq \mathbb{N}_{[0, T_c-1]}, \\ &\text{(Sensing cycle count):} \quad k \in \mathcal{T}_s \triangleq \mathbb{N}_{[0, K-1]}, \\ &\text{(Grid cell location):} \quad l \in \mathcal{G} \triangleq \mathbb{N}_{[1, |\mathcal{G}|]}, \end{aligned} \quad (6)$$

where $x_{ijkl} = 1$ if sensor i is at cell l at time step j in the sensing cycle k , $x_{ijkl} = 0$ otherwise; $y_{abkl} = 1$ if station a is at cell l at time step b in the sensing cycle k , $y_{abkl} = 0$ otherwise; and, $\lambda_k = 1$ if sensing cycle k is necessary for the search team to solve (5), $\lambda_k = 0$ otherwise. Here, K is a user-specified upper bound on the number of sensing cycles. In what follows, the constraints are enforced for all indices i, j, k, l, a , and b as described in (6), unless stated otherwise.

To simplify notation and discussion, (6) associates a binary variable for every $l \in \mathcal{G}$ instead of associating only binary variables within \mathcal{C} for the sensors and within \mathcal{R} for the stations. The number of variables in the resulting IP for each epoch p is $K(1 + |\mathcal{G}|(N_d T_d + N_c T_c))$.

While an integer program formulation of (5) may require non-trivial computational effort (especially for large teams and large search environments), we remind the reader that our setup has relatively permissible bounds on solve time for (5), compared to traditional motion planning problems. Specifically, at each epoch, all sensors will be charging their

batteries after their rendezvous with a station. Thus, it suffices to solve (5) within the time to recharge the batteries of sensors, which may be in the order of tens of minutes.

Motion constraints: The following constraints enforce (5b),

$$x_{i00l} = 1 \text{ and } y_{a00d} = 1, \quad \forall l \in \mathcal{X}_0, d \in \mathcal{Y}_0, \quad (7a)$$

$$\sum_{m \in \mathcal{N}(l)} x_{i(j-1)km} \geq x_{ijkl}, \quad (7b)$$

$$\sum_{m \in \mathcal{N}_c(l)} y_{a(b-1)km} \geq y_{abkl}, \quad (7c)$$

$$\sum_{a \in \mathcal{C}} y_{abkl} \leq 1 \quad (7d)$$

$$\sum_{i \in \mathcal{D}} x_{ijkl} \leq 1, \quad \forall j \in \mathbb{N}_{[1, T_d-2]} \quad (7e)$$

$$\sum_{l \in \mathcal{G}} x_{ijkl} = \sum_{l \in \mathcal{G}} y_{abkl} = \lambda_k, \quad (7f)$$

$$\lambda_k \geq \lambda_{k+1}, \quad \forall k \in \mathcal{T}_s. \quad (7g)$$

Constraint (7a) sets the sensor and the station locations to the team configuration $(\mathcal{X}_0, \mathcal{Y}_0)$ at epoch p , where $\mathcal{X}_0, \mathcal{Y}_0 \subset \mathcal{G}$ is the set of cells occupied by the sensors and stations respectively at epoch p . Constraints (7b) and (7c) require that the sensors and the stations on cell $l \in \mathcal{G}$ move only to a neighboring cell in $\mathcal{N}(l)$ and $\mathcal{N}_c(l)$ respectively. Constraint (7d) requires no two stations to occupy the same cell at any time to avoid collisions. Constraint (7e) enforces a similar collision avoidance among sensors, but is relaxed at the beginning and the end of epochs to facilitate multiple sensors to rendezvous with the same station. Alternatively, we may guarantee collision avoidance among sensors by requiring them to fly at different altitudes [5]. Constraint (7f) link the binary variables corresponding to sensing cycles λ_k to the team paths. Specifically, $\lambda_k = 0$ requires the team to visit no cells in \mathcal{G} during the sensing cycle k , i.e., the path of the team terminates prior to sensing cycle k . On the other hand, when $\lambda_k = 1$, each sensor and station visit exactly one cell in \mathcal{G} at each time step in the sensing cycle k . Constraint (7g) encodes the temporal constraint among sensing cycles, i.e., $\lambda_{m+1} = 1$ for any $m \in \mathcal{T}_s$ implies that $\lambda_k = 1, \forall k \in \mathbb{N}_{[0, m]}$.

Visit constraints: The following constraints enforce (5c) and (5d) by requiring that every cell $m \in \mathcal{E}_p$ is visited by some sensors at some time step in some sensing cycle, the team (both sensors and stations) never visits any cell in \mathcal{O} , and the stations only visit cells in \mathcal{R} ,

$$\sum_{(i,j,k) \in \mathcal{D} \times \mathcal{T}_d \times \mathcal{T}_s} x_{ijkl} \geq 1, \quad \forall l \in \mathcal{E}_p, \quad (8a)$$

$$x_{ijkl} = y_{abkl} = 0, \quad \forall l \in \mathcal{O}, \quad (8b)$$

$$y_{abkl} = 0, \quad \forall l \in \mathcal{G} \setminus \mathcal{R}. \quad (8c)$$

Battery constraints: The following constraints enforce (5e),

$$\sum_{(j,l) \in \mathcal{T}_d \times \mathcal{G}} x_{ijkl} \leq T_d, \quad \sum_{(b,l) \in \mathcal{T}_c \times \mathcal{G}} y_{abkl} \leq T_c, \quad (9a)$$

$$|x_{i0(k+1)l} - x_{iT_dkl}| \leq 2(1 - \lambda_{k+1}), \quad \forall k \in \mathbb{N}_{[1, K-1]}, \quad (9b)$$

$$|y_{a0(k+1)l} - y_{aT_ckl}| \leq 2(1 - \lambda_{k+1}), \quad \forall k \in \mathbb{N}_{[1, K-1]}, \quad (9c)$$

$$x_{i0kl} \leq \sum_{a \in \mathcal{C}} y_{a0kl}, \quad x_{iT_{d-1}kl} \leq \sum_{a \in \mathcal{C}} y_{a(T_c-1)kl} \quad (9d)$$

Constraint (9a) requires the path lengths of sensors and stations to satisfy the energy and velocity constraints. Constraints (9b) and (9c) require the positions of sensors and stations to remain unchanged between sensing cycles, and are trivially satisfied when $\lambda_{k+1} = 0$. Constraints (9d) require every sensor to start

Algorithm 2 Low-level planner: Integer program

Input: Epoch goals \mathcal{E}_p and search team configuration $(\mathcal{X}_0, \mathcal{Y}_0)$ at epoch p , obstacle set \mathcal{O} , station admissible set \mathcal{R} , search team parameters $(T_d, T_c, N_d, N_c, K, \mathcal{G})$

Output: Paths for the search team

1: Compute $x_{ijkl}^*, y_{abkl}^*, \lambda_k^*$ by solving the integer program,

$$\begin{aligned} & \text{minimize} && \sum_{k \in \mathbb{N}_{[1, K]}} \lambda_k, \\ & \text{subject to} && (7), (8), \text{ and } (9). \end{aligned} \quad (11)$$

2: Compute paths for the sensors and the stations by extracting all indices (i, j, k, l) and (a, b, c, d) such that $x_{ijkl}^* = 1$ and $y_{abcd}^* = 1$, and creating a sequence of cells to traverse through indices i, j, k, l, a, b, c, d

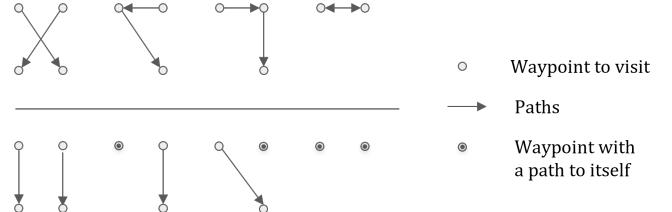


Fig. 2: (Top) Four types of potential collisions in the paths computed by the low-level motion planner. (Bottom) Alternative collision-free paths that visit the same waypoints as in the top figure, identified via a linear assignment.

and end its path within a sensing cycle on a cell occupied by one of the stations.

Algorithm 2 summarizes the IP-based low-level planner. We can also extend the integer program (11) to impose additional constraints on the team. For example,

$$\sum_{i \in \mathcal{D}} x_{i0kl} \leq C, \quad \sum_{i \in \mathcal{D}} x_{iT_dkl} \leq C \quad (10)$$

requires that the number of sensors rendezvous on each station does not exceed a pre-specified limit $C \in \mathbb{N}$. Such constraints may be motivated by resource constraints, such as the number of chargers on each station being limited to C .

C. Collision avoidance for the sensors

While (7d) and (7e) ensure that no sensors and stations occupy the same cell for collision avoidance purposes, we may still have collisions during transitions. Figure 2 (top) shows four potential collision cases in paths computed by the low-level planner, involving any pair of sensors. We avoid such collisions via a linear assignment-based reassignment of the waypoints to the robots, as shown in Figure 2 (bottom). Similar reassignment-based corrections to multi-agent motion plans have been used in literature [25].

Given the paths of the N_d sensors of any sensing cycle k , let \mathcal{T}_{jk} and $\mathcal{T}_{(j+1)k}$ be the set of cell locations the sensors will visit at time j and $j+1$ respectively. The linear assignment problem amounts to finding a bijection $f : \mathcal{T}_{jk} \rightarrow \mathcal{T}_{(j+1)k}$ that minimizes the cost function $\sum_{l \in \mathcal{T}_{jk}} C(l, f(l))$, where $C(l, f(l))$ denotes the Euclidean distance between cell location l and $f(l)$. Recall that linear assignment problems are a special

class of integer linear program that admit polynomial time solutions [26]. The proposed reassignment ensures that the new paths for N_d sensors are corrected simultaneously, the resulting paths have minimal interaction in the continuous space, and that the sensors collectively visits the same set of waypoints as if executing the original paths. We can also use a similar reassignment strategy to ensure collision avoidance among pairs of stations.

IV. THEORETICAL GUARANTEES

We now state various theoretical guarantees provided by the proposed approach and address Problem 2.

Proposition 1 (ANYTIME ALGORITHM). *At any epoch $p \in \mathbb{N}$, the sets $\mathcal{K}(p)$ and $\mathcal{R}(p)$ in Algorithm 1 satisfy $\mathcal{K}(p) \subseteq \mathcal{S}_{\theta-\epsilon}$ and $\mathcal{R}(p) \subseteq \mathcal{C} \setminus \mathcal{S}_{\theta+\epsilon}$, with probability of at least $1 - \delta$.*

We provide the proof of Proposition 1 in Appendix A. The key takeaway from Proposition 1 is that the keep set $\mathcal{K}(p)$ computed by Algorithm 1 always inner-approximates $\mathcal{S}_{\theta-\epsilon}$. Consequently, Algorithm 1 may be prematurely terminated if required, and the intermediate solution $\mathcal{K}(p)$ and $\mathcal{R}(p)$ will contain only interesting and uninteresting cells respectively (up to the tolerance ϵ), with high probability.

Proposition 2 (FINITE TIME GUARANTEES). *Algorithm 1 terminates within $P^{\max} \in \mathbb{N}$ epochs with probability of at least $1 - \delta$, and satisfies the labeling error criterion (2) where*

$$P^{\max} = \frac{1}{D} \sum_{l \in \mathcal{C} \setminus \mathcal{D}_\Delta} P_l + \max_{l \in \mathcal{D}_\Delta} P_l, \quad (12)$$

$$P_l = \frac{16}{B \Delta_l^2} \log \left(4 \sqrt{\frac{3|\mathcal{C}|}{\delta}} \log \left(\frac{192}{\Delta_l^2} \sqrt{\frac{3|\mathcal{C}|}{\delta}} \right) \right), \quad (13)$$

$$\Delta_l = |\mu_l - \theta| + \epsilon, \quad (14)$$

where P_l, Δ_l is defined for every $l \in \mathcal{C}$, and \mathcal{D}_Δ is the union of the cell with the smallest Δ_l among all cells $l \in \mathcal{C}$ and a set of $D - 1$ cells with the largest Δ_l among all cells $l \in \mathcal{C}$.

We provide the proof of Proposition 2 in Appendix B. By Proposition 2, Algorithm 1 terminates at some $p_{\text{term}} \leq P^{\max}$ epochs, and returns $\mathcal{K}(p_{\text{term}})$ that satisfies the labeling error criterion (2). Specifically, $\mathcal{K}(p_{\text{term}})$ outer-approximates $\mathcal{S}_{\theta+\epsilon}$ and inner-approximates $\mathcal{S}_{\theta-\epsilon}$ with high probability. Also, P^{\max} decreases (Algorithm 1 terminates faster) when Δ_l increases (μ_l is far away from θ), and the dependence of $|\mathcal{C}|$ and δ on P^{\max} is sub-linear.

Corollary 1. *Given an environment and a team, let K^{\max} be the maximum number of sensing cycles needed by Algorithm 2 to cover any set of epoch goals starting from any configuration of the team. Then, Algorithm 1 with Algorithm 2 as the low-level planner terminates while satisfying (2) in not more than $P^{\max} K^{\max}$ time steps with probability of at least $1 - \delta$.*

Corollary 1 bounds the time steps t required by a team that are deployed using Algorithms 1 and 2 to complete the search, and satisfy (2). Determining K^{\max} involves a min-max computation, and while its solution always exists and is finite, it may be hard to find. However, one can obtain reasonable estimates of K^{\max} via Monte-Carlo simulations.

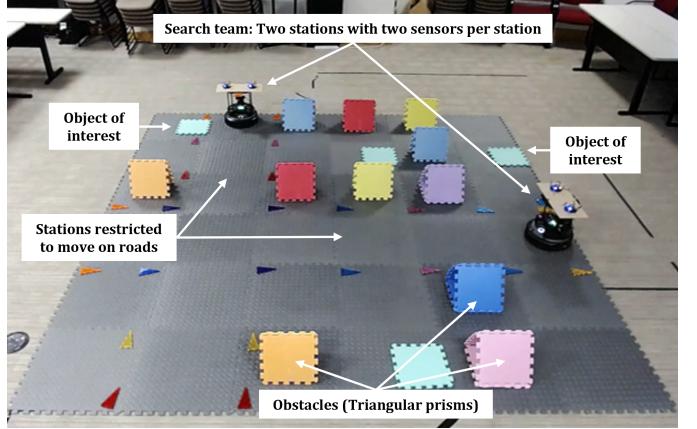


Fig. 3: Snapshot of the hardware experiment with two Turtlebot4 robots as stations and four Crazyflie drones as sensors. The colored triangular prisms represent the obstacles or no-fly zones for the sensors. The second column and the third row (delimited by small triangular tiles) represent the roads where the stations must stay. The four green tiles represent the interesting cells, i.e., they contain the search objective. All other cells are considered uninteresting. See video of the physical experiment at <https://youtu.be/gzulpOcVYzg>.

V. HARDWARE-BASED VALIDATION

We validate the proposed approach using a team comprised of four drones and two ground robots in a 3.6×3.6 meter workspace partitioned into a 6×6 grid environment representing a search-and-rescue application. Figure 3 shows the experiment setup, which includes 11 obstacles or no-fly zones (colored triangular prisms), 4 interesting regions (green tiles), 2 roads (delimited by triangular tiles), and 10 uninteresting regions (grey tiles). We excluded \mathcal{R} from \mathcal{C} for ease of implementation. We impose the motion constraints on the team with $T_d = 8$ and $T_c = 4$.

We use Crazyflie2.1 quadrotors [27] as the sensors. Each Crazyflie is equipped with one IR-reflective marker detected by an OptiTrack motion capture system running at 60 Hz. We use Turtlebot4 robots [28] as stations. Each Turtlebot is equipped with two IR-reflective markers and two wireless chargers. We use a combination of Crazyswarm2 [29] and custom ROS2 packages to track and control the motion of the robots over WiFi. All computations were performed in an Ubuntu 22.04 LTS workstation with an AMD Ryzen 9 5950X 16-core CPU and 128GB of RAM.

We use Algorithm 1 to deploy the team and solve Problem 1. During each epoch, the high-level multi-armed bandit planner identifies six epoch goals ($D = 6$). Then, the low-level planner coordinates the motion of the drones and ground stations to visit these cells as soon as possible. After the sensors visit all epoch goals and return to the stations for charging, we use the collected data to compute the next epoch goals as well as update the keep and reject sets, until the termination criterion in Algorithm 1 was met.

At each unclassified cell along a sensor's path, the sensor captures 30 images using its onboard camera (Crazyflie's AI deck) and transmits them to the central computer via WiFi.

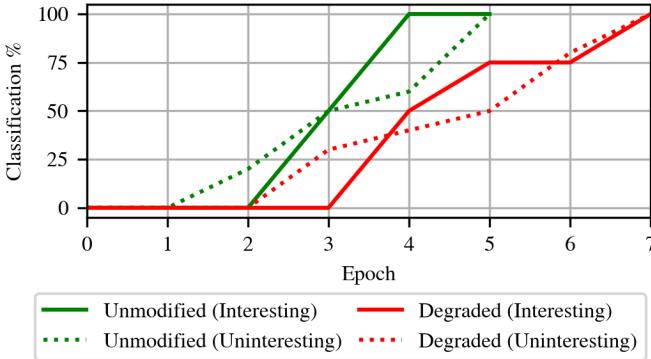


Fig. 4: Classification progress of Algorithm 1 in the hardware experiments with the unmodified and degraded sensors.

Using OpenCV [30], the computer counts the number of green pixels in each image to obtain noisy measurements of whether the visited cell is interesting or not. Thus, the sensor collects $B = 30$ realizations of ν_l at each visited cell $l \in \mathcal{C} \setminus (\mathcal{K} \cup \mathcal{R})$. We also performed an additional experiment with a *degraded sensor*, where we introduced an additional 5% classification noise to the measurements to mimic a poor-quality sensor.

Figure 4 shows the classification progress during the hardware experiments (the percentage of the classified cells) over the course of Algorithm 1 with the two sensors. For the unmodified sensor, Algorithm 1 classifies interesting cells faster than uninteresting cells despite the sparser spatial distribution of the interesting cells. For the degraded sensor, Algorithm 1 takes more epochs to complete the classification, as expected. However, even with the degraded sensors, Algorithm 1 correctly identifies most of the interesting cells at the start of epoch $p = 5$. Algorithm 2 required two sensing cycles for the team to visit the epoch goals for $p = 3$ with the unmodified sensor and $p = 5$ with the degraded sensor. In all other epochs, Algorithm 2 was able to find paths for the team to visit all epoch goals within a single sensing cycle.

VI. SIMULATION-BASED PERFORMANCE ANALYSIS

For a more extensive assessment of the performance of the proposed approach, we also perform a simulation-based analysis. First, we study the effect of sensor accuracy (variation of μ_l for a fixed threshold δ), and show that our approach is robust to noise in measured data. Second, we study the effect of varying the number of epoch goals D and demonstrate a trade-off between computation time as well as utilization of the sensors. Third, we study the effect of agility on the robot teams, and empirically show that more agile sensors typically lead to faster identification of interesting cells. Finally, we demonstrate that our approach scales reasonably for varying team sizes. To easily generate random scenarios, we relaxed the station admissible set constraints in this section.

To study the impact of various parameters on Algorithm 1, we report the results from $M \in \mathbb{N}$ Monte-Carlo simulations. Unless specified otherwise, we consider a search environment 10×10 grid \mathcal{G} with randomly chosen obstacle set \mathcal{O} with $|\mathcal{O}| = 16$, no station admissible set restriction with $\mathcal{R} = \mathcal{G} \setminus \mathcal{O}$, and 10

TABLE I: Median computation time needed to solve (11) at each epoch and the epochs needed for classification when varying the worst-case sensor accuracy $\mu_{l,\text{interest}}^{\text{worst}}$ (along with 0.1, 0.9 quantiles). The epochs needed decrease with increasing sensor accuracy.

Worst-case sensor accuracy $\mu_{l,\text{interest}}^{\text{worst}}$	Computation time (s) per epoch	Epochs needed to classify	
		All interesting cells	All cells
0.6	9.70 (4.83, 487.21)	74 (15, 141)	116 (79, 141)
0.8	11.37 (6.46, 387.38)	17 (12, 21)	26 (23, 30)
1.0	11.72 (6.18, 488.06)	9 (7, 12)	15 (12, 19)

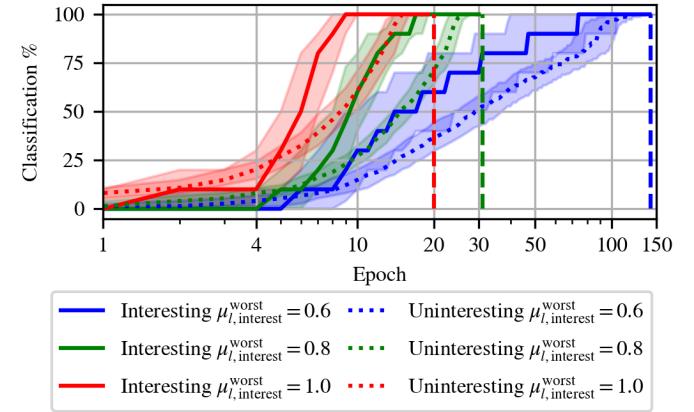


Fig. 5: Classification progress of Algorithm 1 when varying worst-case sensor accuracy $\mu_{l,\text{interest}}^{\text{worst}}$. Epoch (x-axis) is shown in log-scale. The epochs needed to classify all cells (vertical dashed lines) decreases with increasing sensor accuracy.

randomly chosen cells in $\mathcal{C} = \mathcal{G} \setminus \mathcal{O}$ chosen as interesting. We eliminated randomly generated scenarios where an interesting cell is rendered inaccessible by the realization of the obstacle locations. Additionally, we randomly choose μ_l , the sensor accuracy at each cell $l \in \mathcal{C}$, from a uniform distribution defined over the interval $[\mu_{l,\text{interest}}^{\text{worst}}, 1]$ for every interesting cell. Similarly, we drew μ_l from a uniform distribution defined over the interval $[0, 1 - \mu_{l,\text{interest}}^{\text{worst}}]$ for every uninteresting cell. Unless specified otherwise, we choose $\mu_{l,\text{interest}}^{\text{worst}} = 0.8$, and fixed the threshold $\theta = 0.5$ to define \mathcal{S}_θ . For the search team, we consider $N_d = 10$ sensors and $N_c = 5$ stations, choose the number of epoch goals generated $D = 8$ and the sample batch size $B = 10$, and impose move constraints on the sensors and stations within each epoch with $T_d = 8$ and $T_c = 4$.

1) *Sensor accuracy (varying the support of μ_l):* We study the effects of sensor accuracy by varying $\mu_{l,\text{interest}}^{\text{worst}} \in \{0.6, 0.8, 1.0\}$ with $M = 100$. Algorithm 1 is not aware of $\mu_{l,\text{interest}}^{\text{worst}}$ or μ_l for any $l \in \mathcal{C}$, but instead uses data to build the necessary confidence intervals to address Problem 1. Consequently, it has no initial benefit even when a perfect sensor is used ($\mu_{l,\text{interest}}^{\text{worst}} = 1.0$).

Figure 5 shows that a team with more accurate sensor (higher μ_l) completes the spatial classification problem faster. We observe similar trends in the epochs needed for classification in Table I. These observations are consistent with Proposition 2, since a higher $\mu_{l,\text{interest}}^{\text{worst}}$ corresponds to a higher Δ_l by construction, which in turn decreases P^{\max} , the upper

TABLE II: Median computation time needed to solve (11) at each epoch and the epochs needed for classification when varying the number of epoch goals D (along with 0.1, 0.9 quantiles). The computation time increases with increasing D , while the epochs needed decrease.

Number of epoch goals D	Computation time (s) per epoch	Epochs needed to classify	
		All interesting cells	All cells
5	9.18 (5.87, 3354.36)	19 (12, 25)	29 (25, 36)
8	11.37 (6.46, 387.38)	17 (12, 21)	26 (23, 30)
10	12.59 (5.83, 324.19)	15 (12, 21)	25 (23, 28)
15	15.71 (6.57, 594.75)	16 (12, 25)	23 (20, 28)

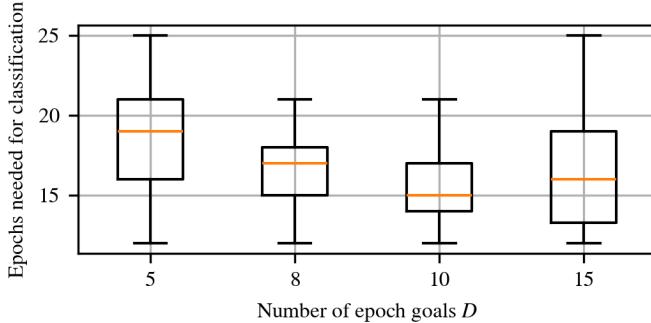


Fig. 6: Epochs needed to classify all interesting cells when varying the number of epoch goals D . The bandit-based high-level planner produces a trade-off between increasing D and the epochs needed to classify interesting cells.

bound on the termination time of Algorithm 1. Finally, we observe empirically that Algorithm 1 classifies most of the interesting cells before classifying all the uninteresting cells, which we attribute to the use of a bandit-based high-level planner in Algorithm 1.

2) *Number of epoch goals D :* Table II reports the computation time and the epochs needed for classification when varying D with $M = 50$. Recall that increasing D reduces the benefit of optimizing cell visits in the high-level planner. Table II shows that increasing D typically increases the burden on the low-level planner while decreasing the epochs needed for classification.

Figure 6 shows a trade-off between the epochs needed to identify all interesting cells and D . The trade-off demonstrates the value of using a bandit-based planner in Algorithm 1.

3) *Agility of the team:* Table III reports the computation time and the epochs needed for classification when varying T_d and T_c with $M = 50$. T_d and T_c constraint the motion of the team and arise from the energy limitations of the sensors as well as the gap between the agility of the sensors and stations. As expected, sensors and stations that are more agile (higher T_d and T_c) typically require lower computation time, possibly because the low-level planner has a simpler integer program to solve. Additionally, the epochs needed also decrease with more agile sensors since the team can now cover larger parts of the search environment at each epoch.

4) *Scalability:* Table IV reports the computation time and the epochs needed for classification when varying team sizes with $M = 50$. Increasing the number of sensors N_d from 5 to 14 almost halves the epochs needed, but nearly doubles the

TABLE III: Median computation time needed to solve (11) at each epoch and the epochs needed for classification when varying the motion constraints on the team T_d and T_c (along with 0.1, 0.9 quantiles). Sensors and stations with more agility (higher T_d , T_c) typically need fewer epochs.

Sensing cycle (T_d, T_c)	T_d/T_c	Computation time (s) per epoch		Epochs needed to classify	
		All interesting cells	All cells	All interesting cells	All cells
(8, 4)	2	11.38 (6.54, 389.24)	17 (12, 21)	26 (23, 30)	
(12, 4)	3	14.26 (8.97, 835.76)	14 (10, 23)	22 (17, 26)	
(16, 4)	4	19.41 (11.85, 243.15)	12.5 (8, 19)	19 (15, 23)	
(12, 6)	2	13.57 (8.82, 194.83)	15 (10, 20)	21 (19, 25)	
(18, 6)	3	20.45 (12.46, 65.20)	12 (8, 17)	17 (14, 21)	
(24, 6)	4	27.95 (15.73, 79.49)	10 (7, 15)	15 (12, 18)	
(16, 8)	2	18.54 (11.64, 50.58)	14 (8, 19)	18 (16, 23)	
(24, 8)	3	27.43 (16.23, 86.60)	10 (5, 14)	14 (12, 20)	
(32, 8)	4	35.56 (21.05, 85.38)	9 (5, 13)	13 (10, 17)	

TABLE IV: Median computation time needed to solve (11) at each epoch and the epochs needed for classification when varying the team sizes (along with 0.1, 0.9 quantiles). Larger teams need fewer epochs but need more compute time.

Team size (N_d, N_c)	Computation time (s) per epoch	Epochs needed to classify	
		All interesting cells	All cells
(5, 5)	6.17 (3.77, 192.55)	23 (17, 33)	41 (36, 46)
(7, 7)	8.07 (5.73, 140.56)	22 (13, 30)	36 (32, 42)
(10, 5)	11.37 (6.46, 387.38)	17 (12, 21)	26 (23, 30)
(10, 10)	12.74 (8.74, 525.27)	18 (13, 27)	30 (25, 36)
(14, 7)	15.73 (10.38, 705.02)	13 (10, 17)	21 (20, 29)

computation time per epoch. However, even for moderately sized teams, the 0.9-quantile computation time per epoch for Algorithm 1 is less than 10 minutes.

VII. CONCLUSION

We presented an iterative algorithm to address the spatial classification problem using constrained robots. We used a combination of multi-arm bandits and optimization-based planning to design the proposed data-driven algorithm. Using hardware and simulation experiments, we demonstrated the efficacy of our approach in a variety of settings. Our future work will focus on improving the high-level planner using constraints from the low-level planner.

REFERENCES

- [1] P. Thaker, S. Di Cairano, and A. Vinod, “Bandit-based multi-agent search under noisy observations,” *IFAC-PapersOnLine*, pp. 2780–2785, 2023.
- [2] D. Drew, “Multi-agent systems for search and rescue applications,” *Curr. Rob. Rep.*, vol. 2, pp. 189–200, 2021.
- [3] J. Queralta, J. Taipalmaa, B. Pullinen, V. Sarker, T. Gia, H. Tenhunen, M. Gabbouj, J. Raitoharju, and T. Westerlund, “Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision,” *IEEE Access*, vol. 8, pp. 191 617–191 643, 2020.
- [4] M. Dunbabin and L. Marques, “Robots for environmental monitoring: Significant advancements and applications,” *IEEE Rob. & Autom. Mag.*, vol. 19, no. 1, pp. 24–39, 2012.
- [5] S. Nayak, M. Greiff, A. Raghunathan, S. D. Cairano, and A. Vinod, “Data-driven monitoring with mobile sensors and charging stations using multi-arm bandits and coordinated motion planners,” in *Amer. Ctrl. Conf.*, 2024, pp. 299–305.
- [6] A. Krause and C. Guestrin, “Near-optimal observation selection using submodular functions,” in *AAAI*, vol. 7, 2007, pp. 1650–1654.
- [7] F. Bullo, J. Cortés, and S. Martinez, *Distributed control of robotic networks*. Princeton Univ. Press, 2009.

- [8] M. Schwager, M. Vitus, S. Powers, D. Rus, and C. Tomlin, “Robust adaptive coverage control for robotic sensor networks,” *IEEE Tran. Ctrl. Netw. Syst.*, vol. 4, no. 3, pp. 462–476, 2015.
- [9] W. Luo, C. Nam, G. Kantor, and K. Sycara, “Distributed environmental modeling and adaptive sampling for multi-robot sensor coverage,” in *Proc. Intl. Conf. Auto. Agents Multi-Agent Syst.*, 2019, pp. 1488–1496.
- [10] R. Rajcsy, Y. Aloimonos, and J. Tsotsos, “Revisiting active perception,” *A. Robots*, vol. 42, no. 2, pp. 177–196, 2018.
- [11] A. Kapoutsis, S. Chatzichristofis, and E. Kosmatopoulos, “DARP: Divide areas algorithm for optimal multi-robot coverage path planning,” *J. Intelli. Robotic Syst.*, vol. 86, no. 3, pp. 663–680, 2017.
- [12] G. Best, J. Faigl, and R. Fitch, “Online planning for multi-robot active perception with self-organising maps,” *A. Robots*, vol. 42, no. 4, pp. 715–738, 2018.
- [13] R. Marchant and F. Ramos, “Bayesian optimisation for intelligent environmental monitoring,” in *IEEE Int'l Conf. Intelli. Robots Syst.*, 2012, pp. 2242–2249.
- [14] R. Ghods, A. Banerjee, and J. Schneider, “Decentralized multi-agent active search for sparse signals,” in *Proc. Conf. Uncertainty Artif. Intelli.*, vol. 161, 2021, pp. 696–706.
- [15] T. Lattimore and C. Szepesvári, *Bandit algorithms*. Cambridge Univ. Press, 2020.
- [16] E. Rolf, D. Fridovich-Keil, M. Simchowitz, B. Recht, and C. Tomlin, “A successive-elimination approach to adaptive robotic source seeking,” *IEEE Tran. Robotics*, vol. 37, no. 1, pp. 34–47, 2021.
- [17] B. Du, K. Qian, H. Iqbal, C. Claudel, and D. Sun, “Multi-robot dynamical source seeking in unknown environments,” in *Intl Conf. Robotics and Autom.*, 2021, pp. 9036–9042.
- [18] A. Locatelli, M. Gutzeit, and A. Carpenter, “An optimal algorithm for the thresholding bandit problem,” in *Intl. Conf. Machine Learning*, 2016, pp. 1690–1698.
- [19] K.-S. Jun, K. Jamieson, R. Nowak, and X. Zhu, “Top arm identification in multi-armed bandits with batch arm pulls,” in *Proc. Intl Conf. Artifi. Intelli. Stats.*, vol. 51, 2016, pp. 139–148.
- [20] B. Mason, L. Jain, A. Tripathy, and R. Nowak, “Finding all ϵ -good arms in stochastic bandits,” *Adv. Neural Info. Process. Syst.*, 2020.
- [21] T. Kundu and I. Saha, “Mobile recharger path planning and recharge scheduling in a multi-robot environment,” in *IEEE Int'l Conf. Intell. Rob. Syst.*, 2021, pp. 3635–3642.
- [22] X. Lin, Y. Yazıcıoğlu, and D. Aksaray, “Robust planning for persistent surveillance with energy-constrained uavs and mobile charging stations,” *IEEE Rob. Autom. Lett.*, vol. 7, no. 2, pp. 4157–4164, 2022.
- [23] K. Yu, A. K. Budhiraja, S. Buebel, and P. Tokekar, “Algorithms and experiments on routing of unmanned aerial vehicles with mobile recharging stations,” *J. Field Rob.*, vol. 36, no. 3, 2018.
- [24] Gurobi Opt., LLC, “Gurobi Optimizer Reference Manual,” <https://www.gurobi.com> (Last accessed: 2025).
- [25] N. Michael, M. Zavlanos, V. Kumar, and G. Pappas, “Distributed multi-robot task assignment and formation control,” in *Int'l Conf. Rob. Autom.*, 2008, pp. 128–133.
- [26] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*. USA: Society for Industrial and Applied Mathematics, 2009.
- [27] Bitcraze, “Crazyflie 2.1,” <https://www.bitcraze.io/products/crazyflie-2-1> (Last accessed: 2025).
- [28] Clearpath Robotics, “Turtlebot 4,” <https://clearpathrobotics.com/turtlebot-4/> (Last accessed: 2025).
- [29] W. Honig, K. McGuire *et al.*, “Crazyswarm2,” <https://github.com/IMRCLab/crazyswarm2> (Last accessed: 2025).
- [30] G. Bradski, “The OpenCV Library,” *Dr. Dobb's J. Soft. Tools*, 2000.

APPENDIX

A. Proof of Proposition 1

Consider two events — $\mathcal{E}_{\mathcal{K}}(p) = \{\mathcal{K}(p) \setminus \mathcal{S}_{\theta-\epsilon} \neq \emptyset\}$ and $\mathcal{E}_{\mathcal{R}}(p) = \{\mathcal{R}(p) \cap \mathcal{S}_{\theta+\epsilon} \neq \emptyset\}$. Specifically, at any epoch p , $\mathcal{E}_{\mathcal{K}}(p)$ and $\mathcal{E}_{\mathcal{R}}(p)$ are the undesirable events that a cell in $\mathcal{C} \setminus \mathcal{S}_{\theta-\epsilon}$ is included in the keep set $\mathcal{K}(p)$ and a cell in $\mathcal{S}_{\theta+\epsilon}$ is included in the reject set $\mathcal{R}(p)$ respectively. We need to show that $\mathbb{P}\left[\left(\bigcup_{p \geq 1} \mathcal{E}_{\mathcal{K}}(p)\right) \cup \left(\bigcup_{p \geq 1} \mathcal{E}_{\mathcal{R}}(p)\right)\right] \leq \delta$.

By Boole’s inequality, it is suffices to show that $\mathbb{P}\left[\bigcup_{p \geq 1} \mathcal{E}_{\mathcal{K}}(p)\right] \leq \delta/2$ and $\mathbb{P}\left[\bigcup_{p \geq 1} \mathcal{E}_{\mathcal{R}}(p)\right] \leq \delta/2$. Observe

that $U_l(p, \delta)$ in (3b) is the upper confidence bound in [19, Eq. (2)] with δ replaced with $\delta/(2|\mathcal{C}|)$. By (3b) and [19, Lem. 1],

$$\mathbb{P}\left[\bigcap_{p \geq 1} \{|\mu_l - \hat{\mu}_l| \leq U_l(p, \delta)\}\right] \geq 1 - \frac{\delta}{2|\mathcal{C}|}, \quad (15)$$

for any cell $l \in \mathcal{C}$. From the definition of $\mathcal{E}_{\mathcal{K}}(p)$, the event $\mathcal{E}_{\mathcal{K}}(p)$ occurs at some $p \geq 1$ if and only there is some cell $l \in \mathcal{C}$ such that $\mu_l < \theta - \epsilon \leq \hat{\mu}_l(p) - U_l(p, \delta)$ at p . Using (15) and Boole’s inequality,

$$\begin{aligned} \mathbb{P}\left[\bigcup_{p \geq 1} \mathcal{E}_{\mathcal{K}}(p)\right] &\leq \mathbb{P}\left[\bigcup_{l \in \mathcal{C}} \bigcup_{p \geq 1} \{|\hat{\mu}_l(p) - \mu_l| \geq U_l(p, \delta)\}\right] \\ &\leq \sum_{l \in \mathcal{C}} \frac{\delta}{2|\mathcal{C}|} \leq \frac{\delta}{2}. \end{aligned} \quad (16)$$

The proof for $\mathbb{P}\left[\bigcup_{p \geq 1} \mathcal{E}_{\mathcal{R}}(p)\right] \leq \delta/2$ follows similarly. ■

B. Proof of Proposition 2

We simplify the analysis by considering the data collected only at \mathcal{E}_p at each epoch p and ignore the effect of data collected along the way on the classification. Due to the monotonicity of U_l and $|\mathcal{H}_l|$, such an analysis holds even when Algorithm 1 uses data collected along the way. Thus, we analyze Algorithm 1 by studying the corresponding bandit problem $(\mathcal{C}, \{\mu_l\}_{l \in \mathcal{C}}, \theta, \delta, \epsilon)$ [19], and ignore the effect of the sensor paths on the classification.

Claim 1: For every $l \in \mathcal{C}$, consider any epoch $p_l \in \mathbb{N}$ such that $U_l(p_l, \delta) \leq \Delta_l/2$. Then, with probability $1 - \delta$, l is assigned either to \mathcal{K} or \mathcal{R} by epoch p_l .

Proof of Claim 1: From (4), l is assigned to either \mathcal{K} or \mathcal{R} by epoch p if $|\hat{\mu}_l(p) - \theta| \geq U_l(p, \delta) - \epsilon$. For any $l \in \mathcal{C}$ and for all $p \geq 1$, we use triangle inequality, the fact that $|x| \geq \pm x$ for all $x \in \mathbb{R}$, (14), and (16) to obtain a lower bound of $|\hat{\mu}_l(p) - \theta|$ with probability $1 - \delta$, i.e., $|\hat{\mu}_l(p) - \theta| \geq |\hat{\mu}_l(p) - \mu_l| - |\mu_l - \theta| \geq |\mu_l - \theta| - |\hat{\mu}_l(p) - \mu_l| \geq (\Delta_l - \epsilon) - U_l(p, \delta)$. Since $U_l(p_l, \delta) \leq \Delta_l/2$, $|\hat{\mu}_l(p_l) - \theta| \geq U_l(p_l, \delta) - \epsilon$, which completes the proof. □

We define $P_l \in \mathbb{N}$ as the number of times Algorithm 1 directs the team to visit cell l . For every cell l and any epoch p_l that satisfies Claim 1, $|\mathcal{H}_l(p_l)| = P_l B$ for sample batch size B by Step 3 of Algorithm 1. We obtain P_l in (13) using $U_l(p_l, \delta) \leq \Delta_l/2$ and [19, Eq. (6)].

Claim 2: With probability $1 - \delta$, Algorithm 1 terminates by $P^{\max} = \frac{1}{D} \sum_{l \in \mathcal{C} \setminus \mathcal{D}_{\Delta}} P_l + \max_{l \in \mathcal{D}_{\Delta}} P_l$ epochs.

Proof of Claim 2: We split the progress of Algorithm 1 into two phases — Phase 1, the initial phase, where all epochs has more than D unclassified cells, and otherwise as Phase 2. Let $\mathcal{D} \subset \mathcal{C}$, $|\mathcal{D}| = D$ denote the set of unclassified cells at the start of Phase 2. Then, using the definition of P_l , Algorithm 1 terminates within P^{\max} epochs with probability $1 - \delta$,

$$P^{\max}(\mathcal{D}) = \frac{1}{D} \sum_{l \in \mathcal{C} \setminus \mathcal{D}} P_l + \max_{l \in \mathcal{D}} P_l. \quad (17)$$

Here, (17) uses the observations that the low-level planner coordinates the team to visit each epoch goal at least once, Phase 1 involves at most $\sum_{l \in \mathcal{C} \setminus \mathcal{D}} P_l$ visits with $|\mathcal{E}_p| = D$, and Phase 2 has $\mathcal{E}_p = \mathcal{C} \setminus (\mathcal{K} \cup \mathcal{R})$ with $|\mathcal{E}_p| \leq D$. We observe that \mathcal{D}_{Δ} maximizes (17), which completes the proof. □

By Proposition 1, Algorithm 1 satisfies the labeling criterion (2) whenever it terminates, which completes the proof. ■