Assessing Reputation to Improve Team Performance in Heterogeneous Multi-Robot Coverage

Mela Coffey and Alyssa Pierson

Abstract— **When agents in a multi-robot team have limited knowledge about their relative performance, their teammates, or the environment, robots must observe individual performance variations and adapt accordingly. We propose *robot reputation* to assess the historical performance of agents and make future adaptations in a persistent coverage task. We consider a heterogeneous multi-robot team, where robots are equipped with different capabilities to serve discrete events in an environment. We utilize a heterogeneous coverage control approach to partition the space according to robot capabilities and the estimated probability density, such that the robot is responsible for serving the events in its assigned region. As the team serves events, we assign each robot a reputation, which is then used to adjust the size of a robot's region, thus adjusting the amount of space a robot is responsible for serving. Our simulations show that using reputation to weigh the size of the Voronoi cells outperforms the case where we neglect reputation.**

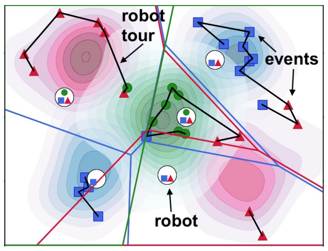
I. Introduction

To make heterogeneous multi-robot teams more robust in deployment, we cannot assume the agents all have equal performance in sensing, actuation, or communication. Suppose we deploy a multi-robot team to visit specific points in an environment and take various measurements or collect samples. It is very possible a ground robot tasked with collecting samples is slowed by mud, or the camera of a drone becomes faulty. Conversely, some robots may perform better than expected, either with variations in design or wear over time. Accounting for these individual variations will improve the overall team performance.

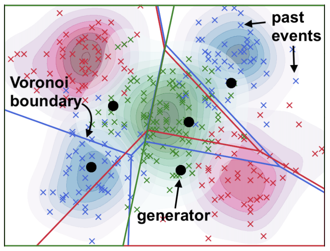
In this paper, we address the problem of observing and adapting to these performance variations by assigning each robot a *reputation*, which agents can use to modify their tasks or behavior. Specifically, we consider a heterogeneous multi-robot team in which each agent is equipped with one or more capabilities. The team is tasked with meeting discrete events in the environment, with each event requiring one or more of the robot capabilities, by traveling to those events. We utilize a weighted heterogeneous Voronoi-based coverage control approach to assign regions of the environment to each robot, such that each robot is responsible for serving events in their assigned cells. The probability of event occurrences is estimated over time, and the assigned robot regions adapt accordingly. As robots serve events, they are assigned a reputation based on their ability to serve events, and their

Mela Coffey and Alyssa Pierson are with the Department of Mechanical Engineering at Boston University, Boston, MA 02115, USA. mcoffey@bu.edu, pierson@bu.edu

This work supported in part by NSF CAREER Award 2235622 and a Boston University startup grant. Their support is gratefully acknowledged.



(a) Robots serving events



(b) Event history and generators

Fig. 1. Our simulations present 5 robots (white discs), each equipped with up to 3 capabilities (green circle, blue square, or red triangle), and tasked with serving scattered events requiring those capabilities. (a) shows the circle, square, and triangle events in the space that need to be served, and the black lines indicate the planned robot paths to serve these events. (b) shows the same instance as (a). The black dots are the generators that make up the weighted Voronoi cells. The green, blue, and red $\times$'s are current and past events used to estimate the density function corresponding to the circle, square, and triangle capabilities, respectively. The contours in both (a) and (b) indicate the estimated probability that an event will occur given those current and past events.

cells are weighted according to their reputation. Intuitively, a robot that has a reputation of performing better than its neighbors will increase its cell size to cover more area and compensate for lack of performance in its neighbors. Our simulations show that modifying cell size based on relative robot reputation allows robots to serve more events than an approach that does not consider reputation.

The contributions of this paper are as follows:

1) A framework for a multi-robot team to meet discrete, heterogeneous events that assigns each robot a *reputation* as robots serve demand, which is then used to create weighted Voronoi partitions;
2) Proof of convergence of the Voronoi generators to critical points of the weighted locational cost;
3) A Kernel Density Estimation (KDE) to continually update the density function as events appear; and
4) Simulations to show that our approach outperforms that which neglects robot reputation.

Related Work

Many studies have explored adaptation to changes in the team or the environment in multi-robot coverage. Prior works invoke weighted Voronoi partitions to accommodate sensing radius [1], power efficiency [2], and overall sensing health [3] by adapting the size of each robot cell accordingly. The author's prior work [4] uses adaptive weights to account for relative sensing and actuation performance. The authors of [5] compute equitable partitions so that each robot has roughly the same amount of "work" to cover before deploying robots to sweep the space.

Other works address the problem of balancing expected and desired behavior. In [6], robots lose their calibration and ability to function as anticipated, and they must adapt to those individual differences. In [7], robots adapt to individual performance changes in task allocation by tracking the "quality" of a robot performing a task. The authors of [8] consider a heterogeneous robot team where robots assist one another in completing tasks that the other cannot.

Some studies consider the situation in which robot information is unknown. In [9], robot performance is unknown, and the proposed algorithm assigns specialization to agents by differentiating between simulated and actual costs of the agent. In [10], the capability of each robot is originally unknown in a task allocation problem, and each agent calculates its capability based on the tasks it completes, then broadcasts this information to its neighbors. While many task allocation problems focus on centralized or auction-based solutions [11], [12] and work under uncertainty [13] and communication constraints [14], our Voronoi tessellation determines the task assignment.

While prior work considers performance variations, a main contribution of this paper is the introduction of reputation. Unlike the author's prior work assessing instantaneous variations in sensing or actuation performance [4], we calculate reputation by observing historical actions of agents. Intuitively, reputation decreases when robots fail to meet assigned tasks, and increases when they complete all objectives. We use reputation as a tool to improve overall performance and serve more events. By observing and tracking each individual robot's reputation, we can further get a sense of heterogeneity in the team, and adjust to reputation differences online to achieve an improved performance.

The remainder of this paper is organized as follows: Section II formulates the problem and provides necessary technical background. Section III details our definition of reputation and outlines our control policy. In Section IV, we describe our density estimation methods. We provide our overall algorithm in Section V and present simulations in Section VI. Finally, we state our conclusions in Section VII.

II. Problem Formulation and Background

We instantiate our problem within a Voronoi-based coverage control framework, which promotes equitable coverage of an evolving demand, while generating new events for robots to serve. Voronoi-based coverage control allows us to assign regions of Q to each robot based on the demand $\mathcal{D}_j$, the robot capabilities, and the robot performance. It also allows us to naturally adapt to changes in the environment, in this case, changes in event appearance probability. Each robot moves within its own cell(s), following a route (tour)[1] to serve the events that appear within its own cell(s), while Voronoi generators maintain the partitions. This section provides a brief introduction to the heterogeneous coverage problem, including generating and maintaining robot cells.

We consider a heterogeneous team of n robots, $i \in \{1, \cdots, n\}$. Each robot has at least one of m capabilities, $j \in \{1, \cdots, m\}$, and we denote the set of capabilities of robot i by $\mathcal{C}_i$. We consider a convex environment $Q \in \mathbb{R}^2$, with points in Q denoted q, $q \in Q$. The positions of the robots are denoted $x = \{x_1, \cdots, x_n\}$, $x_i \in Q$. The robots are tasked with serving discrete events (demand) in Q, which require service from one of the m capabilities. We denote the set of events requiring capability j by $\mathcal{D}_j$, $\mathcal{D}_j \subset Q$.

Traditional Voronoi-based coverage control [15] deploys a homogeneous team of robots to cover an environment given some static density function $\phi(q) : \mathbb{R}^2 \to \mathbb{R}$ using Lloyd's algorithm [16]. To account for heterogeneity within the team, specifically the various capabilities of the robots, we invoke a heterogeneous extension of traditional Voronoi-based coverage control, introduced in [17], [18]. We provide this technical background here, which we build upon in Section III. We define one Voronoi partition for each capability j, and assign robot i to partition j if robot i has capability j. Thus, we have m Voronoi partitions. Formally, we define the set of robots assigned to partition j as $\mathcal{P}^j = \{i \mid j \in \mathcal{C}_i\}$. Then, we define the Voronoi cell of robot i in partition j by $V_i^j = \{q \in Q \mid \|q - g_i\| \leq \|q - g_k\|, \forall k \in \mathcal{P}^j, \forall k \neq i\}$, where g_i is the generator [19] associated with robot i[2]. The probability that an event at point q will require capability j is represented by the density function $\phi_j(q)$. Then each robot i can compute its mass and centroid associated with partition j by $M_i^j = \int_{V_i^j} \phi_j(q) dq$ and $C_i^j = \frac{1}{M_i^j} \int_{V_i^j} q\phi_j(q) dq$, respectively. The robots use Lloyd's algorithm to minimize the locational cost function of the generators: $\mathcal{H}(g) = \sum_{i=1}^n \sum_{j=1}^m \int_{V_i^j} f(\|q - g_i\|)\phi_j(q) dq$, where g is the set of generator positions, $g = \{g_1, \cdots, g_n\}$, $g_i \in Q$, and $f : \mathbb{R} \to \mathbb{R}$. By letting $f(\|q - g_i\|) = \|q - g_i\|^2$, then the generators follow the dynamics

$$\dot{g}_i = \kappa \sum_{j \in \mathcal{C}_i} M_i^j (C_i^j - g_i) \tag{1}$$

to minimize $\mathcal{H}(p)$, where κ is a positive constant.

Recall that we also wish to adjust the size of the robot cells using weighted Voronoi diagrams, ultimately adjusting the amount of space a robot i covers. To accomplish this, we propose the use of weighted Voronoi partitions [19]. When using weighted Voronoi diagrams for coverage control involving only one Voronoi partition, each robot i is assigned a weight w_i that adjusts the size of its cell. Intuitively, a larger weight w_i results in a larger cell size. Then, the environment is partitioned by $W_i = \{q \in Q \mid \|q - g_i\| - w_i \leq \|q - g_k\| - w_k, \forall k \neq i\}$. We can use this partition when computing the mass and centroid of a cell. We extend this weighted Voronoi control policy in Section III to account for the heterogeneous robot capabilities.

[1]Note the generation of robot tours is not a main contribution of this paper. We use a traveling salesman problem (TSP) in our simulations to generate tours between events.

[2]Typically, the robot position serves as the generator for the Voronoi partition (i.e., $g_i = x_i$) as in [18], [20]. In this paper, however, robots move within their own cell to meet discrete events by generating a tour between the events, and we utilize generators to maintain the Voronoi configuration.

III. Reputation to Improve Performance

In this work, each robot is responsible for serving the events that appear within their weighted Voronoi cell(s). As robots serve events, we assign each robot a *reputation* based on their ability to meet events. We use this reputation to adjust the weights of each robot cell, ultimately adjusting the amount of area a robot covers. In this section, we first define the robot reputation, then introduce a weighted Voronoi-based coverage control algorithm for heterogeneous teams.

A. Defining robot reputation

While prior works accommodate instantaneous variations or differences in physical characteristics, we propose reputation, which one can generally think of as a history of performance. By tracking reputation, we can more readily observe variations and heterogeneity, and even differentiate between types of robots. For example, in a team of drones and ground robots, we may assume agents with a history of serving demand significantly faster than others are drones. Definition 1 formalizes our idea of reputation.

Definition 1 (Reputation)**.** *The reputation of robot i associated with a task j is a history of a robot i's ability to perform and/or quality of performing a task j.*

Because reputation considers performance history, we first define the performance of a robot within the scope of serving demand. Since our robots are tasked with serving demand requiring various capabilities, we compute the performance of robot i associated with each of its capabilities $j \in \mathcal{C}_i$ every T units of time. In this work, we define the performance p_i^j of robot i for capability j within a time window T_k as the ratio of the number of events served by robot i of type j to the total number of events of type j that appear in the robot cell. Formally, the performance p_i^j within a time window T_k, $k = 1, \cdots, \infty$, is given by

$$p_i^j(T_k) = \frac{n_{i,\text{served}}^j(T_k)}{n_i^j(T_k)}, \tag{2}$$

where $n_{i,\text{served}}^j(T_k)$ is the number of events served by robot i with capability j within time window T_k, and $n_i^j(T_k)$ is the total number of events that appear in the robot cell V_i^j within time window T_k. Note if no event occurs in cell V_i^j in time window T_k (i.e., $n_i^j(T_k) = 0$), then robot i is not assigned a performance for that time window.

We then define the *reputation memory* $\mathcal{M}_i^j$ for robot i for each of its capabilities j as the set of the N most recent performance metrics. Thus, we can think of the reputation memory $\mathcal{M}_i^j$ as a short-term performance memory, where we only remember the last N time windows where $n_i^j(T_k) > 0$. Note $\mathcal{M}_i^j$ is initially empty. Thus, for early iterations of deployment, specifically when $k < N$ and/or $n_i^j(T_k) = 0$ for small k, then the length of $\mathcal{M}_i^j$ will be less than N.

We define the reputation r_i^j as the average of the reputation memory $\mathcal{M}_i^j$. More formally, we define the reputation as

$$r_i^j = \frac{\epsilon}{|\mathcal{M}_i^j|} \sum \mathcal{M}_i^j, \tag{3}$$

where ϵ is a positive constant adjusted to the size of the environment Q, and $|\mathcal{M}_i^j|$ denotes the length of $\mathcal{M}_i^j$. Intuitively, as a robot i serves more events requiring capability j, its reputation associated with capability j will increase.

B. Weighing Voronoi cells based on reputation

We now wish to use the robot reputation to modify the amount of area a robot covers. Intuitively, as the robot reputation increases, the amount of area it covers should increase, and as reputation decreases, coverage area should decrease. We use weighted Voronoi diagrams to adjust the amount of area a robot covers, and we use the robot reputation (3) to determine the cell weights. Recall we use generators g to maintain the Voronoi partitions while robots follow tours within their cell(s) to serve events. Therefore, we use the generators to construct the weighted Voronoi partition.

When we have one Voronoi partition, we can simply use the partitioning W_i introduced in Section II. However, because we have m partitions, and we wish to account for each robot i's performance with capability j, then we assign one weight w_i^j for each robot i corresponding to its reputation using capability j. We thus partition the space by:

$$\begin{aligned} W_i^j = \{q \in Q \mid \|q - g_i\| - w_i^j \leq \|q - g_k\| - w_k^j, \\ \forall k \in \mathcal{P}^j, \forall k \neq i\}. \end{aligned} \tag{4}$$

We then must modify our control policy to instead utilize the weighted Voronoi partition. The mass and centroid are computed from the weighted partition by:

$$M_i^j = \int_{W_i^j} \phi_j(q)dq, \qquad C_i^j = \frac{1}{M_i^j} \int_{W_i^j} q\phi_j(q)dq. \tag{5}$$

The generators move according to the control policy (1) to minimize the locational cost

$$\mathcal{H}_W(g) = \sum_{i=1}^{n} \sum_{j=1}^{m} \int_{W_i^j} \frac{1}{2} \left(\|q - g_i\|^2 - w_i^j\right) \phi_j(q)dq \tag{6}$$

using Lloyd's algorithm.

Our goal is to adjust the size of a robot's cell based on its ability to serve events requiring capability j over the last N time windows (i.e., its reputation). If a robot serves more events than its neighbors over the last N time windows, then its coverage area should increase. Similarly, robots who do not serve as many events as its neighbors should decrease in cell size. As such, we want the relative weights to converge to the relative reputations:

$$(w_i^j - w_k^j) \to (r_i^j - r_k^j), \forall i, j, k. \tag{7}$$

It was shown in the author's prior work [4, Theorem 1] that the weightings satisfy (7) under a single partition when the weights adhere to the adaptation law

$$\dot{w}_i^j = \frac{-\alpha\kappa}{2M_i^j} \sum_{k \in \mathcal{N}_i^j} \left[(w_i^j - r_k^j) - (w_k^j - r_k^j)\right] d_{ik}^j, \tag{8}$$

where κ is the proportional gain introduced in (1), $\mathcal{N}_i^j$ is the set of neighbors of robot i in partition j, d_{ik}^j is the length of the Voronoi boundary shared by robots i and k

in partition j, and M_i^j is given by (5). Note because we adjust the reputation r_i^j frequently, the weights w_i^j may not have time to converge. Note also the control policy for the generators is distributed, as the robots need only track the reputation of themselves and their neighbors.

We now show in Theorem 1 that under the control policy (1), our algorithm allows generators to converge to critical points of the weighted locational cost function (6).

Theorem 1. *Given n generators following the control policy* (1) *in a bounded, convex environment Q partitioned by* (4)*, the generators converge to critical points of the locational cost function* (6).

Proof. To prove that the generators converge to critical points of the locational cost, we use LaSalle's invariance principle [21]. First, consider the weighted locational cost (6) as a candidate function to prove convergence. We can compute the time derivative to be

$$\begin{aligned}\frac{d\mathcal{H}_W(g)}{dt} =& \sum_{i=1}^{n}\sum_{j=1}^{m}\int_{W_i^j}(q-g_i)^{\mathsf{T}}\phi_j(q)dq\dot{g}_i \\ &-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}\dot{w}_i^j\int_{W_i^j}\phi_j(q)dq.\end{aligned} \tag{9}$$

Letting the generators follow dynamics (1), and given that $-\int_{W_i^j}(q-g_i)^{\mathsf{T}}\phi_j(q)dq = -M_i^j(C_i^j - g_i)$ [22], then the first term in (9) becomes $\frac{\partial\mathcal{H}_W}{\partial g}\dot{g} = -\sum_{i=1}^{n}\sum_{j=1}^{m}\kappa(M_i^j)^2\|C_i^j - g_i\|^2$, implying that $\frac{\partial\mathcal{H}_W}{\partial g}\dot{g} \leq 0$. Then, using the weightings adaptation law (8), the second term becomes $\frac{\partial\mathcal{H}_W}{\partial w}\dot{w} = \sum_{i=1}^{n}\sum_{j=1}^{m}\frac{\alpha\kappa}{4}\sum_{k\in\mathcal{N}_i^j}\left[(w_i^j - r_i^k) - (w_k^j - r_k^j)\right]d_{ik}^j$. Since we sum over all neighbors and all robots, the weights and reputation values cancel in each partition j, and we can conclude that $\frac{\partial\mathcal{H}_W}{\partial w}\dot{w} = 0$. Therefore, $\frac{d\mathcal{H}_W(g)}{dt} \leq 0$.

By LaSalle's invariance principle, the generators g will converge to the largest invariant set contained within the set of all points such that $\frac{d\mathcal{H}_W(g)}{dt} = 0$, i.e. the generators will converge to critical points of $\mathcal{H}_W(g)$, proving the theorem. □

We showed in Theorem 1 that the generators converge to critical points of the weighted locational cost (6). Note while it is indeed possible for generators to converge to saddle points when a robot belongs to more than one partition, in our simulations, we observe that robots converge to one of their centroids. Furthermore, when a generator i converges to a critical point of the locational cost, the corresponding robot i is set up to optimally cover demand as predicted by the density function $\phi_j(q)$.

IV. Estimating Event Probability

Recall the density function $\phi_j(q)$ represents the probability that an event requiring capability j will appear at any point q. Rather than assuming robots have knowledge of event density functions, we estimate the density functions over time based on current and past events. This allows robots to adapt to evolving, stochastic demand while optimally positioning themselves to serve events. In this section, we enable robots to estimate the density function $\phi_j(q)$ as events occur.

Several previous works focus on learning density functions. In [23], each robot estimates the density using basis functions and local information. In [24] robots use Mixture of Gaussian Processes to learn the density online by continuously taking observations. The authors of [25] enable robots to learn the density through exploration and sample collection. In [26], robots balance learning the density function and moving to their centroid. These works all focus on learning the density through measuring samples, such as temperature to build a heat map. In our work, robots need not collect measurements; rather, we learn the density through discrete events (i.e. whether an event occurs at a point q). We therefore turn to kernel density estimation (KDE) [27] to estimate the probability that events will occur in Q.

Similar to keeping a reputation memory of robot performance, in our implementation of KDE, we keep a memory of event occurrences $\mathcal{D}_j$. By tracking only the last n_j events, where n_j is the number of elements in $\mathcal{D}_j$, we assume only "new" events contribute to the overall probability of an event, and any event happening before the last n_j events is outdated.

Given the set of events $\mathcal{D}_j$, we can estimate the density using KDE. The kernel density estimator creates one kernel K for each event location $\mu_l^j \in \mathcal{D}_j \subset Q$, sums over each of the kernels, and smooths the final resulting sum. Similar to [24], [25], [26], we assume a 2D Gaussian kernel

$$K(q,\mu;h) = \frac{1}{2\pi h^2}\exp\left[-\frac{\|q-\mu\|^2}{2h^2}\right], \tag{10}$$

where μ is the location of the Gaussian peak center and h is the bandwidth, also called the window width or smoothing parameter [27]. Given the set of events requiring capability j, $\mathcal{D}_j$, we can estimate the density function ϕ_j by

$$\phi_{j,\text{est}}(q) = \frac{1}{n_j h}\sum_{l=1}^{n_j}K(q,\mu_l^j;h), \tag{11}$$

where K is the kernel function (10) that satisfies the condition $\int_Q K(q,\mu;h)dq = 1$.

Finally, we note the estimated density (11) can be computed in a distributed fashion. We assume each robot can subscribe to the events corresponding to their capabilities (for example, a user can submit requests through an app, and the user location and capability request are streamed to the robots with that capability), then each robot can keep track of $\mathcal{D}_j$, and estimate the corresponding density $\phi_{j,\text{est}}$.

V. Overall Algorithm

In this section, we detail our overall algorithm, outlined in Algorithm 1, for adapting to robot reputation in covering discrete events. In the following paragraphs, we detail each section of our algorithm.

1) Deploy to centroids (lines 2-5): We begin our algorithm by first deploying robots using Voronoi-based coverage control, until robots converge to critical points of (6). The robots use a heterogeneous version of Lloyd's algorithm,

Algorithm 1 Covering Discrete Events

```
1: initialize $t = t_0$, $p_i(t_0)$, $\dot{p}_i(t_0) = [1, 1]^\mathsf{T}$, $g_i(t_0)$, $\mathcal{C}_i$, $T$, $\mathcal{D}_j = \varnothing$, $\mathcal{M}_i^j = \varnothing$, $\mathcal{T}_i = \varnothing$ $n_{i,\text{served}}^j = 0$, $n_i^j = 0$, $w_i^j = r_i^j = \epsilon\ \forall i, j$
2: while $\|\dot{p}_i\| > 0\ \forall i$ do    ▷ deploy robots to CVT
3:     $\dot{g}_i \leftarrow getGeneratorDynamics(g)$
4:     $u_i \leftarrow \dot{g}_i$
5: end while
6: while true do
7:     if new event(s) then
8:         update event memory $\mathcal{D}_j$ for each $j \in \mathcal{C}_i$
9:         $n_i^j \leftarrow n_i^j + 1$ for each new $\mu_l^j \in W_i^j$
10:    end if
11:    update tour $\mathcal{T}_i$ and number of events in cell $n_i^j$
12:    if $\mathrm{mod}(\frac{t}{T}) = 0$ then    ▷ every $T$ seconds
13:        $r_i^j \leftarrow (3)$    ▷ compute reputation
14:        $n_{i,\text{served}}^j \leftarrow 0$, $n_i^j \leftarrow 0$    ▷ reset performance
15:    end if
16:    $\dot{g}_i \leftarrow getGeneratorDynamics(g, \mathcal{D}_j, r_i^j)$
17:    $g_i \leftarrow g_i + \dot{g}_i \Delta t$    ▷ move generators
18:    if $\mathcal{T}_i \neq \varnothing$ then
19:        $d_{i,\text{next}} \leftarrow \mathcal{T}_i(0)$
20:        $u_i \leftarrow \frac{d_{gi,\text{next}} - p_i}{\|d_{i,\text{next}} - p_i\|} v_{i,\max}$    ▷ move to next event
21:        if $\|d_{i,\text{next}} - p_i\| < \lambda$ then    ▷ robot at event
22:            $n_{i,\text{served}}^j \leftarrow n_{i,\text{served}}^j + 1$ for $j \mid d_{i,\text{next}} \in \mathcal{D}_j$
23:            remove $d_{i,\text{next}}$ from $\mathcal{T}_i$
24:        end if
25:    else
26:        $u_i \leftarrow g_i - p_i$    ▷ move to generator
27:    end if
28:    $t \leftarrow t + \Delta t$    ▷ increment time
29: end while
```

Algorithm 2 $getGeneratorDynamics(g, \mathcal{D}_j, r_i^j)$

```
1: for each $j \in \mathcal{C}_i$ do
2:     Estimate the density $\phi_j(q)$ given events $\mathcal{D}_j$ (11)
3:     $\dot{w}_i^j \leftarrow (8)$    ▷ compute weight dynamics
4:     $w_i^j \leftarrow w_i^j + \dot{w}_i^j \Delta t$    ▷ update weights
5:     Compute weighted Voronoi partition $W_i^j$ (4)
6:     Compute mass $M_i^j$ and centroid $C_i^j$ (5)
7: end for
8: Compute generator dynamics $\dot{g}_i = \kappa \sum_{j \in \mathcal{C}_i} M_i^j (C_i^j - g_i)$
9: return $\dot{g}_i$
```

outlined in Algorithm 2, in which they estimate the density given the set of events $\mathcal{D}_j$, compute the mass and centroid given the updated weighted Voronoi partition, and follow a move-to-centroid policy. Here, the robot positions serve as the generators, so the robot dynamics $\dot{x}_i = u_i$ is equivalent to the generator dynamics $\dot{g}_i$. Once the robots have converged to their critical points, the robots then begin to serve discrete events within their cell(s) W_i^j, while adapting to performance variations, a process which is repeated until termination.

2) Update events (lines 7-10): At each iteration, the robots check for new events that may have appeared and assign them to robots. If a new event requiring capability j appears, then the robot updates the event memory $\mathcal{D}_j$ while ensuring the number of elements in $\mathcal{D}_j$ is equivalent to n_j, thus keeping a sliding memory of events that contribute to the estimated density $\phi_{j,\text{est}}$. If a new event μ_l^j appears in robot i's cell W_i^j, we add to the total number of events n_i^j for the current time window T_k.

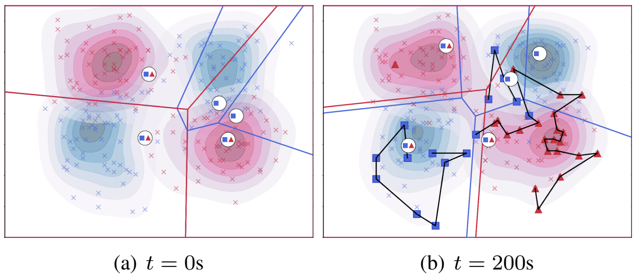
3) Update tour (line 11): The robots then update their tour $\mathcal{T}_i$, which is the queue of events that robot i is to serve within all its cells W_i^j, $\forall j \in \mathcal{C}_i$. As new events appear in a robot's cell, or as events are removed from its cell, each robot updates its own tour to accommodate changes in events. Note events can be removed from the tour by either meeting the event, or by a transfer of events between neighboring robots as cell boundaries move. A loss of an event by transfer would result in the gain of said event by the corresponding neighboring robot. Note this tour update can be computed in a distributed fashion.

4) Compute reputation (lines 12-15): Each robot computes the reputation of all robots in the team, including itself. Every T seconds, we compute the robot performance of the previous time window T_{k-1} using (2), then update the reputation memory $\mathcal{M}_i^j$, which contains the performance values p_i^j from up to the last N time windows. If there is currently no reputation memory ($\mathcal{M}_i^j = \varnothing$), or the number of elements in $\mathcal{M}_i^j$ is less than N, then we append the new reputation value of time window T_{k-1} to $\mathcal{M}_i^j$. If there are already N performance values in $\mathcal{M}_i^j$, then we append the new reputation value, and pop the first element from $\mathcal{M}_i^j$. We use $\mathcal{M}_i^j$ in (8) to update the weights (lines 3-4 in Algorithm 2). The number of events served and total number of events are reset prior to starting the new time window T_k.

5) Move generators and robots (lines 16-27): Finally, we update the generators using Algorithm 2, which also updates the weighted cells. If the robot has events in its cell, the robot sets the next event in its cell $d_{i,\text{next}}$ as its destination, and the robot moves at maximum speed to this event. If the robot is already at that event, we consider the event served. We then add this event to the total number of events served and remove this event from the tour $\mathcal{T}_i$. If there are no events in any of robot i's cells ($\mathcal{T}_i = \varnothing$), the robot moves toward its generator, where it will be optimally positioned based on the estimated probability density.

VI. Simulations

We performed a series of simulations comparing our approach (Algorithm 1) with one that does not track robot reputation. Specifically, the comparison algorithm is Algorithm 1 without a weighted Voronoi diagram, thus we keep w_i^j constant for the duration of the comparison trial. We consider two scenarios: one with five robots and up to two capabilities ($n = 5$, $m = 2$), and one with five robots and up to three capabilities ($n = 5$, $m = 3$). We randomized robot positions, robot capabilities, and maximum robot velocities. We also randomly generated events every 10 seconds, with the probability of an event popping up at point q being weighted by a Gaussian Mixture Model (GMM). Capabilities


(a) $t = 0$s (b) $t = 200$s

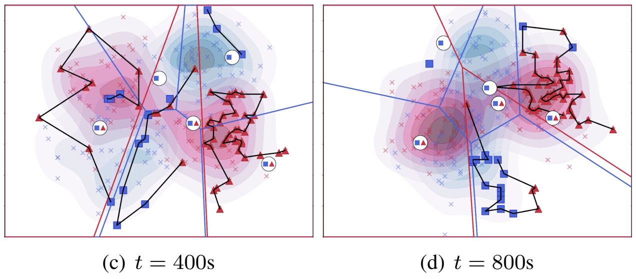
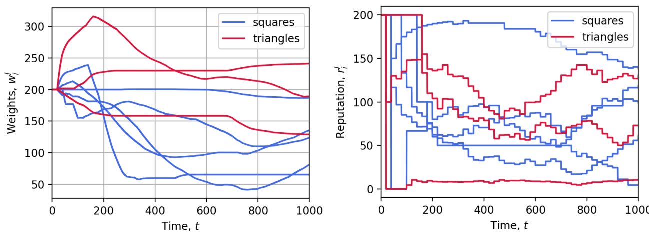

(c) $t = 400$s (d) $t = 800$s

Fig. 2. Time series of an example trial with $n = 5$ robots and $m = 2$ capabilities under our approach (Algorithm 1). Here, the robots are represented by white discs, with their capabilities (blue square and/or red triangle) printed on top of the discs. The solid blue and red lines indicate the Voronoi boundaries for the square and triangle capabilities, respectively. The event history $\mathcal{D}_j$ locations are denoted by the $\times$'s, with red $\times$'s contributing to the triangle density, and the blue $\times$'s contributing to the square density. The contour lines represent probability of event occurrences as estimated by KDE from $\mathcal{D}_j$. Events that have yet to be served are denoted by the red and blue triangles, and the robot tours are denoted by the black solid lines.



(a) Weights over time. (b) Reputation over time.

Fig. 3. Weights and reputation values for the trial presented in Figure 2. Each line indicates the weights $w_i^j(t)$ associated with robot i serving capability j. In this case, j is either a square or a triangle (see Figure 2).

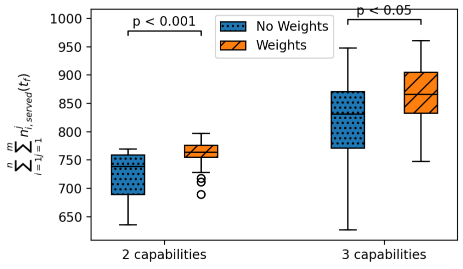


Fig. 4. Randomized simulations comparing the total events met under weighted Voronoi cells and a non-weighted cells. Each box consists of 25 trials. Statistical significance is indicated between the two cases.

$j = 1$ and $j = 2$ each had two Gaussian peaks, as shown in Figure 2, and capability $j = 3$ had a single peak at the center of the environment, as shown in Figure 1. The GMMs slowly migrated across the space and were effectively tracked by our KDE (see Figure 1). Note the multiple, overlapping partitions open up the possibility for collisions between agents. While we do not account for collision avoidance in our implementation, as collisions are highly unlikely in large environments, in practice, we assume robots can carry out collision avoidance locally when necessary.

We carried out 25 randomized simulations of each of the two scenarios under both our approach (with weights) and the comparison algorithm (without weights). As a performance metric, we recorded the total number of events met by the end of the trial. All computation was performed in Python. To compute the tour $\mathcal{T}_i$, each robot solves a traveling salesman problem (TSP) given the events in all its cells using Gurobi[3] optimizer. We use KDEpy[4] for our density estimation.

Figure 2 shows a time series of a randomized simulations for $n = 5$ robots and $m = 2$ capabilities. Full simulations of this trial, along with a trial showing $m = 3$ capabilities (also shown in Figure 1), are provided in the supplemental video. We plot the weights w_i^j and reputation values r_i^j over time for this trial in Figure 3. Here, weights and reputation values start at 200 because we use an ϵ value of 200, which allows us to scale the weightings to the size of the environment.

We present results from all simulations in Figure 4. We performed a statistical analysis comparing the number of events met in the weights and no-weights cases. We observe that our weighted approach outperforms the non-weighted approach, with a statistical significance of $p < 0.001$ in the scenario where $m = 2$, and $p < 0.05$ when $m = 3$, as indicated in Figure 4. We therefore conclude that, by adjusting the size of a Voronoi cell based on robot reputation, the number of events the team meets as a whole increases.

VII. Conclusions

In this work, we introduce the concept of robot reputation to improve overall performance of heterogeneous teams tasked with serving discrete events. We use a weighted, heterogeneous Voronoi-based coverage control approach to optimally assign robots to cell(s) based on their individual capabilities. We show that the Voronoi partition generators converge to critical points of the weighted heterogeneous locational costs, allowing the environment to be optimally partitioned for robots to serve events. Each robot is responsible for serving the events that appear in their individual cell(s). We use KDE to estimate the probability of event occurrences across the environment, which continually evolves as events appear. Robots track the reputation of each agent over time and use the reputation to weigh their Voronoi cells, and ultimately adjust the amount of space they cover. Our simulations show that weighing cells using reputation improves overall team performance.

[3]https://www.gurobi.com/
[4]https://kdepy.readthedocs.io/en/latest/index.html

References

[1] L. Pimenta, V. Kumar, R. C. Mesquita, and G. Pereira, "Sensing and coverage for a network of heterogeneous robots," in *47th IEEE Conference on Decision and Control (CDC)*. IEEE, 2008, pp. 3947–3952.

[2] A. Kwok and S. Martinez, "Energy-balancing cooperative strategies for sensor deployment," in *2007 46th IEEE Conference on Decision and Control*, 2007, pp. 6136–6141.

[3] J.-S. Marier, C. A. Rabbath, and N. Léchevin, "Health-Aware Coverage Control With Application to a Team of Small UAVs," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 5, pp. 1719–1730, 2013.

[4] A. Pierson, L. C. Figueiredo, L. C. A. Pimenta, and M. Schwager, "Adapting to sensing and actuation variations in multi-robot coverage," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 337–354, 2017.

[5] J. M. Palacios-Gasós, D. Tardioli, E. Montijano, and C. Sagüés, "Equitable persistent coverage of non-convex environments with graph-based planning," *The International Journal of Robotics Research*, vol. 38, no. 14, pp. 1674–1694, 2019.

[6] M. Raoufi, P. Romanczuk, and H. Hamann, "Individuality in Swarm Robots with the Case Study of Kilobots: Noise, Bug, or Feature?" 2023.

[7] L. E. Parker, "Lifelong Adaptation in Heterogeneous Multi-Robot Teams: Response to Continual Variation in Individual Robot Performance," *Autonomous Robots*, vol. 8, no. 3, pp. 239–267, 2000.

[8] M. E. Cao, X. Ni, J. Warnke, Y. Han, S. Coogan, and Y. Zhao, "Leveraging Heterogeneous Capabilities in Multi-Agent Systems for Environmental Conflict Resolution," in *2022 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2022, pp. 94–101.

[9] Y. Emam, S. Mayya, G. Notomista, A. Bohannon, and M. Egerstedt, "Adaptive Task Allocation for Heterogeneous Multi-Robot Teams with Evolving and Unknown Robot Capabilities," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France, 2020, pp. 7719–7725.

[10] J. Li, Y. Li, Y. Weng, and J. He, "Adaptive Task Allocation for Multi-agent Cooperation with Unknown Capabilities," in *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, 2020, pp. 1–5.

[11] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International journal of robotics research*, vol. 23, no. 9, pp. 939–954, 2004.

[12] D. P. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Annals of operations research*, vol. 14, no. 1, pp. 105–123, 1988.

[13] A. Prorok, "Redundant robot assignment on graphs with uncertain edge costs," in *Distributed autonomous robotic systems*. Springer, 2019, pp. 313–327.

[14] M. Otte, M. J. Kuhlman, and D. Sofge, "Auctions for multi-robot task allocation in communication limited environments," *Autonomous Robots*, vol. 44, no. 3, pp. 547–584, 2020.

[15] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 243–255, 2004.

[16] S. P. Lloyd, "Least Squares Quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[17] M. Santos and M. Egerstedt, "Coverage Control for Multi-Robot Teams with Heterogeneous Sensing Capabilities Using Limited Communications," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 5313–5319.

[18] A. Sadeghi and S. L. Smith, "Coverage Control for Multiple Event Types with Heterogeneous Robots," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 3377–3383.

[19] M. Pavone, A. Arsie, E. Frazzoli, and F. Bullo, "Equitable partitioning policies for robotic networks," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 2356–2361.

[20] M. Santos, Y. Diaz-Mercado, and M. Egerstedt, "Coverage Control for Multirobot Teams With Heterogeneous Sensing Capabilities," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 919–925, 2018.

[21] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2002.

[22] Z. Drezner, *Facility Location: A Survey of Applications and Methods*. New York, NY: Springer, 1995, no. 11.

[23] M. Schwager, D. Rus, and J.-J. Slotine, "Decentralized, Adaptive Coverage Control for Networked Robots," *The International Journal of Robotics Research*, vol. 28, no. 3, pp. 357–375, 2009.

[24] W. Luo and K. Sycara, "Adaptive Sampling and Online Learning in Multi-Robot Sensor Coverage with Mixture of Gaussian Processes," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 6359–6364.

[25] M. Schwager, M. P. Vitus, S. Powers, D. Rus, and C. J. Tomlin, "Robust Adaptive Coverage Control for Robotic Sensor Networks," *IEEE Transactions on Control of Network Systems*, vol. 4, no. 3, pp. 462–476, 2017.

[26] M. Santos, U. Madhushani, A. Benevento, and N. E. Leonard, "Multi-robot Learning and Coverage of Unknown Spatial Fields," in *2021 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 2021, pp. 137–145.

[27] B. Silverman, *Density Estimation for Statistics and Data Analysis*, 1st ed. New York, NY: Routledge, 1998.