Cooperative Search by UAV Teams: A Model Predictive Approach Using Dynamic Graphs*

James R. Riehl⋆, Gaemus E. Collins†, and João P. Hespanha‡

Abstract

This paper presents a receding-horizon cooperative search algorithm that jointly optimizes routes and sensor orientations for a team of autonomous agents searching for a mobile target in a closed and bounded region. By sampling this region at locations with high target probability at each time step, we reduce the continuous search problem to a sequence of optimizations on a finite, dynamically updated graph whose vertices represent waypoints for the searchers and whose edges indicate potential connections between the waypoints. Paths are computed on this graph using a receding horizon approach, in which the horizon is a fixed number of graph vertices. To facilitate a fair comparison between paths of varying length on non-uniform graphs, the optimization criterion measures the probability of finding the target per unit travel time. Using this algorithm, we show that the team discovers the target in finite time with probability one. Simulations verify that this algorithm makes effective use of agents and outperforms previously proposed search algorithms. We have successfully hardware tested this algorithm in two small UAVs with gimbaled video cameras.

I. Introduction

This paper addresses the problem in which a team of agents is searching for a target in a bounded region with the objective of finding the target in minimum time. Each agent is equipped with a gimbaled sensor that can be aimed at nearby areas within the search region. As they move around and take sensor measurements, the agents gather information on the likely locations of

*Partial support for this work comes from the Air Force Office of Scientific Research (AFOSR), Contract FA9550-06-C-0119 and from the Institute for Collaborative Biotechnologies through grant W911NF-09-D-0001 from the U.S. Army Research Office.

⋆jriehl@att.com, AT&T Government Solutions, 5383 Hollister Ave, Santa Barbara, CA 93111.

†gcollins@toyon.com, Toyon Research Corporation, 6800 Cortona Drive, Goleta, CA 93117.

‡hespanha@ece.ucsb.edu, Center for Control, Dynamical Systems, and Computation, Electrical and Computer Engineering Department, University of California, Santa Barbara, CA 93106-9560.

the target. The objective of each agent is to move itself and control its sensor in a way that minimizes the expected time for the team to discover the target. In this paper, we present an algorithm to approximately solve this optimization problem. The algorithm is *cooperative* in that the agents share information and jointly optimize their routes for the benefit of the team; it is *graph-based* because the search routes are defined by a sequence of vertices in a dynamically updated graph; and we use a receding horizon optimization over a constantly changing target pdf similar to model predictive control. Hence we call the algorithm Cooperative Graph-Based Model Predictive Search (CGBMPS). We also describe our implementation of this algorithm on a test platform consisting of 2 unmanned air vehicles (UAVs) with gimbal-mounted cameras.

We begin with a brief review of search theory. Modern search theory was pioneered by Koopman [14], Stone [23], and others, and was initially motivated by the desire to develop efficient search methods to find enemy marine vessels. More recently, agencies such as the U.S. Coast Guard have applied search theory to search and rescue missions with great success, measured in saved lives [6]. Other search applications include exploration, mining, and surveillance [9].

Early search theory focused on the allocation of search attention to limited areas within a large search region, which is appropriate to cases in which searcher motion is unconstrained. If this is not the case, we are presented with the more difficult problem of finding optimal paths for the searchers. For regular unimodal probability distributions for the target position, lawnmower-like paths are optimal. However, when these probabilities result from sensor fusion with false alarms and non-Gaussian noise, one is often faced with highly irregular distribution for which the computation of optimal paths is difficult. Mangel formulated the continuous search problem in [16] as an optimal control problem on the searcher's velocity subject to a constraint in the form of a partial differential equation, but this problem is only solvable for very simple initial target probability distributions. A more practical approach is to discretize the search space and formulate the search problem on a graph. Under such a formulation, the set of search paths is restricted to piecewise linear paths connecting a finite set of graph vertices. Although this discretization simplifies the problem to some extent, it is still computationally difficult. Trummel and Weisinger showed in [26] that the single-agent search problem on a graph is NP-hard even

for a stationary target. In [5], Eagle and Yee were able to solve somewhat larger problems than what had previously been possible by formulating the problem as a nonlinear integer program and using a branch and bound algorithm to solve it. However, the size of computationally feasible problems was still severely limited. DasGupta *et al.* presented an approximate solution for the continuous stationary target search based on an aggregation of the search space using a graph partition [4]. In [21], we used a similar approach, but with a "fractal" formulation, which allowed the partitioning process to be implemented on multiple levels. All of the results mentioned so far are for a single search agent.

The problem gains additional complexity when we consider multiple agents that are cooperatively searching for a target. There are several methods for reducing this complexity. Some researchers have achieved this through emergent behavior, which is particularly effective when communication is severely limited [22] [8]. Chandler and Pachter approached this problem by splitting the agents (UAVs) into sub-teams and applying hierarchical decomposition to the tasks [2]. Another effective way to reduce the computation involved in finding optimal search paths to locate a moving target is to restrict the optimization to a finite, receding horizon. Somewhat surprisingly, even very short optimization horizons can yield good results. Hespanha *et al.* showed in [10] that in pursuit-evasion games played on a discrete grid, one step Nash policies result in finite-time evader capture with probability one. Using slightly longer horizons, Polycarpou *et al.* introduced a finite-horizon look-ahead approach in [19] similar to model predictive control as part of a cooperative map learning framework. Several other researchers have considered similar look-ahead policies for coordinated search based on Bayesian learning models [24] [7]. A challenge faced by these receding horizon algorithms is that because they require the solution to an NP-hard search problem at each time step, the prediction horizon must be kept short to ensure computational feasibility. This means that if there are regions of high target probability separated by a distance that is much greater than the search horizon, the algorithm's performance may degrade significantly. One does not have to produce pathological examples to generate such a situation, because even a uniform initial distribution can evolve into a highly irregular distribution as the searchers move through the region and update their target pdf estimates based on sensor

measurements.

With the CGBMPS algorithm, we address this issue by performing the receding-horizon optimization on a dynamically changing graph whose nodes are carefully placed at locations in the search region having the highest target probability. The prediction horizon is defined as a fixed number of graph edges, but the edges of the graph are not uniform in length. The algorithm chooses paths that maximize the probability of finding the target per unit time. This dynamic graph structure facilitates two key strengths of our algorithm: (i) search agents only perform detailed searches in regions with high target probability, and (ii) long paths can be efficiently evaluated and fairly compared to short paths.

The remainder of the paper is organized as follows. Section II presents the cooperative search problem in a general discrete time setting. Section III describes our approach to this problem in the form of a receding horizon search algorithm on a graph. In Section IV, we present the main results on finite-time target discovery. Section V provides the numerical simulations of the algorithm and discusses the results. Section VI describes a hardware implementation of the algorithm along with the results of several field tests using two UAVs with gimbal-mounted cameras in a hardware-in-the-loop simulation environment. In Section VII, we provide conclusions and some ideas for future research.

II. Search Problem Formulation

This section formulates the search problem as a discrete-time optimization in which a team of autonomous agents controls their positions and sensor orientations, with the goal of finding a target in minimum time.

A. Agent Dynamics and Sensor Coverage

Suppose a team of M agents is searching for a mobile target in a bounded planar region $\mathcal{R} \subseteq \mathbb{R}^2$. Each agent is equipped with a gimbaled sensor that it can aim at a limited area of the search region. The region that a sensor can view at a particular time instant is called the *field of view (FOV)*, and the subset of $\mathcal{R}$ viewable by the sensor as it is swept through its entire range

of motion (while the agent on which the sensor lies is stationary) is called the sensor's *field of regard (FOR)*. Figure 1 shows the relationship between agent, FOV, and FOR.

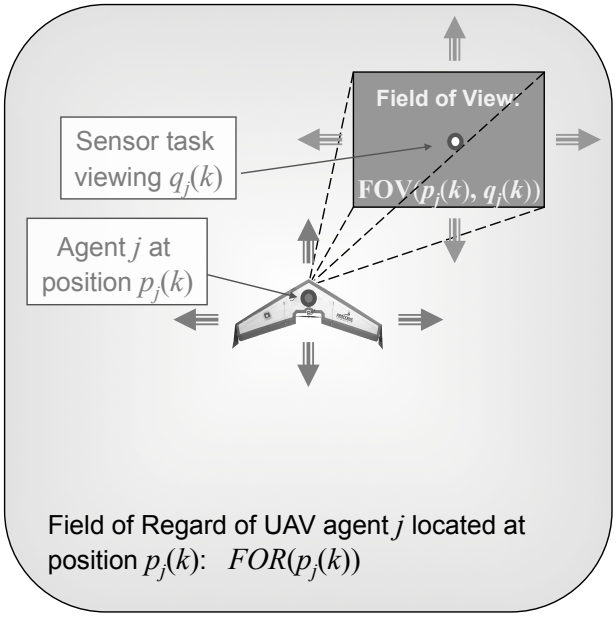


Fig. 1. **Diagram of a sensor field of regard (FOR) and a field of view (FOV) for a search agent located at position $p(k)$.**

Let $\mathcal{A} \subseteq \mathbb{R}^3$ be the space in which the agents move.[‡] We denote the position state of the search team by $\mathbf{p}(k) := [p_1(k), p_2(k), \ldots, p_M(k)]$, where $p_a(k) \in \mathcal{A}$ is agent a's position at time k, and $k \in \mathbb{Z}^{\geq 0}$ is a discrete time variable belonging to the nonnegative integers. We assume a simple kinematic model for the agents, which can move in any direction with unit velocity. For each agent, we define a *sensor task* $q_a(k) \in \mathcal{R}$ that specifies the starepoint where agent a will point its sensor at time k, and we denote the vector of sensor tasks for the team by $\mathbf{q}(k) := [q_1(k), q_2(k), \ldots, q_M(k)]$. The field of view associated with a particular sensor task will generally depend not only the starepoint $q_a(k)$, but also on the agent's position $p_a(k)$ and the characteristics of the sensor. For simplicity of presentation, we assume that there are no additional degrees of freedom associated with the sensor, such as zooming, that could change the field of view while the agent position and sensor task remain fixed. Denoting the set of

[‡]The agents are allowed to move in three dimensions for generality of the approach, and since the sensor properties may be affected by altitude.

all possible sensor tasks by $\mathcal{Q}$, and the set of all subsets of $\mathcal{R}$ by $\mathcal{P}(\mathcal{R})$, we define a function $FOV : \mathcal{A} \times \mathcal{Q} \to \mathcal{P}(\mathcal{R})$ that maps an agent's position and the point at which its sensor is aimed to the subset of $\mathcal{R}$ in view of that sensor. Similarly, we will use the function $FOR : \mathcal{A} \to \mathcal{P}(\mathcal{R})$ to denote the subset of the search region lying in the field of regard of an agent.

B. Sensor Model and Target Detection

Suppose an agent at position $p(k)$ aims its sensor at the point $q(k)$ at some instant in time k. A target located at a point $x \in FOV\,(p(k), q(k))$ will have some probability of being correctly detected by the sensor. We will generally refer to this value as the *probability of detection*, and denote it by $P_D\,(p(k), q(k), x(k)) \in [\rho_{\min}, 1]$, where $\rho_{\min} > 0$ denotes a minimum value for the probability of detection. Although it is common to assume that P_D is a constant, we allow for a more a general case to account for changes in the size and shape of the FOV for different vehicle altitudes and sensor orientations, as well as possible line-of-sight obstructions within the FOV. We refer the reader to [15] for a more detailed discussion of these issues in the context of target detection with a video sensor. The probability of detecting a target that does not lie in the sensor's FOV is zero. It is notationally useful to have an expression for the probability of detection for a target located anywhere in the search region; hence we define a general detection function

$$D\,(p(k), q(k), x(k)) := \begin{cases} P_D\,(p(k), q(k), x(k))\,, & x(k) \in FOV\,(p(k), q(k)) \\ 0, & \text{otherwise} \end{cases}. \tag{1}$$

In this paper, we assume that the sensors do not produce *false positives*; *i.e.*, the sensors do not detect the presence of a target where none exists.

C. Target State Estimation

Let the random variable $\mathbf{X}(k) \in \mathcal{R}$ denote the state of the target. By the nature of the search problem, the target state is unknown, and the objective of the agents is to gather information on $\mathbf{X}(k)$ using their sensors. In order to optimize how the agents and sensors should move,

we need a method for estimating the probability distribution of $\mathbf{X}(k)$ as it evolves over time. With the possible exception of some very simple cases, optimal estimators such as the Kalman filter do not apply to the search problem because target motion may be nonlinear and Gaussian assumptions on the probability distribution do not hold. Even if one assumes a Gaussian initial distribution, sensor measurement updates will cause future probability distributions to be non-Gaussian. Consequently, target estimation is a challenging problem and a rich field of study in itself. We refer the reader to [1] and [11] for a deeper analysis of the problem and several good examples of target estimators. For the purposes of this paper, rather than choosing a specific approach, we provide an algorithm that applies to a wide range of estimation methods. In particular, we work with the class of estimators that characterize the target probability density function (pdf) with a discrete approximation to an underlying continuous probability density function. We choose this class of estimators, which includes grid-based probabilistic maps as well as particle filters, because of their ability to accurately incorporate nonlinear target dynamics and environmental constraints such as road networks, as well as accurately model non-Gaussian distributions.

In the remainder of this section, we assume that the pdf of the target state $\mathbf{X}(k)$ is approximated by a function $\hat{f}_k(\cdot)$ of the following form:

$$\hat{f}_k(x) = \sum_{i=1}^{n} w_i(k)\delta(x_i(k) - x), \qquad \forall k \in \mathbb{Z}_{\geq 0}, \quad \forall x \in \mathcal{R} \tag{2}$$

where each $x_i(k) \in \mathcal{R}$, each $w_i(k) \in [0, 1]$, and $\delta(\cdot)$ denotes a function that integrates to one over the search region:

$$\int_{\mathcal{R}} \delta(x_i(k) - x)dx = 1, \quad \forall i \in \{1, \ldots, n\}. \tag{3}$$

For example, $\delta(\cdot)$ could take the form of a Dirac delta function, for use with a point-mass approximation to a probability distribution, or it could take the form of a two-dimensional rectangular impulse function, for the case of grid-based approximation. The pdf approximation in (2) is a type of *probability mixture model* [17], which also has the property that the sum of

all weights $w_i(k)$ is equal to one:

$$\sum_{i=1}^{n} w_i(k) = 1. \tag{4}$$

We define the Target Probability Mixture Model (TPMM) as the pair $[\mathbf{x}(k), \mathbf{w}(k)]$, where $\mathbf{x}(k) = [x_1(k), x_2(k), \ldots, x_n(k)]$ and $\mathbf{w}(k) = [w_1(k), w_2(k), \ldots, w_n(k)]$. Where there is no ambiguity, we may also refer to the TPMM as the target pdf in the remainder of the paper.

The initial state of the TPMM, $[\mathbf{x}(0), \mathbf{w}(0)]$ is based on any prior knowledge of target location, *e.g.* an initial probability distribution, or a uniform distribution if no prior target information is available. Future TPMM states depend on sensor tasks $\mathbf{q}(k)$ and the prior state. The dynamics of $\mathbf{x}(k)$ and $\mathbf{w}(k)$ will depend on the estimator used, but can be expressed in general form as:

$$\begin{cases} \mathbf{x}(k{+}1) = f_x\big(\mathbf{x}(k), \mathbf{w}(k), \mathbf{q}(k)\big) \\ \mathbf{w}(k{+}1) = f_w\big(\mathbf{x}(k), \mathbf{w}(k), \mathbf{q}(k)\big) \end{cases}, \tag{5}$$

where the functions f_x and f_w must preserve the requirements $x_i(k{+}1) \in \mathcal{R}$ and $w_i(k{+}1) \in [0, 1]$ for each i, as well as condition (4). In (5), we have defined a general model that is compatible with many of the typical target models used in the search literature, including Markov models [7] and particle filters [25]. In the appendix, we provide example dynamics of (5) for a simple particle filter and a probabilistic map.

D. Search Reward

The objective in the search problem is to find the target in minimum time. To achieve this, the agents must be able to evaluate the probability of finding the target for various sequences of control actions. In this section, we define a criterion called *search reward* to measure this probability.

1) Single Agent Target Detection: Recall from Section II-B that when an agent at position p aims its sensor at the point q, the probability that a target located at the point x will be detected

is given by $D(p,q,x)$, which was defined as

$$D(p,q,x) := \begin{cases} P_D(p,q,x), & x \in FOV(p,q) \\ 0, & \text{otherwise} \end{cases},$$

for a given instant in time (we omit the time variable k for simplicity of notation). Since the location of the target is unknown in the search problem, we now define the probability of detection conditioned on our best estimate of the target position, $[\mathbf{x}, \mathbf{w}]$ as discussed in Section II-C. First, we note that the portion of the i^{th} TPMM component that lies within the sensor's field of view can be expressed as follows:

$$\int_{\mathcal{R}} \delta(x_i - x) D(p,q,x) dx$$

Let $\mathcal{D}$ be the event that the single agent discovers the target. One can compute the probability of $\mathcal{D}$ by applying the total probability theorem over the n components of the TPMM:

$$P(\mathcal{D}) = \sum_{i=1}^{n} w_i \int_{\mathcal{R}} \delta(x_i - x) D(p,q,x) dx. \tag{6}$$

2) Multi-Agent Target Detection: Consider now the case in which M agents are searching for the same target, cooperatively. That is, they share a common goal of finding the target in minimum time, regardless of which agent is the first to discover it. The agents cooperate by sharing a common target pdf $[\mathbf{x}, \mathbf{w}]$, which each agent updates as it gathers new information from its sensors. In this section, we are interested in calculating the *team probability of detection*, denoted by $P(\mathcal{D}_{\text{team}})$, where $\mathcal{D}_{\text{team}}$ is the event that the target is discovered by at least one of the agents at a given instant in time.

Suppose M agents at positions $(p_1, \ldots, p_M)$ aim their sensors at the points $(q_1, \ldots, q_M)$. The FOVs associated with each agent's sensor task may or may not overlap in the search region. The following sequence of joint events enumerates all situations that lead to $\mathcal{D}_{\text{team}}$, keeping in mind that once one agent discovers the target, it is irrelevant to our problem whether another agent simultaneously discovers the same target:

$\mathcal{D}_1$ = the target is discovered by agent 1;

$\mathcal{D}_2$ = the target is not discovered by agent 1, but it is discovered by agent 2;

$\mathcal{D}_3$ = the target is not discovered by agents 1 or 2, but it is discovered by agent 3;

$\vdots$

$\mathcal{D}_M$ = the target is not discovered by agents 1 through $M-1$, but it is discovered by agent M.

Since these events are mutually exclusive, the event that the target is detected by at least one agent is simply the union of the events listed above:

$$\mathcal{D}_{\text{team}} = \bigcup_{a=1}^{M} \mathcal{D}_a.$$

The key to evaluating the probability of each event $\mathcal{D}_a$ is to notice that, while the individual events are *not* independent[‡], they become independent when each $\mathcal{D}_a$ is conditioned on a value of the target state. Using this property, it is straightforward to show from (6) that

$$P(\mathcal{D}_a) = \sum_{i=1}^{n} w_i \int_{\mathcal{R}} \Big(\delta(x_i - x) D(p_a, q_a, x) \prod_{\alpha=1}^{a-1} \big(1 - D(p_\alpha, q_\alpha, x)\big) \Big) dx \tag{7}$$

for each $a \in \{1, \ldots, M\}$. Since the sensor tasks $\{q_a\}$ happen simultaneously, the order in which we consider the target detected by the various tasks is irrelevant and the arbitrary ordering of agents incurs no loss of generality. The team's probability of detection may be calculated by summing the probabilities of detection for each agent:

$$\begin{aligned} P(\mathcal{D}_{\text{team}}) &= \sum_{a=1}^{M} P(\mathcal{D}_a | \mathbf{x}, \mathbf{w}, \mathbf{p}, \mathbf{q}) \\ &= \sum_{a=1}^{M} \left\{ \sum_{i=1}^{n} w_i \int_{\mathcal{R}} \Big(\delta(x_i - x) D(p_a, q_a, x_i) \prod_{\alpha=1}^{a-1} \big(1 - D(p_\alpha, q_\alpha, x_i)\big) \Big) dx \right\}. \end{aligned} \tag{8}$$

3) Cumulative Reward: Thus far we have considered instantaneous probabilities of detection, but the quantity we wish to minimize is the time until the target is detected. This requires formulating a search reward function $r(k)$ in terms of a *cumulative* probability of detection, as

[‡]individual events $\mathcal{D}_a$ are not independent because knowledge about one event occurring (or not occurring) gives us clues about the value of the target state, which affects the outcome of the other events.

follows.

Let T^* denote a hypothetical time at which the target is found by any of the search agents. We define the *search reward* as the probability that the target is found at time k:

$$r(k) := P(T^* = k).$$

We can express the search reward as

$$r(0) := 0, \tag{9}$$

$$r(k) := P(T^* = k) = P(T^* = k|T^* \geq k)P(T^* \geq k). \tag{10}$$

Since the events of finding the target at different times are mutually exclusive, we can express $P(T^* \geq k)$, the probability that the target has not been found before time k, as

$$P(T^* \geq k) = 1 - \sum_{\kappa=0}^{k-1} P(T^* = \kappa) = 1 - \sum_{\kappa=0}^{k-1} r(\kappa).$$

We can now rewrite the search reward as

$$r(k) := P(T^* = k|T^* \geq k) \left(1 - \sum_{\kappa=0}^{k-1} r(\kappa)\right). \tag{11}$$

The term $P(T^* = k|T^* \geq k)$ is the conditional probability that the target will be found at time k given that it was not found previously. Since $[\mathbf{x}, \mathbf{w}]$ is conditioned on the target not yet being found,

$$P(T^* = k|T^* \geq k) = P\big(\mathcal{D}_{\text{team}}(k)\big). \tag{12}$$

Combining (8), (11), and (12) results in the following expression for the search reward:

$$r(k) = \sum_{a=1}^{M} \left\{ \sum_{i=1}^{n} w_i(k) \int_{\mathcal{R}} \Big(\delta(x_i - x) D(p_a(k), q_a(k), x_i(k)) \prod_{\alpha=1}^{a-1} \big(1 - D(p_\alpha(k), q_\alpha(k), x_i(k))\big) \Big) dx \right\} \times \left(1 - \sum_{\kappa=0}^{k-1} r(\kappa)\right). \tag{13}$$

It is also useful to have an expression for the reward collected by each individual agent. Therefore we define $r_a(k)$ as the search reward for agent a at time k. We can then rewrite (13) as

$$r(k) = \sum_{a=1}^{M} r_a(k), \quad \text{where}$$

$$r_a(k) = \sum_{i=1}^{n} w_i(k) \int_{\mathcal{R}} \Big(\delta(x_i - x) D(p_a(k), q_a(k), x_i(k)) \prod_{\alpha=1}^{a-1} \big(1 - D(p_\alpha(k), q_\alpha(k), x_i(k))\big) \Big) dx \times \left(1 - \sum_{\kappa=0}^{k-1} r(\kappa)\right). \tag{14}$$

E. Problem Statement

The goal of the search problem is to optimally control the agents and their sensors to locate the target in minimum time. We define a *search policy* to be a function μ that maps the current team state $\mathbf{p}(k)$ and the TPMM $[\mathbf{x}(k), \mathbf{w}(k)]$ to the next set of controls to be executed by the team. That is, each agent's next position and sensor task is given by $[\mathbf{p}(k+1), \mathbf{q}(k+1)] = \mu(\mathbf{p}(k), \mathbf{x}(k), \mathbf{w}(k))$. The objective in the search problem is to compute the search policy that results in the fastest possible discovery of the target by the team.

T^* denotes the time at which the target is discovered by any agent. This time is unknown at the beginning of the search, but we can write its expected value as

$$E_\mu[T^*] = \sum_{k=0}^{\infty} k r(k).$$

We can now express the objective of the search problem as

$$\min_{\mu} E_\mu[T^*] \tag{15}$$

subject to

$$p_a(k+1) \in \mathcal{P}_a(k) \tag{16}$$

$$q_a(k+1) \in \mathcal{Q}_a(p_a(k+1)), \tag{17}$$

where the set $\mathcal{P}_a(k) \subset \mathcal{A}$ denotes the set of all reachable points and $\mathcal{Q}_a(p_a(k+1)) \subset \mathcal{R}$ the set

of all feasible sensor tasks for agent a at position $p_a(k+1)$. These constraints are imposed by physical limitations of the agents and their gimbaled sensors.

For agents with such sensor and motion constraints, the problem posed in (15) is a difficult combinatorial optimization problem. Even for the case of a single agent, stationary target, and fixed sensor, this search problem is known to be NP-hard [26]. The mobile target and movable sensor introduce additional complexity to the problem.

III. Cooperative Graph-based Model Predictive Search Algorithm

In this section, we present a receding horizon graph-based search algorithm that approximately solves the search problem (15)-(17). This algorithm plans paths and sensor tasks for a team of cooperating search agents based on predicted future target states. Agent paths are formulated on a Euclidian graph as a sequence of graph edges. The prediction horizon is set to a fixed number of graph edges, rather than a fixed time, so that paths of varying duration may be considered. This allows agents to travel between regions of high-reward that are separated by large areas with low reward using just one path step.

This approach is designed to reduce computation by optimizing over a smaller number of graph edges and therefore on a much smaller state space, while also using careful vertex placement to maintain route flexibility in regions of high reward. Although our algorithm does not necessarily result in a policy that minimizes $E[T^*]$, the resulting policy does lead to a finite value for this expected time and we provide an upper bound for this quantity in Section IV. Furthermore, Section V provides simulations showing that the CGBMPS algorithm outperforms previously proposed search algorithms.

A. Graph Search Formulation

Let $G := (V, E)$ be a Euclidian Graph for path-planning with vertex set V and edge set E. The vertices of the graph will serve as waypoints for the agents, which are connected by graph edges to form paths. When there is no ambiguity, we use the notation v to denote both the vertex in the set V and its location in $\mathcal{A}$, the space in which the agents move. Similarly, $e = (v, v')$

will denote both the edge in the set E and the line segment connecting its endpoint vertices v and v'.

A *path in* G is a sequence of vertices $P := (v_1, v_2, \ldots, v_\ell)$ such that $(v_i, v_{i+1}) \in E$. Each path segment (v_i, v_{i+1}) will be considered as one *step* in the prediction horizon. The *path cost in* G is the time duration of the path and is defined as

$$C(P) := \sum_{i=1}^{\ell-1} c(v_i, v_{i+1}), \tag{18}$$

where $c(v_i, v_{i+1})$ is an edge-cost function $c\colon E \to [0, \infty)$ that represents the transit time between vertices, assuming the agents move with unit velocity[‡].

Starting from an agent's current location, the algorithm selects the agent's future route as an ℓ-step path in G that maximizes a *path reward* function. In an attempt to minimize the expected time $E[T^*]$ to find the target, we will define a path reward function corresponding to the team's probability of finding the target per unit time [see criterion (20) below]. This optimization criterion enables the algorithm to compare paths of various lengths.

As each agent travels along its path P, it executes a sequence of sensor tasks, which we call a *sensor schedule* and denote by $S(P, k) := \big(q(k), q(k+1), \ldots, q(k+T_P)\big)$, where each $q(\cdot)$ is a scheduled sensor task belonging to $\mathcal{Q}(k)$, the set of all feasible sensor tasks for the agent at time k, and $T_P := \lfloor C(P) \rfloor$ is the total number of sensor tasks that may be executed along the path P. In this discrete time setting, we assume that the duration of each sensor task is one unit of time. Figure 2 shows a diagram of an agent executing a sensor schedule along its path P. The sensor tasks $q(k)$ may be chosen arbitrarily or computed by an optimization. This allows for several possible methods of algorithm implementation, including

1) **Fixed Sensor Tasking:** Construct $S(P, k)$ assuming a fixed pattern of sensor motion, such as slewing side to side within the field of regard.
2) **Joint Routing and Sensor Optimization:** Construct $S(P, k)$ by optimizing over all

[‡]UAVs in the presence of wind may move with unit air velocity, but have different ground velocities. This can be easily incorporated into the algorithm so that predicted paths use UAV ground velocity (accounting for wind). In this case upwind paths have higher costs.

possible sensor schedules along the path P.

Fixed sensor tasking requires much less computation, but when the speed of the agents is too fast for the sensors to view everything inside the agents' FORs, or urban scenarios with line-of-sight blockages, method 2 may yield significant benefit over method 1. Whichever method is used, the sensor tasking algorithm must contain a model of the sensor gimbal so that it will only generate candidate sensor tasks within the gimbal range limits.

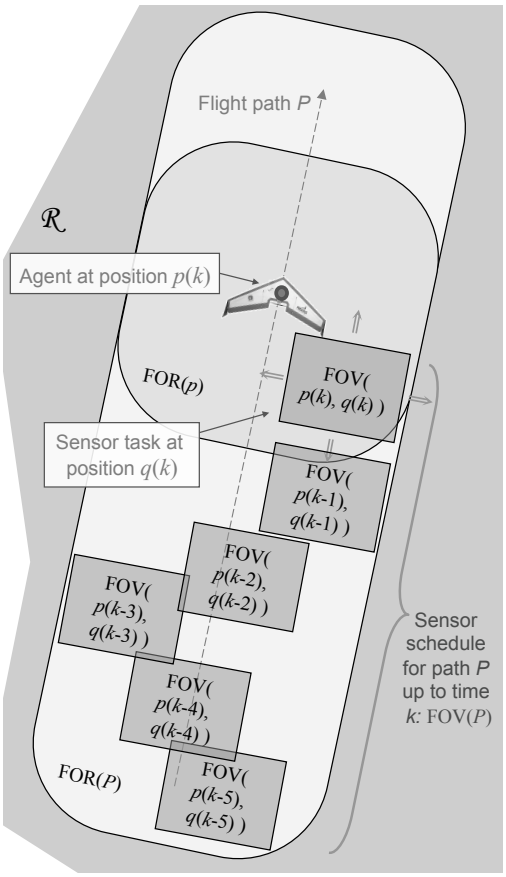


Fig. 2. **Example sensor schedule for an agent traveling along path P.**

The computation of the path reward depends on which method of sensor tasking is used. Suppose that the search agents $\{1, ..., a-1\}$ have planned paths $\{P_i\}_{i=1}^{a-1}$ and corresponding sensor schedules $\{S(P_i, k)\}_{i=1}^{a-1}$. For fixed sensor tasking, we define the *path reward* for agent

a traveling along candidate path P_a as

$$R_a(P_1, \ldots, P_a) := \sum_{\kappa=k}^{k+T_{P_a}} r_a(\kappa), \tag{19}$$

where $r_a(\kappa)$ is computed using (14) with a sensor schedule $S(P_a, k)$ that is computed by some fixed algorithm. Note that any paths $\{P_i\}_{i=a+1}^{M}$ for the agents $\{a+1, \ldots, M\}$ do not affect (19). The path reward for joint routing and sensor optimization includes an optimization over all sensor schedules, and can be expressed as follows:

$$R_a(P_1, \ldots, P_a) := \max_{S(P_a,k)} \sum_{\kappa=k}^{k+T_{P_a}} r_a(\kappa), \tag{20}$$

To provide a fair measure of reward over paths of various lengths, we define the *normalized path reward* $\bar{R}_a$ for the path P_a as the path reward divided by the path cost:

$$\bar{R}_a(P_1, \ldots, P_a) := \frac{R_a(P_1, \ldots, P_a)}{C(P_a)}. \tag{21}$$

Since the time-step is defined as the duration of one sensor task, the re-weighting in (21) is equivalent to dividing by the number of sensor tasks in the plan. One can think of (21) as a reward-per-sensor-task function, and optimizing this function should yield collections of tasks that give the highest reward per task.

As mentioned previously, each agent computes a new path whenever it reaches a waypoint. Since the graph edges have nonuniform length, the agents will generally arrive at waypoints, and hence compute paths, at different times. At each time k, we therefore define two groups of agents: the set $A_{\text{plan}}(k)$ of agents that have arrived at waypoints and need to plan new paths, and the set $A_{\text{en}}(k)$ of the remaining agents that are en route to waypoints. In the CGBMPS algorithm, whenever the set $A_{\text{plan}}(k)$ is nonempty, the agents in that set compute new paths assuming that the agents in the set $A_{\text{en}}(k)$ will continue on their current paths. With this formulation, we can express each step in the receding horizon optimization as

$$\{P_a^* : a \in A_{\text{plan}}(k)\} := \underset{\{P_a : a \in A_{\text{plan}}(k)\}}{\arg\max} \sum_{i=1}^{M} \bar{R}_i(P_1, \ldots, P_i). \tag{22}$$

Often, the set $A_{\text{plan}}(k)$ consists of a single agent, reducing (22) to an optimization for the path of that agent. If this is not the case, a computationally practical approximation to (22) consists of a sequential optimization by the planning agents, which we can express as

$$P_a^* := \arg\max_{P_a} \bar{R}_a(P_1^*, \ldots, P_{a-1}^*, P_a) \quad \forall a \in A_{\text{plan}}(k), \tag{23}$$

where each P_i^* is either the current path of an en route agent $i \in A_{\text{en}}(k)$ that was computed at a previous time or a path computed using (23) by an agent earlier in the sequence $A_{\text{plan}}(k)$. For simplicity of notation and without loss of generality, we assume in (23) that the agents $\{1, \ldots, M\}$ are re-indexed (if necessary) so that the agents in $A_{\text{en}}(k)$ have lower indices than those in $A_{\text{plan}}(k)$.

Notice that since each agent evaluates reward over a different time period T_{P_a}, some agents may plan paths further into the future than others. This does not pose a problem for the algorithm, since (21) is evaluated by accounting for only the reward that prior agents have planned to collect up until their respective time horizons. However, there is the potential for an agent A to "steal" reward from another agent B if agent A's path covers a longer time period and cuts in front of where agent B would be expected to go had it planned a longer path. In this case, at its next waypoint, agent B must simply plan a course that takes agent A's impinging path into account.

B. Dynamic Graph Generation

Although the CGBMPS algorithm will work with any graph, the structure of the graph has a significant impact on computation, and the vertex and edge placements will affect the algorithm's performance. Also, since the target probability distribution is constantly changing due to sensor measurements and predicted target motion, the graph should be periodically updated to keep the agents searching only high-reward areas. This graph update should occur at each time k at which the set of agents $A_{\text{plan}}(k)$ is nonempty. We now present a dynamic graph construction process that guarantees desirable performance properties of the CGBMPS algorithm. Figure 3 shows an example of this process.

In the following procedure, we will use the function $W(\mathbf{x}, \mathbf{w}, v)$ to denote the sum of the

weights of all components of the TPMM lying inside the FOR of a point $v \in \mathcal{A}$, that is

$$W(\mathbf{x}, \mathbf{w}, v) := \sum_{i=1}^{n} w_i \int_{\mathcal{R}} \delta(x_i - x) \cdot \text{inFOR}(v, x) dx,$$

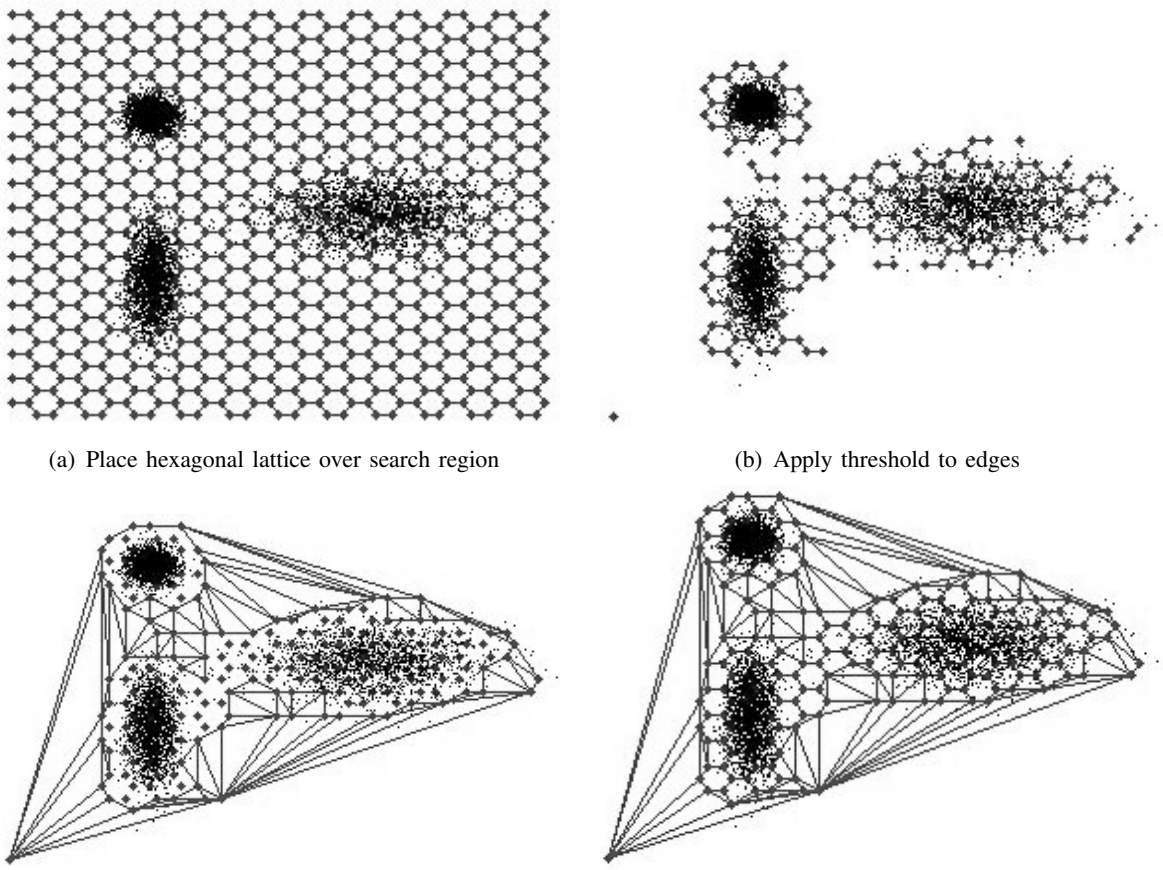
where $\text{inFOR}(v, x) \in \{0, 1\}$ indicates whether or not the point x is inside the FOR of the point v, and is expressed by

$$\text{inFOR}(v, x) := \begin{cases} 1, & x \in FOR(v) \\ 0, & \text{otherwise} \end{cases}. \tag{24}$$

Graph Construction Process

1) Construct a uniform lattice graph $G := (V, E)$ over the search region $\mathcal{R}$ having the property that for every point r in $\mathcal{R}$, there exists a vertex $v \in V$ such that the point r falls within the FOR of v. This can always be achieved by choosing a sufficiently small vertex spacing in the lattice [cf. Figure 3(a)].
2) Choose a reward threshold $\tau > 0$ and let $G_k^\tau := (V_k^\tau, E_k^\tau)$ be the subgraph of G induced by the vertex set $V_k^\tau := \{v \in V : W\big(\mathbf{x}(k), \mathbf{w}(k), v\big) \geq \tau\} \cup V^1$, the set of vertices for which the reward is no less than τ combined with the set $V^1 := \{v_1^1, \ldots, v_1^2\}$ of vertices that serve as initial waypoints for the agents. The edge set E_k^τ consists of all edges in E that connect pairs of vertices that both belong to the set V_k^τ [cf. Figure 3(b)].
3) Let $G_k^{\text{Del}} := (V_k^\tau, E_k^{\text{Del}})$ be the graph generated by a Delaunay triangulation of V_k^τ [18]. Next, let $G_k^{\text{Del}<d} := (V_k^\tau, E_k^{\text{Del}<d})$ be the subgraph of G_k^{Del} obtained by keeping only the edges connecting vertices of degree less than d, where d is the degree of the lattice graph G. That is, $E_k^{\text{Del}<d} := \{(v, v') \in E_k^{\text{Del}} : \deg(v) < d \text{ and } \deg(v') < d\}$ [cf. Figure 3(c)].
4) The final graph $G_k := (V_k, E_k)$ is the combination of graphs G_k^τ and $G_k^{\text{Del}<d}$, with vertex set $V_k := V_k^\tau$ and edge set $E_k := E_k^\tau \cup E_k^{\text{Del}<d}$ [cf. Figure 3(d)].

In step 1, any type of lattice graph will suffice, but we have found that a degree-3 hexagonal lattice is a particularly good choice. The threshold τ in step 2 should be carefully chosen so that it includes only high-reward edges but does not exclude so many edges that the number



(a) Place hexagonal lattice over search region

(b) Apply threshold to edges

(c) Connect using a Delaunay triangulation

(d) Combine graphs b and c for final graph

Fig. 3. **Step-by-step example of the graph construction process. The black points represent the states of a particle filter approximation to the target pdf.**

of feasible paths is severely restricted. The purpose of step 3 is to ensure that the graph G_k is connected and provides paths between unconnected areas of high reward. We include only the Delaunay edges connecting vertices of degree less than 3 because these are the edges between boundary vertices of the disconnected components in the graph. The remaining Delaunay edges are unnecessary. Step 4 combines the graphs of steps 2 and 3 to produce the final graph.

An additional consideration in the graph construction is that to ensure achievable flight paths, the initial graph vertex spacing must be chosen coarse enough that a agent can move from any vertex to any other vertex without looping. This is easy to compute from the minimum turn radius of the agents, and in our experience with UAVs, results in a vertex spacing that still satisfies the requirement of step 1.

C. Nonzero-Reward Paths

In this section, we show that for graphs constructed by the above process, there always exist search paths starting from any vertex on which the agents can collect positive reward, provided the following assumption holds.

Assumption 1. There exists a constant $\gamma > 0$ such that for every point $v \in \mathcal{A}$ and for every $m \in \mathbb{N}$, the evolution of $[\mathbf{x}(k), \mathbf{w}(k)]$ governed by (5) satisfies the property

$$W\big(\mathbf{x}(k+m), \mathbf{w}(k+m), v\big) \geq \gamma^m W\big(\mathbf{x}(k), \mathbf{w}(k), v\big).$$

This assumption is a requirement on the dynamics of $[\mathbf{x}(k), \mathbf{w}(k)]$; it essentially states that the total target weight within the FOR of a vertex cannot drop below a threshold value in a finite number of time steps. In a grid-based probabilistic map, for example, this assumption is satisfied provided that there is at least one nonzero entry in the columns of the transition probability matrix for the cells inside the FOR of v. The assumption will hold in a particle filter provided a large enough number of particles and also enough randomization in the model to make sure that while many particles may leave a region, other may enter so that the total weight does not decay to zero. Under this assumption on the TPMM, The following lemma provides a lower bound on the conditional probability that the target will be found at some time in the interval $[k+1, k+T]$ given that it was not found at or before time k. This result will be instrumental in proving that the CGBMPS algorithm is able to find the target with probability one.

Lemma 1. There exists a constant $\epsilon > 0$ and a time $T > 0$ such that for every graph G_k generated by the Graph Construction Process and every collection of ℓ-step paths $\{P_1, ..., P_M\}$ in G_k, there exists a sequence of sensor tasks $\{\mathbf{q}(k+j)\}_{j=0}^{T-1}$ for which

$$\sum_{\kappa=k}^{k+T-1} P\big(\mathcal{D}_{\textit{team}}(\kappa) | \mathbf{x}(\kappa), \mathbf{w}(\kappa), \mathbf{p}(\kappa), \mathbf{q}(\kappa)\big) \geq \epsilon.$$

Proof: Let v_1 designate the first vertex in path P_1, and choose a vertex v_2 for which there

exists an edge $e \in E_k$ connecting v_1 and v_2. This is always possible because the Delaunay triangulation guarantees that G_k is connected. The graph construction process ensures that $W\big(\mathbf{x}(k), \mathbf{w}(k), v_2\big) \geq \tau$, where τ is the reward threshold selected in Step 2 of the Graph Construction Process. Let $R_{\max}$ be the maximum Euclidean distance between any two vertices in G_k. Using the fact that agents travel with unit velocity, denote the maximum number of time steps between two vertices by $T := \lfloor R_{\max} \rfloor$. By Assumption 1, we have that $W\big(\mathbf{x}(k+T), \mathbf{w}(k+T), v_2\big) \geq \gamma^T W\big(\mathbf{x}(k), \mathbf{w}(k), v_2\big)$, and thus $W\big(\mathbf{x}(k), \mathbf{w}(k), v_2\big) \geq \gamma^T \tau$. We now choose a sensor schedule $S(P_a, k)$ that includes the sensor task

$$q(k+m) := \underset{q \in \mathcal{Q}_e, p \in e}{\arg\max} \sum_{i=1}^{n} w_i(k+m) \int_{\mathcal{R}} \delta(x_i(k+m) - x) D(p, q, x)) dx$$

for some $m \leq T$, where $\mathcal{Q}_e$ is the set of all feasible sensor tasks for an agent along e. Recall from Section II-B that the minimum value for the probability of detection is denoted by ρ_{min}. The minimum amount of target weight contained within the sensor task $q(k+m)$ is achieved for a uniform target weight distribution, and is given by $\gamma^T \tau \rho_{\text{min}} (A_{\text{FOV}}/A_{\text{FOR}})$, where A_{FOV} is the minimum area of the FOV and A_{FOR} is the maximum area of the FOR for any agent/sensor configuration. We conclude from (14) and (20) that $\sum_{a=1}^{M} R_a(P_1, \ldots, P_a) \geq \gamma^T \tau \rho_{\text{min}} (A_{\text{FOV}}/A_{\text{FOR}}) \left(1 - \sum_{\kappa=0}^{k} r(\kappa)\right)$. Therefore, Lemma 1 holds with

$$\epsilon = \gamma^T \tau \rho_{\text{min}} (A_{\text{FOV}}/A_{\text{FOR}}) \left(1 - \sum_{\kappa=0}^{k} r(\kappa)\right).$$

■

D. CGBMPS Algorithm

The Cooperative Graph-Based Model Predictive Search algorithm is described in Table I. It begins with the initialization of the graph and each agent's path and sensor schedule in steps 2 and 3. At each subsequent time step k, whenever the set of agents $A_{\text{plan}}(k)$ is nonempty, the graph is updated, and the agents in $A_{\text{plan}}(k)$ select new paths and sensor schedules according to (22). This process repeats until the target is found at time T^*, which is proven to be finite with

probability one in Theorem 1.

TABLE I
COOPERATIVE GRAPH-BASED MODEL PREDICTIVE SEARCH ALGORITHM

Algorithm 1 CGBMPS

$k := 0$
Construct $G_0 = (V_0, E_0)$ as in Section III-B
For each agent $a \in \{1, \ldots, M\}$, assign an arbitrary initial path $P_a = (v_1^a, v_2^a)$ on G_0 and a corresponding sensor schedule $S(P_a, 0)$
while target has not been discovered **do**
  Each agent a points its sensor at $q_a(k)$ and takes measurements
  Compute the set of agents arriving at waypoints $A_{\text{plan}}(k)$
  if the set $A_{\text{plan}}(k)$ is nonempty **then**
    Update G_k as in Section III-B
    for each agent a in the set $A_{\text{plan}}(k)$ **do**
      Compute the path P_a^* starting from v_2^a and sensor schedule $S(P_a^*, k)$ using (22)
    end for
  end if
  Agents move toward next waypoint in path
  Evaluate $\mathbf{x}(k+1)$, $\mathbf{w}(k+1)$, and $r(k+1)$ using (5) and (13)
  $k := k+1$
end while
$T^* := k-1$

Whenever a path is computed, it starts from the waypoint after the one that was reached. There are two key reasons for implementing the algorithm in this way. First, this allows time for computation of the next path, avoiding situations where an agent has reached a waypoint and does not have any destination until it completes a computation. Second, it is advantageous to have one waypoint planned in advance so that sharp turns in the upcoming path may be smoothed.

E. Computational Complexity

The CGBMPS algorithm provides a computationally efficient method for approximating the original search problem posed in (15) by using a graph that is designed to allow agents to optimize

over a set containing only the most rewarding paths. Let K_e denote the maximum time required to compute the reward collected along an edge e in the graph. Performing an exhaustive search over this set of paths results in a bound on the computation time that depends on the maximum degree of the graph d, the length of the prediction horizon ℓ, and the number of agents M, by the expression $M!K_e\ell d^\ell$. Note that this is a worst-case computation bound because in a non-uniform graph, it is quite rare that all agents will arrive at waypoints simultaneously and hence compute new paths all at the same time. The factor $M!$ corresponds to the optimization given in (22), in which an exhaustive search is performed over every possible order of agents. If one opts to use the computationally simpler, sequential approximation (23) instead, the expression becomes $MK_e\ell d^\ell$. Additionally, in the computation of the optimal path P_a^*, we can take advantage of the property that the reward for the paths (v_1, v_2, v_3) and (v_1, v_2, v_4) is the same for the subpath between vertices v_1 and v_2. The fact that one only needs to compute the reward for the segment (v_1, v_2) once, further reduces the total computation required for this algorithm. This results in the slightly improved bound of $MK_e \sum_{i=1}^{\ell} d^i$ to compute paths for the team of agents. It is clearly computationally advantageous to keep the degree d of the graph low and the prediction horizon ℓ short. One can also reduce the total number paths to evaluate by doing some reasonable pruning on the set of candidate paths $\mathcal{P}_a$, such as eliminating backtracking and paths with sharp turns that the agents cannot physically follow. Here, we have shown computation results for an exhaustive search over the prediction horizon, but it should be noted that algorithms such as branch and bound [5] may further reduce complexity.

F. Searching for Multiple Targets

Although we presented the CGBMPS algorithm in the context of a search for a single target, the algorithm is easily adapted to work for multiple targets. The main difference in implementation is that instead of using a TPMM to model the pdf of one target, one should model the combined pdfs of each target or a target occupancy map as described in [11]. Simulations in Section V-B show that the CGBMPS algorithm performs well on multiple-target searches. We leave for future work the problem of determining multiple-target discovery results analogous to the single-target

results of Section IV.

IV. Persistent Search

In this section we show that the CGBMPS algorithm results in finite-time target discovery with probability one. We use the notion of a *persistent search policy*, which is inspired by the persistent pursuit policies discussed in [10].

A search policy μ, as defined in Section II-E is said to be *persistent* if there exists some $\epsilon > 0$ such that

$$P\big(\mathcal{D}_{\text{team}}(k)|\mathbf{x}(k), \mathbf{w}(k), \mathbf{p}(k), \mathbf{q}(k)\big) > \epsilon \qquad \forall k \in \mathbb{Z}^{\geq 0},$$

is satisfied for the positions $\mathbf{p}(k)$ and sensor tasks $\mathbf{q}(k)$ that are generated by the search policy μ. In other words, the probability of locating the target at time k, given that it was not found previously, is always greater than ϵ. While it may be difficult for a search policy to satisfy this property, we can guarantee the following slightly weaker property when the agents are guaranteed to collect positive reward over some finite time horizon. A search policy is said to be *persistent on the average* if there is some $\epsilon > 0$ and some $T \in \mathbb{N}$ such that

$$\sum_{i=0}^{T-1} P\big(\mathcal{D}_{\text{team}}(k+i)|\mathbf{x}(k+i), \mathbf{w}(k+i), \mathbf{p}(k+i), \mathbf{q}(k+i)\big) > \epsilon \qquad \forall k \in \mathbb{Z}^{\geq 0}, \tag{25}$$

is satisfied for the positions $\mathbf{p}(k)$ and sensor tasks $\mathbf{q}(k)$ that are generated by the search policy μ. The time T is called the *period of persistence*.

Let $F_\mu(k) := \mathbf{P}(T^* \leq k \,|\, \mu)$ denote the distribution function of T^* given the search policy μ. We can alternatively write this as

$$F_\mu(k) = \sum_{\kappa=1}^{k} r(\kappa) = 1 - \prod_{\kappa=1}^{k} (1 - r(\kappa)).$$

The following lemma is proved in [10].

Lemma 2. For a persistent on the average search policy μ with period T, $\mathbf{P}(T^* < \infty \,|\, \mu) = 1$, $F_\mu(k) \geq 1 - (1-\epsilon)^{\lfloor \frac{k}{T} \rfloor} \;\; \forall k \in \mathbb{Z}^{\geq 0}$, and $E_\mu[T^*] \leq T\epsilon^{-1}$, with ϵ given in (25).

We now state the main result on finite-time target discovery, which proves that the CGBMPS algorithm terminates with probability one.

Theorem 1. *The CGBMPS algorithm results in target discovery in finite time with probability one and*

$$E[T^*] \leq \frac{T}{\gamma^T \tau \rho_{min}(A_{\mathrm{FOV}}/A_{\mathrm{FOR}})},$$

where

γ *is defined by Assumption 1,*

T, *defined in Lemma 1, is the period of persistence,*

τ *is the threshold selected in Step 2 of the Graph Construction Process, and*

ρ_{min} *is the lower bound on the probability of detection.*

Proof: From Lemma 1, we conclude that the search policy generated by the CGBMPS algorithm, using optimal sensor schedules, satisfies

$$\sum_{\kappa=k}^{k+T-1} P\big(\mathcal{D}_{\text{team}}(\kappa)|\mathbf{x}(\kappa), \mathbf{w}(\kappa), \mathbf{p}(\kappa), \mathbf{q}(\kappa)\big) \geq \gamma^T \tau \rho_{\min}(A_{\mathrm{FOV}}/A_{\mathrm{FOR}}) \left(1 - \sum_{\kappa=0}^{k} r(\kappa)\right).$$

At time $k = 0$ we have

$$\sum_{\kappa=0}^{T-1} P\big(\mathcal{D}_{\text{team}}(\kappa)|\mathbf{x}(\kappa), \mathbf{w}(\kappa), \mathbf{p}(\kappa), \mathbf{q}(\kappa)\big) \geq \gamma^T \tau \rho_{\min}(A_{\mathrm{FOV}}/A_{\mathrm{FOR}}).$$

From the definition given in (25), we conclude that the search policy is persistent on the average, with period T and

$$\epsilon = \gamma^T \tau \rho_{\min}(A_{\mathrm{FOV}}/A_{\mathrm{FOR}}).$$

Theorem 1 then follows directly from Lemma 2 with ϵ as defined above. ■

V. SIMULATIONS

The CGBMPS algorithm was coded in MATLAB and simulations were performed to evaluate the performance and scalability of the search. The algorithm was then coded in C++ to test against Toyon's SLAMEM® simulation, a high-fidelity entity-level simulation environment.

A. MATLAB Simulations

The CGBMPS algorithm was initially tested in MATLAB. A MATLAB prototype algorithm was created to simulate a cooperative search scenario with four agents searching for a target in a 5 km by 5 km square region. The FOR of each agent was selected to be a circle 1 km in diameter and the FOV a circle 200 m in diameter. All agents start from the same initial position, move at a constant speed of 10 m/s and took measurements every 5 seconds. The initial target pdf consisted of a weighted sum of three randomly placed Gaussian distributions with random covariances, and the TPMM was constructed using a particle filter as described in Appendix A. For path planning, we used a degree-3 hexagonal lattice graph, which is dynamically updated using the process described in Section 3, and the agents used a three-step prediction horizon. Figure 4 shows snapshots of two simulations using different representations of the target pdf. Notice how the graph construction allows the agents to plan paths across low-interest regions to regions of high target likelihood that would not be reachable in three steps on a uniform lattice graph.

The first set of Monte Carlo tests is designed to test the effectiveness of the cooperation between agents. Table II shows the average discovery times of the CGBMPS algorithm for teams of one to four cooperating agents. Notice that two agents find the target in just over half the time it takes one agent, and three agents find the target in about one third of the time. In this particular scenario, the addition of a fourth agent is nearing the point of diminishing returns, but in general, the algorithm makes efficient use of agents. This efficiency can be attributed to the sharing of target pdf information between agents.

Table III compares the performance of the CGBMPS algorithm with previously proposed search algorithms, each using three agents. The greedy search algorithm in line 1 chooses one-

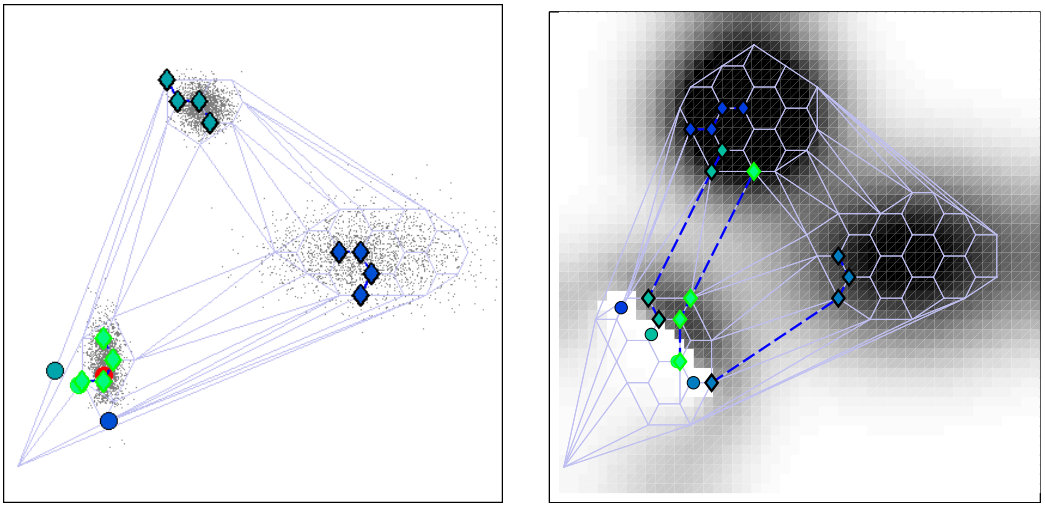


Fig. 4. **Snapshots of the MATLAB search simulation using a particle filter (left) and a probabilistic map (right). The agents are represented by the solid circles, and the diamonds are their waypoints.**

TABLE II
AVERAGE TIME TO TARGET DISCOVERY FOR TEAMS OF 1-4 AGENTS, AVERAGED OVER 500 RUNS.

Number of Agents (M)	**Average Time to Discovery (s)**
1	1401
2	713
3	486
4	440

step paths on a static square lattice graph that maximize the probability of finding the target. This is equivalent to giving each agent the choice of moving straight, left, or right at fixed time intervals and is similar to the greedy pursuit policies proposed in [10]. The results of line 2 were generated using a three-step receding horizon algorithm on a static square lattice graph, which is similar to algorithms described in [19], [24], and [7]. Lines 3 and 4, when compared to lines 1 and 2, show that the CGBMPS algorithm performs significantly better than the other algorithms that use a fixed time horizon over static graphs with edges of uniform length. The key advantage of the CGBMPS algorithm is its ability to evaluate longer high-reward paths than the fixed-step methods, but with roughly the same amount of computation. Furthermore, the method of joint routing and sensor tasking resulted in almost 10 percent faster target discovery than fixed sensor

tasking.

TABLE III
ALGORITHM PERFORMANCE COMPARISON, RESULTS AVERAGED OVER 500 RUNS

Search Method (with 4 agents)	Average Time to Discovery (s)
Greedy search	1092
3-step fixed RH Search	846
CGBMPS (fixed sensor)	476
CGBMPS (joint sensor optimization)	432

B. SLAMEM Simulations

We also implemented the CGBMPS algorithm in Toyon's high-fidelity SLAMEM simulation environment. SLAMEM contains detailed models for ground targets, surveillance platforms, sensors, attack aircraft, UAVs, data exploitation, multi-source fusion, sensor retasking, and attack nomination. SLAMEM models road networks, foliage cover, wind, buildings, and terrain (using the terrain elevation data – DTED). Testing in SLAMEM exercised the CGBMPS algorithm against detailed, realistic scenarios with multiple UAVs and targets, and helped us prepare the algorithm for flight testing.

SLAMEM has been used as the center for our system development on the hardware implementation of the CGBMPS algorithm. Our control and data fusion algorithms work with SLAMEM sensor and communications models to emulate a complete UAV autonomous control system. Video sensor exploitation models produce measurements to determine the presence of objects in a region, and communications models provide realistic data link constraints on the network to determine the feasibility of our decentralized processing and control architecture. SLAMEM asset models accurately represent the Unicorn UAV platforms and other standard military platforms in the simulation and execute commands from our control algorithms. GVS^{TM} supplies realistic targets of interest to the simulation.

In the SLAMEM simulations, one or more UAVs were autonomously controlled by the CGBMPS algorithm to locate one or more targets in a specified area of interest (AOI). The

parameters for the simulations match closely with the parameters of the hardware tests (cf. Section VI). The search region (AOI) spanned 10.5 km^2 of gently sloping terrain. The UAV airspeed was set to approximately 50 km/hr, and the flight altitude was fixed to 100m – 120m above ground level. At this altitude, the sensor FOV footprint on the ground covers approximately 0.1 km^2, meaning that the UAVs see only 1% of the search region with each camera view. A screenshot of the CGBMPS SLAMEM simulation appears in Figure 5.

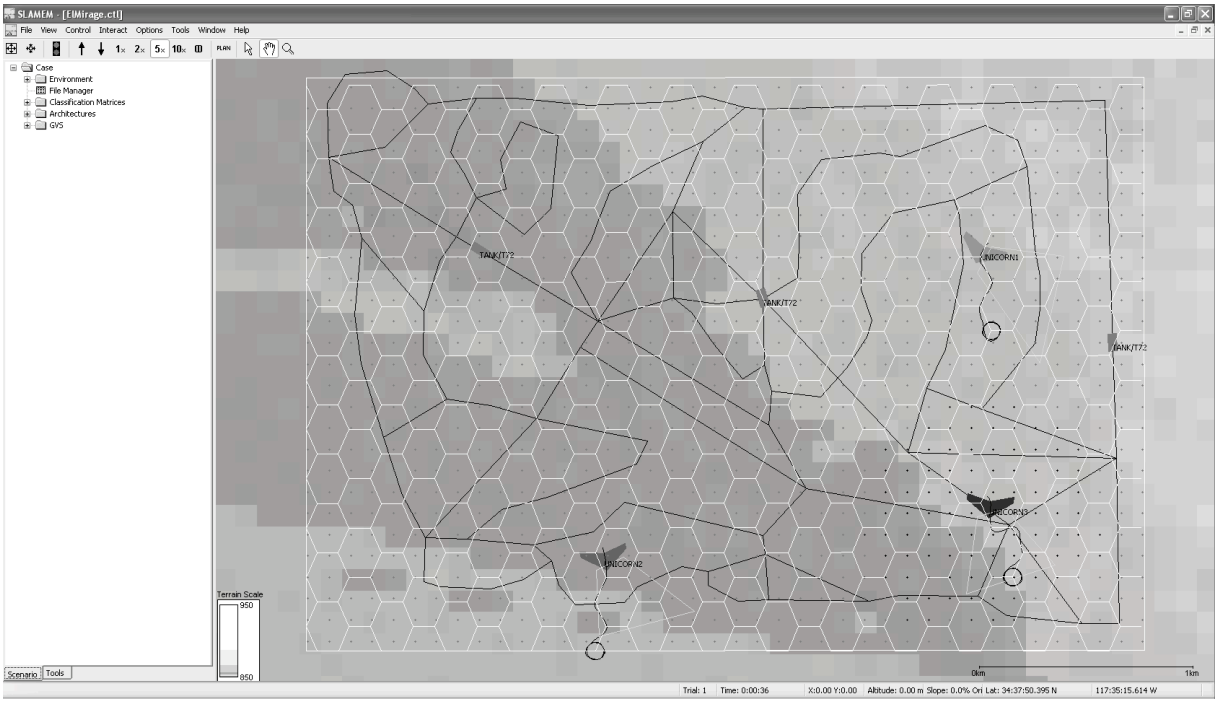


Fig. 5. **A snapshot of the CGBMPS operating in the SLAMEM simulation environment**

Simulation results measured T^*, the amount of time it took the UAV or UAV team to detect the target(s). In each trial, T^* results were averaged over 100 Monte Carlo trials. Several variations of UAV teams, ground targets, and search algorithms were simulated. In these tests, because each Monte Carlo run required a large computation effort, we only compared CGBMPS with algorithms that we knew would perform very well for the specific model used for the target motion.

▷ **Random Search.** As a UAV approaches a waypoint, the Random Search algorithm selects

the next UAV waypoint at random within the AOI. This algorithm tends to send the UAV through the middle of the AOI very often. By matching the random target motion with the motion of the random searchers, we obtained an algorithm that is quite effective. In practice such matching could not be done for real targets.

▷ **Raster Search.** The Raster Search algorithm scans the entire AOI side-to-side, bottom-to-top, as in Figure 6. This is actually the optimal search algorithm to discover a stationary target when the initial target probability distribution in uniform over the search region.

▷ **CGBMPS.** The algorithm described in this paper. This algorithm was tested with one, three, and five-step prediction horizons.

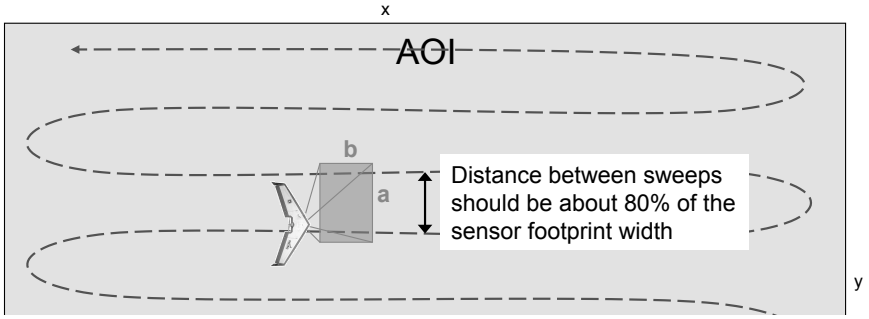


Fig. 6. **UAV path for the Raster Search**

These results appear in Table IV. It is important to note the search results shown in Tables IV and V depend on the target motion models. Targets moving randomly along the road network tend to spend more time in the middle of the AOI, making them more easily detected by the Random Search. A different target motion model that kept targets more stationary, or moving near the perimeter of the AOI, would give more advantage to the Raster Search. Our results are very encouraging because they show that the CGBMPS algorithm compares favorably with algorithms that explore specific motion models for the target. In contrast, the CGBMPS algorithm will perform well with nearly any target motion model because of its versatile target state estimation framework (Section II-C).

Table V shows the results for multiple UAVs and multiple targets. One can observe that each subsequent target takes longer too find, and that the team of three agents cooperates effectively.

TABLE IV
THE FOLLOWING TABLE SHOWS SIMULATION RESULTS FOR SEARCH ALGORITHMS RUNNING IN SLAMEM. THE NUMERIC ENTRIES SHOW T^* IN MINUTES, WHERE T^* = THE TIME REQUIRED TO DETECT A SINGLE TARGET, AVERAGED OVER 100 MONTE CARLO TRIALS.

Search Algorithm	$T^* =$ **time to target detection in minutes**
Random	21.1
Raster scan	25.7
CGBMPS, Dynamic graph, 1-step	24.7
CGBMPS, Dynamic graph, 3-step	18.7
CGBMPS, Dynamic graph, 5-step	18.7

While, in this case, three UAVs found the targets on average more than three times faster than a single UAV, this can be attributed to advantageous starting positions for the second and third UAVs.

TABLE V
THIS TABLE SHOWS SLAMEM SIMULATION RESULTS FOR THE CGBMPS ALGORITHM WITH VARYING NUMBERS OF UAVS AND TARGETS, AVERAGED OVER 100 MONTE CARLO TRIALS.

# UAVs	**# targets**	**Time to (1^{st}, 2^{nd}, 3^{rd}) target detection in minutes**
3	1	7.5
1	3	7.0, 26.2, 55.2
3	3	1.8, 5.7, 14.1

VI. HARDWARE IMPLEMENTATION

The CGBMPS algorithm has been successfully field-tested using both Unicorn [27] and Raven [20] UAV platforms. This section describes the hardware-software system setup and implementation used for testing with Unicorns, Toyon's UAV test platform. We also present some results from the field tests.

To facilitate hardware-in-the-loop (HIL) testing of the search algorithm, the SLAMEM simulation environment was modified to allow integration of real UAVs and ground vehicles with other simulated assets and vehicles, all as part of one complex scenario [3]. SLAMEM communicates

with Virtual Cockpit^{TM}[28], the UAV ground control software, to receive position updates for the real UAVs and then render them in simulation. SLAMEM can also receive GPS updates from a real target and render this target in simulation. In this way, SLAMEM can act as the HIL display and control interface for the real UAVs in the air and real vehicles on the ground. Meanwhile, SLAMEM can also simulate additional assets and targets. The real entities and simulated entities may all interact within SLAMEM, facilitating mixed HIL+simulation scenarios, as shown in Figure 7.

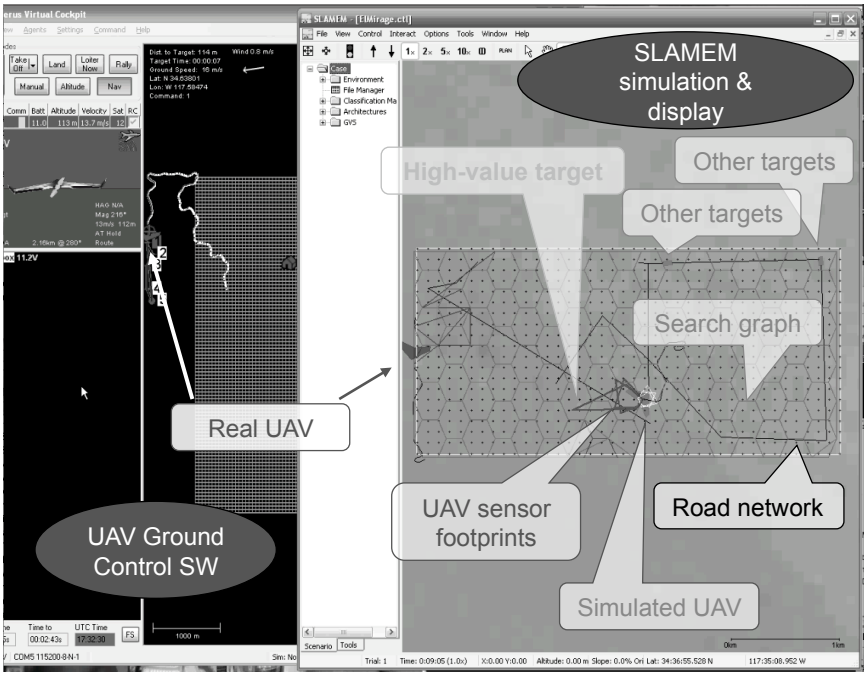


Fig. 7. **An annotated laptop screen capture showing SLAMEM and Virtual Cockpit running on our ground control station. Virtual Cockpit (on the left) displays real-time UAV statistics and prior flight paths. SLAMEM (on the right) shows real and simulated UAVs, real and simulated targets, roads, and other control parameters.**

A. UAV platform

Our primary test platform is the Unicorn UAV (Figure 8). This platform is based on the Unicorn expanded polypropylene (EPP) foam wing [27]. This is a radio controlled, 60" EPP foam wing powered by four lithium-polymer battery packs driving an electric motor attached to a propeller. This power-train propels the UAV at about 50 km/hr (airspeed). On a full charge, the batteries will provide roughly one hour of flight time. Our Unicorn UAVs also house the

Kestral^{TM}Autopilot control board, a gimbaled video camera, two radio modems, batteries, and wiring to connect the various components. The UAV uses separate channels for the telemetry data and the video stream, thus necessitating two radio modems.

Fig. 8. **Unicorn wing outfitted with Procerus® Kestrel^{TM} autopilot system and gimbaled video camera**

B. Gimbaled Video Camera

A small 480 line, 5volt, CCD, NTSC video camera is attached to a two degree of freedom gimbal, mounted on the bottom of each UAV. The camera and gimbal can be seen in figure 9. The elevation has unconstrained movement in the range $-90^\circ - 0^\circ$. The azimuth is limited to movement in the range $-135^\circ - 45^\circ$ (with 0° looking straight out the nose of the UAV). The camera has a fixed zoom that can be changed by replacing the camera lens.

The UAVs are typically flown at 100m altitude, above ground level (AGL). Intersecting the camera FOV with the earth from this altitude yields a FOV footprint on the ground covering about 0.1 km^2. Analog video from the camera is transmitted wirelessly to a receiver on the ground.

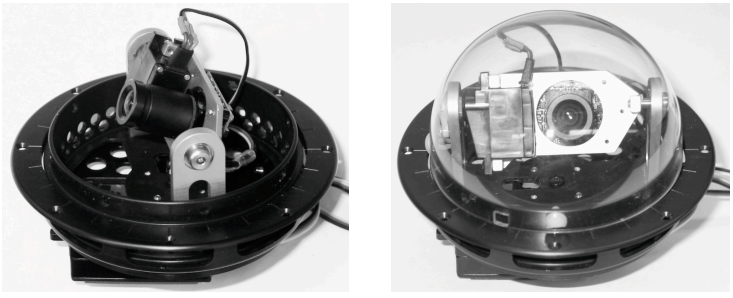


Fig. 9. **Miniature video camera and gimbal mechanism.**

C. Kestral Autopilot

Low level control logic is handled by the Kestrel Autopilot [12]. Navigation controls for the UAV are communicated to the Autopilot in the form of *waypoint* commands: geodetic locations above the ground in latitude/longitude/altitude (above the geoid). Gimbal targeting commands can be sent either in azimuth/elevation (relative to the platform), or as a latitude/longitude/altitude in an earth-based coordinate frame. The system also allows a limited degree of manual flight and gimbal control using a gamepad.

The CGBMPS algorithm controls the UAV by selecting waypoints and sending them to the autopilot. These waypoints come from nodes in the search graph G (Section III-A), and are selected by the optimization in (22) or (23).

D. Field Test Results

A team of two Unicorns were autonomously controlled by the CGBMPS algorithm to cooperatively discover and locate targets in a 10.5 km^2 region. The UAVs search using dynamically updated probabilistic maps and particle filters. Each UAV controller runs a separate instance of the CGBMPS algorithm, and each instance used estimator-based decentralized control to estimate current states and future search actions for the other UAV. Cooperation occurs by UAVs sharing their states and predicting each other's sensor actions.

One of the UAVs was real, while the other was simulated in SLAMEM. The demo included

- 1 real target (a pickup truck carrying a GPS unit to render it in SLAMEM),
- 2 simulated targets,
- 1 real Unicorn UAV, and
- 1 simulated Unicorn UAV.

UAV airspeed was approximately 50 km/hr, flying at 100m AGL. The sensor FOV footprint on the ground covers approximately 0.1 km^2, meaning that the UAVs see only 1% of the search region with each camera view. All target motions were random, including the motions of the real target. The first target detection occurred by the real UAV at 15:54 into the trial; the second target detection was by a simulated UAV at 22:17.

The following Figures show screenshots that were generated using recorded data from the field tests of the CGBMPS algorithm.

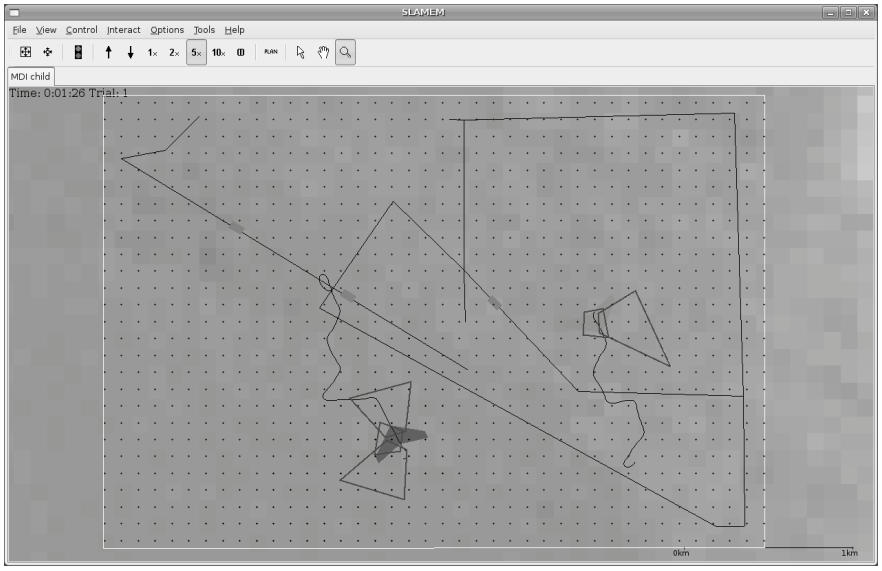


Fig. 10. **Two Unicorn UAVs begin a search of the AOI (white rectangle) for three targets (gray rectangles). Curved UAV search paths are shown as black lines connecting vertices in the search graph (not displayed). UAV sensor task footprints appear as quadrilaterals.**

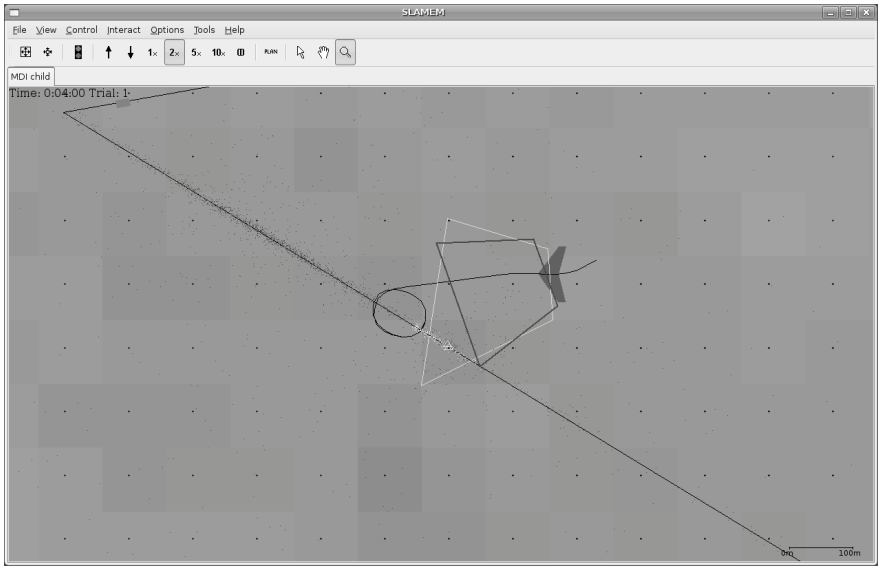


Fig. 11. **Once a target is detected and a track on that target has been instantiated, a particle filter is used to update and predict the track state. The particle cloud appears in the figure as a cluster of dots near the road. The UAV has transitioned from search to bird-dogging, and SLAMEM shows it's future path as a loop around the predicted position of the target.**

VII. Conclusions and Future Work

We have presented a cooperative search algorithm that uses receding horizon optimization on a dynamically updated graph and achieves finite-time target discovery with probability one. The novel contributions of this algorithm are (1) optimizing on a dynamic graph updated based on changing target probability distribution, (2) using a waypoint-based prediction horizon allowing for comparison between paths of various lengths, and (3) incorporating the use of gimbaled sensors whose orientations can be jointly optimized with the paths of the agents. Simulations showed that the waypoint-based prediction horizon with a strategic placement of graph vertices significantly improved search algorithm performance over traditional implementations having a fixed time horizon on graphs with edges of uniform length. Using joint routing and sensor optimization provides an additional increase in performance. Furthermore, we have successfully tested the CGBMPS algorithm on a physical system consisting of two UAVs with gimbal-mounted cameras.

The decentralized version of the algorithm, in which each agent maintains estimates on the positions and sensor measurements of its teammates and updates these estimates whenever it receives information from other agents, warrants further study. However, the communication bandwidth requirements of the centralized algorithm we have presented are relatively low. The central computer only needs to send waypoints (latitude,longitude,altitude) to the UAVs, amounting to no more than a few hundred bytes. The only data returned from the UAVs are the positive detections, since negative detections are assumed whenever no data is received.

The frequency with which one must upload these waypoint packets depends on the coarseness of the search graph. Our graph construction process specifies a vertex spacing approximately twice the diameter of the sensor FOR. For a Raven UAV, this distance is about 400m. Flying at a nominal speed of about 15 m/s, this leaves about 26 seconds between waypoints. Before the UAV reaches each waypoint, a new waypoint or multi-waypoint path should be uploaded. So even if the search algorithm determined a complete flight path change at each recalculation, the total communication amounts to a few hundred bytes every 26 seconds.

Moreover, for UAVs, low-power long-range communication is available because one generally

has line of sight. For example, in another project [13], we are using relatively weak radios that still have 60 miles range – this is possible because we have of line-of-sight from UAV to UAV and because in this work we never need to transmit large volumes of data.

VIII. Acknowledgements

The authors would like to thank Craig Agate for assistance with target state estimation and probability modeling.

Appendix: Target Probability Mixture Model Examples

A. Example 1: Particle Filter.

Particle filtering is a sequential Monte Carlo-based method for state estimation that is especially well-suited to systems with non-linear dynamics and non-Gaussian probability distributions. It involves modeling a system with a large number of dynamic "particles" whose states evolve according to some stochastic model. Weights are typically updated upon the arrival of new measurements and are then normalized so that the total weight of the particles is equal to one. For a particle filter that estimates the location of a mobile target, the probability that the target is located within some region of the state space can be estimated by summing the weights of the particles lying in that region. Hence regions in which the particles are densely spaced represent areas with high target likelihood, while low density regions indicate low target likelihood. Following is a generalized model of a simple particle filter.

Let $\mathbf{x}(k) = [x_1(k), x_2(k), \ldots, x_n(k)]$ represent the positions of the particles whose corresponding weights are given by $\mathbf{w}(k) = [w_1(k), w_2(k), \ldots, w_n(k)]$. The system dynamics can be expressed as follows:

$$\text{POSITION UPDATE: } x_i(k+1) = f_x(x_i(k), v_i(k)), \tag{26}$$

$$\text{WEIGHT UPDATE: } \tilde{w}_i(k+1) = w_i(k) \prod_{a=1}^{M} \big(1 - D(p_a(k), q_a(k), x_i(k))\big) \tag{27}$$

$$\text{RE-NORMALIZATION: } \mathbf{w}(k+1) = \tilde{\mathbf{w}}(k+1) \frac{1}{\sum_{i=1}^{n} \tilde{w}_i(k+1)}, \tag{28}$$

where $\tilde{\mathbf{w}}(k) := (\tilde{w}_1(k), \ldots, \tilde{w}_n(k))$ is an intermediate vector of unnormalized target weights, and $\mathbf{v}(k) := (v_1(k), \ldots, v_n(k))$ is a process noise sequence. The function f_x is used to propagate the particle positions according to a model of the target dynamics, randomized with $\mathbf{v}(k)$. For our purposes, the noise sequence $\mathbf{v}(k)$ is known by all agents. In this case, the weight update equation (27) uses the fact that the delta function defined in (3) takes the form of a Dirac delta function.

We omit from this section a discussion of specific particle filter implementations in favor of a more general model. However, much of the power of particle filter estimation lies in the refined sampling and resampling techniques used in the more advanced models. We refer the reader to [1] for a survey of several particle filtering methods.

B. Example 2: Grid-based Probabilistic Map

A grid-based probabilistic map is another method for estimating the states of an uncertain system. It involves representing a system's state-space by a grid of cells, each of which has a weight corresponding to the probability of the target being located inside that cell. The cell weights are updated based on incoming measurements and predicted future states. Following is a generalized implementation of a probabilistic map.

Let $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ represent the static center points of the cells, whose corresponding weights are given by $\mathbf{w}(k) = [w_1(k), w_2(k), \ldots, w_n(k)]$. The system dynamics can be expressed by:

$$\text{WEIGHT UPDATE: } \tilde{w}_i(k+1) = w_i(k) \int_{\mathcal{R}} \delta(x_i - x) \prod_{a=1}^{M} \big(1 - D(p_a(k), q_a(k), x(k))\big) dx \quad (29)$$

$$\text{DIFFUSION: } \hat{\mathbf{w}}(k+1) = f_w(\tilde{\mathbf{w}}(k+1)), \quad (30)$$

$$\text{RE-NORMALIZATION: } \mathbf{w}(k+1) = \hat{\mathbf{w}}(k+1) \frac{1}{\sum_{i=1}^{n} \hat{w}_i(k+1)} \quad (31)$$

where $\tilde{\mathbf{w}}(k)$ and $\hat{\mathbf{w}}(k)$ are intermediate vectors of target weights. For this grid-based method, the delta function $\delta(\cdot)$ is a two-dimensional rectangular impulse function the size of one grid cell. The function f_w can be used to diffuse target weights between adjacent cells according to a

transition probability matrix. Note that since the x_i are fixed in a grid-based probabilistic map, there is no position update. See [11] for a more detailed description.

REFERENCES

[1] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2), 2002.

[2] P. Chandler and M. Pachter. Hierarchical control for autonomous teams. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2001.

[3] G. E. Collins, P. S. Vegdahl, and J. R. Riehl. A mixed simulation and hardware-in-the-loop display and controller for autonomous sensing and navigation by unmanned air vehicles. In Kevin Schum and Dawn A. Trevisani, editors, *Modeling and Simulation for Military Operations II*, volume 6564, page 65640X. SPIE, 2007.

[4] B. DasGupta, J. P. Hespanha, J. R. Riehl, and E. Sontag. Honey-pot constrained searching with local sensory information. *Nonlinear Analysis: Hybrid Systems and Applications*, 65(9):1773–1793, Nov. 2006.

[5] J. Eagle and J. Yee. An optimal branch-and-bound procedure for the constrained path, moving target search problem. *Operations Research*, 28(1), 1990.

[6] J. R. Frost and L. D. Stone. Review of search theory: Advances and applications to search and rescue decision support. Technical report, U.S. Coast Guard Research and Development Center, Groton, CT, Sept. 2001.

[7] T. Furukawa, F. Bourgault, B. Lavis, and H. F. Durrant-Whyte. Recursive bayesian search-and-tracking using coordinated uavs for lost targets. In *Proceedings of the 38th IEEE Conference on Robotics and Automation*, pages 2521–2526, 2006.

[8] P. Gaudiano, B. Shargel, E. Bonabeau, and B. Clough. Control of uav swarms: What the bugs can teach us. In *Proceedings of the 2nd AIAA "Unmanned Unlimited" Systems, Technologies, and Operations–Aerospace, Land, and Sea Conference and Workshop*, 2003.

[9] K. B. Haley and L. D. Stone, editors. *Search Theory and Applications*. Plenum Press, New York, 1980.

[10] J. P. Hespanha, H. J. Kim, and S. Sastry. Multiple-agent probabilistic pursuit-evasion games. In *Proc. of the 38th Conf. on Decision and Contr.*, volume 3, pages 2432–2437, December 1999.

[11] J. P. Hespanha and H. Kızılocak. Efficient computation of dynamic probabilistic maps. In *Proceedings of the 10th Mediterranean Conference on Control and Automation*, 2002.

[12] Kestrel Autopilot. http://procerusuav.com/productsKestrelAutopilot.php.

[13] D. Klein, J. Isaacs, S. Venkateswaran, J. Burman, T. Pham, J. P. Hespanha, and U. Madhow. Source localization in a sparse acoustic sensor network using uav-based semantic data mules. Submitted to SenSys'10, Apr 2010.

[14] B. O. Koopman. *Search and Screening*. Operations Evaluations Group Report No. 56, Center for Naval Analyses, Alexandria, VA, 1946.

[15] M. Mallick. Geolocation using video sensor measurements. In *Proceedings of the 10th International Conference on Information Fusion*, 2007.

[16] M. Mangel. Search theory: A differential equations approach. In Chudnovsky and Chudnovsky, editors, *Search Theory: Some Recent Developments*, pages 55–101. Marcel Dekker, New York, 1989.

[17] G. J. McLachlan and D. Peel. *Finite Mixture Models*. Wiley, 2000.

[18] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* Wiley, New York, 1992.

[19] M. M. Polycarpou, Y. Yang, and K. M. Passino. A cooperative search framework for distributed agents. In *Proc. of the IEEE International Symposium on Intelligent Control*, 2001.

[20] Raven UAV. http://www.globalsecurity.org/intell/systems/raven.htm.

[21] J. R. Riehl and J. P. Hespanha. Fractal graph optimization algorithms. In *Proceedings of the 44th Conference on Decision and Control*, 2005.

[22] J. Schlecht, K. Altenburg, B. M. Ahmed, and K. E. Nygard. Decentralized search by unmanned air vehicles using local communication. In *Proceedings of the International Conference on Artificial Intelligence, Las Vegas, NV*, pages 757–762, 2003.

[23] L. D. Stone. *Theory of Optimal Search*. Academic Press, New York, 1975.

[24] T. H. Chung and J. W. Burdick. A Decision-Making Framework for Control Strategies in Probabilistic Search. In *Intl. Conf. on Robotics and Automation*. ICRA, April 2007.

[25] J. Tisdale, A. Ryan, Z. Kim, D. Tornqvist, and J. K. Hedrick. A multiple uav system for vision-based search and localization. In *Proceedings of the American Control Conference*, 2008.

[26] K. E. Trummel and J.R. Weisinger. The complexity of the optimal searcher path problem. *Operations Research*, 34(2):324–327, 1986.

[27] Unicorn Flying Wing. http://unicornwings.stores.yahoo.net/.

[28] Virtual Cockpit. http://procerusuav.com/productsGroundControl.php.

James Riehl received a B.S. degree in Engineering from Harvey Mudd College in 2002 and went on to study Control Systems at the University of California, Santa Barbara. There, under the advisement of Professor Joo Hespanha, he developed several new algorithms for approximating the solutions to computationally difficult graph-based optimization problems with applications to cooperative search and path planning. He received his M.S. degree in 2004 and Ph.D. in 2008 from the Department of Electrical and Computer Engineering. Also while at UCSB, he consulted with Toyon Research Corporation on cooperative search by UAV teams and helped implement and test the algorithms. Since the Fall of 2007, Dr. Riehl has worked for AT&T Government Solutions where he has conducted research in fields ranging from network optimization to machine learning.

Gaemus Collins is a Senior Analyst at Toyon research Corporation. He received a B.S degree in mathematics from Salisbury University in 1996, and subsequently received M.S. and Ph.D. degrees in mathematics from the University of California, Santa Barbara (UCSB) in 1999 and 2002 respectively. Working under Mihai Putinar, Dr. Collins' graduate research proved new bounds for the numerical range of the Orr-Sommerfeld and Squire operators, which model viscous fluid flow in a parallel channel. Following his Ph.D., Dr. Collins held a two-year post-doctorate position at University of California, San Diego (UCSD). Working with Bill McEneaney, Dr. Collins studied Hamilton-Jacobi equations using max-plus algebraic techniques, and developed a state-feedback dynamic program for DARPA's Mixed Initiative Control for Automa-teams (MICA) program. As a senior analyst at Toyon Research Corp., Dr. Collins is an algorithm designer and program manager for several government-funded programs developing cooperative multi-agent, multi-sensor UAV search and tracking systems.

João P. Hespanha was born in Coimbra, Portugal, in 1968. He received the Licenciatura

in electrical and computer engineering from the Instituto Superior Tcnico, Lisbon, Portugal in 1991 and the Ph.D. degree in electrical engineering and applied science from Yale University, New Haven, Connecticut in 1998. From 1999 to 2001, he was Assistant Professor at the University of Southern California, Los Angeles. He moved to the University of California, Santa Barbara in 2002, where he currently holds a Professor position with the Department of Electrical and Computer Engineering. Prof. Hespanha is Associate Director for the Center for Control, Dynamical-systems, and Computation (CCDC), Vice-Chair of the Department of Electrical and Computer Engineering, and a member of the Executive Committee for the Institute for Collaborative Biotechnologies (ICB). From 2004-2007 he was an associate editor for the IEEE Transactions on Automatic Control. His current research interests include hybrid and switched systems; the modeling and control of communication networks; distributed control over communication networks (also known as networked control systems); the use of vision in feedback control; and stochastic modeling in biology. Dr. Hespanha is the recipient of the Yale University's Henry Prentiss Becton Graduate Prize for exceptional achievement in research in Engineering and Applied Science, a National Science Foundation CAREER Award, the 2005 best paper award at the 2nd Int. Conf. on Intelligent Sensing and Information Processing, the 2005 Automatica Theory/Methodology best paper prize, the 2006 George S. Axelby Outstanding Paper Award, and the 2009 Ruberti Young Researcher Prize. Dr. Hespanha is a Fellow of the IEEE and an IEEE distinguished lecturer since 2007.