

A UAV Routing and Sensor Control Optimization Algorithm for Target Search

Gaemus E. Collins^a, James R. Riehl^b, and Philip S. Vegdahl^a

^aToyon Research Corporation, 6800 Cortona Dr., Goleta, CA, USA;

^bDepartment of Electrical and Computer Engineering (ECE), University of California, Santa Barbara, CA, USA

ABSTRACT

An important problem in unmanned air vehicle (UAV) and UAV-mounted sensor control is the target search problem: locating target(s) in minimum time. Current methods solve the optimization of UAV routing control and sensor management independently. While this decoupled approach makes the target search problem computationally tractable, it is suboptimal.

In this paper, we explore the target search and classification problems by formulating and solving a joint UAV routing and sensor control optimization problem. The routing problem is solved on a graph using receding horizon optimal control. The graph is dynamically adjusted based on the target probability distribution function (PDF). The objective function for the routing optimization is the solution of a sensor control optimization problem. An optimal sensor schedule (in the sense of maximizing the viewed target probability mass) is constructed for each candidate flight path in the routing control problem.

The PDF of the target state is represented with a particle filter and an “occupancy map” for any undiscovered targets. The tradeoff between searching for undiscovered targets and locating tracks is handled automatically and dynamically by the use of an appropriate objective function. In particular, the objective function is based on the expected amount of target probability mass to be viewed.

Keywords: Unmanned Air Vehicle (UAV); Cooperative control; Sensor management; Model predictive control; Graph theory; Particle filtering; Information, Surveillance, and Reconnaissance (ISR)

1. INTRODUCTION

We consider a target search problem in which a team of autonomous search agents (e.g. UAVs) are searching a bounded region for one or more mobile targets. The target states are unknown to the searchers, but each agent is initialized with a target state estimate. Search agents are equipped with a gimbaled sensor that they can aim at a various points in the search region. The sensor can detect targets with some probability of detection < 1 if the targets fall within the sensor’s field of view. As agents move around and make sensor measurements, they gather information about the likely locations of targets, which they can communicate to other agents over a possibly low-bandwidth and lossy network. We further suppose that the agents have a cost constraint, such as limited amount of fuel or time to complete the search mission. The objective of each agent is to move itself and control its sensor in a way that maximizes the team’s probability of finding the targets, subject to this cost constraint. In this paper, we introduce a graph-based receding horizon search algorithm to approximately solve this problem.

Further author information: (Send correspondence to G.E.C.)

G.E.C: E-mail: gcollins@toyon.com, Telephone: 805-968-6787

J.R.R: E-mail: jriehl@ece.ucsb.edu

P.S.V: E-mail: pvegdahl@toyon.com, Telephone: 805-968-6787

Note: Emails from international addresses cannot be received by the Toyon server. International interests should call.

1.1. Search Theory

Search theory, as we know it today, began with work by Koopman¹(1946), Stone²(1975), and others. The work was initially motivated by the desire to develop efficient search methods to find enemy marine vessels. Agencies such as the U.S. Coast Guard have applied search theory to search and rescue missions with great success, measured in saved lives.³ Other search applications include exploration and surveillance.⁴

Early search theory focused on the allocation of search effort to *areas* within the search region. This problem formulation is appropriate for special cases in which finding optimal search paths on these areas is intuitive, or 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 subject to constraints. There are some studies of the search problem in continuous time and space,⁵ but these generally apply to a very restrictive class of problems. A more practical approach is to discretize the search space and formulate the search problem on a graph. Then the search path describes a sequence of regions to search, and an amount of time that should be spent in each region. Although this discretization simplifies the problem to some extent, it is still computationally very difficult. Trummel and Weisinger showed that the single-agent search problem is NP-Hard even for a stationary target.⁶ Eagle and Yee⁷ and Stewart⁸ formulated the moving target problem as a nonlinear integer program and proposed branch and bound algorithms to solve it. DasGupta *et al.* presented an approximate solution for the stationary target search based on an aggregation of the search space using a graph partition.⁹ We used a similar approach in Ref. 10, but with a fractal formulation, that is, the partitioning process could be implemented on multiple levels.

All of the results mentioned so far are for a single searcher. The problem becomes even more complex when we consider a team of agents that are cooperatively searching for a target. For this reason, literature on the cooperative search problem typically tries to find good approximate solutions. We extended our fractal decomposition results to the cooperative search problem in Ref. 11, but this result does not directly apply to the case of mobile targets. A natural 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.*¹² showed that in pursuit-evasion games, one step Nash policies resulted in finite-time evader capture with probability one. Using slightly longer horizons, Polycarpou *et al.* introduced a finite-horizon look-ahead approach similar to model predictive control where the searchers share information with each other.¹³ A potential disadvantage to these algorithms is that because all possible search paths are evaluated at each time step (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 an agent can cover on this horizon, the algorithm's performance may suffer.

1.2. Overview of Proposed Solution

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 selected graph nodes serve as candidate waypoints for the searchers. At each time step, the algorithm chooses the l -step future path ($l \in \mathbb{N}$) leading to the highest probability of target detection. These l -step paths may represent very short prediction horizons (in the case of tightly spaced graph nodes), or very long prediction horizons (in the case of sparsely distributed graph nodes). This dynamic graph structure facilitates two key strengths of our algorithm: (a) search agents only perform detailed searches of regions with high target probability, and (b) long prediction horizons can be efficiently evaluated and honestly compared to short prediction horizons. Multiple searchers cooperate by choosing paths that “divide and conquer” the search space, minimizing path overlap. This is facilitated by forcing each agent's plan to be conditioned on any existing plans for the other agents. This is not a fully cooperative solution, but it operates with far less computation resources than the fully cooperative solution, which has exponential complexity in the number of agents. Simulations show that this conditional cooperation makes effective use of assets.

Another important aspect of the search problem is *sensor control*. As the sensors mounted onboard UAVs have become more sophisticated and dynamically controllable, enabling the sensing neighborhood to be steered about the UAV platform, the UAV target search/ID problem has expanded from a 2- or 3-DOF optimization problem for the UAV position, to a 4-, 5-, or 6-DOF optimization problem for the UAV *and sensor* position.

While most search methods in the literature deal only with the UAV routing problem, this paper explores two variants of an algorithm that optimizes control for both the UAV routing and UAV sensor control.

The final goal of the search problem is to locate and/or identify targets in the search region. This goal is facilitated by repeatedly updating a target state estimate with new sensor measurements or empty sensor dwells. Many methods exist to maintain a target state estimate; in this paper we employ a general target state equation that can be used with a grid-based target occupancy map or a particle filter. Both methods can be used simultaneously to facilitate UAV search and tracking.

The paper is organized into five sections. Following the introduction, Section 2 presents a detailed formulation of the problem, including the discretization approach used in our algorithms. Section 3 outlines the algorithms used to solve the problem. This is followed by Section 4 that includes some preliminary simulation results. Section 5 concludes the paper and suggests directions of further research on the UAV target search problem.

2. PROBLEM FORMULATION AND DISCRETIZATION

Suppose a team of M agents are searching for N mobile targets in a bounded region \mathcal{R} of a plane. Each agent will be equipped with one gimbaled sensor that can be steered in both azimuth (Az) and elevation (El). The instantaneous sensing region of the sensor on the ground, known as the *field of view (FOV)*, will be modeled as an oval or quadrilateral “footprint,” as in Figure 1. This FOV will move around on the ground as a result of the motion the UAV platform and the steerable gimbal mechanism that houses the sensor. There are mechanical and practical limits on how far the sensor can be steered in Az or El. The angular region between these limits is the camera’s range of motion. The complete region on the ground that could be viewed by the camera as it is swept through its entire range of motion (while the UAV was stationary) is called the *field of regard (FOR)* of the sensor.

The number of targets (N) as well as the locations of the targets in \mathcal{R} are unknown to the UAV controller. Therefore, the controller will hypothesize and update a target state estimate, denoted by $\mathbf{x}(k) := [x_1(k), x_2(k), \dots, x_n(k)]$, where $k \in \mathcal{T} := \{0, k_1, k_2, \dots, T\}$, a set of discrete times at which all events take place. The target state estimate consists of position and weight components $\mathbf{x}(k) := [\mathbf{x}^p(k), \mathbf{x}^w(k)]$. The individual $x_i(k) = (x_i^p(k), x_i^w(k))$ may represent particles in a particle filter, gridpoints in a grid-based probabilistic map, or some other type of estimator. Specific methods of generating \mathbf{x} are treated in references such as Ref. 14 and Ref. 15. The requirements we impose on \mathbf{x} are

$$\sum_{i=1}^n x_i(k) = N \quad \forall k \in \mathcal{T}, \quad (1)$$

$$x_i^p(k) \in \mathcal{R} \quad \forall i \in \{1, \dots, n\}, \forall k \in \mathcal{T}. \quad (2)$$

Condition (1) can be achieved by a re-normalization at each time step. The initial target state estimate $\mathbf{x}(0)$ is based on any prior knowledge of target locations. Future state estimates depend on control inputs $\mathbf{u}(k)$ for each agent, previous state estimates, and a process noise sequence $\mathbf{w}(k)$:

$$\mathbf{x}(k) = f(\mathbf{x}(k-1), \mathbf{u}(k), \mathbf{w}(k)). \quad (3)$$

The function f models the evolution of the target probability distribution and must preserve conditions (1) and (2).

The state of the search team is given by $\mathbf{p}(k) := [p_1(k), p_2(k), \dots, p_M(k)]$, where $p_j(k) = (p_{j,x}(k), p_{j,y}(k)) \in \mathbb{R}^2$ is agent j ’s position at time step k . The FOR of an agent located at position $p_j(k)$ at time k is denoted by $FOR(p_j(k))$. The control inputs to the system at time k include controls for all agents: $\mathbf{u}(k) := [u_1(k), u_2(k), \dots, u_M(k)]$. The control input $u_j(k)$ for the j^{th} agent at time step k consists of the UAV heading and the point on the ground at which the UAV sensor points:

$$u_j(k) = (v_j(k), q_j(k)).$$

The UAV heading $v_j(k) = (v_{j,x}(k), v_{j,y}(k))$ will be specified in terms of a path or future waypoints (as described in Section 2.6), and the the UAV sensor control $q_j(k) = (q_{j,x}(k), q_{j,y}(k))$ is a point on the ground in \mathcal{R} where the

sensor should point at time k . The footprint or FOV of the sensor pointed at $q_j(k)$ is denoted by $FOV(q_j(k))$. A target appearing in $FOV(q_j(k))$ will have some probability of being correctly detected/ID'ed/tracked. We will generally refer to this value as *probability of detection* ($P_D(q_j(k))$) for all three applications, and further enforce that $P_D(q_j(k)) = 0$ for targets outside of $FOV(q_j(k))$.

In order to optimize how the agents and sensors should be moved, we define a *search reward* representing the probability of finding a target at a specific location and time. Let $r(k)$ denote a cumulative state of the collected search reward for agent a , with $r(0) = 0$ and

$$r(k+1) = g(\mathbf{x}(k), \mathbf{u}(k), r(k)) \quad \forall k > 0 \in \mathcal{T}; \quad (4)$$

the function g will be explained in specific detail in Section 2.4. With this designation, the search objective becomes

$$\max_{\mathbf{u} \in \mathcal{U}}(r_T), \quad (5)$$

where \mathcal{U} is the set of all admissible control sequences and T is the time limit on the search. Implicit in \mathcal{U} are the constraints imposed by the dynamics of the agent and sensor, that is, the control inputs $\mathbf{u}(k)$ must belong to the set of feasible controls $\mathcal{U}(k)$ determined by the positions of the agents and each sensor's FOR. To achieve the search objective, we define a *search policy* to be a function $\mathbf{u}(k) = \mu(\mathbf{p}(k), \mathbf{x}(k))$ that maps the team state and target state estimate to the next control inputs to be executed by the agents. The problem posed in (5) is to find the search policy that results in the maximum total reward collected by the team over the entire time allotted for the search.

For agents with motion constraints, this a very 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.⁶ Our approach to this problem approximate the solution by optimizing over a finite, receding horizon on a graph. This method is designed to reduce computation by optimizing over a shorter time horizon on a finite set of graph vertices, while also using careful vertex placement to maintain route flexibility in regions of high search reward.

2.1. Particle Filter Representation of the Target State

Particle filtering is a standard method of Monte Carlo state estimation that is especially well-suited to systems with non-linear dynamics and non-gaussian distributions. It involves modeling a system with a large number of dynamic ‘‘particles’’ whose states evolve according to some stochastic model. Each particle also has a weight representing the likelihood that the actual state is near its own state. Weights may be updated upon the arrival of new measurements and are then often normalized so that the total weight of the particles is equal to the number of targets. See Ref. 15 for a more detailed treatment of particle filtering. Below is an example of the search problem dynamics using particle filter estimation for the target states.

Let $\mathbf{x}(k) := [\mathbf{x}^p(k), \mathbf{x}^w(k)]$ represent the states of a simple particle filter (PF), where $x_i^p(k)$ is the position and $x_i^w(k)$ is the weight of the i^{th} particle. The system dynamics are then

$$\text{POSITION UPDATE: } x_i^p(k+1) = f_p(x_i^p(k), w_i(k)), \quad (6)$$

$$\text{WEIGHT UPDATE: } \tilde{x}_i^w(k+1) = (1 - P_D(q(k))) \cdot x_i^w(k) \quad (\text{with } P_D(q(k)) = 0 \text{ for } x_i^p \notin FOV(q(k))), \quad (7)$$

$$\text{RE-NORMALIZATION: } \mathbf{x}^w(k+1) = \tilde{\mathbf{x}}^w(k+1) \frac{1}{\sum_{i=1}^n \tilde{x}_i^w(k+1)}, \quad (8)$$

where $\mathbf{w}(k) = [w_1(k), \dots, w_n(k)]$ is a process noise sequence. The function f_p is typically used to propagate the particle positions according to a model of the expected target dynamics, randomized with $\mathbf{w}(k)$.

2.2. Grid-based Occupancy Map Representation of the Target State

A grid-based *occupancy map* is another method for estimating the states of an uncertain system. It involves representing a system's state-space on a grid, which typically partitions the search region \mathcal{R} . If $\mathcal{C} := \{c_1, c_2, \dots, c_{n_c}\}$, $c_i \in \mathcal{R} \quad \forall i \in \{1, \dots, n_c\}$ is a partition of \mathcal{R} into cells, then

$$\bigcup_i c_i = \mathcal{R} \quad \text{and} \quad c_i \cap c_j = \emptyset \text{ for } i \neq j.$$

The occupancy map representation of the target state decomposes $\mathbf{x}(k)$ into *position* and *weight* components: $\mathbf{x}(k) := [\mathbf{x}^p, \mathbf{x}^w(k)]$. The position component x_i^p represents a static position within cell c_i , typically the center of the cell. With some abuse of notation, we will write $x_i \in \mathcal{C}$ for $x_i \in c_i \subset \mathcal{C}$. The weight $x_i^w(k)$ represents the expected value (in the probabilistic sense) of $\mathbf{x}(k)$ in the cell; in the case of a *target* occupancy map, $x_i^w(k)$ represents the expected number of targets in the cell (which may be fractional, since this is a probabilistic expectation). Cell weights are dynamically updated based on incoming measurements and predicted future states. The system dynamics are

$$\text{WEIGHT UPDATE: } \tilde{x}_i^w(k+1) = (1 - P_D(q(k))) \cdot x_i^w(k) \quad (\text{with } P_D(q(k)) = 0 \text{ for } x_i^p \notin \text{FOV}(q(k))), \quad (9)$$

$$\text{RE-NORMALIZATION: } \mathbf{x}^w(k+1) = \tilde{\mathbf{x}}^w(k+1) \frac{1}{\sum_{i=1}^n \tilde{x}_i^w(k+1)}, \quad (10)$$

See Ref. 14 for more on target occupancy maps.

2.3. Dual Representation of the Target State

Both particle filters and a grid-based target occupancy map will be used in this paper to estimate the target state \mathbf{x} . Hypothesized but undiscovered targets will be represented on the grid, as in (9)-(10). When some information exists on a target (such as an initial detection), the target's state will be represented with a particle filter, as in (6)-(8), and the corresponding target value will be removed from the occupancy map, so that

$$\sum_{x_i \in \mathcal{C}} x_i^w(k) + \sum_{j \text{ PFs}} \left\{ \sum_{x_i \text{ in PF } j} x_i^w(k) \right\} = N.$$

2.4. Discretization of the Sensor Control Space

Let the UAV's path P be represented by a curve in the flight plane of the UAV. As a UAV moves along a path, it will execute a sequence of sensor tasks that depend on the current position and orientation of the UAV. Sensor controls $q(k) = (q_x(k), q_y(k))$ will be selected to center the FOV footprint $\text{FOV}(q(k))$ on gridpoints $x_i^p \in \mathcal{C}$, or on the center of mass of each particle cloud. A sensor dwell on the region $\text{FOV}(q(k))$ will be called a *sensor task*, and may be represented simply by $q(k)$ when there is no ambiguity. A sequence of sensor tasks will be called a *sensor schedule*, and represented by the points that define the centers of the sensor tasks composing the schedule: $S = \{q(k_1), q(k_2), \dots, q(k_n)\}$. Slew time for the sensor to move between sensor tasks in the schedule is also modeled in the discretization. There may be many different possible sensor schedules for a given path P .

The region $\text{FOV}(q(k))$ will be modeled on the ground as an oval or quadrilateral generated by intersecting the camera viewing frame with the ground. If the UAV platform flies at a fixed altitude with a typical 640 x 480 pixel video camera, $\text{FOV}(q(k))$ has minimum, approximately rectangular dimensions on the ground (w, h), achieved when the sensor is pointing straight down. We then construct the occupancy map grid \mathcal{C} as a tessellation of squares of side length $s = h/\sqrt{2}$, for $h < w$. This grid spacing ensures that: the entire c_i cell is contained in $\text{FOV}(c_i)$ regardless of how the UAV platform is aligned with the grid (Figure 1); and

$$\mathcal{R} \subseteq \bigcup_{i=1}^{n_c} \text{FOV}(c_i),$$

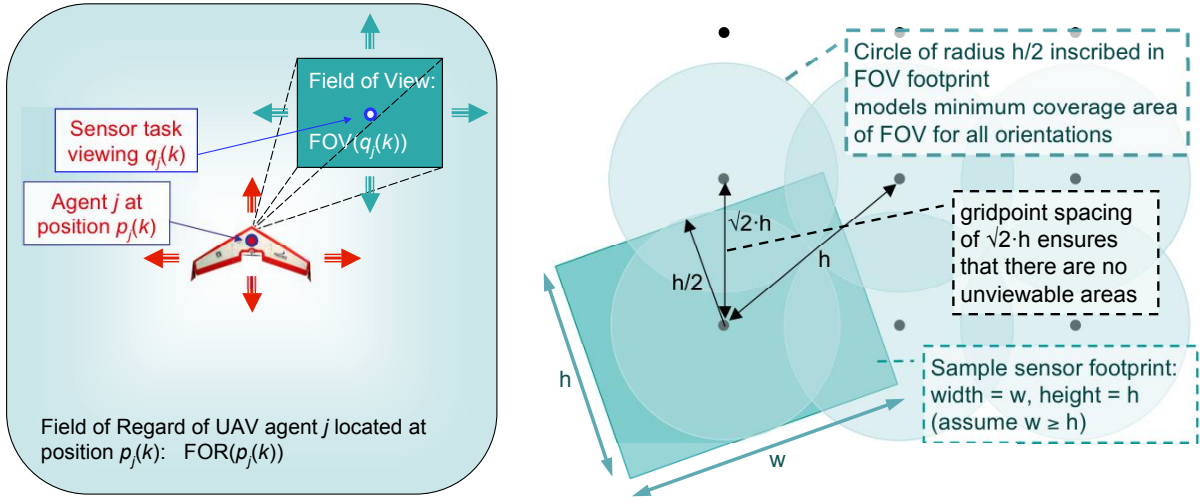
so that every point in \mathcal{R} is viewable by the sensor.

The reward collected by a sensor task $\text{FOV}(q(k))$ is

$$r(q(k)) = \sum_{x_i^p(k) \in \text{FOV}(q(k))} P_D(q(k)) \cdot x_i^w(k), \quad (11)$$

resulting in the detected target values being decreased in the target state estimate:

$$x_i^w(k+1) = (1 - P_D(q(k))) \cdot x_i^w(k), \quad (12)$$



(a) Sensor Field of View (FOV) and Field of Regard (FOR)

(b) Target occupancy map \mathcal{C}

Figure 1. UAV motion and sensing model, and target occupancy map

with the understanding that $P_D(q(k)) = 0$ for $x_i^p(k) \notin FOV(q_k)$.

Reward for the sensor schedule $S = \{q(k), q(k+1), \dots, q(k+n)\}$ will be computed by tallying up the reward received from each sensor task q composing the schedule:

$$r(S) := \sum_{j=0}^n \left\{ \sum_{x_i^p(k+j) \in FOV(q(k+j))} P_D(q(k+j)) \cdot x_i^w(k+j) \right\}, \quad (13)$$

in which $x_i^w(k+j)$ is computed from $x_i^w(k+j-1)$ for $j > 0$ using (12). Written in terms of $\mathbf{x}(k)$, equation (13) becomes

$$r(S) = \sum_{j=0}^n \left\{ \sum_{x_i^p(k+j) \in FOV(q(k+j))} P_D(q(k+j)) \cdot \left[\prod_{l=0}^{j-1} (1 - P_D(q(k+l))) \right] \cdot x_i^w(k) \right\}. \quad (14)$$

If $S(P)$ is the sensor schedule for path P , then $r(P) := r(S(P))$.

For multiple agents acting cooperatively, reward for one agent's schedule or path depends on the actions of the other agents. If two agents count reward for the same region without conditioning the reward values on the actions of the other agent, there is a danger that they will choose redundant actions that degrade the performance of the team. One solution to this problem is to evaluate the reward of paths after subtracting out reward that other agents plan to collect. In cases where multiple agents are planning simultaneously, we require that they plan in the pre-specified order $1, 2, \dots, M$.

For agent α planning with respect to existing plans for agents a_1, \dots, a_m , let P_{a_1}, \dots, P_{a_m} be the planned paths and $S(P_{a_1}), \dots, S(P_{a_m})$ be the planned sensor schedules for agents a_1, \dots, a_m respectively. If each schedule consists of tasks $S(P_{a_j}) = \{q_{a_j}(k_{a_j}), \dots, q_{a_j}(k_{a_j} + n_{a_j})\}$, then agent α will compute the reward of a candidate path by conditioning on the sensor schedule

$$S := \{q_{a_1}(k_{a_1}), \dots, q_{a_1}(k_{a_1} + n_{a_1}), \dots, q_{a_m}(k_{a_m}), \dots, q_{a_m}(k_{a_m} + n_{a_m})\} \quad (15)$$

consisting of all sensor tasks for agents a_1, \dots, a_m . The reward of candidate path P_α and corresponding sensor schedule $S(P_\alpha) = \{q_\alpha(k_\alpha), \dots, q_\alpha(k_\alpha + n_\alpha)\}$ is therefore given by

$$r(S(P_\alpha)) = \sum_{j=0}^{n_\alpha} \left\{ \sum_{x_i^p(k_\alpha+j) \in FOV(q(k_\alpha+j))} P_D(q(k_\alpha + j)) \cdot \left[\prod_{l=0}^{j-1} (1 - P_D(q(k_\alpha + l))) \right] \cdot x_i^w(k_{a_m} + n_{a_m}) \right\}, \quad (16)$$

where $x_i^w(k_{a_m} + n_{a_m})$ has been updated for all sensor tasks in \mathcal{S} using (12). Equation (14) will be used to evaluate candidate sensor schedules for path P_α and select the optimal sensor schedule $S^*(P_\alpha)$, in the sense of collecting the most reward $r(S)$.

2.5. Routing and Sensor Control

For a given path P , one may construct a simplified version of (16) for $r(P)$ by assuming that the sensor can view every target state element x_i exactly once. The region of sensor coverage on the ground corresponding to this designation will be called $FOR(P)$ to represent the field of regard of the entire path P . It is defined by

$$FOR(P) := \bigcup_{\text{points } p \in P} FOR(p),$$

where the union is taken over all points p in the plane curve P . The reward collected for $FOR(P)$ can be computed from (11) as

$$r(FOR(P)) = \sum_{x_i^p(k) \in FOR(P)} P_D(P) \cdot x_i^w(k), \quad (17)$$

where $P_D(P)$ is a fixed probability of detection that is assumed for any targets within the path P . The state update corresponding to (17) is given by

$$x_i^w(k+1) = (1 - P_D(P)) \cdot x_i^w(k),$$

with $P_D(P) = 0$ for $x_i^p(k) \notin FOR(P)$. Rewards for viewing each x_i^p are computed from $x_i^p(k)$, in contrast to (13) in which the reward for viewing x_i^p after time k will be conditioned on prior sensor tasks within the schedule. If multiple agents are acting cooperatively, then (17) must be conditioned on the prior actions \mathcal{S} of other agents.

Depending on the relative speeds of the UAV and the camera gimbal, a UAV agent may not be able to view every target state element x_i in $FOR(P)$ while moving along the path P ; alternatively, the UAV may be able to view certain x_i repeatedly. Therefore, the actual sensor schedule $S(P)$ may radically differ from $FOR(P)$. Based on experience with platforms in Toyon's UAV testbed, our team has found that even if the servos controlling the camera gimbal are able to slew the camera fast enough to view all points in $FOR(P)$, automatic video tracking software processing such a video stream may not be able to perform frame registration on such a quickly translating frame sequence. In these cases, evaluating the path P based on the reward that could be collected in $FOR(P)$ is inaccurate, and instead the path P should be evaluated based on the the reward that could be collected in $S(P)$.

In Section 3, we describe two different routing algorithms. The first is referred to as a the *Decoupled Routing and Sensor Tasking Algorithm* because these two optimization steps are performed independently. Candidate UAV paths P are evaluated based on scoring the sensor schedule $FOR(P)$ using (16); once an optimal path P^* is selected with this criterion, then the actual sensor schedule for P^* is determined by running a separate optimization of (16) over all implementable sensor schedules for P^* . The second algorithm is called a *Coupled Routing and Sensor Tasking Algorithm* because the routing and sensor tasking problems are solved together. For each candidate path P , an optimal (implementable) sensor schedule $S^*(P)$ is selected using (16). The score of $S^*(P)$ is used to compare each candidate path P , and the selected path will be the one with the best score for an implementable schedule. Thus, in the coupled algorithm, candidate UAV paths are evaluated using a more accurate reward model.

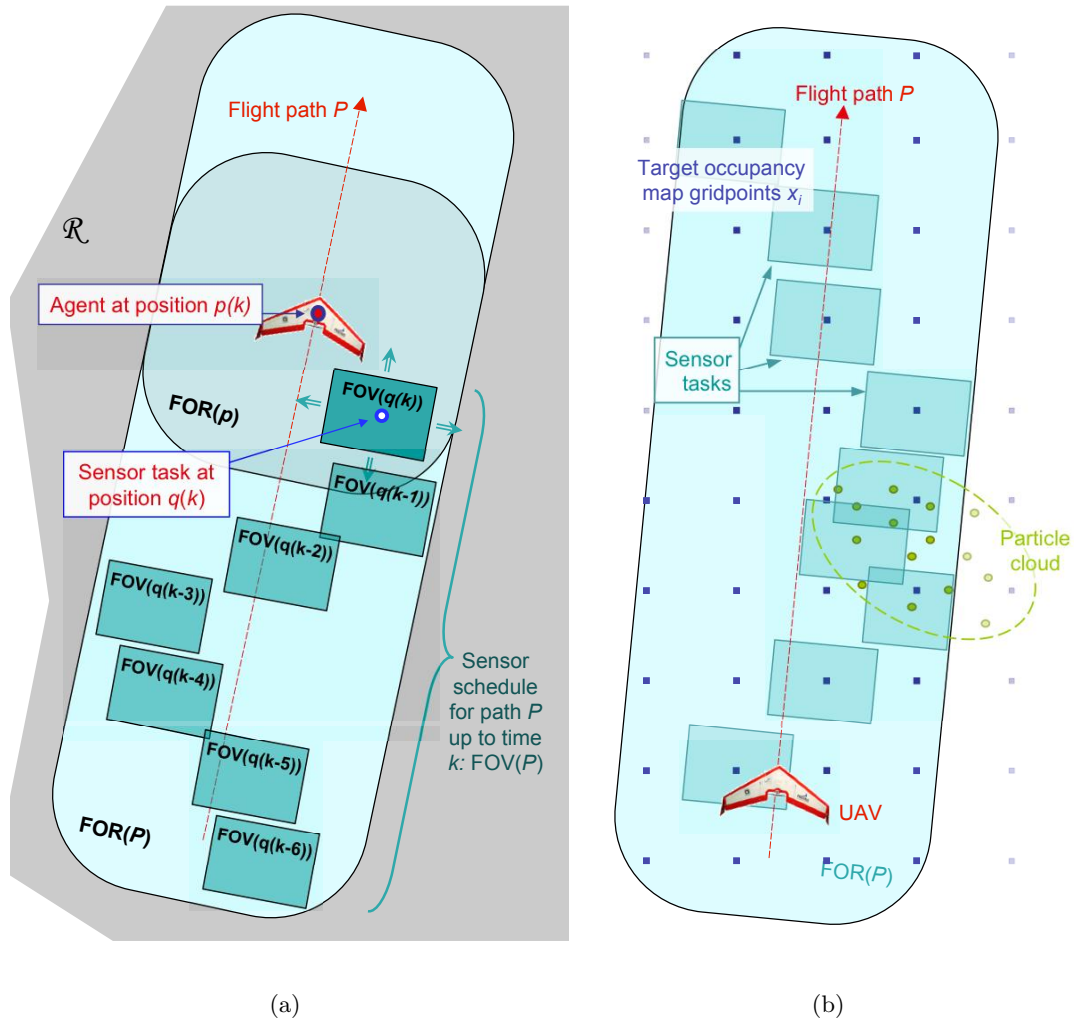


Figure 2. (a) Field of Regard for UAV path P , shown with $FOR(p(k))$ and individual sensor tasks $FOV(q(k-j))$, $j = 0, \dots, 6$. (b) Target occupancy map gridpoints and particles in a particle filter.

2.6. Graph-Based Discretization of the Routing Control Space

UAV routing optimization problems, such as the one discussed in this paper, are commonly solved at the “waypoint level.” *Waypoints* are points in space, typically specified in Lat/Lon/Alt, through which the UAV should fly. These waypoints can be wirelessly transferred to a commercial autopilot system that will determine the avionics controls (rudder, throttle, ailerons, etc) required to fly the UAV through the waypoints, plus/minus some constant error. Thus, waypoint level is an appropriate level of detail to specify a UAV flightpath. The flight paths constructed in this paper will therefore be modeled as a sequence of straight line segments joining the waypoints. (The actual flightpath will have slightly rounded corners to adhere to the physical turn rate limitations of the UAV platform. This correction is made automatically by the autopilot.)

Waypoint-level routing lends itself nicely to graph-based routing optimization approaches. We define a base path-planning graph $G := (V, E)$ with vertices v in vertex set V and edges $e = (v, v')$ in edge set E . The vertices will serve as potential waypoints for the agents and the existence of an edge between two vertices means that an agent is allowed to move between them. When there is no ambiguity, we will use the notation v to denote both the vertex in the set V and its spatial location in \mathbb{R}^2 . Similarly, $e = (v, v')$ will denote both the

edge in the set E and the set of points on the line segment connecting its endpoint vertices v and v' . Positions and relative distances between graph vertices are selected in order to avoid sensor coverage gaps in the search region. In particular, we require that for every point $r \in \mathcal{R}$, there exists an edge $e \in E$ satisfying the condition $r \in FOR(e)$. With this condition, the graph vertices v will be spaced by approximately the diameter of the sensor FOR. Lastly, we define an *edge cost function* $\tau : E \rightarrow \{0, 1, 2, \dots\}$, representing the transit time between vertices.

A *path in G* is a sequence of vertices $P := (v_1, v_2, \dots, v_{l-1}, v_l)$ such that $(v_j, v_{j+1}) \in E$. In the model predictive search algorithm introduced in Section 3, whenever an agent reaches a waypoint, it will compute a new l -step path on the graph, where l is the length of the prediction horizon. The *path cost in G* is the total duration of the path and is given by

$$c(P) := \sum_{i=1}^{q-1} \tau(v_i, v_{i+1}).$$

The *path reward in G* is given by (16) (conditioned on the actions of the preceding agents) with P defined as the curve in \mathbb{R}^2 sequentially joining $(v_1, v_2, \dots, v_{l-1}, v_l)$.

We wish to compare paths of varying lengths in G . To create an honest comparison, we define the *average path reward*

$$\bar{r}(P) := \frac{r(P)}{c(P)}. \quad (18)$$

To perform numeric optimization on G , the planing algorithm will select the optimal path

$$P^* := \arg \max_{P \in \mathcal{P}} \bar{r}(P),$$

where \mathcal{P} is the set of all l -step candidate paths for an agent at time k .

2.7. Dynamic Graph Generation

Although any graph will work with the algorithm introduced in this paper, the structure will have a major effect on computation, and the node and edge placements will affect performance. In preliminary tests of the algorithm, we used a degree-3 hexagonal lattice graph as the base path-planning graph G . From G , we generate a dynamic path-planning graph $G_k := (V_k, E_k)$ by deleting vertices in areas where the target state estimate is below a certain threshold. To maintain connectivity of the graph, we reconnect any resulting disconnected subgraphs using a Delaunay algorithm. Since the target state estimate is constantly changing, the graph G_k will need to be periodically updated. This allows the agents to search new high-reward areas that were excluded in the previous version of the graph.

3. GRAPH-BASED MODEL PREDICTIVE SEARCH ALGORITHMS

In this section we present two graph-based model predictive search (GBMPS) algorithms to solve the UAV routing and sensor tasking problem formulated in Section 2. Both algorithms in this paper share the same basic structure for UAV routing. This structure is similar to model predictive control in that the algorithms compute the expected outcome of a set of paths emanating from the search team's current position state, choose the paths that maximize an objective function based on search reward, and implement a portion of the chosen paths while computing a new set of paths. An advantage to this type of algorithm is that it can be adapted to the computation time available, and as the computation applied to the problem increases, the solution approaches the optimum on the graph.

One such approach was presented in Ref. 13. A potential disadvantage to this type of algorithm is that because it requires 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 an agent can cover on this horizon, the algorithm's performance will

suffer. We address this issue by performing the optimization on a graph whose nodes are placed at high-reward locations and dynamically updated at each time step. We also allow for independent control of the gimbaled sensor. This allows for several possible methods of algorithm implementation, as discussed in Sect. 2.5):

1. **Decoupled Routing and Sensor Tasking:** Evaluating paths based on all search reward that lies within an agent’s field of regard along the entire path.
2. **Coupled Routing and Sensor Tasking:** Evaluating paths based on optimization of the sensor tasks for the duration of the path.

The first method requires less computation, but the second method provides a more accurate model of what the sensor can view. For situations in which the relative speed of the agents is too fast for the sensors to view everything inside their FOR, method 2 may yield significant benefit over method 1. The algorithm presented here works with either implementation.

3.1. Routing Optimization

The steps of the graph-based model predictive search algorithm are listed below.

1. (INITIALIZATION) Generate a base path-planning graph $G(V, E)$, such that for every point $r \in \mathcal{R}$, there exists an edge $e \in E$ satisfying the condition $r \in FOR(e)$.
2. (INITIALIZATION) Set $j = 0$. For each agent $a \in \{1, \dots, N\}$, let v_j^a denote the starting vertex of the search agent. Pick two initial waypoints v_{j+1} and v_{j+2} that are reachable from v_j by traveling on graph edges.
3. Update the dynamic graph G_k from the base graph G using the process described in Section 2.7, keeping all current waypoint nodes.
4. At waypoint v_i , begin flying to v_{i+1} and commit to a new waypoint v_{i+2} .
5. On the way from v_i to v_{i+1} , compute the future path $P := (v_{i+2}, v_{i+3}, \dots, v_{i+l})$ and corresponding sensor schedule $S(P)$ that maximizes the expected average reward $\bar{r}(P)$, as described in Section 2.6. This future path must include at least one new waypoint, v_{i+3} .
6. Once the agent has arrived at waypoint v_{i+1} , set i equal to $i + 1$ and return to step 3.

In the above algorithm, there are two key reasons for implementing the first two computed waypoints instead of just one. First, this allows time for computation of the next asset path, avoiding situations where the agent has reached a waypoint and does not know where to go next until it completes a computation. Second, it may be helpful to have two waypoints planned in advance so that curves in the path may be interpolated if there is a change in heading from waypoint 1 to waypoint 2.

Our routing algorithms differ in their implementations of Step 5. The decoupled algorithm computes path rewards using only the geometry of the candidate sensor path, by assuming $S(P) = FOR(P)$ for path scoring. The coupled algorithm computes path rewards by modeling the actual sensor geometry over the candidate paths, optimizing (14) over all possible sensor schedules, for each candidate path.

4. SIMULATIONS RESULTS

We simulate a cooperative search with four agents searching a 75 km by 75 km square region \mathcal{R} for a single target. The FOR of each agent is a 6 km diameter circle and its FOV is a 1.5 km diameter circle. The search region is partitioned as in Section 2.2 to form a grid-based occupancy map representation for the target state. Square cells c_i with side length $(1.5)\sqrt{2}$ are chosen as in Section 2.4 to ensure that the UAV sensors can see every point in \mathcal{R} . This grid discretization generates a 70 by 70 grid of points x_i for the target occupancy map.

The path-planning graph is a degree-3 hexagonal lattice, spaced according to the FOR, and dynamically updated using the process described in Section 2.7. The agents use a three step prediction horizon and take

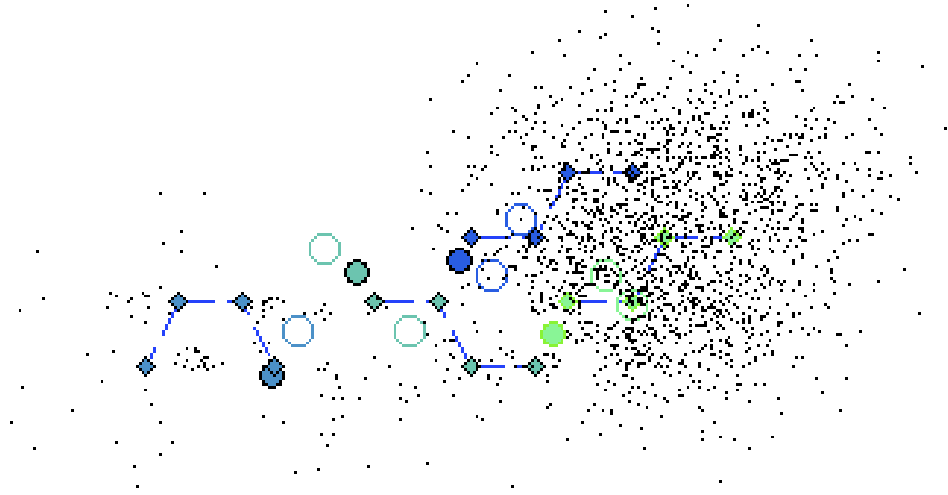


Figure 3. Display of Search Simulation.

measurements every 15 seconds. The target state estimates are modeled by a simple particle filter as described in Section 2.1. Figure 3 shows a snapshot of the simulation display.

Search agents (UAVs) are represented by solid circles. Diamonds show agent waypoints, connected by dashed lines. Open circles represent the footprints of the scheduled sensor tasks on the immediate graph edge. Particles x_i in the particle filter are represented by black dots, but if a particle weights x_i^w drop below a certain threshold due to negative sensor detections, then the particle is no longer displayed.

Table 1. Results of a GBMPS for 1-4 Agents. T^* represents the time at which the target is first discovered by a search agent.

Number of Agents (M)	T^* (seconds)
1	1592
2	1181
3	975
4	931

Table 1 shows the results of the graph-based model predictive search algorithm for one to four cooperating agents. The fact that expected time to capture drops significantly with each increase in the number of searchers shows that this algorithm makes effective use of additional agents.

Method	Reward (max = 1)
GBMPS on a static square lattice graph using decoupled routing and sensor optimization	0.32
GBMPS on a dynamically generated graph, as in Section 2.7, using decoupled routing and sensor optimization	0.75
GBMPS on a dynamically generated graph, as in Section 2.7, using coupled routing and sensor optimization	0.78

Table 2. Algorithm Performance Comparison

The GBMPS method applied to a static square lattice graph is similar to previously proposed cooperative search algorithms such as in Ref. 13 and the pursuit policies of Ref. 16. Lines 2 and 3 of table 2 vs. line 1 show that placing graph nodes only in high reward areas makes a significant increase in algorithm performance.

Additionally, in this scenario, the method of coupled routing and sensor optimization performed 4% better than the decoupled algorithm.

5. CONCLUSIONS AND FUTURE RESEARCH

The goal of this research is to determine how to optimally route UAV platforms and simultaneously control their sensors to cooperatively locate/ID one or more targets. In this paper we have made several key contributions to this effort. We proposed a cooperative search algorithm that uses receding horizon optimization on a periodically updated graph. This algorithm optimizes paths on a dynamic graph that is updated based on the current target probability distribution, and uses a vertex-based prediction horizon allowing for comparison of paths of different lengths. We also incorporated the use of gimbaled sensors whose tasks can be optimized and used in the evaluation of candidate paths, thereby routing the platform to make the best use of its sensors. Simulations showed that the strategic placement of graph vertices causes a significant boost in performance over traditional implementations having a fixed time horizon. Using joint routing and sensor optimization provides an additional increase in performance.

This algorithm was designed with a decentralized implementation as the end goal, but have only presented the centralized version in this paper. In future work we will present a decentralized implementation of the cooperative GBMPS algorithm with simulations and analysis on robustness to communication loss.

Another possible future extension of this work would involve improving the fidelity of the search paths. The discretization of the UAV paths onto our search graph restricts the possibilities for where the platform can travel and the specific route it can take to get there. Incorporating a sensor-based dynamic graph generation may further improve the search performance.

Another extension of our work would improve the sensor schedule model used in the schedule prediction and scoring stage. Since our sensor scheduling algorithm uses branch-and-bound optimization, it does not necessarily build the optimal sensor schedule over the entire scheduling horizon. An interesting comparison for future research would pit the algorithm in this paper against a similar coupled routing and tasking algorithm that employed a sensor scheduler capable of searching the entire sensor control space.

REFERENCES

1. B. O. Koopman, *Search and Screening*, Operations Evaluations Group Report No. 56, Center for Naval Analyses, Alexandria, VA, 1946.
2. L. D. Stone, *Theory of Optimal Search*, Academic Press, New York, 1975.
3. J. R. Frost and L. D. Stone, "Review of search theory: Advances and applications to search and rescue decision support," tech. rep., U.S. Coast Guard Research and Development Center, Groton, CT, Sept. 2001.
4. K. B. Haley and L. D. Stone, eds., *Search Theory and Applications*, Plenum Press, New York, 1980.
5. M. Mangel, "Search theory: A differential equations approach," in *Search Theory: Some Recent Developments*, D. V. Chudnovsky and G. V. Chudnovsky, eds., pp. 55–101, Marcel Dekker, New York, 1989.
6. K. E. Trummel and J. Weisinger, "The complexity of the optimal searcher path problem," *Operations Research* **34**(2), pp. 324–327, 1986.
7. J. Eagle and J. Yee, "An optimal branch-and-bound procedure for the constrained path, moving target search problem," *Operations Research* **28**(1), 1990.
8. T. J. Stewart, "Experience with a branch and bound algorithm for constrained searcher motion," in *Search Theory and Applications*, K. B. Haley and L. D. Stone, eds., pp. 247–254, Plenum Press, New York, 1980.
9. B. DasGupta, J. Hespanha, J. Riehl, and E. Sontag, "Honey-pot constrained searching with local sensory information," *Nonlinear Analysis: Hybrid Systems and Applications* **65**, pp. 1773–1793, Nov. 2006.
10. J. R. Riehl and J. P. Hespanha, "Fractal graph optimization algorithms," in *Proceedings of the 44th Conference on Decision and Control*, 2005.
11. J. R. Riehl and J. P. Hespanha, "Cooperative graph search using fractal decomposition." To be presented at the ACC'07, July 2007.
12. J. P. Hespanha and M. Prandini, "Optimal pursuit under partial information," in *Proceedings of the 10th Mediterranean Conference on Control and Automation*, 2002.

13. 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.
14. J. P. Hespanha and H. Kızıloca, "Efficient computation of dynamic probabilistic maps," in *Proceedings of the 10th Mediterranean Conference on Control and Automation*, 2002.
15. 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.
16. 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.*, **3**, pp. 2432–2437, Dec. 1999.