2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Sampling-based search for a semi-cooperative target

Isaac Vandermeulen[1], Roderich Groß[2], and Andreas Kolling[3]

***Abstract*— Searching for a lost teammate is an important task for multirobot systems. We present a variant of rapidly-expanding random trees (RRT) for generating search paths based on a probabilistic belief of the target teammate's position. The belief is updated using a hidden Markov model built from knowledge of the target's planned or historic behavior. For any candidate search path, this belief is used to compute a discounted reward which is a weighted sum of the connection probability at each time step. The RRT search algorithm uses randomly sampled locations to generate candidate vertices and adds candidate vertices to a planning tree based on bounds on the discounted reward. Candidate vertices are along the shortest path from an existing vertex to the sampled location, biasing the search based on the topology of the environment. This method produces high quality search paths which are not constrained to a grid and can be computed fast enough to be used in real time. Compared with two other strategies, it found the target significantly faster in the most difficult 60% of situations and was similar in the easier 40% of situations.**

I. INTRODUCTION

Communication is essential for the successful completion of many tasks performed by teams of mobile robots. In real environments, robots often communicate over inexpensive ad-hoc networks which have limited connectivity that is affected by distance and line of sight [1]. The robots may lose connectivity as they move throughout their environment. There are several possible solutions to this problem.

- **Constant connectivity** [2], [3], [4]: The robots' motions are restricted to maintain connectivity. This constraint enables constant communication but makes the team less effective at other tasks as they cannot spread out.
- **Periodic connectivity** [5], [6], [7]: The team can separate temporarily with a plan of where they will meet back up. This strategy provides some flexibility but is inconvenient when tasks take unpredictable times, as some robots will be forced to wait for others.
- **Intermittent connectivity** [8]: The team can separate without a reconnection plan. This approach requires robots to search for each other to communicate.

The best communication strategy for real robots depends on a range of factors, including the size of the environment and how predictably the robots behave. In predictable controlled situations, a conservative strategy with preplanned meetings may be best. However, in unstructured environments where robots must modify plans in response to unpredictable events, intermittent connectivity provides the most flexibility.

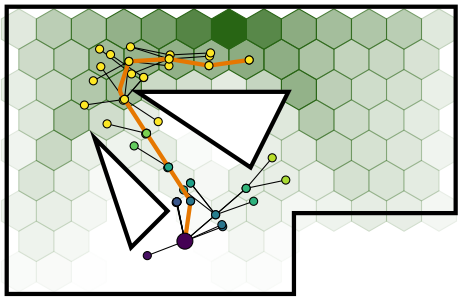

Fig. 1. A searcher (purple) has a belief (green cells) of a target robot's position. Using this belief, it constructs a planning tree using a discounted search reward (low reward = blue, high reward = yellow) and follows the tree's best path (orange) to search for the target.

When robots communicate intermittently, they do not have a prearranged meeting and must instead search for each other. Within a team, robots are cooperative and, when connected, can share information such as a planned path or behavior, which will make search easier. However, intermittent communication is only useful in unpredictable situations where robots do not follow their plans exactly. Therefore, the search target can be considered *semi-cooperative*. It may provide some useful information about its behavior, but does not guarantee that it will behave exactly as planned.

In this paper, we present a sampling-based search algorithm that can be used by teams of robots operating in unpredictable environments. This algorithm is inspired by rapidly-exploring random trees (RRT) [9] but includes important modifications for the search problem. The planning tree is based on a probabilistic belief of the target's position and attempts to maximize a reward function related to the probability and time needed to find the target (Figure 1). As the objective is maximization instead of RRT's usual objective of minimization, the algorithm is modified to avoid adding all vertices to the longest branch of the tree. When searching, the searcher follows a path in the tree and can reuse parts of the tree for planning future search paths.

II. RELATED WORK

Search theory dates back to the 1940s motivated by the US navy's antisubmarine missions during World War II [10]. Most early efforts were focused on searching for stationary targets [11], [12] or targets which moved randomly, indifferent to the searcher [13], [14]. This one-sided search problem has recently been studied by the robotics community [15], [16]. The typical approach to search uses a belief of a target's location, updated using a motion model, to plan a path that

[1] Isaac Vandermeulen is with iRobot, Pasadena, California, USA isaac.vandermeulen@gmail.com
[2] Roderich Groß is with the Department of Automatic Control and Systems Engineering, The University of Sheffield, Sheffield, UK r.gross@sheffield.ac.uk
[3] Andreas Kolling is with Amazon Robotics, North Reading, Massachusetts, USA andreas.kolling@gmail.com

maximizes the probability of finding the target over some time horizon.

A *belief* is a probabilistic description of a target robot's position based on information known by a searcher. Beliefs evolve over time as the searcher expects its target to move. Three methods of describing and updating beliefs are:

(a) **Markov models** use a probability vector over a graph—corresponding to a discretization of the environment—which is updated using transition probabilities that only depend on the robot's current state [17], [18], [19]. Variants such as second-order Markov models [20], semi-Markov models [8], and hidden Markov models [21] provide more realistic models in the same framework.

(b) **Particle filtering algorithms** use a finite set of particles, each representing one possible target behavior, moving in continuous space according to the target's dynamics, with each particle using different control inputs [22], [23], [24]. A probability hypothesis density filter replaces the point particles with Gaussian distributions [25], [26].

(c) **Historical data** can also be used to build a model, if enough data is available or can be simulated [27].

Using any of these methods, the searcher can maintain a belief used to plan its search path. This path maximizes the probability of finding the target over a finite horizon [17], [23], [27], [20], [8], the probability of finding the target per unit time [22], a discounted reward which values finding the target quickly [18], [28], or a fairness-based reward which values regularly observing multiple different targets [19]. Finding the optimal path is NP-hard, so branch-and-bound [17], [18], [20], mixed integer linear programming [27], multi-level optimization [29], and depth-first search [23] are used to plan near-optimal paths on grids.

Our search algorithm uses a sampling-based planner. Sampling-based planners have been popular in robotics since the development of probabilistic roadmaps (PRM) [30] and rapidly-exploring random trees (RRT) [9] in the late 1990s. Both techniques use random samples to construct a graph, which in the case of RRT is a tree. By rewiring the tree as vertices are added, RRT* is asymptotically optimal [31]. Although originally used to minimizing distance, RRT has also been used to minimize localization error [32], mechanical work over uneven terrain [33], probability of capture in pursuit-evasion [34], and distance from a moving target [35]. In this paper we use a sampling-based planner to *maximize* a discounted reward function based on finding the target robot as quickly as possible.

III. COMMUNICATION MODELS

Depending on hardware, robots communicate using one of a variety of wireless signals. This signal's strength determines the probability of successful communication [36]. Distance and line of sight can influence signal strength [37] resulting in a variety of communication models (Figure 2).

For two robots located at positions, q_0, q_1, in an environment, $\mathcal{Q}$, let $C(q_0, q_1) \in [0, 1]$ denote the probability that they can communicate. As C can depend on many factors, such as the environment's physical properties and the type of wireless signal used, we will assume that a good estimate of C is available for any pair of locations. This assumption is reasonable as C can be estimated experimentally from signal strength data at a few points [1], [38].

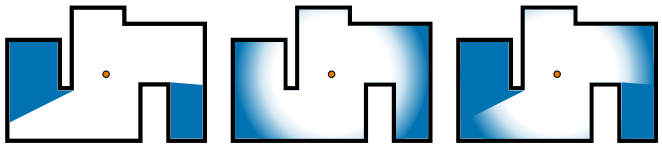

Fig. 2. Communication between robots can be limited by line of sight (left), distance (center), or both (right) depending on the type of wireless communication used.

When planning a search path, the searcher needs the probability of communication if it were at some $q_{\text{sea}} \in \mathcal{Q}$. As the target's position is not known, the searcher can estimate this probability by

$$\mathbb{P}(\text{communication}) = \int_{\mathcal{Q}} C(q_{\text{tar}}, q_{\text{sea}}) \mathrm{d}\mu(q_{\text{tar}}) \qquad (1)$$

where the searcher's belief of the target's location is described by a probability measure $\mu : \Sigma \to [0, 1]$ over Σ, a σ-algebra on $\mathcal{Q}$.

The method of approximating the integral in (1) depends on what kind of target belief is used. We will store the belief as a probability vector, $\boldsymbol{b}_{\text{tar}} \in \mathbb{R}^{|\mathcal{Y}|}$, over $\mathcal{Y}$, a cellular decomposition of $\mathcal{Q}$. Although q_{sea} is known, we can also represent it using a vector $\boldsymbol{b}_{\text{sea}} \in \mathbb{R}^{|\mathcal{Y}|}$ whose elements are all 0 except the element corresponding to the cell containing q_{sea} which is 1. Using these vectors, we approximate (1) by

$$\mathbb{P}(\text{communication}) = \boldsymbol{b}_{\text{tar}}^{\top} \boldsymbol{C} \boldsymbol{b}_{\text{sea}} \qquad (2)$$

where $\boldsymbol{C}$ is the *communication matrix* whose $(i, j)^{\text{th}}$ element is the probability of communication between cells y_i and y_j. This approximation is valid if the cells of $\mathcal{Y}$ are small enough that communication strength is similar everywhere in a cell.

IV. TRACKING AN UNSEEN TARGET

To plan its search, the searcher needs probabilistic descriptions of where it believes its target is and how it moves. We assume that the target robot is *semi-cooperative* and is not actively trying to avoid the searcher so its behavior is independent of the searcher's location. In this paper, the decomposition, $\mathcal{Y}$, is a polygonal lattice and the belief is a probability vector, $\boldsymbol{b} \in \mathbb{R}^{|\mathcal{Y}|}$, with elements

$$b_i = \mathbb{P}\left(q_{\text{tar}} \in y_i \mid \text{information known by searcher}\right) \qquad (3)$$

If the target and searcher can communicate, then

$$b_i = \begin{cases} 1 & \text{if } q_{\text{tar}} \in y_i \\ 0 & \text{otherwise.} \end{cases}$$

Otherwise, $\boldsymbol{b}$ will have many non-zero entries and will need to be updated using a probabilistic motion model.

In this paper, we use a hidden Markov Model (HMM) to update $\boldsymbol{b}$. HMMs augment the cells, $\mathcal{Y}$, of a basic Markov model with a set of hidden states, $\mathcal{X}$. These are "hidden" because they can include information about the target that cannot be directly observed—a planned path, objective, or

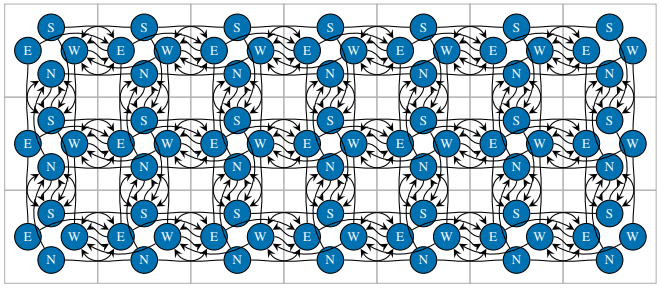

Fig. 3. State transition graph of an HMM using direction states to model momentum.

velocity—in addition to its observable location. Similar to the cell belief vector, $\boldsymbol{b} \in \mathbb{R}^{|\mathcal{Y}|}$, we define a state belief vector, $\boldsymbol{w} \in \mathbb{R}^{|\mathcal{X}|}$. As $\boldsymbol{w}$ contains more information than $\boldsymbol{b}$, we first update the state belief with a Markov update rule

$$\boldsymbol{w}[\tau] = \boldsymbol{A}\boldsymbol{w}[\tau - 1] \tag{4}$$

where $\boldsymbol{A} : \mathbb{R}^{|\mathcal{X}|} \to \mathbb{R}^{|\mathcal{X}|}$ is a sparse left-stochastic *update matrix* whose $(i,j)^{\text{th}}$ element is the probability that a robot in state x_j at time $\tau - 1$ will be in state x_i at time τ. Each state corresponds to a physical location so the cell belief is

$$\boldsymbol{b}[\tau] = \boldsymbol{P}\boldsymbol{w}[\tau] \tag{5}$$

where $\boldsymbol{P} : \mathbb{R}^{|\mathcal{X}|} \to \mathbb{R}^{|\mathcal{Y}|}$ is a sparse *projection matrix* mapping each state to one or a few nearby cells. Since both $\boldsymbol{A}$ and $\boldsymbol{P}$ are sparse with fewer than some constant number of non-zero elements in each row and column, updating a belief using (4)–(5) is $\mathcal{O}(|\mathcal{Y}|)$. Therefore, for a fixed cell size, the complexity of updating the belief scales linearly with the area of the environment.

The HMM update law (4)–(5) is *Markovian* because it assumes the belief at time τ only depends on the belief at time $\tau - 1$ and not any older information. For ordinary Markov models, where $\mathcal{X} = \mathcal{Y}$ and $\boldsymbol{P} = \boldsymbol{I}$, this assumption is restrictive because most robots' motion does depend on events from more than one time step ago. However, for HMMs, it is not restrictive because the states of $\mathcal{X}$ can include any relevant information.

Example 1 (Momentum)**.** *Real robots have second order dynamics so momentum effects how they move. For the vast majority of robotic tasks—exploration, coverage, search, delivery—the robot moves in straight lines much more than it goes back-and-forth over the same location. To incorporate momentum and bias the HMM towards straight paths, we use $\mathcal{X} = \mathcal{Y} \times \Theta$, where Θ is a set of directions (Figure 3), and the transition probabilities $a_{i,j}$ are higher between states with the same direction. HMMs with these* direction states *are equivalent to second-order Markov models.*

Example 2 (Variable velocity)**.** *Real robots do not travel at fixed velocities. To incorporate variable velocity into an HMM, we add a chain of* transit states *(Figure 4) between two states that would otherwise be adjacent. Each transit state can either transition to the next transit state in the chain or to the original end state. The transition probabilities are computed from the distribution of times that it takes to transitioning between the two original states which depends on the robot's velocity distribution. HMMs with transit states are equivalent to semi-Markov models.*

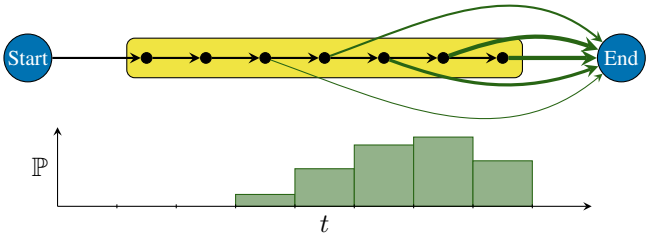

Fig. 4. HMMs can include a chain of transit states (top) to approximate the distribution of times (bottom) needed to transition between two states.

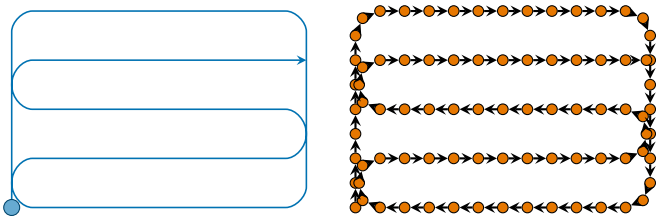

Fig. 5. A known target path (left) can be incorporated into an HMM by adding a chain of path states (right).

Example 3 (Planned path)**.** *Cooperative robots which are localized in the same map can share their planned paths with each other before separating. The searcher can use its target's planned path to generate a set of* path states *along the path (Figure 5). Each state transitions to the next path state and multiple path states can be in the same location if the path intersects itself.*

Example 4 (Multiple behaviors)**.** *If a target has m possible behaviors, with behavior i described by $\mathcal{X}_i$, the searcher can model all the behaviors using $\mathcal{X} = \mathcal{X}_1 \cup \cdots \cup \mathcal{X}_m$. Each behavior can be thought of as a different* layer *of the HMM with some low probability transitions between layers to represent changing behavior. For example, a target with a known initial planned path has a top layer of path states which transitions to a middle layer of direction states which transitions to a bottom layer of* stuck states *which only transition to themselves (Figure 6). The transitions between layers are unidirectional because the robot will not return to an abandoned path and will not start moving after getting stuck.*

Example 5 (Historic data)**.** *If a robot behaves repetitively, historic or simulated data of a its behavior can be used to build a more accurate HMM. The structure of this HMM may consist of multiple layers including direction, path, transit, stuck, or other kinds of states. The transition probabilities between these states are determined using discretized versions of the historic paths. The model based on historic data can be verified by comparing the paths to the HMM's* stationary distribution *(Figure 7). A stationary distribution does not change when updated using $\boldsymbol{A}$ and it can be computed from the eigenvectors of $\boldsymbol{A}$ [39].*

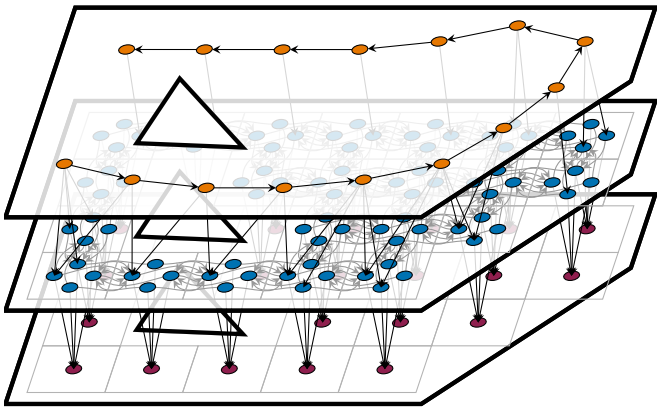

Fig. 6. A layered HMM containing a layer of path states (orange) which transitions to a layer of direction states (blue) which translates to a layer of stuck states (red). Each layer represents a different behavior.

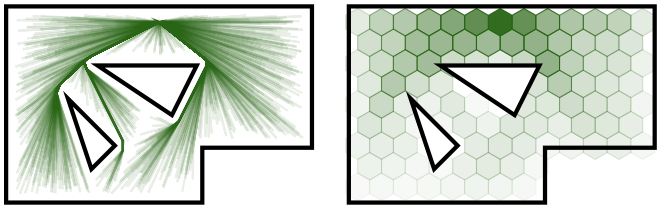

Fig. 7. Historic paths (left) can be used to determine transition probabilities of an HMM. The model's stationary distribution (right) closely resembles the density of paths.

V. EFFECTS OF OBSERVATIONS

The searcher uses an HMM, (4)–(5), to predict its target's behavior and location. However, it also has another useful source of information: its own observations. Whenever the searcher is not able to communicate with its target, it can use this negative observation to narrow the belief distribution and avoid searching in places it has been recently.

Bayes filtering is the general problem of using observations to update a belief [40]. Given some observation, we use the Bayes' filter to update each element of the belief by

$$w_i[\tau] \propto \mathbb{P}(\text{observation} \mid x[\tau] = x_i)\big(\boldsymbol{a}_i \boldsymbol{w}[\tau - 1]\big) \quad (6)$$

where $\boldsymbol{a}_i$ is the i^{th} row of $\boldsymbol{A}$ and w_i is the i^{th} element of $\boldsymbol{w}$, the belief that the robot's true state, x is x_i. The way that the first term of (6) is defined depends on the kind of observation.

A *negative observation* is when the searcher is not able to communicate with its target. The communication matrix, $\boldsymbol{C}$, defines the probability of communication by (2). Using the HMM projection, (5), the probability of communication in terms of the state belief is $\boldsymbol{w}_{\text{tar}} \boldsymbol{P}^\top \boldsymbol{C} \boldsymbol{b}_{\text{sea}}$. For a single state of $\mathcal{X}$, the first term in (6) is then [8]

$$\mathbb{P}(\text{no communication} \mid x[\tau] = x_i) = 1 - (\boldsymbol{p}^i)^\top \boldsymbol{C} \boldsymbol{b}_{\text{sea}}$$

where $\boldsymbol{p}^i$ is the i^{th} column of the projection matrix, $\boldsymbol{P}$. Substituting this expression back into (6), and aggregating over i the update rule for the whole vector is

$$\boldsymbol{w}[\tau] \propto \big(\mathbf{1} - \boldsymbol{P}^\top \boldsymbol{C} \boldsymbol{b}_{\text{sea}}[\tau]\big) \odot \big(\boldsymbol{A} \boldsymbol{w}[\tau - 1]\big) \quad (7)$$

where $\mathbf{1} \in \mathbb{R}^{|\mathcal{X}|}$ is the vector consisting of all ones and $\odot : \mathbb{R}^{|\mathcal{X}|} \times \mathbb{R}^{|\mathcal{X}|} \to \mathbb{R}^{|\mathcal{X}|}$ is the elementwise product.

VI. EVALUATING SEARCH PATHS

When a robot decides to search, it uses its belief, $\boldsymbol{b}$, to plan a finite length search path. Ideally, we would minimize the expected time required to find the target, but since it is impossible to guarantee that a searcher will find a mobile target in finite time, expected time is not a useful optimization criterion. Instead, we could maximize the probability of success [17], [23] but this criterion does not value finding the target quickly. To prioritize paths that find the target quickly, we use a discounted reward function [18]. For an infinitely long discrete path, p, the discounted reward is

$$J(p) = \sum_{\tau=1}^{\infty} \beta^{\tau-1} \Delta\phi[\tau] \quad (8)$$

where $\beta \in (0, 1]$ is the discount factor and $\Delta\phi[\tau]$ is the probability of finding the target at time τ when following p. Smaller β results in greedier behavior whereas β close to 1 results in more conservative behavior.

The probability of connecting to the target for the first time at time τ, $\Delta\phi[\tau]$, can be expressed as

$$\Delta\phi[\tau] = \phi[\tau] - \phi[\tau - 1] \quad (9)$$

where $\phi[\tau]$ is the probability that the two robots have been connected at least once by time τ. When updating the state belief using (7), the new belief must be normalized and the normalization factor is the inverse of the probability that the robots did not connect in the previous time step. Without normalizing, the update law,

$$\widehat{\boldsymbol{w}}[\tau] = \big(\mathbf{1} - \boldsymbol{P}^\top \boldsymbol{C} \boldsymbol{b}_{\text{sea}}[\tau]\big) \odot \big(\boldsymbol{A} \widehat{\boldsymbol{w}}[\tau - 1]\big) \quad (10)$$

with $\widehat{\boldsymbol{w}}[0] = \boldsymbol{w}[0]$ gives a vector, $\widehat{\boldsymbol{w}}[\tau]$, that sums to the probability of never connecting by time τ. Therefore

$$\phi[\tau] = 1 - \mathbf{1}^\top \widehat{\boldsymbol{w}}[\tau]. \quad (11)$$

Using (9)–(11) we can iterate through the vertices of a path and use the $\widehat{\boldsymbol{w}}$ and ϕ at the previous vertex (with $\widehat{\boldsymbol{w}}[0] = \boldsymbol{w}[0]$ and $\phi[0] = 0$) to compute $\widehat{\boldsymbol{w}}$, ϕ and $\Delta\phi$ at each vertex.

The discounted reward in (8) assumes an infinite length path. When searching, the searcher plans finite length paths, but can extend a path if it does not find the target. Therefore, we should compare finite length paths based on all the possible paths starting with a given finite-length path. Any path starting with p is guaranteed to have a reward of at least

$$J_{\min}(p) = \sum_{\tau=1}^{|p|} \beta^{\tau-1} \Delta\phi[\tau] \quad (12)$$

which would be achieved by a path with no chance of finding the target after the end of p. Similarly, it is impossible for any path starting with p to have a reward greater than

$$J_{\max}(p) = J_{\min}(p) + \beta^{|p|} \left(1 - \phi[\tau]\right) \quad (13)$$

which would be achieved by a path guaranteed to find the target one time step after the end of p. By their definitions, $J_{\min}$ increases monotonically and $J_{\max}$ decreases monotonically as more vertices are added to the end of a path.

Using these bounds, if $J_{\min}(p_1) \geq J_{\max}(p_2)$, then every path starting with p_1 is at least as good as any path starting with p_2 so there is no need to consider any paths starting with p_2. For paths starting with the same two vertices, this inequality is useful when using one planning tree as the basis of a new one.

Theorem 1. *Let p_1 and p_2 be two paths of length T whose first two vertices are $q[0], q[1]$ and let p_1' and p_2' be the same paths with $q[0]$ removed. Suppose that $J_{\min}(p_1) > J_{\max}(p_2)$ for some initial belief $\boldsymbol{w}[0]$. Then $J_{\min}(p_1') > J_{\max}(p_2')$ for the initial belief $\boldsymbol{w}[1]$ obtained by updating $\boldsymbol{w}[0]$ using* (7)*.*

Proof. Let $\Delta\phi_i[\tau]$ and $\Delta\phi_i'[\tau-1]$ represent the marginal probabilities at the same location along p_i and p_i'. The difference between these probabilities is conditioned on the fact that the robots were not connected at $q[1]$ so

$$\Delta\phi_i'[\tau] = \frac{\Delta\phi_i[\tau]}{1-\Delta\phi_i[1]}.$$

Using this relationship in (12)–(13), we can similarly express the reward bounds as

$$J_{\min}(p_i) = \Delta\phi_i[1] + \frac{1}{\beta(1-\Delta\phi_i[1])} J_{\min}(p_i') \tag{14}$$

$$J_{\max}(p_i) = \Delta\phi_i[1] + \frac{1}{\beta(1-\Delta\phi_i[1])} J_{\max}(p_i'). \tag{15}$$

Since both paths start with the same vertices, $\Delta\phi_1[1] = \Delta\phi_2[1]$ and the reward bounds get transformed by the same linear relationship. As linear relationships preserve inequalities, if $J_{\min}(p_1) > J_{\max}(p_2)$ then $J_{\min}(p_1') > J_{\max}(p_2')$. □

VII. SAMPLED SEARCH PATHS

To plan a search path that maximizes the discounted reward, we first construct a search tree using a sampling-based algorithm (Algorithm 1). Vertices are added to the tree based on the values of $J_{\min}$ and $J_{\max}$ for the path to that vertex from the root vertex. Each vertex contains a location, q, and the data—$\widehat{\boldsymbol{w}}$, ϕ, $\Delta\phi$, $J_{\min}$, and $J_{\max}$—required to compute the bounds iteratively from the parent vertex's data using (9)–(13). The root vertex is located at the searcher's location with its current belief, $\phi = \Delta\phi = J_{\min} = 0$, and $J_{\max} = 1$. As the tree grows, it keeps track of its best vertex, v^*, and prunes the tree whenever v^* is updated. The stopping criterion is either the size of the tree, the reward of v^*, or the number of rounds of the algorithm.

A. Growing the tree

The planning tree grows (Algorithm 2) by adding new vertices based on a randomly sampled location, $q_{\text{rand}} \in \mathcal{Q}$. The sampled location is used to create one candidate vertex, v_{new}, for each existing vertex of the tree. Unlike RRT which chooses v_{new} along the straight line from v to q_{rand}, our search algorithm chooses v_{new} along the shortest path in $\mathcal{Q}$. This method of selecting v_{new} biases the locations of new vertices based on the topology of $\mathcal{Q}$ resulting in vertices in topologically central locations which tend to be the most useful for search.

Algorithm 1: Search tree

Input: Environment, $\mathcal{Q} \subset \mathbb{R}^2$; searcher's location, $q_0 \in \mathcal{Q}$; and initial belief, $\widehat{\boldsymbol{w}}[0]$
Output: Search tree, $\mathcal{T}$

```
v_0 ← root vertex at q_0 with belief ŵ[0]
T ← planning tree consisting of v_0
v*(T) ← v_0                          /* Maximizes J_min */
while stopping criterion is not met do
    Grow T                           /* Algorithm 2 */
    if v*(T) has been updated then
        Prune T                      /* Algorithm 3 */
return T
```

Algorithm 2: Grow tree

Input: Environment, $\mathcal{Q} \subset \mathbb{R}^2$; and planning tree, $\mathcal{T}$
Output: Planning tree, $\mathcal{T}$, with new vertices added

```
q_rand ← uniformly random location in Q
for τ ∈ {0, ..., τ_max(T)} do
    for v ∈ T at depth τ do
        p ← shortest path from v to q_rand within Q
        q_new ← location one time step along p
        v_new ← vertex at q_new with parent v
        if J_min(v_new) > J_min(v*(τ)) then
            v*(τ) ← v_new
    Add v*(τ) to T
    if J_min(v*(τ)) > J_min(v*(T)) then
        v*(T) ← v*(τ)
return T
```

Candidate vertices are compared using $J_{\min}$. Since $J_{\min}$ increases monotonically along a path, most vertices at depth τ have a larger $J_{\min}$ than any vertices at depth $\tau' < \tau$. If we added a single vertex per round like in RRT, the algorithm would favor adding vertices deep in the tree. Similarly, if we added a single vertex based on $J_{\max}$, it would favor vertices adjacent to the root vertex. To avoid these biases and better balance growth and branching, we add multiple vertices—one at each possible depth—for each sampled location. After k rounds, the tree will have $\mathcal{O}(k^2)$ vertices and we will have checked $\mathcal{O}(k^3)$ candidate vertices. As each candidate vertex is checked in constant time, it takes $\mathcal{O}(n^{3/2})$ to construct a tree with n vertices.

B. Pruning the tree

We saw in Section VI that if $J_{\min}(v_1) \geq J_{\max}(v_2)$ then every path through v_1 is better than any path through v_2 could ever be. Using this criterion, we *prune* the tree to remove poor quality vertices from previous rounds. To prune, we keep track of the vertex, $v^*(\mathcal{T})$, that maximizes $J_{\min}$. Initially, in Algorithm 1, $v^*(\mathcal{T})$ is the root vertex; when Algorithm 2 adds vertices to the tree, it also updates $v^*(\mathcal{T})$. After $v^*(\mathcal{T})$ is updated, we prune the tree by removing all

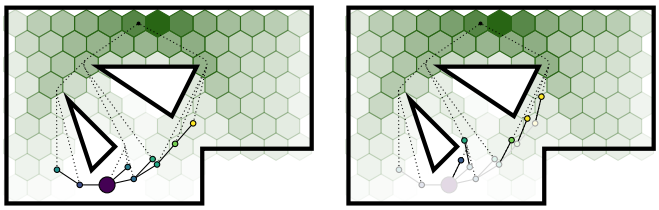

Fig. 8. A randomly sampled location is used to generate candidate vertices along the shortest paths to that location (left). Algorithm 1 adds the candidate vertex at each depth that maximizes $J_{\min}$ (low is blue; high is yellow).

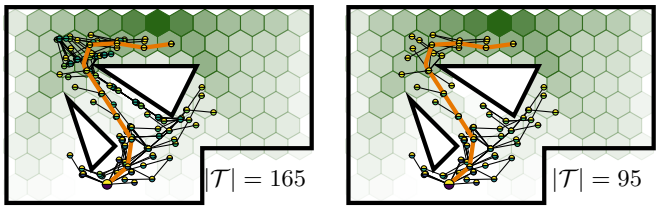

Fig. 9. Before pruning (left) a tree has some vertices whose $J_{\max}$ (bottom half of circle) is lower than some other vertex's $J_{\min}$ (top half of circle). Pruning removes these vertices, resulting in a smaller tree (right).

vertices with $J_{\max}(v) \leq J_{\min}(v^*(\mathcal{T}))$ as they are no longer useful (Algorithm 3). By pruning, we often remove close to half of the tree's vertices (Figure 9) making every future round of Algorithm 2 more efficient.

Algorithm 3: Prune tree

Input: Planning tree, $\mathcal{T}$
Output: Planning tree, $\mathcal{T}$, with some vertices removed

```
for vertex v ∈ T do
    if J_max(v) < J_min(v*(T)) then
        Remove v from T
return T
```

C. Re-rooting the tree

Using a search tree, $\mathcal{T}$, the best search strategy is to follow the path to v^*. After moving to the first vertex, v_1, of this path, if the searcher has not found its target, it can replan by building a new search tree, $\mathcal{T}'$ with root $v'_0 = v_1$. Rather than build $\mathcal{T}'$ from scratch, it can reuse any vertex of $\mathcal{T}$ which is a child of v'_0. This *re-rooting* process can save many of the original vertices (Figure 10).

By Theorem 1 all the reward bounds for children of v'_0 are updated in the same way by (14)–(15). Similar equations are used to update each vertex's other properties $\widehat{\boldsymbol{w}}$, ϕ and $\Delta\phi$. To re-root the tree (Algorithm 4), we update the properties of children of v'_0 using these rules and remove all other vertices.

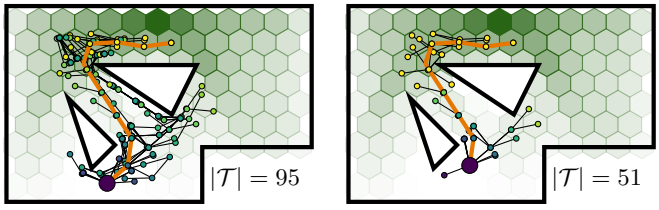

Fig. 10. Re-rooting takes an existing tree (left) and only keeps vertices that are children of the new root vertex. This new tree (right) has its vertices properties updated based on the new root.

This process is easiest if the vertices are sorted by depth so that children get updated after their parent and can be updated using the parent's updated data.

Algorithm 4: Re-root tree

Input: Planning tree, $\mathcal{T}$; and new root vertex, $v'_0 \in \mathcal{T}$
Output: Planning tree, $\mathcal{T}'$ rooted at v'_0

```
Create new tree T' with v'_0 as root vertex
Sort vertices of T by depth
for vertex v ∈ T do
    if v's parent is in T' then
        v' ← re-rooted version of v with v'_0 as root
        Add v' to T'
return T'
```

VIII. RESULTS

The search algorithm described in this paper could be used to reconnect teams of robots who are performing a variety of tasks, such as search, coverage, surveillance, or delivery. To evaluate its effectiveness we applied it to a simple relay scenario in an environment with communication limited by distance and line of sight (Figure 11). The target robot wanders by repeatedly following the shortest path to a randomly selected location. The searcher acts as a relay between the wandering robot and a stationary base station. After finding the target, it returns to the base station and then searches again. The search is successful when the two robots are within communication range (blue in Figure 11) and there are no obstacles between them. Both robots' linear velocities vary between 0.8–$1.2\,\mathrm{m/s}$ and angular velocities vary between 36–$54\,^\circ/\mathrm{s}$. In this semi-cooperative scenario, the searcher knows the target's general behavior—that it travels between randomly selected locations—but it does not know the target's current location or planned path. Based on this knowledge, the searcher uses an HMM derived from simulated target paths. The searcher is also aware that its communication is limited by distance and line of sight and uses this knowledge when evaluating potential search paths.

We compared the sampling-based search with $\beta = 5/6$ with two baseline algorithms:

- **Random**: The searcher chooses random locations in $\mathcal{Q}$ and follows the shortest paths between these locations until it finds its target.
- **Greedy**: The searcher uses a belief of the target robot updated using (7) and greedily follows the shortest path to the location with the highest belief.

For the sampling-based search, we tested various values of β and found that for $\beta > 1/2$, the performance was not very sensitive to β and performance decreased with β below $1/2$. For each algorithm, we simulated a scenario where the searcher finds the target and returns to the base station 500 times and recorded the times needed to find the target after leaving the base stations (Figure 12). The sampling-based mean time-to-find of $67.8\,\mathrm{s}$ was significantly faster than both

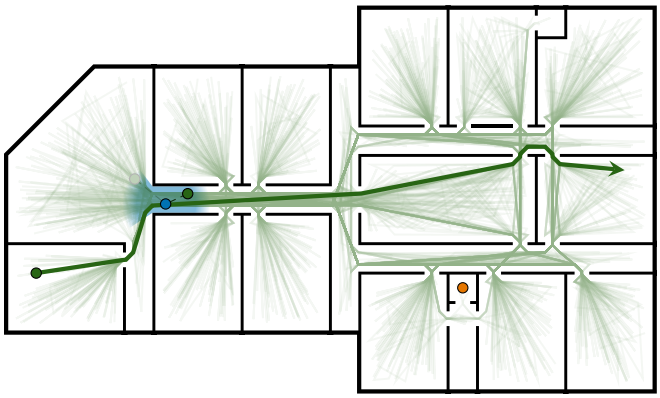

Fig. 11. The search simulations are set in an environment measuring 44 m by 26 m. Assuming there is a line of sight, the searcher (blue) can always connect with its target (green) if they are less than 2 m apart and the probability of connection decreases linearly to 0 probability at 3 m. The searcher must return to its base station (orange) whenever it finds its target. Its beliefs are updated using an HMM based on simulated target paths (green lines).

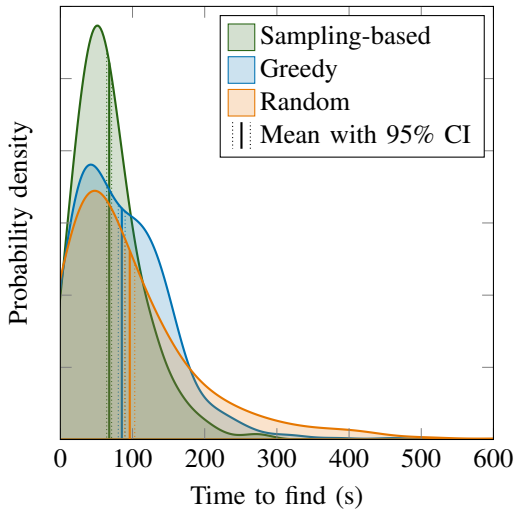

Fig. 12. Time-to-find distributions for three search algorithms based on 500 searches in the environment in Figure 11. The smoothed distributions were obtained by kernel density estimation using a Gaussian kernel with bandwidth 0.4. The 95% confidence interval for the means of each distribution were computed by bootstrapping with 1000 resamplings of the data.

the greedy (85.2 s) and random (96.2 s) searchers at the 95% confidence level using a two-sided t-test.

Although the mean times-to-find indicate better performance, the distributions in Figure 12 are not normal. Therefore, we also computed the 9 deciles for each distribution (Table I). The differences between the first 4 deciles, representing the fastest 40% of the searches, are not statistically significant at the 95% confidence level. These searches represent *easy* scenarios where the target robot happens to be relatively close to the base station and any searcher leaving the base station is likely to find it quickly. For the remaining slowest 60% of the searches, the sampling-based searcher is significantly faster at the 95% confidence level. These searches represent *difficult* searches where the target is far away and the searcher needs to follow a relatively long path while using a much wider belief distribution than in the easy searches. It is in these scenarios that the planning strategy can have the largest effect and for these scenarios the sampling-based strategy is significantly faster than either benchmark strategy.

TABLE I

DECILES, D_i, FOR THE DISTRIBUTIONS IN FIGURE 12. THE p-VALUES ARE THE PROBABILITIES THAT THE SAMPLING-BASED DECILE IS LOWER THAN THE OTHER ALGORITHM'S DECILE.

	Sampling	Greedy		Random	
i	D_i (s)	D_i (s)	p	D_i (s)	p
1	16.94	14.38	0.418	14.82	0.457
2	30.92	28.52	0.542	27.36	0.172
3	40.28	42.62	0.468	35.36	0.061
4	50.40	55.12	0.334	52.18	0.688
5	59.48	74.68	**0.001**	66.84	**0.047**
6	68.24	96.36	**0.000**	86.28	**0.005**
7	82.68	112.70	**0.000**	112.66	**0.000**
8	97.12	133.38	**0.000**	144.74	**0.000**
9	127.08	156.52	**0.000**	214.64	**0.000**

The total time needed for the searcher to find the target and return to the base station 500 times ranged from 6.8–8.1 h. The simulations were performed in C++ using a standard laptop computer running Linux and took between 2.4–3.5 h to perform. As the simulation times, which include the planning times, were lower than the real time needed to follow the planned search paths, our sampling-based search planner could be used in real time on a robot with hardware comparable to a standard laptop computer. Future work will involve more extensive evaluation on real robots using search while performing various kinds of tasks, and the application of RRT-like planning algorithms to other maximization problems such as data acquisition or coverage.

IX. CONCLUSIONS

The ability to search for a disconnected teammate is useful for robotic teams with limited communication which need to separate temporarily and cannot guarantee their exact behavior after separating. In this paper we presented a method of planning search paths over continuous space based on a *belief* of a semi-cooperative target robot's location. The belief is updated using an HMM built from historic data of the target's behavior or a planned path previously sent to the searcher. At every time step where the searcher does not find the target, it uses this negative observation to narrow the belief and help guide the search.

Using the belief, the searcher plans a path to maximize a discounted reward function that uses a discount factor to prioritize finding the target quickly. Search paths are planned by building a tree of possible search paths using a process similar to RRT. The locations of candidate vertices are based on the shortest path in the environment between an existing vertex and a sampled point, biasing the tree based on the environment's topology which tends to consider directions that are more useful for search. Candidate vertices are evaluated using bounds on the discounted rewards of every path passing through that vertex. Since the reward bounds are strongly dependent on depth in the tree, multiple vertices—one at each depth—are added per round. The reward bounds are also used to prune the tree by removing old vertices which

are no longer useful. The searcher searches by following the best path in the tree and can re-root the tree to use it as the basis of a new search tree at a new location. We compared this algorithm against benchmark greedy and random searchers. The three approaches required similar times to find the target in the fastest 40% of cases where the belief was narrow or the target was nearby and so search was easy. In the slower 60% of cases where the target was more difficult to find, the sampling-based algorithm significantly outperformed the two benchmarks.

REFERENCES

[1] Y. Mostofi, M. Malmirchegini, and A. Ghaffarkhah, "Estimation of communication signal strength in robotic networks," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2010, pp. 1946–1951.

[2] E. Stump, A. Jadbabaie, and V. Kumar, "Connectivity management in mobile robot teams," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2008, pp. 1525–1530.

[3] M. De Gennaro and A. Jadbabaie, "Decentralized control of connectivity for multi-agent systems," in *IEEE Conference on Decision and Control (CDC)*. IEEE, 2006, pp. 3628–3633.

[4] Y. Pei, M. Mutka, and N. Xi, "Coordinated multi-robot real-time exploration with connectivity and bandwidth awareness," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2010, pp. 5460–5465.

[5] I. Rekleitis, A. New, E. Rankin, and H. Choset, "Efficient boustrophedon multi-robot coverage: An algorithmic approach," *Annals of Mathematics and Artificial Intelligence*, vol. 52, no. 2, pp. 109–142, 2008.

[6] G. Hollinger and S. Singh, "Multirobot coordination with periodic connectivity: Theory and experiments," *IEEE Transactions on Robotics*, vol. 28, no. 4, pp. 967–973, 2012.

[7] Y. Kantaros and M. Zavlanos, "Distributed intermittent connectivity control of mobile robot networks," *IEEE Transactions on Automatic Control*, vol. 62, no. 7, pp. 3109–3121, 2017.

[8] I. Vandermeulen, R. Groß, and A. Kolling, "Re-establishing communication in teams of mobile robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 7947–7954.

[9] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Department, Iowa State University, Tech. Rep., 1998.

[10] B. Koopman, *Search and screening*. Center for Naval Analysis, Alexandria, Virginia, 1946.

[11] L. Stone, "OR forum—what's happened in search theory since the 1975 Lanchester prize?" *Operations Research*, vol. 37, no. 3, pp. 501–506, 1989.

[12] S. Benkoski, M. Monticino, and J. Weisinger, "A survey of the search theory literature," *Naval Research Logistics (NRL)*, vol. 38, no. 4, pp. 469–494, 1991.

[13] S. Brown, "Optimal search for a moving target in discrete time and space," *Operations Research*, vol. 28, no. 6, pp. 1275–1289, 1980.

[14] A. Washburn, "Search for a moving target: The FAB algorithm," *Operations Research*, vol. 31, no. 4, pp. 739–751, 1983.

[15] T. Chung, G. Hollinger, and V. Isler, "Search and pursuit-evasion in mobile robotics," *Autonomous Robots*, vol. 31, no. 4, pp. 299–316, 2011.

[16] C. Robin and S. Lacroix, "Multi-robot target detection and tracking: Taxonomy and survey," *Autonomous Robots*, vol. 40, no. 4, pp. 729–760, 2016.

[17] H. Lau, S. Huang, and G. Dissanayake, "Probabilistic search for a moving target in an indoor environment," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2006, pp. 3393–3398.

[18] G. Hollinger, S. Singh, J. Djugash, and A. Kehagias, "Efficient multi-robot search for a moving target," *The International Journal of Robotics Research*, vol. 28, no. 2, pp. 201–219, 2009.

[19] J. Banfi, J. Guzzi, A. Giusti, L. Gambardella, and G. Di Caro, "Fair multi-target tracking in cooperative multi-robot systems," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 5411–5418.

[20] H. Yu, R. Beard, M. Argyle, and C. Chamberlain, "Probabilistic path planning for cooperative target tracking using aerial and ground vehicles," in *American Control Conference (ACC)*. IEEE, 2011, pp. 4673–4678.

[21] A. Bayoumi, P. Karkowski, and M. Bennewitz, "Speeding up person finding using hidden Markov models," *Robotics and Autonomous Systems*, vol. 115, pp. 40–48, 2019.

[22] J. Riehl, G. Collins, and J. Hespanha, "Cooperative graph-based model predictive search," in *IEEE Conference on Decision and Control (CDC)*. IEEE, 2007, pp. 2998–3004.

[23] C. Geyer, "Active target search from UAVs in urban environments," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2008, pp. 2366–2371.

[24] B. Charrow, V. Kumar, and N. Michael, "Approximate representations for multi-robot control policies that maximize mutual information," *Autonomous Robots*, vol. 37, no. 4, pp. 383–400, 2014.

[25] Y. Sung and P. Tokekar, "Algorithm for searching and tracking an unknown and varying number of mobile targets using a limited FoV sensor," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 6246–6252.

[26] P. Dames, "Distributed multi-target search and tracking using the PHD filter," in *International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. IEEE, 2017, pp. 1–8.

[27] J. Royset and H. Sato, "Route optimization for multiple searchers," *Naval Research Logistics (NRL)*, vol. 57, no. 8, pp. 701–717, 2010.

[28] T. Bandyopadhyay, N. Rong, M. Ang, D. Hsu, and W. Lee, "Motion planning for people tracking in uncertain and dynamic environments," in *IEEE International Conference on Robotics and Automation (ICRA), Workshop on People Detection & Tracking*. IEEE, 2009.

[29] A. Sarmiento, R. Murrieta-Cid, and S. Hutchinson, "An efficient motion strategy to compute expected-time locally optimal continuous search paths in known environments," *Advanced Robotics*, vol. 23, no. 12-13, pp. 1533–1560, 2009.

[30] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[31] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *Robotics Science and Systems VI*, vol. 104, 2010.

[32] Y. Huang and K. Gupta, "RRT-SLAM for motion planning with motion and map uncertainty for robot exploration," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2008, pp. 1077–1082.

[33] L. Jaillet, J. Cortés, and T. Siméon, "Transition-based RRT for path planning in continuous cost spaces," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2008, pp. 2145–2150.

[34] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for a class of pursuit-evasion games," in *Algorithmic Foundations of Robotics IX*. Springer, 2011, pp. 71–87.

[35] A. Sintov and A. Shapiro, "Time-based RRT algorithm for rendezvous planning of two dynamic systems," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 6745–6750.

[36] F. Amigoni, J. Banfi, and N. Basilico, "Multirobot exploration of communication-restricted environments: A survey," *IEEE Intelligent Systems*, vol. 32, no. 6, pp. 48–57, 2017.

[37] Y. Mostofi, A. Gonzalez-Ruiz, A. Gaffarkhah, and D. Li, "Characterization and modeling of wireless channels for networked robotic and control systems—a comprehensive overview," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2009, pp. 4849–4854.

[38] A. Li, P. Penumarthi, J. Banfi, N. Basilico, J. O'Kane, I. Rekleitis, S. Nelakuditi, and F. Amigoni, "Multi-robot online sensing strategies for the construction of communication maps," *Autonomous Robots*, pp. 1–21, 2019.

[39] J. Norris and J. R. Norris, *Markov chains*. Cambridge University Press, 1998, no. 2.

[40] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.