Rapidly-exploring Random Cycles: Persistent Estimation of Spatio-temporal Fields with Multiple Sensing Robots

Xiaodong Lan and Mac Schwager

***Abstract*—This paper considers the problem of planning trajectories for both single and multiple sensing robots to best estimate a spatio-temporal field in a dynamic environment. The robots use a Kalman filter to maintain an estimate of the field value, and to compute the error covariance matrix of the estimate. Two new sampling-based path planning algorithms (RRC and RRC*) are proposed to find periodic trajectories for the sensing robots that minimize the largest eigenvalue of the error covariance matrix over an infinite horizon. The algorithms are proven to find the minimum infinite horizon cost cycle in a random graph, which grows by successively adding random points. The algorithms leverage recently developed methods for periodic Riccati recursions to efficiently compute the infinite horizon cost of the cycles, and they use the monotonicity property of the Riccati recursion to efficiently compare the costs of different cycles without explicitly computing their costs. The algorithms are demonstrated in a study using National Oceanic and Atmospheric Administration (NOAA) data to plan sensing trajectories in the Caribbean Sea. Our algorithms significantly outperform random, greedy, and receding horizon approaches in this environment.**

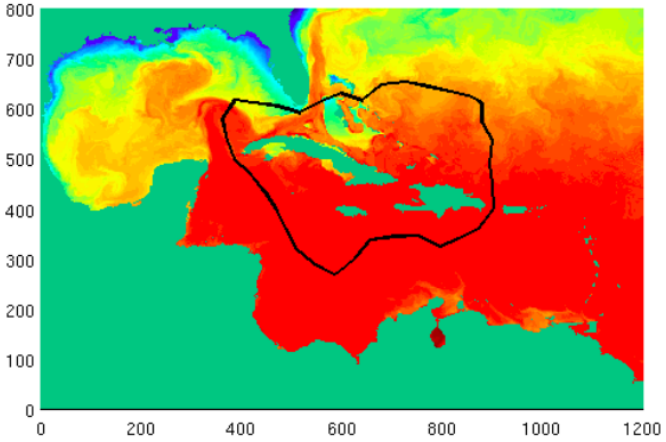


Fig. 1. The Caribbean Sea surface water temperature. Colormap denotes the temperature value with red standing for high values and blue standing for low values. The green area denotes land. One unit length in the plot is equal to 2.304 miles in geography. The black curve is a trajectory planned by our RRC* algorithm.

I. Introduction

In this paper we present two algorithms to plan periodic trajectories for sensing robots to monitor a continually changing field in their environment. Consider, for example, a team of Unpiloted Aerial Vehicles (UAV) with radiation sensors that must continually fly over the site of a nuclear accident to maintain an estimate of the time changing levels of radiation in the region. The objective of the vehicles is to execute a trajectory that optimizes, over an infinite horizon, the estimation accuracy. We build upon recent results concerning infinite horizon sensor scheduling [1], [2] and recent advances in sampling-based path planning [3] to design new path planning algorithms that return periodic trajectories with guaranteed infinite horizon sensing performance.

We model the environmental field with a model common in geological and environmental sciences which consists of a fixed number of spatial basis functions. We show that this model with Gaussian noise can be treated as a linear-Gaussian dynamical system. Furthermore, each sensing robot carries a point sensor which can only measure the field value at its current position with additive Gaussian white noise. This is a suitable model for a broad range of environmental scalar fields, including temperature fields, radio signal strength or electric field strength, radioactivity around a nuclear accident site, or fields of the concentration of a chemical contaminant such as oil around a leaking well in the ocean, or carbon monoxide in a region of the atmosphere, just to name a few. The model can also be applied to estimate the density of a population of interest, such as the concentration of people in different areas of a city, the density of vegetation over a forest, or the density of some species of animal over a wilderness area.

The sensing robots estimate the field using a Kalman filter, which is well known to minimize the mean squared estimation error for a linear system with Gaussian noise. However, our problem is distinguished from a typical estimation setting by the fact that our robot's sensing performance is a function of its position. Intuitively, the robot only obtains information about the part of the environment that is closest to it. Since the environment is dynamic the robot must perpetually move to maintain an estimate of the field value at different places. This is an instance of persistent monitoring [4], due to the infinite horizon and spatially distributed nature of the sensing task.

Our main contribution in this paper is two algorithms which we call Rapidly-expanding Random Cycles (RRC) and an improved variant, RRC*. These algorithms run offline, before the sensing agents are deployed, in order to determine periodic trajectories that the agents ought to take to collect the most valuable information for estimating the field. The output of both of these algorithms is a discrete time periodic trajectory (see Figure 1) for either a single agent or a group of agents to execute in order to minimize the asymptotic supremum (the $\limsup$) of a measure of the quality of their field estimate. We choose the spectral radius (i.e. the largest eigenvalue) of the error covariance matrix of the Kalman filter

X. Lan is with the Department of Mechanical Engineering, Boston University, Boston, MA 02215, USA, xlan@bu.edu.

M. Schwager is with the Department of Mechanical Engineering and the Division of Systems Engineering, Boston University, Boston, MA 02215, USA, schwager@bu.edu.

for the performance metric, although many other choices are possible. We specifically focus on finding periodic trajectories because recent results in optimal sensor scheduling [1], [2] have shown that an optimal infinite horizon sensor schedule can be arbitrarily closely approximated by a periodic schedule.

Finding the optimal infinite horizon trajectory is intractable in general, however the RRC and RRC* algorithms produce a series of periodic trajectories with monotonically decreasing cost. The algorithms work by expanding a tree of randomly generated candidate waypoints over the environment, a strategy inspired by popular sampling-based planning algorithms [3]. Given any two nodes in this tree, a cycle can be created by taking the unique path through the tree between the nodes, and adding to it an edge connecting the nodes. Our algorithms maintain a record of the minimal cost cycle that can be created from the tree in this way and iteratively search for a lower cost cycle by successively adding randomly sampled candidate waypoints to the tree.

The RRC* variant of our algorithm introduces a rewire procedure, similar to the rewire procedure from RRT* [3]. The rewire procedure in RRT* allows it to be asymptotically optimal since it rewires any vertex as long as a lower cost path leading to that vertex is found. However, due to the non-additivity of our cost function, the rewire procedure in RRC* only rewires the vertex along the currently minimal cost cycle, thus it does not lead to asymptotic optimality. As demonstrated in the numerical simulations, the estimation performance along the trajectories returned by RRC and RRC* significantly outperforms random search, greedy, and receding horizon algorithms.

In RRC and RRC* we need to evaluate the infinite horizon cost along different cycles. However, naively computing the infinite horizon cost is computationally intensive. Instead we leveraging a recent technique, called the Structure-preserving Swap and Collapse Algorithm (SSCA) and the Structure-preserving Doubling Algorithm (SDA) [5], for finding the asymptotic solution to periodic Riccati recursions efficiently. We also propose several efficient tests to compare the cost of one cycle with another without explicitly calculating its asymptotic cost.

Specifically, the main contributions of this paper are as follows:

- We propose the Rapidly-exploring Random Cycles (RRC) algorithm to find periodic trajectories for a single sensing robot to estimate a dynamic spatio-temporal field (Algorithm 1).
- We propose an improved variant of RRC, called RRC*, that incorporates a *rewire* procedure to take advantage of performance improvements from local changes in the tree (Algorithm 3).
- We prove that RRC and RRC* both give the minimal cost simple cycle that can be obtained from an expanding tree of random points in the environment (Theorem 1).
- We adapt the SSCA and SDA algorithm to evaluate the cost of cycles when needed, and we give a procedure for comparing the asymptotic cost of two cycles that does not require their cost to be computed explicitly.
- We extend the RRC and RRC* algorithms to give periodic trajectories for multiple sensing robots working together to estimate a spatio-temporal field.
- We use RRC and RRC* to plan periodic trajectories for estimating surface temperatures over the Caribbean Sea. The temperature model is identified by applying subspace identification algorithm [6], [7] with the raw observation buoy station data from US National Oceanic and Atmospheric Administration (NOAA). Our algorithms are shown to significantly outperform random search, greedy, and receding horizon algorithms in this environment.

A preliminary version of some of the material in this paper was presented in [8]. This paper presents much new material, including introduction of the RRC* algorithm, the multi-agent extensions of both RRC and RRC*, new cycle cost comparison techniques, and new numerical simulation results on a real spatio-temporal dynamic field (derived from NOAA data). This paper is organized as follows. Related work is discussed in Section II. The problem is formulated in Section III. Section IV describes our trajectory planning algorithms, and efficient computation of the asymptotic cost of periodic trajectories is described in Section V. Section VI presents the results of numerical simulations on a real ocean surface temperature model derived from raw NOAA data, and our conclusions are given in Section VII.

II. Related Work

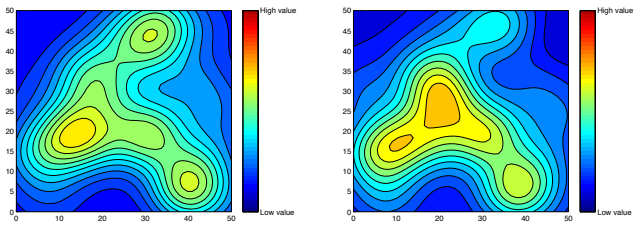
Our RRC and RRC* algorithms are inspired by incremental sampling-based path planning algorithms, also known as randomized path planners, particularly the Rapidly-exploring Random Tree (RRT) algorithm [9], and its recent variant with optimality guarantees, RRT* [3]. This algorithm is guaranteed to asymptotically approach the minimum cost feasible path from an initial point to a goal region almost surely if one exists. Another common algorithm called Probabilistic Roadmaps (PRM) is presented in [10], and a variant called PRM* is also proposed in [3] and shown to converge to an optimal path almost surely. However, unlike the path planners above, our algorithm uses a path cost related to sensing quality rather than distance. This sensing quality depends on the path history, which leads to the non-additivity of the path cost. Also, the trajectory that our algorithm returns is periodic by design, rather than having a fixed origin and destination.

Our search for a periodic trajectory is motivated by recent results in sensor scheduling, in which one among a set of senors must be chosen to take a measurement at each time step. Our problem is similar to sensor scheduling, except we have a continuum of possible "sensors" to select (i.e. positions from which to take measurements) and our measurements must be taken along a path that can be executed by our sensing agent. The authors of [1], [2], [11], [12] exploit well known monotonicity and concavity properties of the Riccati recursion to design sensor scheduling algorithms. The authors in [1], [2] prove that under some mild conditions, the optimal infinite horizon estimation cost can be approximated arbitrarily closely by a periodic schedule with a finite period. Optimization techniques are employed to find optimal or suboptimal periodic

sensor schedules in [13], [14], [15]. These algorithms are computationally expensive and they are only suitable for a finite number of sensors, while in our case we have an infinite number of "sensors" (possible sensing locations).

The error covariance matrix that indicates our sensing quality along a trajectory is given by a Discrete-time Periodic Riccati Equation (DPRE). The paper [16] proves that as long as the system is stabilizable and detectable, there exists a unique symmetric periodic positive semidefinite solution to the DPRE. There are several efficient and numerically stable approaches in the literature to solve for the DPRE. For example, [17] proposes the famous Kleinman procedure for the DPRE, [18] proposes the periodic QZ algorithm to solve for the DPRE, and [19] uses the QR-SWAP approach. Recently, [5] proposes the structure-preserving swap and collapse algorithm (SSCA) to reduce the DPRE into an equivalent Discrete-time Algebraic Riccati Equation (DARE) first, and then apply the structure-preserving doubling algorithm (SDA) to the DARE to solve for the solution. To the best of our knowledge, SSCA+SDA is the most efficient algorithm to solve the DPRE in the literature. We use the SSCA+SDA technique in RRC and RRC* to calculate the infinite horizon cost of periodic trajectories when necessary.

Our work is situated in the larger field of information gathering and persistent monitoring. In [4], [20], the authors present an algorithm for persistent monitoring by controlling the speed of a sensing agent along a given closed path. A similar problem is solved using optimal control techniques in [21], [22], [23]. In [21], the authors use infinitesimal perturbation analysis (IPA) to obtain a set of switching locations and waiting time for each agent in a 1-D mission space. In [22], given a closed path it is proved that multi-agent persistent monitoring with the minimum patrol period can be achieved by optimizing the agents' moving speed and initial locations on this path. In [23], the matrix version of Pontryagin's Minimum Principle is used to find a set of necessary conditions the optimal trajectory for persistent monitoring has to satisfy. Other flavors of persistent monitoring problems are reported in [24], [25]. For information gathering, different approaches are proposed in [26], [27], [28], [29], [30], [31], [32], [33]. Among them, the authors in [30], [33] use sampling-based technique to search for an informative path, but they do the sampling in a known, static information space. This is different from our work since the information space is dynamic in our case. In our problem we can not plan in the information space directly. One reason is that the information space (error covriance matrix cone) is unbounded and there is no way to sample this space uniformly. The second reason is that there may be no path existed to connect two sample points in the information space. The third reason is that sampling in the information space is computationally expensive because of the high dimensionality of the information space. Therefore, in this work we will do sampling and planning in the sensing agent's motion space. The price to pay is that we have to calculate the corresponding information gathered along a trajectory in the motion space. We calculate this using the SSCA+SDA algorithm to solve for the Riccati equation updating the error covariance matrix. Our goal is that we can plan a trajectory for the sensing agent in its motion space such that it can gather as more information as possible, thus minimizing the estimation uncertainties.



(a) Dynamic field at time t_0 (b) Dynamic field at time t_1

(c) Dynamic field at time t_2 (d) Dynamic field at time t_3

Fig. 2. An example of a dynamic spatio-temporal field modeled using Gaussian radial basis functions is shown at different time instants. The color map indicates the field values and points on the same contour line have the same field value.

III. PROBLEM FORMULATION

A. Spatio-temporal Field

First, we introduce the mathematical model of the spatiotemporal field. We adopt a form that is commonly used to model environmental fields in the geological and environmental sciences [34], [35]. Consider a discrete time dynamic scalar field $\phi_t(q) : \mathcal{Q} \times \mathbb{R}_{\geq 0} \mapsto \mathbb{R}$, where $\mathcal{Q} \subset \mathbb{R}^2$ is the domain of interest in the environment, and $q \in \mathcal{Q}$ is an arbitrary point in the domain. The domain of interest may be nonconvex, and may have obstacles with complex, nonconvex geometries. Suppose that the field can be decomposed as the dot product of a row vector of *static* spatial basis functions $C(q) = [c_1(q) \cdots c_n(q)]$ and a column vector of *time-changing* weights $a_t = [a_t^1 \cdots a_t^n]^T$, so that

$$\phi_t(q) = C(q)a_t. \tag{1}$$

Although it may appear to be limiting, this decomposition is quite general. For example, given any analytic spatiotemporal field, $\phi_t(q)$, we can take a truncated Taylor series expansion, or Fourier series expansion, (or many other kinds of expansions) to yield $C(q)a_t$. In this case $C(q)$ will be the vector of 2D polynomial bases for the Taylor expansion, or a vector of 2D Fourier bases for the Fourier expansion, and a_t is the vector of Taylor or Fourier coefficients at each time. In this work we choose Gaussian radial basis functions as the bases, so that the ith element of $C(q)$ is given by $c_i(q) = Ke^{-\|q-q_i\|^2/2\sigma_c^2}$, where q_i is the center position of the basis function, σ_c is its variance and K is a scaling constant. An example of a spatiotemporal scalar field modeled in this way is shown in Figure 2.

To model the time changing coefficients a_t, we let them be the state of a linear discrete-time stochastic system of the form

$$a_{t+1} = Aa_t + w_t \tag{2}$$

where A is a $n \times n$ matrix and $w_t \in \mathbb{R}^n$ is Gaussian white noise process with distribution $w_t \sim N(0, Q)$, and Q is the covariance matrix which is positive semidefinite. The dynamics matrix A and the noise error covariance matrix Q have to be known for us to estimate the vector a_t. In this work we will use subspace identification algorithms [6], [7] to learn A and Q first from real data before we apply our path planning algorithms to estimate a_t. Let $t_0 = 0$ denote the initial time and a_0 the initial state vector. It is assumed that a_0 is Gaussian distributed, independent of w_t. Since the basis functions $C(q)$ are fixed and known, estimating the field $\phi_t(q) = C(q)a_t$ is equivalent to estimating the unknown time-varying state vector a_t.

B. Robot Dynamics and Sensor Model

The sensing robot moves along a trajectory in the environment and takes measurements. Assume we have $N_s \geq 1$ mobile sensors, with simple first order dynamics given by

$$x^i_{t+1} = x^i_t + u^i_t, \quad \|u^i_t\| \leq \eta, \quad x^i(t_0) = x^i_0$$

where $x^i_t \in \mathbb{R}^2$ is the position of robot i at time t, x^i_0 is the initial position of the robot i, u^i_t is the control input, and $\eta > 0$ is the largest distance the robot can travel in one time step. We consider these simple dynamics so as not to complicate the scenario, although more complex dynamics $x^i_{t+1} = f^i_t(x^i_t, u^i_t)$ can be accommodated with some modifications. Each robot carries a point sensor with which it can measure the field value at its current location, with some additive sensor noise

$$y^i_t = \phi_t\big(x^i_t\big) + v^i_t = C\big(x^i_t\big)a_t + v^i_t \tag{3}$$

where v^i_t is Gaussian white noise, uncorrelated with w_t, and with Gaussian distribution $v^i_t \sim N(0, R^i)$. Our goal is to plan trajectories for the sensing agents such that by moving along the planned trajectories while estimating the field, they minimize a measure of the estimation uncertainty. In this paper, we will consider planning in the joint state space of the mobile sensors $x_t = \big((x^1_t)^T, (x^2_t)^T, \ldots, (x^{N_s}_t)^T\big)^T \in \mathbb{R}^{2N_s}$. Similarly, $y_t = \big((y^1_t)^T, (y^2_t)^T, \ldots, (y^{N_s}_t)^T\big)^T \in \mathbb{R}^{N_s}$. We also let $C(x_t) = \big(C^T(x^1_t), C^T(x^2_t), \ldots, C^T(x^{N_s}_t)\big)^T \in \mathbb{R}^{N_s \times n}$, and $R = \text{diag}(R^1, R^2, \ldots, R^{N_s})$, where diag($\bullet$) is the diagonal matrix with the diagonal entries being $\bullet$.

C. Trajectory Quality

In order to measure the estimation performance from moving along a trajectory, an estimation performance metric must be defined. We use the Kalman filter to estimate the time-varying weights a_t, which evolve according to (2). The Kalman filter is known to be the optimal estimator for this model in the sense that it minimizes the mean squared estimation error [36]. Then let $\hat{a}_t = \mathbb{E}[a_t \mid y_0, \ldots, y_{t-1}]$ be the one-step-ahead predicted mean from the Kalman filter, and $\Sigma_t = \mathbb{E}[(a_t - \hat{a}_t)(a_t - \hat{a}_t)^T]$ is the so-called *a priori* error covariance matrix associated with this estimate.[1] From the Kalman filter equations, we can obtain the well-known Riccati equation to update the *a priori* error covariance matrix

$$\Sigma_{t+1} = A\Sigma_t A^T - A\Sigma_t C(x_t)^T \times \big(C(x_t)\Sigma_t C(x_t)^T + R\big)^{-1} C(x_t)\Sigma_t A^T + Q. \tag{4}$$

We choose the spectral radius (i.e. the largest eigenvalue) of the *a priori* error covariance matrix, $\rho(\Sigma_t)$, as the objective function. We choose this instead of, for example, the trace of the covariance matrix because we want to maintain a good estimate evenly over the environment. It is possible to have a good estimate in one location and a poor estimate in another and still have a relatively small trace, whereas the spectral radius is an indication of the worst estimation in the environment. Our goal is to design trajectories for the N_s agents such that when they move along their trajectories the spectral radius of the error covariance matrix will be minimized over an infinite horizon. However, in order not to be influenced by initial transient effects we consider the asymptotic limit of the spectral radius

$$J(\sigma, \Sigma_0) = \limsup_{t \to \infty} \rho(\Sigma^\sigma_t), \tag{5}$$

where σ denotes a trajectory for the agent, and Σ^σ_t is the error covariance attained at time t along the trajectory σ from initial condition Σ_0. Note that the limit of the supremum always exists even though Σ^σ_t does not necessarily converge as t goes to infinity.

Denote the robot position state space by $X \subset \mathbb{R}^{2N_s}$, the free state space and the obstacle region by X_{free} and X_{obs}, respectively. A feasible trajectory is $\sigma : \mathbb{Z}_{\geq 0} \mapsto X_{free}$, that is, $\sigma_t \in X_{free}$ for $t \in \mathbb{Z}_{\geq 0}$. In our case, trajectories are characterized by a sequence of discrete waypoints, $\{x_0, x_1, \ldots, x_\infty\}$, connected by straight lines which are traversed at a speed bounded by η. Denote by $M^\infty(x_0)$ all the feasible trajectories with initial condition $x_0 \in X_{free}$ in infinite time horizon. Denote by Σ^σ_t the covariance matrix along the trajectory σ at time t from initial condition Σ_0. Denote by $\mathcal{A}$ the cone of semi-definite matrices. The main notation used in this paper is summarized in Table I.

Next we give a formal statement of the problem considered in this paper.

Problem 1 (Persistent Estimation Problem)**.** *Given a bounded and connected domain of interest $\mathcal{Q}$ with a spatio-temporal field $\phi_t(q)$, find an optimal feasible trajectory σ^* such that*

$$\sigma* \in \underset{\sigma \in M^\infty(x_0)}{\textit{argmin}} \; J(\sigma, \Sigma_0) \tag{6}$$

subject to

$$\Sigma^\sigma_{t+1} = A\Sigma^\sigma_t A^T - A\Sigma^\sigma_t C(x_t)^T \times \big(C(x_t)\Sigma^\sigma_t C(x_t)^T + R\big)^{-1} C(x_t)\Sigma^\sigma_t A^T + Q$$
$$\eta \geq \|x^i_{t+1} - x^i_t\|, i = 1, 2, \ldots, N_s.$$

[1] We could also use the *a posteriori* error covariance matrix, $\mathbb{E}[(a_t - \mathbb{E}[a_t \mid y_0, \ldots, y_t])(a_t - \mathbb{E}[a_t \mid y_0, \ldots, y_t])^T]$, but the expressions are simpler with the *a priori* form.

TABLE I
MAIN NOTATION USED IN THIS PAPER.

N_s	Number of sensing agents.
x_t	Column vector of size $2N_s$ by 1, representing concatenated positions of N_s sensing agents.
$\phi_t(q)$	Scalar field value at point q at time t.
a_t	Column vector of size n by 1, time-varying weights put on the n spatial basis functions.
$C(x_t)$	Matrix of size N_s by n, parametric spatial basis functions depending on positions in the environment.
y_t	Column vector of size N_s by 1, concatenated measurements from N_s sensors.
σ	A collision-free discrete trajectory consisting of a set of waypoints in the joint state space.
$\tilde{\sigma}$	A **periodic** collision-free discrete trajectory with period T.
η	The maximum distance one sensing agent can travel in one time step in $\mathbb{R}^2$.
$M^{\infty}(x_0)$	All the collision-free trajectories in infinite time horizon starting at position x_0.
$\rho(\bullet)$	Largest eigenvalue of matrix $\bullet$.
$J(\sigma, \Sigma_0)$	Infinite horizon estimation uncertainty along a trajectory σ starting with initial covariance matrix Σ_0.
$\mathcal{A}$	Cone of semi-definite matrices.
g_{x_t}	Riccati mapping.
$G_x^{\tau_1:\tau_2}$	Composite Riccati mapping over an interval of time steps $\tau_1, \tau_1+1, \ldots, \tau_2$.
Σ_t	Error covariance matrix.
Σ_t^{σ}	Error covariance matrix along trajectory σ.
$\Sigma_{\infty}^{x_i}$	Asymptotic covariance matrix at position x_i of a periodic trajectory $\tilde{\sigma}$.

Remark 1. *Note that in Problem 1, the cost function will keep increasing because of the process noise w_t unless measurements are taken infinitely often in enough locations in the environment to decrease estimation uncertainties, hence this will drive the sensing robots to continuously move around in the environment.*

IV. RAPIDLY-EXPLORING RANDOM CYCLES (RRC)

In this section we propose an incremental sampling-based planning algorithm to give approximate solutions to Problem 1 by finding periodic trajectories with finite periods. Our algorithm is built upon the RRT* algorithm [3]. However, unlike RRT*, our algorithm returns a cycle instead of a path. Before we discuss our algorithm, we first introduce the motivation to plan for periodic trajectories.

A. Motivation to Search for Periodic Trajectory

First, we define a Riccati map $g_{x_t} : \mathcal{A} \mapsto \mathcal{A}, t \in \{1, 2, \ldots\}$ as

$$g_{x_t} := A\Sigma_t A^T - A\Sigma_t C(x_t)^T \times \left(C(x_t)\Sigma_t C(x_t)^T + R\right)^{-1} C(x_t)\Sigma_t A^T + Q \quad (7)$$

where $\mathcal{A}$ is the cone of semi-definite matrices. Initially, when $t = 0$, then $g_{x_0} = A\Sigma_0 A^T + Q$. Consider the error covariance over an interval of time steps $\tau_1, \tau_1+1, \ldots, \tau_2$, then we define the repeated composition of g_{x_t} over the interval as

$$G_x^{\tau_1:\tau_2}(\Sigma_{\tau_1}) := g_{x_{\tau_2}}(g_{x_{(\tau_2-1)}}(\cdots g_{x_{\tau_1}}(\Sigma_{\tau_1}))), \quad (8)$$

so that $\Sigma_{(\tau_2+1)} = G_x^{\tau_1:\tau_2}(\Sigma_{\tau_1})$.

Two useful properties of the Riccati recursion (7) are given in the following Lemma.

Lemma 1 (Riccati Properties)**.** *For any $\Sigma_1, \Sigma_2 \in \mathcal{A}$ and $\alpha \in [0, 1]$, we have*

1) *(monotonicity) If $\Sigma_1 \preceq \Sigma_2$, then $g_{x_t}(\Sigma_1) \preceq g_{x_t}(\Sigma_2)$,*
2) *(concavity) $g_{x_t}(\alpha\Sigma_1 + (1-\alpha)\Sigma_2) \succeq \alpha g_{x_t}(\Sigma_1) + (1-\alpha) g_{x_t}(\Sigma_2)$.*

Proof. These two properties are well-known results and their proofs can be found in [37]. □

These properties carry over directly to the composition of several Riccati maps, and stated in the following Corollary.

Corollary 1 (Composite Riccati Properties)**.** *For any $\Sigma_1, \Sigma_2 \in \mathcal{A}$, $\alpha \in [0, 1]$, and $\tau_1, \tau_2 \in \{0, 1, 2, \ldots\}$, where $\tau_1 < \tau_2$ we have*

1) *(monotonicity) If $\Sigma_1 \preceq \Sigma_2$, then $G_x^{\tau_1:\tau_2}(\Sigma_1) \preceq G_x^{\tau_1:\tau_2}(\Sigma_2)$,*
2) *(concavity) $G_x^{\tau_1:\tau_2}(\alpha\Sigma_1 + (1-\alpha)\Sigma_2) \succeq \alpha G_x^{\tau_1:\tau_2}(\Sigma_1) + (1-\alpha) G_x^{\tau_1:\tau_2}(\Sigma_2)$.*

Proof. This follows directly by applying Lemma 1 repeatedly. □

Based on these two properties of the composite Riccati map, we extend the results from [1], [2] to our problem in the following corollary. Let $\tilde{\sigma}$ be a periodic trajectory with period T, so that $\tilde{\sigma}(t) = \tilde{\sigma}(t+T)$. Let the T waypoints in $\tilde{\sigma}$ be given by $(x_1, x_2 \ldots, x_T)$, and denote the composite Riccati map that starts at x_i and ends at x_i after moving around the cycle once by $G_{x_i}^{\tilde{\sigma}} := G_x^{i:T-i+1}$.

Corollary 2. *For any $\delta > 0$, Σ_0 and x_0, if there exists an optimal trajectory for Problem 1 with optimal cost J^*, then there exists a periodic trajectory $\tilde{\sigma}$ consisting of waypoints $(x_1, x_2, \ldots, x_T)$ with a finite period $T(\delta) \in \mathbb{Z}_{>0}$ such that*

1) *$0 \le J(\tilde{\sigma}) - J^* \le \delta$,*
2) *The trajectory of the covariance matrix $\Sigma_t^{\tilde{\sigma}}$ converges exponentially to a unique limit cycle, $\Sigma_{\infty}^{x_1}, \Sigma_{\infty}^{x_2}, \ldots, \Sigma_{\infty}^{x_T}$, where $\Sigma_{\infty}^{x_i}$ is the fixed point of the composite Riccati map $\Sigma_{\infty}^{x_i} = G_{x_i}^{\tilde{\sigma}}(\Sigma_{\infty}^{x_i})$, for all $i = 1, 2, \ldots T$,*
3) *J^* is independent of Σ_0.*

Proof. The proof of the first two parts follows directly from theorem 4 in [1]. The proof of the third part follows from theorem 2 in [1]. □

Remark 2. *Part 1 of this corollary indicates that if there exists an optimal trajectory for the persistent estimation problem, then we can always find a closed periodic trajectory whose cost is arbitrarily close to the optimal cost. The period of the periodic path is determined by how close we want the two costs to be.*

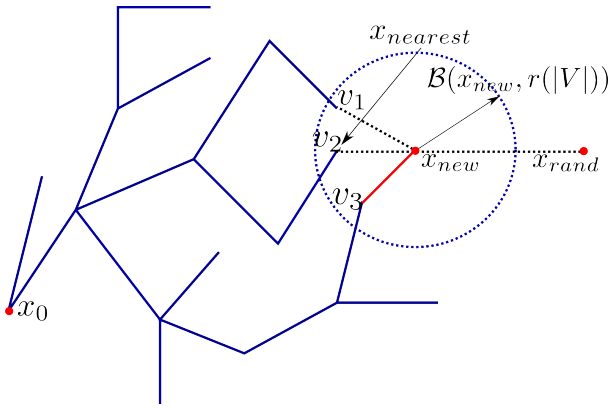


Fig. 3. This figure shows the expanding tree in the RRC algorithm.

Remark 3. *Part 2 of this corollary indicates that the cost of the periodic trajectory can be expressed as*

$$J(\tilde{\sigma}) = \rho(\Sigma_\infty^{\tilde{\sigma}}) = \max_{i\in\{1,2,\ldots,T\}} \rho(\Sigma_\infty^{x_i}) \tag{9}$$

Remark 4. *Part 3 of this corollary indicates that if a trajectory is optimal for some initial covariance matrix Σ_0, then it is also optimal for any other initial covariance matrix $\Sigma_0^{'}$.*

Based on this result, next we propose an algorithm called Rapidly-exploring Random Cycles (RRC) to search for periodic trajectories to estimate the spatio-temporal field.

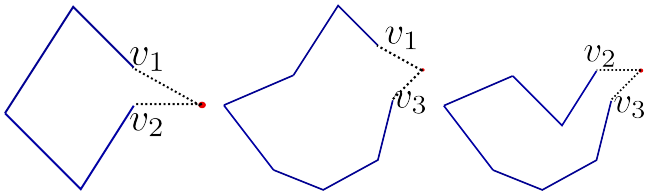
B. Rapidly-exploring Random Cycles (RRC)

Since the cost of a periodic trajectory can be arbitrarily close to the optimal cost of Problem 1, in [8], we propose an algorithm called *Rapidly-exploring Random Cycles (RRC)* to return a periodic trajectory for a single agent to estimate the field. Here we first review the algorithm, then we extend the algorithm to account for multiple sensing agents. RRC incrementally builds a tree graph $G = (V, E)$ embedded in X_{free}. At each iteration, we generate a new vertex to expand the tree and connect the new vertex to the tree in order to get new simple cycles[2] with better estimation performance. The tree is maintained so that all edges are feasible (they do not cause the agent to intersect with an obstacle) and they respect the kinematic constraints of the agent, $\|x_{t+1}^i - x_t^i\| \leq \eta$ for $i = 1, 2, \ldots, N_s$. Before describing the operation of the algorithm in detail, we provide a description of the primitive procedures used in the algorithm. Among all the primitive procedures, FindCycle and PersistentCost are unique to RRC and all the other procedures are the same with the primitives used in RRT*.

1) Primitive Procedures:

Sampling: Let $\mathsf{SampleFree} : \mathbb{Z}_{>0} \to X_{free}$ return independent identically distributed (i.i.d.) samples from X_{free}. In this paper, the samples are assumed to be uniformly distributed. In Figure 3, it returns x_{rand}.

[2]A simple cycle is one that visits no vertex and edge more than once.



(a) Potential cycle 1. (b) Potential cycle 2. (c) Potential cycle 3.

Fig. 4. This figure shows the new potential cycles found by adding the new point x_{new} to the tree in Figure 3.

Nearest Neighbor: Given a graph $G = (V, E)$ and a point $x \in X_{free}$, the function $\mathsf{Nearest} : (G, x) \to v$ returns a vertex $v \in V$ which is closest to x in terms the Euclidean distance, $\mathsf{Nearest} : (G = (V, E), x) = \text{argmin}_{v\in V} \|x - v\|$. In Figure 3, it returns $x_{nearest}$.

Steering: Given two points $x, z \in X$, a positive real number $\eta \in \mathbb{R}_{>0}$, the function $\mathsf{Steer} : (x, z, \eta) \to x'$ returns a point $x' \in X$ such that $\|x' - z\|$ is minimized while $\|x' - x\| \leq \eta$. That is, $\mathsf{Steer}(x, z, \eta) = \text{argmin}_{x'\in\mathcal{B}_{x,\eta}} \|x' - z\|$, where $\mathcal{B}_{x,\eta} := \{x' \in X | \|x' - x\| \leq \eta\}$. In Figure 3, it returns x_{new}.

Near Vertices: Given a graph $G = (V, E)$, a point $x \in X_{free}$ and a positive real number $r \in \mathbb{R}_{>0}$, the function $\mathsf{Near} : (G, x, r) \to V^{'} \subseteq V$ returns the vertices in V that are contained in a closed ball of radius r centered at x. That is, $\mathsf{Near}(G = (V, E), x, r) = \{v \in V, v \in \mathcal{B}_{x,r}\}$. Similarly to RRT*, we choose r as a function of the number of vertices in the graph: $r(|V|) = \min\{\gamma(\log|V|/|V|)^{1/d}, \eta\}$, where $\gamma > \gamma^* = 2(1 + 1/d)^{1/d}(\mu(X_{free})/\zeta_d)^{1/d}$, $\mu(X_{free})$ is the volume of the free space, ζ_d is the volume of the unit ball in $\mathbb{R}^d$, and $|V|$ denotes the number of vertices in V. In Figure 3, it returns v_1, v_2, v_3.

Collision Test: Given two points $x, z \in X_{free}$, the Boolean function $\mathsf{CollisionFree}(x, z)$ returns True if the straight line segment between x and z lies in X_{free} and False otherwise.

Find Cycles: Given a spanning tree $T = (V, E')$ of a graph $G = (V, E)$, and a point $x \in X_{free}$, the function $\mathsf{FindCycle} : (v_1, v_2, x) \to C_{v_1,v_2,x}$ returns the index of the vertices in a cycle, where v_1 and v_2 are any two vertices in the spanning tree $T = (V, E')$, and $C_{v_1,v_2,x}$ is a cycle which includes vertices v_1, v_2 and x. That is, $\mathsf{FindCycle}(v_1, v_2, x) = (P_{v_1,v_2}, (v_1, x), (v_2, x))$, where P_{v_1,v_2} is the unique path between v_1 and v_2, and (v_i, x) is the edge between v_i and x. In Figure 3, it returns 3 cycles, shown in Figure 4.

Calculating Cycle Cost: Given a cycle returned by the function FindCycle, the function PersistentCost returns the infinite horizon cost of the cycle. We will discuss how to calculate this cost in detail in Section V. That is, $\mathsf{PersistentCost} : C_{v_1,v_2,x} \to \mathbb{R}_{>0}$.

2) Generating a Periodic Path: We now describe the RRC algorithm in detail (as shown in Algorithm 1), using the primitive procedures introduced above. The algorithm starts

with an initial position x_0. We also initialize the algorithm by $minCycleIndex = \emptyset$ and $minCycleCost = \infty$. The variable $minCycleIndex$ is used to store the index of all the vertices in the cycle with minimum cost, and $minCycleCost$ is used to store the cost of the cycle. In this algorithm, we will keep a tree structure. At each step, we generate a new random point x_{rand} by the function SampleFree. With this new random point and the function Nearest, we find the nearest vertex $x_{nearest}$ of the graph to the random point. Then we apply the Steer function to get a potential new vertex x_{new} for the tree. These procedures are the same with the corresponding procedures in RRT*.

From the function Near, we get a set X_{near} which includes the near neighbors of x_{new}. If there is only one vertex in X_{near}, i.e. $x_{nearest}$, then we just connect x_{new} to $x_{nearest}$. If there are more than one vertices in X_{near}, then we do several connecting tests. We call the procedure RRC extend, see Algorithm 2. That is, we do a test by connecting x_{new} to any two vertices inside X_{near}. By connecting the new point with two vertices, we form a single cycle, which is made up of the two edges connecting the vertices to the new point plus the unique path through the tree between the two vertices. If there are k vertices in X_{near}, then we will have to consider $\frac{k(k-1)}{2}$ cycles formed in this way, see Figure 4. We compare the cost of these cycles and choose the one with the minimum cost. We can use the procedure PersistentCost to calculate these costs. After we find out the cycle with minimum cost among the $\frac{k(k-1)}{2}$ cycles, we give the value of the minimum cost to $cycleCost$. We also look for the indices of the vertices inside this cycle using the function FindCycle, and give them to $cycleIndex$. The variable $cycleVertexIndex$ is used to store either of the two vertices which belong to X_{near} and $cycleIndex$. That is, $cycleVertexIndex \in (X_{near} \cap cycleIndex)$. Then we connect x_{new} to the vertex in $cycleVertexIndex$. So we only add one new edge to the spanning tree, and the new graph will still be a spanning tree.

After the extend procedure, we compare the cost of the cycle $cycleIndex$ with $minCycleCost$. If it is smaller, then we update the $minCycleCost$ and make it equal to the cost of the new cycle. That is, $minCycleCost = cycleCost$. We also update $minCycleIndex$ by $minCycleIndex = cycleIndex$. We repeat this process for $numSteps$ times and we get a tree with $numSteps + 1$ vertices and a fundamental cycle of this tree, i.e., $minCycleIndex$. This cycle is the periodic trajectory for Problem 1. A pseudo-code implementation of the algorithm is shown in Algorithm 1. The following theorem characterizes the performance of the algorithm.

Theorem 1 (Properties of RRC algorithm)**.** *At each iteration, the RRC algorithm gives the minimal cost feasible simple cycle that can be created from the graph G by adding a single edge. This cost is monotonically non-increasing in the number of iterations.*

Proof. The proof is by induction. Let $\tilde{\sigma}_i$ be the trajectory given by the algorithm at iteration i, and denote by Ξ_i the set of all feasible simple cycles that can be created by adding one edge to G_i. Assume that $\tilde{\sigma}_i = \text{argmin}_{\tilde{\sigma}\in\Xi_i} J(\tilde{\sigma})$. In iteration $i+1$ a single vertex x_{new} is added to the graph, during which all feasible cycles that include this vertex are compared. Denote these new cycles by $\Delta\Xi_{i+1}$, and note that $\Xi_{i+1} = \Xi_i \cup \Delta\Xi_{i+1}$. The cycle with minimal cost among $\{\tilde{\sigma}_i, \Delta\Xi_{i+1}\}$ is taken to be $\tilde{\sigma}_{i+1}$, hence $\tilde{\sigma}_{i+1} = \text{argmin}_{\tilde{\sigma}\in\{\tilde{\sigma}_i,\Delta\Xi_{i+1}\}} J(\tilde{\sigma}) = \text{argmin}_{\tilde{\sigma}\in\Xi_i\cup\Delta\Xi_{i+1}} J(\tilde{\sigma}) = \text{argmin}_{\tilde{\sigma}\in\Xi_{i+1}} J(\tilde{\sigma})$. The initial case for the induction follows from the fact that x_0 is the minimum cost cycle in its own trivial graph. To prove monotonicity, notice that $\Xi_{i+1} \supset \Xi_i$, therefore $\min_{\tilde{\sigma}\in\Xi_{i+1}} J(\tilde{\sigma}) \leq \min_{\tilde{\sigma}\in\Xi_i} J(\tilde{\sigma})$. □

Algorithm 1 Rapidly-exploring Random Cycles (RRC)

$V \leftarrow x_0$; $E \leftarrow \emptyset$; $minCycleIndex \leftarrow \emptyset$; $minCycleCost \leftarrow \infty$;
for $i = 1$ **to** $numSteps$ **do**
  $x_{rand} \leftarrow$ SampleFree;
  $x_{nearest} \leftarrow$ Nearest$(G = (V, E), x_{rand})$;
  $x_{new} \leftarrow$ Steer$(x_{nearest}, x_{rand}, \eta)$;
  $X_{near} \leftarrow$ Near$(G = (V, E), x_{new}, r)$;
  if card$(X_{near}) == 1$ **then**
    if CollisionFree$(x_{nearest}, x_{new})$ **then**
      $V \leftarrow V \cup x_{new}$; $E \leftarrow E \cup (x_{nearest}, x_{new})$;
    end if
  else
    $V \leftarrow V \cup x_{new}$;
    ($cycleCost$, $cycleVertexIndex$, $cycleIndex$) $\leftarrow$ RRC extend(G, X_{near}, x_{new});
    $E \leftarrow E \cup (cycleVertexIndex, x_{new})$;
    if $cycleCost < minCycleCost$ **then**
      $minCycleCost \leftarrow cycleCost$;
      $minCycleIndex \leftarrow cycleIndex$;
    end if
  end if
end for
return $G = (V, E)$, $minCycleIndex$, $minCycleCost$;

C. RRC for Multiple Sensing Agents

We now extend the RRC algorithm to deal with multiple sensing agents. The main difference is that we maintain a tree in the joint state space of all the agents, so that each node in the tree represents a multi-agent state. Two of the primitive procedures must be modified for this case, the Steer procedure and the Near procedure. We modify the Steer procedure to account for the kinematic constraint for each agent, that is, the maximum distance one agent can move forward in $\mathbb{R}^2$ is bounded by η.

Steering: Denote two points in $\mathbb{R}^{2N_s}$ by $x = \left((x^1)^T, (x^2)^T, \ldots, (x^{N_s})^T\right)^T$ and $z = \left((z^1)^T, (z^2)^T, \ldots, (z^{N_s})^T\right)^T$, $x^i, z^i \in \mathbb{R}^2$, the function Steer : $(x, z, \eta) \rightarrow v$ returns a point v on the line connecting x and z such that $\|v - z\|$ is minimized while $\max_{i\in\{1,2,\ldots,N_s\}} \|x^i - v^i\| \leq \eta$. Specifically, we choose v as follows. Denote the unit vector along the direction

Algorithm 2 RRC extend

Input: $G = (V, E)$, X_{near} (card$(X_{near}) \geq 2$), and x_{new};
Output: $cycleCost$, $cycleVertexIndex$, $cycleIndex$;

$cycleCost \leftarrow \inf$;
$cycleVertexIndex \leftarrow \emptyset$;
$cycleIndex \leftarrow \emptyset$;
for any two $(x^1_{near}, x^2_{near}) \in X_{near}$ **do**
  if CollisionFree(x^1_{near}, x_{new}) **and** CollisionFree(x^2_{near}, x_{new}) **then**
    $\tilde{\sigma} \leftarrow$ FindCycle$(x^1_{near}, x^2_{near}, x_{new})$;
    $J(\tilde{\sigma}) \leftarrow$ PersistentCost$(\tilde{\sigma})$;
    if $J(\tilde{\sigma}) < cycleCost$ **then**
      $cycleVertexIndex \leftarrow x^2_{near}$ (or x^1_{near});
      $cycleCost \leftarrow J(\tilde{\sigma})$;
      $cycleIndex \leftarrow \tilde{\sigma}$;
    end if
  end if
end for

from x to z by $e = \frac{z-x}{\|z-x\|}$, denote the elements of e by $e = (e^1_x, e^1_y, \ldots, e^{N_s}_x, e^{N_s}_y)^T$, then

$$v = \begin{cases} z & \text{if } \|x - z\| \leq \tilde{\eta} \\ x + e \cdot \tilde{\eta} & \text{otherwise} \end{cases}$$

where $\tilde{\eta} = \eta / \max\limits_{i \in \{1,2,\ldots,N_s\}} \sqrt{(e^i_x)^2 + (e^i_y)^2}$. Here $\tilde{\eta}$ means the maximum distance between two vertices in $\mathbb{R}^{2N_s}$ while making sure that the maximum distance is bounded by η in $\mathbb{R}^2$.

Near Vertices: Similarly, we also modify the Near procedure by choosing the ball radius as $r(|V|) = \min\{\gamma(\log|V|/|V|)^{1/(2N_s)}, \tilde{\eta}\}$.

By these modifications, RRC algorithm can be extended to return periodic trajectories for multiple agents. Note that in this case the sampling takes place in the high dimensional joint state space, which belongs to $\mathbb{R}^{2N_s}$. Because of the rapidly-exploring feature of randomized algorithms, our algorithm can explore this high dimensional joint state space very quickly.

D. Rapidly-exploring Random Cycles Star (RRC)*

In the RRC algorithm above, we search for better simple cycles by adding one random vertex to the tree and comparing the cycles formed by this new vertex with the current lowest cost cycle maintained in the tree. This procedure tends to be myoptic in searching for lower cost cycles since it only considers the cycles that can be formed in the neighborhood of the new vertex. As a result, a connection which is the best in the current step may inhibit the search for better cycles in future iterations. RRT* also has this drawback, but it uses a procedure called *rewire* to eliminate this side affect, thus making RRT* asymptotically optimal. In order to make RRC more efficient in searching for better simple cycles, we analogously propose a *rewire* procedure to the original RRC algorithm. We call this new algorithm RRC*. The *rewire* procedure in RRC* is different from the *rewire* procedure in RRT*, because in RRT* the *rewire* procedure is to rewire the parent of all the vertices inside the near ball while in RRC* the *rewire* procedure is to rewire the parent of the vertices on the current lowest cost cycle. The reason why we can not rewire the parent for every vertex inside the near ball is that the cost in our problem is not additive. As a result, the *rewire* procedure in RRC* cannot eliminate the side effect of the myopic connection. However, it does provide an improvement for the current lowest cost cycle.

RRC* works similarly to RRC. First, we generate random samples in the free state space, then by using the Nearest, Steer and Near procedure, we obtain vertices X_{near} inside the near ball. If there is only one vertex inside the ball, we connect the new random vertex x_{new} to this vertex directly. If there is more than one verticex inside the ball, then we check whether there are at least two vertices that belong to the current lowest cost cycle, that is, check whether the cardinality of the set $X_{near} \cap minCycleIndex$ is larger than or equal to two. If it is not, then we perform the normal RRC extend procedure and try to connect the new random vertex to every two vertices inside the near ball, and compare the cost of the potential cycles. However, if the cardinality is larger than or equal to two, then we check whether we can perform the *rewire* procedure to find a lower cost cycle than the current lowest cost cycle $minCycleIndex$ in the tree. We call this procedure *rewireCheck* (see Algorithm 4).

The *rewireCheck* procedure works in the following way. Denote the waypoints in $minCycleIndex$ by $\{x_1, x_2, \ldots, x_T\}$, denote the intersection set between X_{near} and $minCycleIndex$ by $X_{near \cap minCycleIndex}$, as discussed above, the cardinality of $X_{near \cap minCycleIndex}$ is greater than or equal to two. For every two vertices x_i and $x_j (j > i)$ in $X_{near \cap minCycleIndex}$, try to connect x_{new} to x_i and x_j. If the cost of the new cycle $\{x_1, x_2, \ldots, x_i, x_{new}, x_j, x_{j+1}, \ldots, x_T\}$ is lower than the current $minCycleCost$, then update the $minCycleCost$ and set the $rewireFlag = 1$, otherwise, set $rewireFlag = 0$, which means rewiring does not help to improve the lowest cost cycle $minCycleIndex$. When $rewireFlag = 1$, it means we can use the rewire procedure to find a cycle whose cost is lower than the current $minCycleIndex$. However, we do not rewire the tree immediately since it is possible that the normal RRC extend procedure may find better cycles than the rewire procedure. Hence, if $rewireFlag = 1$, we still perform the normal RRC extend procedure. Note that if $rewireFlag = 1$, then after the rewireCheck procedure, $minCycleCost$ is updated. After performing the RRC extend procedure, we compare the cost of the cycle $cycleIndex$ with the new updated $minCycleCost$. If the normal RRC extend procedure returns lower cost cycles than the updated $minCycleCost$, then we connect x_{new} to the $cycleVertexIndex$ returned by the RRC extend procedure. Otherwise, we use the rewire procedure to connect x_{new} to one vertex in $X_{near \cap minCycleIndex}$.

The rewire procedure works as follows, see Algorithm 5. Denote the two vertices from rewireCheck by $rewireVertex = \{x_i, x_j\}$. First we use a procedure called isInSamePath to check whether x_i and x_j are in the same path to the root. If they are, then isInSamePath returns True, otherwise, it returns False. When x_i and x_j are in the same

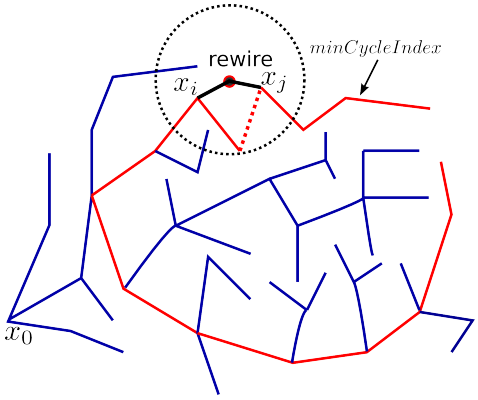


Fig. 5. Red curve: current lowest cost cycle obtained from the tree, solid black curve: new edges after the rewiring, dotted red curve: deleted edges in rewiring.

path to the root, next we find out which one is the ancestor and which one is the descendant. Then we make x_{new} as the new parent of the descendant and delete the edge between the descendant and its old parent. We also make x_{new} as the child of the ancestor. On the other hand, if x_i and x_j are not in the same path, then we make x_{new} as the parent of either x_i or x_j. In this way, we rewire the tree. This procedure is illustrated graphically in Figure 5. The pseudo-code for RRC* is given in Algorithm 3. Similar to RRC, RRC* can be extended to plan trajectories for multiple sensing agents by considering the joint state space of these agents.

V. Efficient Computation of Cycle Cost

In the primitive procedure PersistentCost, we need to calculate the infinite horizon cost of a cycle. Let's consider a cycle $\tilde{\sigma}$ with waypoints $(x_1, x_2, \ldots, x_T)$ and period T, when taking measurements along this cycle, we will get a discrete-time stochastic periodic system with period T.

$$a_{t+1} = Aa_t + w_t, w_t \sim N(0, Q) \tag{10a}$$

$$y_t = C(x_t)a_t + v_t, v_t \sim N(0, R) \tag{10b}$$

where $C(x_t) = C(x_{t+T})$. Note that when having multiple sensors taking measurements, y_t is the composite measurements from all the sensors. Also, by doing Kalman filtering along this cycle, we will get a discrete-time periodic Riccati equation (DPRE) with period T.

$$\Sigma_\infty^{x_{i+1}} = A\Sigma_\infty^{x_i}A^T - A\Sigma_\infty^{x_i}C(x_i)^T \times \left(C(x_i)\Sigma_\infty^{x_i}C(x_i)^T + R\right)^{-1} C(x_i)\Sigma_\infty^{x_i}A^T + Q \tag{11}$$

In [16], the authors prove that there exists a unique symmetric periodic positive semidefinite (SPPS) solution to this DPRE if and only if the periodic system (10) is stabilizable and detectable. In the case that system (10) is not detectable, there is no SPPS solution to the DPRE and the covariance matrix will become unbounded, which means the cost of the corresponding cycle is infinity. Therefore, we do not need to calculate the cost of these undetectable cycles. Next we discuss how to calculate the cost of detectable cycles, that is, how to solve for the DPRE equation.

Algorithm 3 Rapidly-exploring Random Cycles Star (RRC*)

$V \leftarrow x_0$; $E \leftarrow \emptyset$; $minCycleIndex \leftarrow \emptyset$; $minCycleCost \leftarrow \infty$;
for $i = 1$ **to** $numSteps$ **do**
  $x_{rand} \leftarrow$ SampleFree;
  $x_{nearest} \leftarrow$ Nearest$(G = (V, E), x_{rand})$;
  $x_{new} \leftarrow$ Steer$(x_{nearest}, x_{rand}, \eta)$;
  $X_{near} \leftarrow$ Near$(G = (V, E), x_{new}, r)$;
  if card$(X_{near}) == 1$ **then**
    if CollisionFree$(x_{nearest}, x_{new})$ **then**
      $V \leftarrow V \cup x_{new}$; $E \leftarrow E \cup (x_{nearest}, x_{new})$;
    end if
  else
    $V \leftarrow V \cup x_{new}$;
    if card$(X_{near \cap minCycleIndex}) < 2$ **then**
      ($cycleCost$, $cycleVertexIndex$, $cycleIndex$) $\leftarrow$ RRC extend(G, X_{near}, x_{new});
      $E \leftarrow E \cup (cycleVertexIndex, x_{new})$;
      if $cycleCost < minCycleCost$ **then**
        $minCycleCost \leftarrow cycleCost$;
        $minCycleIndex \leftarrow cycleIndex$;
      end if
    else
      $(rewireFlag, rewireVertex, minCycleCost) \leftarrow$ rewireCheck;
      if $rewireFlag == 0$ **then**
        ($cycleCost$, $cycleVertexIndex$, $cycleIndex$) $\leftarrow$ RRC extend(G, X_{near}, x_{new});
        $E \leftarrow E \cup (cycleVertexIndex, x_{new})$;
        if $cycleCost < minCycleCost$ **then**
          $minCycleCost \leftarrow cycleCost$;
          $minCycleIndex \leftarrow cycleIndex$;
        end if
      else
        ($cycleCost$, $cycleVertexIndex$, $cycleIndex$) $\leftarrow$ RRC extend(G, X_{near}, x_{new});
        if $cycleCost < minCycleCost$ **then**
          $minCycleCost \leftarrow cycleCost$;
          $minCycleIndex \leftarrow cycleIndex$;
          $E \leftarrow E \cup (cycleVertexIndex, x_{new})$;
        else
          $(G(V, E), minCycleIndex) \leftarrow$ rewire;
        end if
      end if
    end if
  end if
end for
return $G = (V, E)$, $minCycleIndex$, $minCycleCost$;

Algorithm 4 rewireCheck

Input: $G = (V, E)$, x_{new}, $X_{near \cap minCycleIndex}$ ($\text{card}(X_{near \cap minCycleIndex}) \geq 2$), $minCycleCost$;
Output: $rewireFlag$, $rewireVertex$, $minCycleCost$;

$rewireFlag \leftarrow 0$;
$rewireVertex \leftarrow \emptyset$;
for any two $(x_i, x_j) \in X_{near \cap minCycleIndex}$ **do**
  if CollisionFree(x_i, x_{new}) **and** CollisionFree(x_j, x_{new}) **then**
    $\tilde{\sigma} \leftarrow \{x_1, x_2, \ldots, x_i, x_{new}, x_j, x_{j+1}, \ldots, x_T\}$;
    $J(\tilde{\sigma}) \leftarrow$ PersistentCost$(\tilde{\sigma})$;
    if $J(\tilde{\sigma}) < minCycleCost$ **then**
      $rewireVertex \leftarrow \{x_i, x_j\}$;
      $minCycleCost \leftarrow J(\tilde{\sigma})$;
      $rewireFlag \leftarrow 1$;
    end if
  end if
end for

Algorithm 5 rewire

Input: $G = (V, E)$, $rewireVertex = \{x_i, x_j\}$, x_{new};
Output: $G = (V, E)$, $minCycleIndex$;

if isInSamePath(G, x_i, x_j) **then**
  For $\{x_i, x_j\}$, find which one is ancestor, which one is descendant;
  if x_i is ancestor of x_j **then**
    // Rewire to make x_{new} as the parent of x_j and as the child of x_i.
    $x_{parent} \leftarrow$ Parent(x_j);
    $E \leftarrow (E \setminus \{(x_{parent}, x_j)\}) \cup \{(x_{new}, x_j)\}$;
    $E \leftarrow E \cup \{(x_i, x_{new})\}$;
  else
    $x_{parent} \leftarrow$ Parent(x_i);
    $E \leftarrow (E \setminus \{(x_{parent}, x_i)\}) \cup \{(x_{new}, x_i)\}$;
    $E \leftarrow E \cup \{(x_j, x_{new})\}$;
  end if
else
  // Choose any one in $\{x_i, x_j\}$ as the ancestor. Here we choose x_i.
  $x_{parent} \leftarrow$ Parent(x_j);
  $E \leftarrow (E \setminus \{(x_{parent}, x_j)\}) \cup \{(x_{new}, x_j)\}$;
  $E \leftarrow E \cup \{(x_i, x_{new})\}$;
end if

By the Sherman-Morrison-Woodbury formula, when $(I + G_i \Sigma_\infty^{x_i})^{-1}$ exists, the DPRE can be rewritten as:

$$\Sigma_\infty^{x_{i+1}} = A\Sigma_\infty^{x_i} (I + G_i \Sigma_\infty^{x_i})^{-1} A^T + Q \tag{12}$$

where $G_i = C(x_i)^T R^{-1} C(x_i) \succeq 0$. To solve for this DPRE, one naive way is to start with an initial covariance matrix and calculate an approximation by iterating (12) until convergence is reached. This approach is conceptually simple, but computationally highly inefficient. In [5], [17], [18], [19], the authors propose efficient methods to solve the DPRE. In this work, we will use the structure-preserving algorithm introduced in [5] to calculate the SPPS solution. To the best of our knowledge, this method is the most efficient approach. It also converges exponentially fast to the SPPS solution.

A. Structure-preserving Algorithms for DPRE

Structure-preserving algorithms consist of the structure-preserving swap and collapse algorithm (SSCA) and the structure-preserving doubling algorithm (SDA). Basically, this algorithm uses the SSCA algorithm to get an equivalent Riccati equation for the periodic Riccati equation, and then it uses the SDA algorithm to solve this equivalent Riccati equation and get $\Sigma_\infty^{x_T}$, from which the other $\Sigma_\infty^{x_i}, (i = T-1, \ldots, 1)$ can be found through equation (11).

Here we restate the SSCA and SDA algorithm in Algorithm 6 and Algorithm 7, respectively. More details can be found in [5]. The output of SSCA $\hat{A}_T, \hat{G}_T, \hat{Q}_T$ is the A, G, Q matrix for the equivalent Riccati equation. We use them as the input of the following SDA algorithm. Hence, by (9), the cost of the cycle $\tilde{\sigma}$ is given by

$$J(\tilde{\sigma}) = \rho(\Sigma_\infty^{\tilde{\sigma}}) = \max_{i \in \{1,2,\ldots,T\}} \rho(\Sigma_\infty^{x_i})$$

where $\Sigma_\infty^{x_i}$ is calculated by the SSCA+SDA algorithm. As discussed in [5], the solution from the SDA algorithm converges to $\Sigma_\infty^{x_T}$ exponentially fast.

Algorithm 6 Structure-preserving Swap and Collapse Algorithm (SSCA)

Input: A, G_i, Q, $i = 1, 2, \ldots, T$;
Output: $\hat{A}_T$, $\hat{G}_T$, $\hat{Q}_T$;

Initialize $\hat{A}_1 \leftarrow A$, $\hat{G}_1 \leftarrow G_1$, $\hat{Q}_1 \leftarrow Q$;
for $i = 2, 3, \ldots, T$ **do**
  Compute $W \leftarrow I + Q\hat{G}_{i-1}$;
  Solve $WV_1 = A^T$, $WV_2 = Q$ for V_1, V_2;
  $\hat{A}_i \leftarrow V_1^T \hat{A}_{i-1}, \hat{G}_i \leftarrow G_i + A\hat{G}_{i-1}V_1$;
  $\hat{Q}_i \leftarrow \hat{Q}_{i-1} + \hat{A}_{i-1}^T V_2 \hat{A}_{i-1}$;
end for

Algorithm 7 Structure-preserving Doubling Algorithm (SDA)

Input: A, G, Q, τ(a small error tolerance);
Output: Positive semidefinite solution $\Sigma_\infty^{x_T}$ to the equivalent Riccati equation;

Initialize $i \leftarrow 0$, $A_0 \leftarrow A$, $G_0 \leftarrow G$, $Q_0 \leftarrow Q$;
repeat
  $W \leftarrow I + G_i Q_i$;
  Solve for V_1, V_2 from $WV_1 = A_i, V_2 W^T = G_i$;
  $A_{i+1} \leftarrow A_i V_1, G_{i+1} \leftarrow G_i + A_i V_2 A_i^T$;
  $Q_{i+1} \leftarrow Q_i + V_1^T Q_i A_i$;
until $\|Q_{i+1} - Q_i\|_F \leq \tau \|Q_{i+1}\|_F$;
Set $\Sigma_\infty^{x_T} \leftarrow Q_{i+1}$.

B. Comparing the Cost of Different Cycles

From our simulation experience, the time consumed to run RRC or RRC* to estimate a spatio-temporal field hinges on the number of cycle cost comparisons. If the radius of the near ball is large, then there will be more vertices inside the

ball and more cycle comparisons are needed. To compare the costs of two cycles, a straightforward way is to calculate their costs respectively and compare the costs directly. Although the SSCA+SDA algorithm is very efficient, it still takes considerable time to compute these costs if the number of cycle comparisons is large. Here we propose a procedure to compare the costs of two cycles on the condition that we know the cost of one cycle. That is, by this procedure, if we know the cost of one cycle, we can compare the costs of the two cycles without calculating the cost of the other cycle. This procedure can help reduce the number of times of using SSCA and SDA algorithm to calculate the cycle cost.

As in Corollary 2, let the composite Riccati map corresponding to one complete cycle of a periodic trajectory $\tilde{\sigma}$ that starts and ends at x_i be denoted by $G^{\tilde{\sigma}}_{x_i}$, and recall that $J(\tilde{\sigma}) = \max_{i\in\{1,2,\ldots,T\}} \rho(\Sigma^{x_i}_\infty)$. We have the following theorem to compare the costs of two cycles.

Theorem 2 (Cycle Cost Comparison)**.** *Consider two periodic paths, $\tilde{\sigma}_x$ with period T_x and points $(x_1, x_2, \ldots, x_{T_x})$, and $\tilde{\sigma}_y$ with period T_y and points $(y_1, y_2, \ldots, y_{T_y})$, and let $k = argmax_{i=1,2,\ldots,T_x}\rho(\Sigma^{x_i}_\infty)$. We have that*

1) *if there exists one $G^{\tilde{\sigma}_y}_{y_i}(\Sigma^{x_k}_\infty) \succeq \Sigma^{x_k}_\infty$, then $J(\tilde{\sigma}_y) \geq J(\tilde{\sigma}_x)$,*
2) *if $G^{\tilde{\sigma}_y}_{y_i}(\Sigma^{x_k}_\infty) \preceq \Sigma^{x_k}_\infty$ for all $i = 1, 2, \ldots, T_y$, then $J(\tilde{\sigma}_y) \leq J(\tilde{\sigma}_x)$,*
3) *if there exists at least one $(G^{\tilde{\sigma}_y}_{y_j}(\Sigma^{x_k}_\infty) - \Sigma^{x_k}_\infty)$ remaining indefinite and there is no $G^{\tilde{\sigma}_y}_{y_i}(\Sigma^{x_k}_\infty) \succeq \Sigma^{x_k}_\infty$ for $i = 1, 2, \ldots, T_y$, then we cannot compare $J(\tilde{\sigma}_y)$ and $J(\tilde{\sigma}_x)$.*

Proof. Here we prove part 1. The proof for part 2 follows in a similar method, and by proving part 1 and part 2, we have proved part 3. Without loss of generality, let us consider point y_1 in the path $\tilde{\sigma}_y$. By part 3 of Corollary 2, $J(\tilde{\sigma_y})$ is independent of where we start. So we choose $\Sigma^{y^1}_0 = \Sigma^{x^k}_\infty$ as the initial covariance matrix for the Riccati recursion along $\tilde{\sigma}_y$. We denote the trajectory of the covariance matrix at y_1 by $\Sigma^{y_1}_0, \Sigma^{y_1}_{T_y}, \Sigma^{y_1}_{2T_y}, \Sigma^{y_1}_{3T_y}, \ldots, \Sigma^{y_1}_\infty$, so

$$\begin{aligned}
\text{if } \Sigma^{y_1}_{T_y} &= G^{\tilde{\sigma}_y}_{y_1}(\Sigma^{x_k}_\infty) \preceq \Sigma^{x_k}_\infty \\
\text{then } \Sigma^{y_1}_{2T_y} &= G^{\tilde{\sigma}_y}_{y_1}(\Sigma^{y_1}_{T_y}) \preceq G^{\tilde{\sigma}_y}_{y^1}(\Sigma^{x_k}_\infty) = \Sigma^{y_1}_{T_y} \preceq \Sigma^{x_k}_\infty \\
&\vdots \\
\Sigma^{y_1}_\infty &\preceq \Sigma^{x_k}_\infty \\
\Rightarrow \rho(\Sigma^{y_1}_\infty) &\leq \rho(\Sigma^{x_k}_\infty),
\end{aligned}$$

where the inequalities are an application of the monotonicity of the composite Riccati map from Corollary 1. Similarly, we can prove that $\rho(\Sigma^{y_i}_\infty) \leq \rho(\Sigma^{x_k}_\infty), i = 2, \ldots, T_y$. So we have

$$J(\tilde{\sigma}_y) = \max_{i=1,2,\ldots,T_y} \rho(\Sigma^{y_i}_\infty) \leq \rho(\Sigma^{x_k}_\infty) = J(\tilde{\sigma}_x)$$

□

By this theorem, when we compare the costs of two cycles, we first calculate an infinite horizon cost of the first cycle using the structure-preserving algorithm. Then starting with the asymptotic covariance matrix which has the largest spectral radius along the first cycle we calculate the cost of moving along the second cycle once, i.e., $\rho(\Sigma^{y^i}_{T_y}), i = 1, 2, \ldots, T_y$. If any of these costs is larger than the infinite horizon cost of the first cycle, then we can conclude that the infinite horizon cost of the second cycle is larger than the first, $J(\tilde{\sigma}_y) \geq J(\tilde{\sigma}_x)$. On the other hand, if all $\rho(\Sigma^{y_i}_{T_y})$ are smaller than the infinite horizon cost of the first cycle, then we can still conclude that the infinite horizon cost of the second cycle is smaller than the infinite horizon cost of the first cycle, $J(\tilde{\sigma}_y) \leq J(\tilde{\sigma}_x)$. Hence, by this theorem, we can avoid calculating the costs of some cycles, thus decreasing the computation burden of the algorithm.

In the above theorem, we still need to calculate the cost of one cycle. Next we prove another theorem which uses the information matrix to compare the costs of two cycles without calculating the cost of any cycle.

Theorem 3. *Consider two cycles $\tilde{\sigma}_x$ and $\tilde{\sigma}_y$ with the same period T, assume they have $N_d (0 \leq N_d \leq T)$ different waypoints and the other $T - N_d$ waypoints are the same, denote these different waypoints in $\tilde{\sigma}_x$ and $\tilde{\sigma}_y$ by $\{x_1, x_2, \ldots, x_{N_d}\}$ and $\{y_1, y_2, \ldots, y_{N_d}\}$, respectively. If for any $i \in \{1, 2, \ldots, N_d\}$, we have*

$$C(x_i)^T R^{-1} C(x_i) \succeq C(y_i)^T R^{-1} C(y_i)$$

then $J(\tilde{\sigma}_x) \leq J(\tilde{\sigma}_y)$.

Proof. By using Sherman-Morrison-Woodbury formula, the Riccati equation in (4) can be rewritten as follows.

$$\Sigma_{t+1} = A(\Sigma_t^{-1} + C(x_t)^T R^{-1} C(x_t))^{-1} A^T + Q$$

Without loss of generality, we assume that we start calculating the asymptotic covariance with the initial covariance matrix equal to Σ_0 for both cycles. Also, we assume that we start at x_i and y_i for $\tilde{\sigma}_x$ and $\tilde{\sigma}_y$, respectively. Hence, if $C(x_i)^T R^{-1} C(x_i) \succeq C(y_i)^T R^{-1} C(y_i)$, by the fact that $C(x_t)^T R^{-1} C(x_t)$ is positive semi-definite, we have

$$\begin{aligned}
&(\Sigma_0^{-1} + C(x_i)^T R^{-1} C(x_i))^{-1} \\
&\preceq (\Sigma_0^{-1} + C(y_i)^T R^{-1} C(y_i))^{-1} \\
&\Rightarrow A(\Sigma_0^{-1} + C(x_i)^T R^{-1} C(x_i))^{-1} A^T + Q \\
&\preceq A(\Sigma_0^{-1} + C(y_i)^T R^{-1} C(y_i))^{-1} A^T + Q \\
&\Rightarrow \Sigma_1^{\tilde{\sigma}_x} \preceq \Sigma_1^{\tilde{\sigma}_y}
\end{aligned}$$

Similarly, we have $\Sigma_t^{\tilde{\sigma}_x} \preceq \Sigma_t^{\tilde{\sigma}_y}, t \in [1, +\infty)$. Therefore, $J(\tilde{\sigma}_x) \leq J(\tilde{\sigma}_y)$.

□

Remark 5. *In the above theorem, $C(x_i)^T R^{-1} C(x_i)$ is called the* information matrix. *Hence, if we can get more information by moving along one cycle, the estimation uncertainty along that cycle will be smaller. Here we want to point out that in RRC and RRC* when doing cycle comparisons, there are many cases such that two cycles have the same period, and with a large portion of their waypoints the same, like the potential cycle 2 and potential cycle 3 in Figure 4. In theses cases,*

as long as we can compare the information matrix, we can compare the costs of the two cycles directly based on the comparison result of the information matrix.

Remark 6. *In Theorem 3, we only discuss the case that two cycles have the same period. For cycles with different periods, as long as the information matrix at every waypoint of one cycle is larger than the information matrix at every waypoint of the second cycle, Theorem 3 still holds. However, from our simulation experience, this condition is rarely satisfied.*

C. Reduce the Number of Cycle Comparisons

In RRC and RRC*, the number of cycle comparisons depends on how many vertices are inside the near ball. When we design our algorithms, we inherit the Near procedure from RRT*. Hence, the radius of the near ball will shrink and will go to zero as the number of samples goes to infinity. The shrinking of the radius prevents the number of vertices inside the near ball from going to infinity. It also leads to the distance between two vertices in the tree becoming smaller and smaller as the number of samples grows. While this has no affect on the path returned by RRT*, it does influence the cycle returned by RRC and RRC*, especially when we use RRC and RRC* to plan periodic trajectories for estimation. When the distance between two vertices becomes very small, it reduces the probability that we get lower cost cycles for estimation since taking measurements at two very close positions does not give new information for estimation. This is why RRC and RRC* do not frequently update the lowest cost cycles they maintain when the number of samples reaches a very large number, see Figure 10. In order to avoid this side effect, as well as reducing the number of cycle comparisons, we propose a procedure called *random k-near neighbors*. This is different from the *k-nearest neighbors* RRT*.

Random k-near Neighbors RRC and RRC*: The basic idea is that we do not shrink the radius of the near ball. In order to prevent the number of cycle comparisons from going to infinity, we randomly choose k vertices inside the near ball. Since the radius of the ball is fixed and we choose the k vertices randomly, the distance between two vertices will not become very small even though the number of samples goes to infinity. Also, since k is fixed, the number of cycle comparisons is bounded.

D. Computational Complexity Analysis

In this section we analyze the computational complexity of RRC and RRC*. First we analyze the computational complexity of some primitive procedures. Then we analyze the computational complexity of the whole algorithm.

As discussed in [3], the CollisionFree procedure can be executed in $O(\log^d M)$ time, M is the number of obstacles in the environment and d is the dimension of the state space. As for the procedure Nearest, it has time complexity $O(\log|V|)$, where $|V|$ is the number of vertices. In addition, $O(\log|V|)$ time is spent on the procedure Near. In the procedure FindCycle, it takes $O(h)$ time to find the lowest common ancestor (LCA) of two nodes in the tree, where h is the maximum height of the tree. Assume we have k vertices in X_{near}, then we need to find out the $\frac{k(k-1)}{2}$ cycles. Hence the time complexity of this procedure is $O(\frac{k(k-1)}{2}h)$. Because k is bounded as a result of the shrinking radius of the ball, we can write the complexity of this procedure as $O(h)$.

Hence, in RRC at each iteration we have $O(\log^d M)$ time spent on collision checking, $O(\log|V|)$ time spent on finding nearest vertex, $O(\log|V|)$ time spent on finding near vertices, $O(h)$ time spent on finding cycles. So with N iterations, the time complexity of RRC is $O(N(\log^d M + \log|V| + \log|V| + h))$. Since d and M are fixed, and $|V| = N$, the worst case time complexity of RRC is $O(N \log N + Nh)$.

As for the time complexity of RRC*, it has one more procedure rewire. In this procedure, we need to check whether two nodes are in the same path to root, that is, the isInSamePath procedure in the rewire procedure. It takes $O(h)$ time to do this checking. Hence, the time complexity for RRC* is $O(N(\log^d M + \log|V| + \log|V| + h + h))$, which is still $O(N \log N + Nh)$.

VI. Numerical Simulations

This section presents numerical simulations of our algorithms using data from National Oceanic and Atmospheric Administration (NOAA). We apply our algorithms to plan periodic trajectories for sensing robots to estimate the sea water temperature on the surface of the Caribbean Sea, see Figure 1. We choose a rectangular region with length equal to 1200 units and width equal to 800 units. One unit length is equal to 2.304 miles in geography. Then we compare the performance of our algorithms with several benchmark algorithms.

Before we use our planning algorithms to plan trajectories to estimate the temperature, we need to have a dynamical model of the field. That is, we need to learn the A and Q matrix in (2). We apply subspace identification algorithms [6], [7] to learn these dynamics based on observational and real-time data from the National Oceanographic Data Center (NODC) [38]. The data set we use is included in the attachment of this submission. It contains the water temperature taken at different buoy stations in the Caribbean Sea in the year of 2014. The time interval between two consecutive measurements is one hour, however we find that the temperature change is minimal over this time window. Hence we subsample the data to one data point every six hours for the sake of learning the model. Using this subsampled data, we apply a subspace identification algorithm to learn the A, Q and R matrix. We refer the readers to [6], [7] for more details.

We also need to know how the water temperature changes in space. That is, we need to know the basis functions. Here we use Gaussian radial basis functions, so the j-th element in the $C(x_t^i)$ matrix is $c_j^i = Ke^{-\|x_t^i - q_j\|^2/2\sigma_c^2}$, where K is a scale factor, x_t^i is the i-th sensor's position at time t, q_j is the position of the j-th basis function and σ_c is the variance of the Gaussian. To best fit the spatial variation observed in the NODC temperature data, we choose K as 10 and σ_c as 100 units. Next we choose the Gaussian radial basis centers by using K-means algorithm to cluster the ocean area based on

the Euclidean distance. The number of clusters is set to be 9. That is, we assume that there are 9 basis functions representing this region, $n = 9$. All the information about the learned A, Q, R and the basis centers is in the attachment of this submission.

We use this model derived from the NODC data to plan trajectories to estimate the sea water temperature on the surface of the Caribbean Sea. We choose the region of interest as a rectangular area with the southwest corner located at $(-98.00deg, 5.00deg)$ and the northeast corner located at $(-58.04deg, 31.63deg)$, where (lon, lat) are the geographic coordinates, see Figure 1. We also know the position of the obstacles, which in this case are islands and continental land. We get the position of these obstacles from the NOAA Operational Model Archive and Distribution System (OceanNOMADS) in [39].

First, we use RRC and RRC* to plan trajectories for a single sensing agent to monitor this region. We initialize the RRC and RRC* algorithm with $x_0 = (500units, 200units)$. For the primitive procedures, we choose $\eta = 50$ units for the Steer function. The r in the function Near is chosen as $r = \min\{\gamma(\log|V|/|V|)^{1/2}, \eta\}$, where $\gamma = (2(1 + 1/2)^{1/2}\mu(X)/\zeta_2)^{1/2}$, $\mu(X)$ is the area of the rectangular region and $\zeta_2 = \pi$. For the PersistentCost function, we choose the error tolerance $\tau = 10^{-4}$ for the structure-preserving algorithm. Figure 6 and Figure 7 show the resulting single agent trajectory from the RRC algorithm and RRC* algorithm, respectively. From these two figures, we can see that the planned trajectory covers most of the region. This can be explained by the requirement for the sensing agent to take measurements in the whole environment to minimize the estimation uncertainties. Local measurements in a small region of the environment will only help decrease the estimation uncertainties in that small area, which will lead to the estimation in other areas becoming inaccurate. Meanwhile, the sensing agent does not have to spend lots of time exploiting one area since the spatial field is correlated. Also, spending more time in one area means the time interval between two consecutive measurements in other areas will be larger, which will result in the estimation uncertainties growing larger in that time span.

Next we apply RRC and RRC* algorithm to plan periodic trajectories for multiple sensing agents. The parameters used in the simulation are the same with that of RRC and RRC* except that r is chosen as $r = \min\{\gamma(\log|V|/|V|)^{1/(2N_s)}, \eta\}$, where $\gamma = (2(1+1/(2N_s))^{1/(2N_s)}\mu(X)/\zeta_d)^{1/(2N_s)}$, $\mu(X)$ is the volume of the $2N_s$-dimensional hypercube, which is equal to $(800 \times 1200)^{N_s}$, ζ_d is the volume of the unit ball in $\mathbb{R}^{2N_s}$, which is equal to $\frac{\pi^{N_s}}{N_s!}$. Figure 8 shows the result of applying the RRC algorithm to plan trajectories for 3 sensing agents. In this simulation, the sampling takes place in $\mathbb{R}^6$ space and we get the trajectory for each sensing agent by projecting the high dimensional trajectory in $\mathbb{R}^6$ space back to the corresponding $\mathbb{R}^2$ space. Because of this, the trajectories for the 3 sensing agents have the same period. This figure also shows that each agent covers a subregion of the area of interest. They take measurements at the same time and a centralized Kalman filter (assumed to be running on a central base station) fuses all the measurements to update the estimate for the temperature field, thus minimizing the estimation uncertainties. Note that the 3

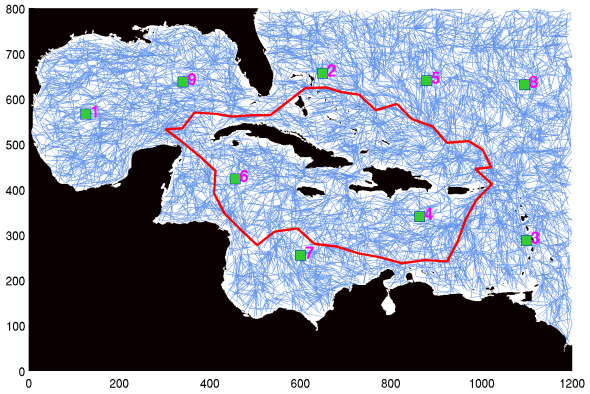


Fig. 6. Red curve: planned trajectory by RRC algorithm. Blue curve: rapidly-exploring random tree with 10000 iterations. Green square: basis centers.

sensing agents do not visit the subregion close to the basis center 6 and 7. The reason is that the noise of the field values in that subregion is small compared to other subregions. This can be verified by checking the sixth and seventh diagonal entry of the learned noise covariance matrix Q (see learned Q in the attachment of this submission).

Figure 9 shows the simulation of applying RRC* algorithm to plan trajectories for 3 sensing agents. In this simulation, the sampling also takes place in the $\mathbb{R}^6$ space and we get the trajectories for each agent by projecting the high-dimensional trajectory back to the $\mathbb{R}^2$ space. Comparing the trajectories planned by RRC and RRC*, we can see that the trajectory planned by RRC* tends to be smoother than RRC, especially when planning trajectories for a single agent. This difference is not that obvious for multiple sensing agents since in the multiple agents case, the projection of the high-dimensional trajectory back to the $\mathbb{R}^2$ space will result in sharp trajectories in $\mathbb{R}^2$. The trajectory difference between RRC and RRC* can be explained by the existence of the *rewire* procedure in RRC*. The *rewire* procedure will rewire the sharp turns to get smoother trajectories since this will incur lower cost for estimation. In estimation, sharp turns will collect repeated information while smooth trajectories tend to get more new information, which will help decrease the estimation uncertainties, thus lowering the cost.

Figure 10 shows how the cycle cost found by RRC and RRC* change with the number of iterations. From this figure, the cost of the cycle returned by RRC and RRC* decreases with the number of iterations, which verifies the monotonicity property of these two algorithms. Also, the cycle returned by RRC* has lower cost than the cycle returned by RRC and the cost of using multiple sensing agents is much lower than the cost of using one single agent. The reason is that using multiple robots to take measurements can gather more information, thus getting lower estimation uncertainties.

Next we compare our algorithms with several benchmark algorithms: random search, a greedy algorithm, and a receding horizon algorithm. For these algorithms, we make their maximum distance between two consecutive steps equal to the segment length in RRC and RRC*, that is, $\eta = 50$ units. For the greedy algorithm and the receding horizon

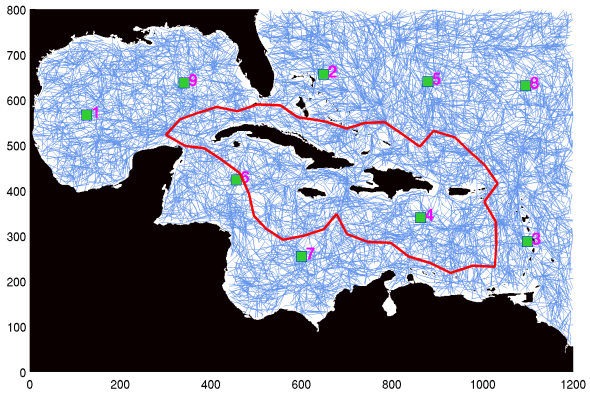


Fig. 7. Red curve: planned trajectory by RRC* algorithm. Blue curve: rapidly-exploring random tree with 10000 iterations. Green square: basis centers.

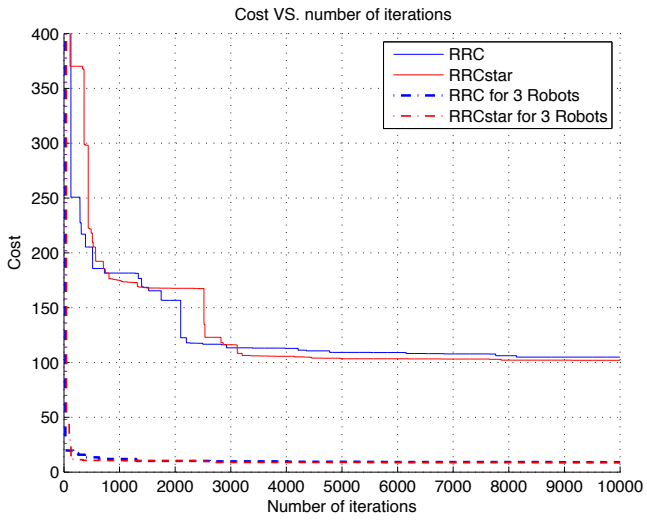


Fig. 10. The cost of the cycle found by RRC and RRC* decreases monotonically with the number of iterations.

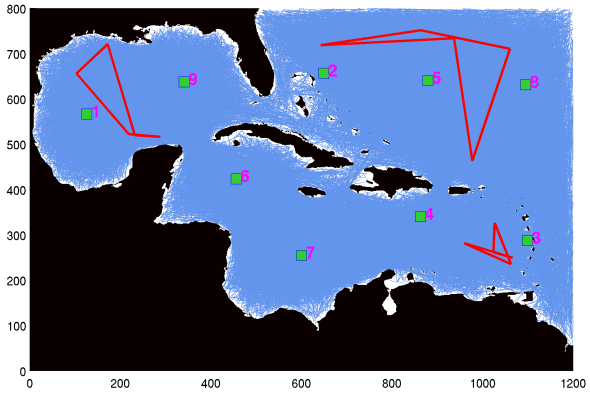


Fig. 8. Red curve: planned trajectories by RRC for 3 sensing agents. Blue curve: rapidly-exploring random tree with 10000 iterations in $\mathbb{R}^6$ space. Green square: basis centers.

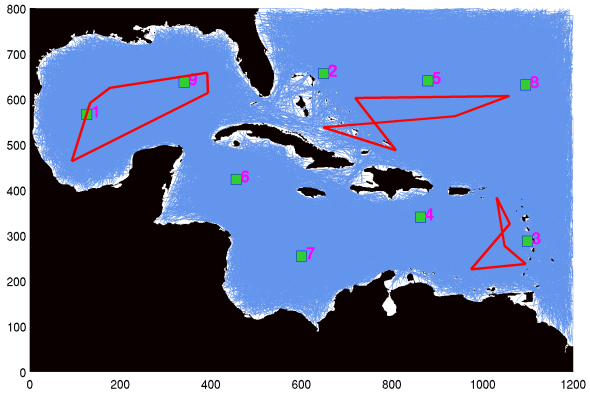


Fig. 9. Red curve: planned trajectories by RRC* for 3 sensing agents. Blue curve: rapidly-exploring random tree with 10000 iterations in $\mathbb{R}^6$ space. Green square: basis centers.

algorithm, we assume that the action list for the robot to take at each step is given by $\{e_1, e_2, \ldots, e_8\}$, where $e_i = (cos(\frac{2(i-1)\pi}{8}), sin(\frac{2(i-1)\pi}{8}))^T$, that is, at one position, the robot can move in the eight directions separated by $\frac{\pi}{4}$ radians with a maximum distance bounded by 50 units. The algorithm chooses the action which can minimize the estimation uncertainty by calculating the largest eigenvalue of the updated covariance matrix. Also, the resulting path under that action needs to be collision-free. If the paths under all actions are in collision with obstacles, we call it *deadlocked*. In these situations, the robot can *escape* by choosing the next step randomly and finding one direction which is collision-free. For the receding horizon algorithm, we let the sensing agent have the same action list as with the greedy algorithm. Instead of taking the next best action, the receding horizon algorithm looks several steps ahead and choose the action which can minimize the estimation uncertainties after these steps. The number of steps the algorithm looks ahead is called the *horizon length*. We set the horizon length to be 4 in this study. Note that the receding horizon algorithm is very computationally expensive since at each step we will have $N_a^{N_h}$ actions to compare to decide which action is the best to minimize estimation uncertainties, where N_a is the cardinality of the action list and N_h is the horizon length.

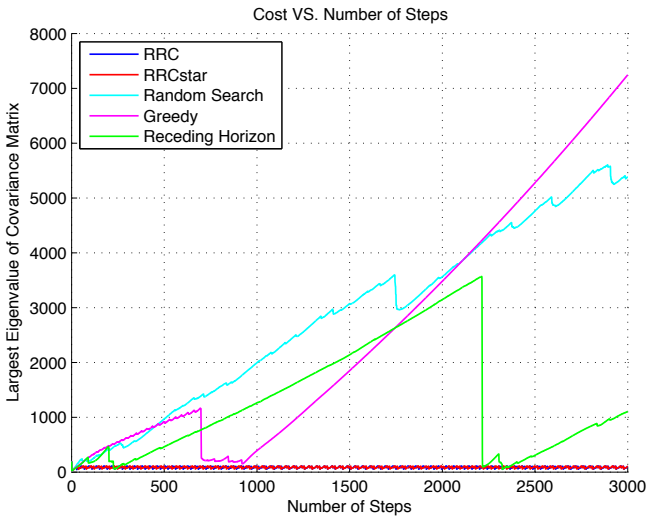
The trajectories produced by these three algorithms are shown in Figure 11. We do Kalman filtering along the trajectories generated by each algorithm and update the covariance matrix at each step. We show how the largest eigenvalue of the covariance matrix changes when the robot moves along the trajectories and takes measurements in Figure 12. Since we are considering the infinite horizon performance, we choose a relatively long simulation time horizon and make the number of time steps 3000. The initial covariance matrix is an arbitrary positive semi-definite matrix. Note that in Corollary 2 it is proved that the asymptotic cost is independent of Σ_0. We also let the robot start at different locations. From this figure, we

can see that the RRC and RRC* trajectories have a bounded and small cost while the cost along the random search, greedy and receding horizon trajectories grow bigger and bigger. The greedy algorithm and the receding horizon algorithm both tend to get stuck in local minima leading to poor global estimation performance. The figure shows that both of our sampling-based algorithms significantly outperform these other methods in this simulation study. Our infinite horizon objective prevents RRC and RRC* from getting caught in local minima, providing a globally accurate estimate of the environmental field over time.

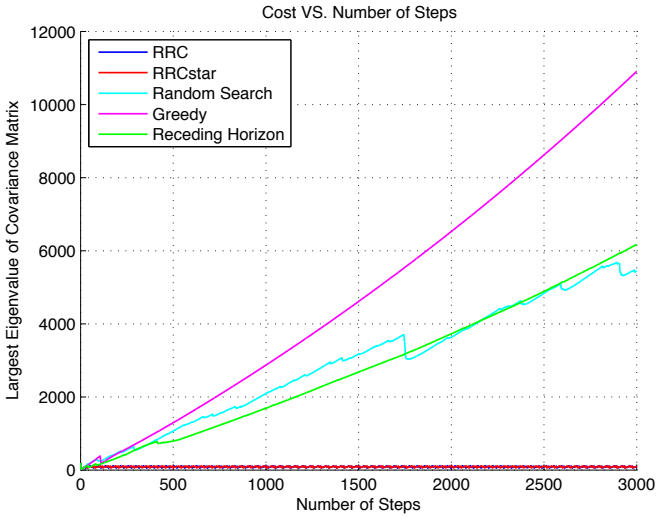
The animation of how the field estimation uncertainty changes when the sensing robot moves along different trajectories is given in the multimedia materials. One screen shot of the animation is shown in Figure 13. The colormap denotes the estimation uncertainties with red hot standing for high variance and dark blue standing for low variance. The estimation uncertainty of the field at one point in the environment is denoted by its variance $\mathsf{Var}(q)$. By equation (1), we have $\mathsf{Var}(q) = E\left[(C(q)(a_t - \hat{a}_t)) \cdot (C(q)(a_t - \hat{a}_t))^T\right] = \mathrm{tr}(\Sigma_t C(q)^T C(q))$. In the animation, the environment is discretized into a 800×1200 grid. We calculate the variance of all the points in the grid based on the current covariance matrix Σ_t and the spatial basis function $C(q)$. From this figure we can see that the worst estimation uncertainty along the random search, greedy and receding horizon trajectories is significantly larger than that along the RRC and RRC* trajectories.

VII. Conclusions and Future Work

In this paper we proposed two incremental sampling-based algorithms to plan periodic trajectories for robots to estimate a spatio-temporal environmental field. The algorithms seek to minimize the largest eigenvalue of the error covariance matrix of the field estimate over an infinite horizon. These algorithms leverage the monotonicity property of the Riccati recursion to efficiently compare the cost of cycles in a tree which grows by successively adding random points. We applied our algorithms to plan periodic trajectories to estimate surface temperatures over the Caribbean Sea using US NOAA data. We also showed that our algorithms significantly out-perform several benchmark algorithms, including random search, a greedy method, and a receding horizon method. In future work we will consider using the sensing agents to learn the dynamics of the spatio-temporal field on-line from field measurements, and estimating this field based on these learned dynamics.



(a) Comparison results when starting at (500, 200).



(b) Comparison results when starting at (800, 600).

Fig. 12. The estimation quality as measured by the largest eigenvalue of the error covariance matrix is shown for the trajectories from our RRC and RRC* algorithms (blue and red, respectively), compared with with random search, greedy, and receding horizon algorithms (cyan, magenta, and green, respectively). The RRC and RRC* trajectories significantly outperform the other methods.

References

[1] W. Zhang, M. P. Vitus, J. Hu, A. Abate, and C. J. Tomlin, "On the optimal solutions of the infinite-horizon linear sensor scheduling problem," in *2010 49th IEEE Conference on Decision and Control (CDC)*, (Atlanta, GA, USA), pp. 396 – 401, 15-17 Dec. 2010.

[2] L. Zhao, W. Zhang, J. Hu, A. Abate, and C. J. Tomlin, "On the optimal solutions of the infinite-horizon linear sensor scheduling problem," *IEEE Transactions on Automatic Control*, vol. 59, no. 10, pp. 2825 – 2830, 2014.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, pp. 846–894, June 2011.

[4] S. L. Smith, M. Schwager, and D. Rus, "Persistent robotic tasks: Monitoring and sweeping in changing environments," *IEEE Transactions on Robotics*, vol. 28, pp. 410–426, April 2012.

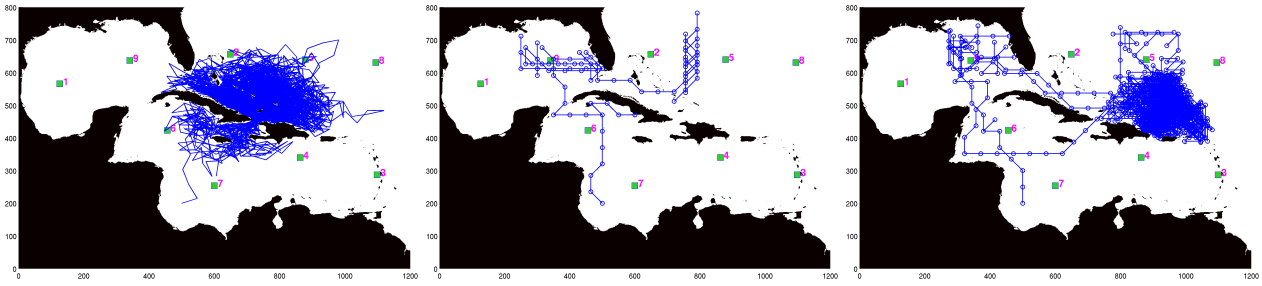
[5] E. K. W. Chu, H. Y. Fan, W. W. Lin, and C. S. Wangs, "Structure-preserving algorithms for periodic discrete-time algebraic riccati equations.," *International Journal of Control*, vol. 77, no. 8, pp. 767 – 788, 2004.

[6] L. Ljung, ed., *System Identification: Theory for the User*. Prentice Hall PTR, 1999.

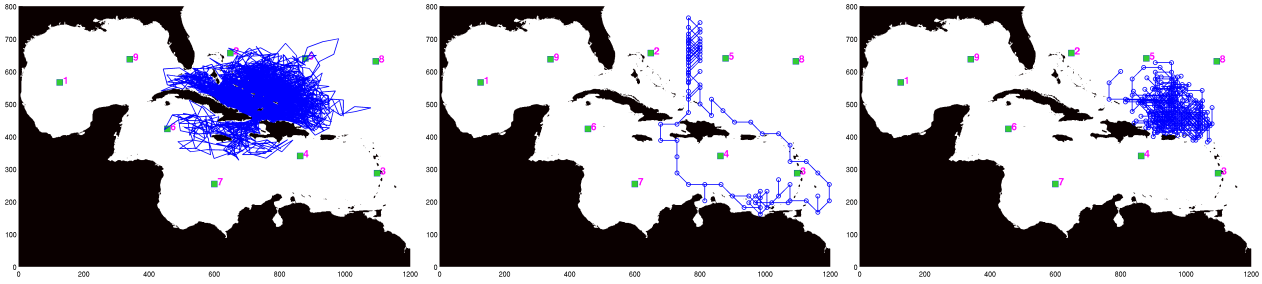
[7] P. Van Overschee and B. L. De Moor, *Subspace identification for linear systems: theory, implementation, applications*, vol. 3. Kluwer academic publishers Dordrecht, 1996.

[8] X. Lan and M. Schwager, "Planning periodic persistent monitoring trajectories for sensing robots in gaussian random fields," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 2415–2420, IEEE, 2013.

[9] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path



(a) Random search starting at (500, 200). (b) Greedy algorithm starting at (500, 200). (c) Receding horizon algorithm starting at (500, 200) with horizon equal to 4.



(d) Random search starting at (800, 600). (e) Greedy algorithm starting at (800, 600). (f) Receding horizon algorithm starting at (800, 600) with horizon equal to 4.

Fig. 11. Comparison of RRC and RRC* algorithm with random search, a greedy algorithm, and a receding horizon algorithm (receding horizon of 4). RRC and RRC* trajectories are those shown in red in Figure 6 and Figure 7, respectively. Blue curve is the trajectory planned by random search algorithm, greedy algorithm and receding horizon algorithm, respectively. Top row and bottom row are trajectories planned with different initial positions.

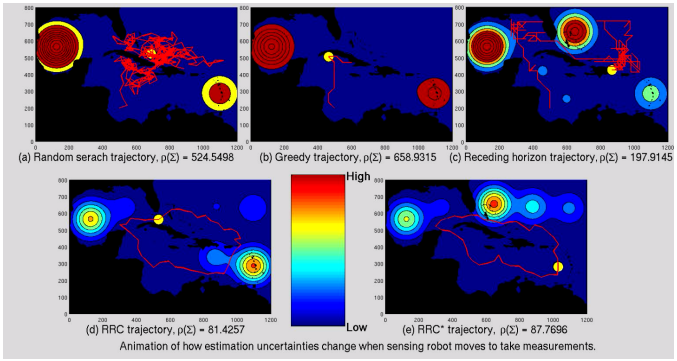


Fig. 13. The colormap indicates the field estimation uncertainties. Red hot stands for high uncertainty while dark blue stands for low uncertainty. The robot moves around in the environment, takes measurements and tries to minimize the estimation uncertainties everywhere in the environment (See multi-media attachment for an animation).

planning." TR 98-11, Computer Science Dept., Iowa State University, October 1998.

[10] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 566–580, August 1996.

[11] V. Gupta, T. H. Chung, B. Hassibi, and R. M. Murray, "On a stochastic sensor selection algorithm with applications in sensor scheduling and sensor coverage," *Automatica*, vol. 42, no. 2, pp. 251–260, 2006.

[12] M. P. Vitus, W. Zhang, A. Abate, J. Hu, and C. J. Tomlin, "On efficient sensor scheduling for linear dynamical systems," *Automatica*, vol. 48, no. 10, pp. 2482–2493, 2012.

[13] D. Shi and T. Chen, "Approximate optimal periodic scheduling of multiple sensors with constraints," *Automatica*, vol. 49, no. 4, pp. 993–1000, 2013.

[14] D. Shi and T. Chen, "Optimal periodic scheduling of sensor networks: A branch and bound approach," *Systems & Control Letters*, vol. 62, no. 9, pp. 732–738, 2013.

[15] S. Liu, M. Fardad, E. Masazade, and P. K. Varshney, "Optimal periodic sensor scheduling in networks of dynamical systems," *IEEE Transactions on Signal Processing*, vol. 62, pp. 3055 – 3068, June 2014.

[16] S. Bittanti, P. Colaneri, and G. De Nicolao, "The difference periodic ricati equation for the periodic prediction problem," *Automatic Control, IEEE Transactions on*, vol. 33, pp. 706 –712, Aug 1988.

[17] S. Bittanti, P. Colaneri, and G. D. Nicolao, "Analysis of the discrete-time periodic riccati equation by a kleinman procedure," in *Decision and Control, 1986 25th IEEE Conference on*, vol. 25, pp. 1444 –1449, Dec. 1986.

[18] J. Hench and A. Laub, "Numerical solution of the discrete-time periodic riccati equation," *Automatic Control, IEEE Transactions on*, vol. 39, pp. 1197 –1210, Jun 1994.

[19] P. Benner, R. Byers, R. Mayo, E. S. Quintana-Ortí, and V. Hernández, "Parallel algorithms for lq optimal control of discrete-time periodic linear systems," *Journal of Parallel and Distributed Computing*, vol. 62, no. 2, pp. 306 – 325, 2002.

[20] R. N. Smith, M. Schwager, S. L. Smith, B. H. Jones, D. Rus, and G. S. Sukhatme, "Persistent ocean monitoring with underwater gliders: Adapting sampling resolution," *Journal of Field Robotics*, vol. 28, pp. 714–741, September-October 2011.

[21] C. G. Cassandras, X. Lin, and X. Ding, "An optimal control approach to the multi-agent persistent monitoring problem," *Automatic Control, IEEE Transactions on*, vol. 58, no. 4, pp. 947–961, 2013.

[22] C. Song, L. Liu, G. Feng, and S. Xu, "Optimal control for multi-agent persistent monitoring," *Automatica*, vol. 50, no. 6, pp. 1663–1668, 2014.

[23] X. Lan and M. Schwager, "A variational approach to trajectory planning for persistent monitoring of spatiotemporal fields," in *American Control Conference (ACC), 2014*, pp. 5627–5632, IEEE, 2014.

[24] S. Alamdari, E. Fata, and S. L. Smith, "Persistent monitoring in discrete environments: Minimizing the maximum weighted latency between observations," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 138–154, 2014.

[25] J. Yu, S. Karaman, and D. Rus, "Persistent monitoring of events with stochastic arrivals at multiple stations," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 5758–5765, IEEE, 2014.
[26] N. E. Leonard, D. A. Paley, F. Lekien, R. Sepulchre, D. M. Fratantoni, and R. E. Davis, "Collective motion, sensor networks, and ocean sampling," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 48–74, Jan. 2007.
[27] K. M. Lynch, I. B. Schwartz, P. Yang, and R. A. Freeman, "Decentralized environmental modeling by mobile sensor networks," *IEEE Transactions on Robotics*, vol. 24, no. 3, pp. 710–724, June, 2008.
[28] J. L. Ny and G. J. Pappas, "On trajectory optimization for active sensing in gaussian process models," in *Joint 48th IEEE Conference on Decision and Control and 28th Chinese Control Conference*, (Shanghai, China), December 16-18, 2009.
[29] A. Singh, A. Krause, C. Guestrin, and W. J. Kaiser, "Efficient informative sensing using multiple robots," *Journal of Artificial Intelligence Research*, pp. 707–755, 2009.
[30] D. S. Levine, "Information-rich path planning under general constraints using rapidly-exploring random trees," Master's thesis, Massachusetts Institute of Technology, 2010.
[31] J. Binney and G. S. Sukhatme, "Branch and bound for informative path planning.," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 2147–2154, IEEE, 2012.
[32] N. Atanasov, J. Le Ny, K. Daniilidis, and G. J. Pappas, "Information acquisition with sensing robots: Algorithms and error bounds," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 6447–6454, IEEE, 2014.
[33] G. A. Hollinger and G. S. Sukhatme, "Sampling-based robotic information gathering algorithms," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1271–1287, 2014.
[34] N. Cressie, "The origins of kriging," *Mathematical Geology*, vol. 22, no. 3, pp. 239–252, 1990.
[35] N. Cressie, *Statistics for Spatial Data*. New York: Wiley, 1993.
[36] R. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.
[37] B. Sinopoli, L. Schenato, M. Franceschetti, K. Poolla, M. Jordan, and S. Sastry, "Kalman filtering with intermittent observations," *Automatic Control, IEEE Transactions on*, vol. 49, no. 9, pp. 1453–1464, 2004.
[38] NODC, "Nodc coastal water temperature guide." http://www.nodc.noaa.gov/cwtg/all_tmap.html.
[39] NOAA, "Noaa operational model archive and distribution system (oceannomads)." http://ecowatch.ncddc.noaa.gov/thredds/catalog/amseas/catalog.html.