Multi-Agent Path Planning for Level Set Estimation Using B-Splines and Differential Flatness

Grant Stagg and Cameron K. Peterson

***Abstract*—In this letter, we present a decentralized multi-agent path planning algorithm for level set estimation (LSE) and environmental monitoring missions. The planned paths are parameterized using B-splines and optimized using a novel objective function designed for LSE path planning that accounts for the exploration/exploitation trade-off while allowing the use of a gradient-based optimizer. We use the differential flatness property of the unicycle model to formulate constraints for our path optimization that ensure planned paths are kinematically feasible. We also employ a block coordinate ascent (BCA) algorithm that enables multi-agent coordination in exploring the environment. Finally, we present simulation and hardware results validating our approach.**

***Index Terms*—Autonomous agents, environment monitoring and management, path planning for multiple mobile robots or agents.**

I. Introduction

MULTI-AGENT systems (MAS) are capable of rapidly exploring and modeling large-scale environments. Cooperating teams of unmanned aerial systems (UASs) or unmanned ground vehicles (UGVs) can explore areas that are too dangerous for human exploration, such as chemical spills or nuclear radiation sites. One particularly interesting use of MAS is environmental monitoring, which involves equipping autonomous agents with sensors to measure phenomena of interest such as radiation, chemical spill concentrations, or harmful algae blooms. Agents in the MAS are tasked with creating a global model of the phenomena. This requires each agent to share information with other agents and make informed decisions on where to travel.

During environmental monitoring missions, it is often important to classify the operational area into high- or low-concentration regions. For example, given a radiation or chemical spill, identifying regions that contain dangerously high concentration levels is important for safety. For other phenomena, areas of high concentration could be of special interest, such as areas that contain high concentrations of harmful algal blooms. The problem of classifying regions into high and low sets is known as level set estimation (LSE) [1]. To perform LSE with MAS, autonomous agents need to plan paths to measurement locations that allow all regions of the operational area to be rapidly classified into high- and low-concentration sets. Past research in LSE and path planning has informed the development of our path planner.

Path planning can be divided into two categories, discrete [1], [2] and continuous [3], [4]. Discrete path planners generate a set of waypoints, whereas continuous path planners parameterize trajectories using continuous functions. The researchers in [2] presented a discrete path planner for LSE that formulated the path planning as an orienteering problem and solved for a discrete set of points for the agents to visit. However, this discrete path planner did not account for kinematically feasible trajectories between waypoints.

Continuous-based approaches plan trajectories for agents to follow instead of discrete sampling locations. In [3] a continuous path planner used B-splines to parameterize paths and mutual information as an objective function. Boundary and path length constraints were incorporated into the objective function as penalties. The paths were optimized using an evolutionary algorithm. However, this path planner did not account for kinematic feasibility constraints. Furthermore, their objective function was not differentiable and could not be used with a more efficient gradient-based optimizer that allows for hard constraints. This approach was extended in [4] to account for the exploration/exploitation trade-off.

Past research in multi-agent coordination for informative path planning includes dividing the environment into Voronoi cells and assigning each agent its own cell [5], [6]. Agents seek to learn the underlying field while simultaneously traveling to the local maximum of the field within their cell. LSE path planning does not seek to find the maximum of the underlying field, it seeks to classify the area into high and low sets. Voronoi partitioning could lead to agents being assigned to areas where the field value is far from the threshold and where not as many measurements are needed. Instead of using Voronoi partitioning, we use block coordinate ascent (BCA) for multi-agent coordination.

In this letter, we extend past researcher's work by planning continuous paths while enforcing kinematic feasibility constraints using differential flatness. We use B-splines to parameterize paths and employ a novel objective function for LSE (described in Section IV). Our objective function accounts for the exploration/exploitation trade-off by rewarding paths that explore new areas, but also penalizing paths that stray from the boundaries between low- and high-concentration regions. The function is designed to be differentiable, which allows it to be optimized using a gradient-based algorithm.

Manuscript received 28 November 2023; accepted 15 March 2024. Date of publication 3 April 2024; date of current version 11 April 2024. This letter was recommended for publication by Associate Editor G. Pereira and Editor M. A. Hsieh upon evaluation of the reviewers' comments. This work was supported in part by the Center for Autonomous Air Mobility and Sensing (CAAMS) and in part by the National Science Foundation Industry/University Cooperative Research Center (I/UCRC) under NSF Award IIP-2139551 along with significant contributions from CAAMS industry members. *(Corresponding author: Cameron K. Peterson.)*

The authors are with the Department of Electrical and Computer Engineering, Brigham Young University, Provo, UT 84602 USA (e-mail: ggs24@byu.edu; cammy.peterson@byu.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2024.3384763, provided by the authors.

Digital Object Identifier 10.1109/LRA.2024.3384763

2377-3766 © 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Brigham Young University. Downloaded on April 25,2025 at 17:24:34 UTC from IEEE Xplore. Restrictions apply.

The cooperative path optimization is achieved using BCA, which we believe is the first time BCA has been used in B-spline path optimization for multi-agent systems. In BCA we iteratively optimize a single agent's path while assuming all other agents' paths are constant [7]. Individual agents can then update their paths without relying on a centralized processor. Previous research used Gaussian process regression (GPR) to form LSE [1]. However, GPR scales poorly and is not suited to decentralized cooperation. We use a decentralized implementation of a sparse GPR algorithm (DSGPR) [8] to account for the complexities of cooperative environmental modeling. In summary, our letter's contributions include:

1) A path planning algorithm that uses differential flatness to plan kinematically feasible informative paths for LSE.
2) A novel objective function for LSE path optimization that accounts for the exploration/exploitation trade-off and is differentiable.
3) A decentralized multi-agent LSE path planner implemented with a block coordinate ascent algorithm.

This letter proceeds as follows. A problem statement is shown in Section II. Background information on DSGPR and LSE is provided in Section III. We present our path optimization algorithm in Section IV. Section V contains simulation and hardware results to validate our path planner and conclusions are given in Section VII.

II. Problem Statement

Consider a group of N_a agents operating in an environment $D \subset \mathbb{R}^2$. There exists an unknown scalar field $f(\boldsymbol{x})$:$D \mapsto \mathbb{R}$ that represents the environmental phenomena of interest. We assume we have a set of test points $\overline{X}_* = (\boldsymbol{x}_{*1}, \ldots, \boldsymbol{x}_{*N_*})$ evenly spaced across D, and a threshold h. We wish to classify the points in $\overline{X}_*$ into high- and low- concentration sets $H = \{x \in \overline{X}_* | f(\boldsymbol{x}) > h\}$ and $L = \{\boldsymbol{x} \in \overline{X}_* | f(\boldsymbol{x}) \leq h\}$. As agents traverse the environment they receive measurements of the underlying field corrupted with Gaussian noise, $z_i = f(\boldsymbol{x}_i) + \eta, \eta \sim \mathcal{N}(0, \sigma_n^2)$. We assume agents have no prior information about the environment. The goal of a LSE path planner is to plan paths to the measurement locations that best help classify points into H and L.

We also assume agent kinematics are represented with a 2D unicycle model, which has been used to model autonomous vehicles [9]. The agents are constrained to a 2D plane with state variables $[x, y, \theta]^\top$, where x is the distance East of the origin, y is the distance North of the origin and θ is the heading of the agent measured counterclockwise from the East axis. Agents obey non-holonomic constraints and only travel in the direction of their heading yielding kinematics of

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} v(t)\cos\theta(t) \\ v(t)\sin\theta(t) \\ u(t) \end{bmatrix}, \tag{1}$$

where $v(t)$ is the speed of the agent at time t and $u(t)$ is the turn rate. Both $v(t)$ and $u(t)$ are inputs to the system.

The goal of our path planning algorithm is to plan paths $\boldsymbol{p}^j(t) \in D, j \in (1, \ldots, N_A)$, for multiple agents to best classify the test points using the LSE algorithm. We plan continuous paths parameterized with B-splines and use constraints on velocity, turn rate and curvature to ensure that planned paths are kinematically feasible.

III. Background

In this section, we present essential background information to our path-planning algorithm. This includes a discussion of the DSGPR algorithm developed in [8], and LSE [1], which was the starting point for our path-planning algorithm.

A. Decentralized Sparse Gaussian Process Regression (DSGPR)

In our work, a level set estimator uses DSGPR to create a model of $f(\boldsymbol{x})$. DSGPR provides a mean and variance value for the model. The goal of DSGPR is to estimate an underlying function $f(\boldsymbol{x})$ at test points $\overline{X}_*$ using noisy measurements taken at locations $\overline{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{N_z})$, where N_z is the number of measurements, and $\boldsymbol{z} = (z_1, \ldots, z_{N_z})$ is a vector of measurements.

GPR uses a kernel function $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ to create a prior distribution between the test points and measurements. The original GPR algorithm scales poorly with the number of data points, so the authors in [10] introduced sparse Gaussian process regression (SGPR). SGPR uses inducing points to summarize the measured points. Using the kernel function, a Gaussian distribution for the inducing points $\overline{X}_u = (\boldsymbol{x}_{u1}, \ldots, \boldsymbol{x}_{uN_u})$ is defined with mean

$$\boldsymbol{m} = {\sigma_n}^{-2} K_{uu} \Sigma K_{uf} \boldsymbol{z} \tag{2}$$

and covariance

$$\Lambda = K_{uu} \Sigma K_{uu}, \tag{3}$$

where $\Sigma = ({K_{uu} + {\sigma_n}^{-2} K_{uf} K_{fu}})^{-1}$, $K_{uf} = [\kappa(\boldsymbol{x}_{ui}, \boldsymbol{x}_j)]_{\boldsymbol{x}_{ui} \in \overline{X}_u, \boldsymbol{x}_j \in \overline{X}}$, $K_{fu} = K_{uf}^\top$, $K_{uu} = [\kappa(\boldsymbol{x}_{ui}, \boldsymbol{x}_{uj})]_{\boldsymbol{x}_{ui}, \boldsymbol{x}_{uj} \in \overline{X}_u}$, and N_u is the number of inducing points. Subscripts u and f denote inducing points and measurement. Thus, K_{uu} is the kernel function between the inducing points and themselves and K_{uf} is the kernel function between the inducing points and the measurements.

In DSGPR, agents use a clustering algorithm to compute their set of inducing points ($\overline{X}_u^j$). They then calculate their local inducing point distributions from (2) and (3) resulting in a mean $\boldsymbol{m}^j$ and covariance Λ^j for $j \in [1, \ldots, N_a]$, where N_a is the number of agents. Each agent transmits its local distribution to neighboring agents using efficient message-passing heuristics. After the local models are transmitted, each agent combines all local models to compute a global model. The mean values of the local inducing point distributions are stacked in the vector $\overline{M} = [{\boldsymbol{m}^1}^\top, {\boldsymbol{m}^2}^\top, \ldots, {\boldsymbol{m}^{N_a}}^\top]^\top$, and $\overline{\Lambda}$ is a block diagonal matrix of Λ^i (the covariance of the local inducing points distributions). The locations of all agents' inducing points are combined into $\overline{X}_{\overline{U}} = (\overline{X}_u^1, \overline{X}_u^2, \ldots, \overline{X}_u^{N_a})$. The $\overline{U}$ subscript denotes the combined list of inducing points. The mean and covariance of the global model are given by

$$\mu(\overline{X}_*) = K_{*\overline{U}} K_{\overline{U}\overline{U}}^{-1} \overline{M}, \tag{4}$$

and

$$cov(\overline{X}_*) = K_{*\overline{U}} K_{\overline{U}\overline{U}}^{-1} \overline{\Lambda} K_{\overline{U}\overline{U}}^{-1} K_{\overline{U}*} + K_{**} - K_{*\overline{U}} K_{\overline{U}\overline{U}}^{-1} K_{\overline{U}*}, \tag{5}$$

Authorized licensed use limited to: Brigham Young University. Downloaded on April 25,2025 at 17:24:34 UTC from IEEE Xplore. Restrictions apply.

where $K_{*\overline{U}} = [\kappa(\boldsymbol{x}_{*i}, \boldsymbol{x}_{uj})]_{\boldsymbol{x}_{*i}\in\overline{X}_*, \boldsymbol{x}_{uj}\in\overline{X}_{\overline{U}}}$, $K_{\overline{U}\overline{U}} = [\kappa(\boldsymbol{x}_{ui}, \boldsymbol{x}_{uj})]_{\boldsymbol{x}_{ui}, \boldsymbol{x}_{uj}\in\overline{X}_{\overline{U}}}$, and $K_{\overline{U}*} = K_{*\overline{U}}^\top$.

Unlike the original LSE algorithm that uses a computationally expensive, centralized algorithm, we use DSGPR, which allows for real-time decentralized estimation.

B. Level Set Estimation (LSE)

We build on the algorithm presented by [1] to create our multi-agent LSE path planner. The goal of LSE is to create a map of high- and low-concentration regions using measurements of the environment. As measurements are taken they are incorporated into a GPR estimate for the underlying field value at test points $\boldsymbol{x}_{*i} \in \overline{X}_*$. This results in a mean value $\mu_{t_c}(\boldsymbol{x}_{*i})$ and a variance value $\sigma_{t_c}(\boldsymbol{x}_{*i})$ for each test point $\boldsymbol{x}_{*i}$ at time t_c. The subscript t_c emphasizes that the mean and covariance are time-dependent and will change as new measurements are included in the estimate.

Using the mean and covariance values, a confidence interval is constructed for each test point representing a range of values within which the true value of the underlying function is likely to fall. This range is given by $Q_{t_c}(\boldsymbol{x}_{*i}) = \mu_{t_c}(\boldsymbol{x}_{*i}) \pm \beta\sigma_{t_c}(\boldsymbol{x}_{*i})$, where β determines the width of the confidence interval. Using this interval, each $\boldsymbol{x}_{*i} \in \overline{X}_*$ can be classified into the high-concentration set $H_{t_c} \subseteq \overline{X}_*$ or the low-concentration set $L_{t_c} \subseteq \overline{X}_*$. If the region $Q_{t_c}(\boldsymbol{x}_{*i})$ is entirely above a predefined threshold h, then $\boldsymbol{x}_{*i}$ is classified into H_{t_c}. Similarly, if the entire region lies below h, $\boldsymbol{x}_{*i}$ is classified into L_{t_c}. The authors in [1] introduced an accuracy parameter ϵ that relaxes the bounds, resulting in the following

$$H_{t_c} = \{\boldsymbol{x}_{*i} \in \overline{X}_* | \min(Q_{t_c}(\boldsymbol{x}_{*i})) + \epsilon > h\} \tag{6a}$$

$$L_{t_c} = \{\boldsymbol{x}_{*i} \in \overline{X}_* | \max(Q_{t_c}(\boldsymbol{x}_{*i})) - \epsilon \leq h\}. \tag{6b}$$

Points that do not meet either criterion are said to be unclassified and added to $U_{t_c} \subseteq \overline{X}_*$: $U_{t_c} = \overline{X}_*/(H_{t_c} \cup L_{t_c})$.

IV. Path Optimization

The goal of our path planner is similar to that of other LSE path planners; however, we plan continuous paths with kinematic feasibility constraints. To achieve this goal, we present a novel path optimization algorithm that uses the property of differential flatness to generate paths that are kinematically feasible (Section IV-A) and B-splines to parameterize those paths (Section IV-B). We then use a gradient-based optimization algorithm to select B-spline control points to maximize our objective function while complying with constraints (Section IV-C). A receding horizon (RH) scheme is employed in combination with a block coordinate ascent (BCA) algorithm to re-plan paths when new information is available and enable multi-agent cooperation (Section IV-E).

A. Differentially Flat Model

We use the property of differential flatness to provide constraints that ensure planned paths are kinematically feasible [11]. A system is differentially flat if all the states and inputs can be expressed as a function of the same flat output and its derivatives. In the case of the unicycle model, we choose the flat output space to be the 2D plane. We can define a trajectory on the plane as $\boldsymbol{p}(t) = [x(t), y(t)]^\top$. The unicycle model has been shown to be differentially flat in [12]. As such, our inputs $v(t)$ and $u(t)$ can be written as a function of $\boldsymbol{p}(t)$ and its derivatives. In these terms, the velocity is

$$v(t) = ||\dot{\boldsymbol{p}}(t)||_2, \tag{7}$$

and the turn rate $u(t)$ is

$$u(t) = \frac{\dot{\boldsymbol{p}}(t) \times \ddot{\boldsymbol{p}}(t)}{||\dot{\boldsymbol{p}}(t)||_2^{\,2}}. \tag{8}$$

We can compute the curvature of the trajectory as

$$\kappa(t) = \frac{u(t)}{v(t)}. \tag{9}$$

Using the property of differential flatness allows us to optimize agents' paths in the flat output space (x, y) instead of the full state space. We can also put constraints on the turn rate, velocity and curvature in terms of the path $\boldsymbol{p}(t)$, planned in the flat output space as we will see in Section IV-C.

B. B-Splines

B-splines are used to define agent paths in the flat output space. B-splines are piecewise polynomial functions defined by a list of control points $\overline{C} = (\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_{N_c})$ and a set of knot points $\boldsymbol{t}_k = (t_1, t_2, \ldots, t_{N_k})$, where there are N_c control points and N_k knot points. For the 2D path $\boldsymbol{c}_i = [c_i^x, c_i^y]^\top$ and $t_i \in \mathbb{R}^+$. The B-spline can be written as a weighted sum of basis functions with the control points as weights,

$$\boldsymbol{p}(t) = \sum_{i=1}^{N_c} B_{i,k}(t)\boldsymbol{c}_i, \tag{10}$$

where k is the order of the b-spline and the basis functions $B_{i,k}$ are defined using the Cox-de Boor recursive formula shown in [13]. We choose to use $k = 3$ because this creates a spline that has two smooth derivatives.

For our path planning application, we use clamped B-splines. This means the first $k+1$ and last $k+1$ knot points are identical: $\boldsymbol{t}_k = (t_1, \ldots, t_1, t_2, \ldots, t_{N_k}, \ldots, t_{N_k})$, where N_k is now the number of distinct knot points. This ensures that the initial and final positions of the path are the first and last control points. The rest of the knot points are spaced out evenly between the first and last knot points. Using clamped B-splines allows us to easily plan paths starting at the agent's current position by setting the first control point to that location.

To ensure that the velocity is continuous we set the initial velocity of the path to the current velocity of the agent. We do this by first noting that the derivative of the B-spline curve is another B-spline of order $k-1$ given by, $\dot{\boldsymbol{p}}(t) = \sum_{i=1}^{N_c-1} B_{i,k-1}(t)\boldsymbol{b}_i$, where $\boldsymbol{b}_i = (\boldsymbol{c}_{i+1} - \boldsymbol{c}_i)k/(t_{i+k} - t_i)$.

Because we are using clamped B-splines we know the curve passes through the first control point. This holds for the derivative, meaning that $\dot{\boldsymbol{p}}(t_1) = \boldsymbol{b}_1$. Let v_{t_c} be the current velocity of the agent and θ_{t_c} be the current heading of the agent. Using the definition of $\boldsymbol{b}_1$ we can solve for the x and y components of $\boldsymbol{c}_2$ as

$$\boldsymbol{c}_2 = \frac{v_{t_c}(t_{k+2} - t_2)}{k} \begin{bmatrix} \cos(\theta_{t_c}) \\ \sin(\theta_{t_c}) \end{bmatrix} + \boldsymbol{c}_1. \tag{11}$$

Authorized licensed use limited to: Brigham Young University. Downloaded on April 25,2025 at 17:24:34 UTC from IEEE Xplore. Restrictions apply.

From (11), we can see that the second control point of the agent's path is constrained by its current velocity. With the agent's path defined in the flat output space, we are now ready to optimize the control points.

C. Path Optimization

Our optimization objective is to determine a path that allows the LSE algorithm to rapidly classify the boundary space into high- and low-concentration regions. To do this we use a novel objective function for LSE.

Our objective function rewards paths in areas where more accurate DSGPR predictions are needed and is differentiable for use with an efficient gradient-based optimizer. LSE creates confidence regions using the mean $\mu_{t_c}(\boldsymbol{x}_{*i})$ and variance $\sigma_{t_c}(\boldsymbol{x}_{*i})$ for each $\boldsymbol{x}_{*i} \in \overline{X}_*$. The closer the mean value is to the threshold h the more certain the estimation needs to be at that location to properly classify the region. This gives rise to a two-part objective function. On one hand, the path planner should favor regions with high uncertainty. However, the path planner also needs to tend towards regions where the mean value is close to the threshold. To achieve this balance, we use the following utility function, for a single point $\boldsymbol{x}$:

$$\Gamma(\boldsymbol{x}) = \alpha\sigma_{t_c}(\boldsymbol{x}) - (1-\alpha)(h - \mu_{t_c}(\boldsymbol{x}))^2, \tag{12}$$

with tuning parameter $\alpha \in [0, 1]$. Other works have combined the mean and variance of GPR as an objective function such as [4] and [5]. However, ours differs from these because instead of seeking to find the maximum value of $f(\boldsymbol{x})$, we seek to quickly classify H and L. Other LSE path planners use a measure called ambuiguity as an objective function [1], [2] or mutual information excluding uninteresting points (points outside of H) [3]. However, these objective functions are not differentiable and we were unable to use numerical derivative approximations to achieve convergence in our experiments.

Our utility function rewards exploring regions of high uncertainty, but also applies a quadratic penalty to sampling points whose mean value is far from the threshold h. This incentivizes the paths in regions where the mean is closer to h, while also allowing exploration. The tuning parameter α quantifies the trade-off between the exploration of new areas and the exploitation of areas where the mean value is close to h. A value of $\alpha = 1$ indicates that only uncertainty is considered without penalty, and a value of $\alpha = 0$ indicates that only the mean value is considered.

Using this utility function, the single-agent path planning problem is defined as

$$\overline{C}_{opt} = \text{argmax}_{\overline{C}} \sum_{i=1}^{N_m} \Gamma(\boldsymbol{p}(t_c + i\Delta_{tf})) \tag{13a}$$

$$\text{subject to } \boldsymbol{c}_1 = \boldsymbol{p}_{t_c}, \boldsymbol{p}(\boldsymbol{t}_s) \in D, v_{lb} \leq v(\boldsymbol{t}_s) \leq v_{ub} \tag{13b}$$

$$u_{lb} \leq u(\boldsymbol{t}_s) \leq u_{ub}, -\kappa_{ub} \leq \kappa(\boldsymbol{t}_s) \leq \kappa_{ub} \tag{13c}$$

$$\boldsymbol{c}_2 = \frac{v_{t_c}(t_{k+2} - t_2)}{k} \begin{bmatrix} \cos(\theta_{t_c}) \\ \sin(\theta_{t_c}) \end{bmatrix} + \boldsymbol{c}_1 \tag{13d}$$

where $\boldsymbol{t}_s = (t_c, t_c + \Delta_{ts}, t_1 + 2\Delta_{ts}, \ldots, t_c + N_s\Delta_{ts})$, N_s is the number of discrete samples for the constraints, t_c is the current time, $\Delta_{ts} = T_p/N_s$, T_p is how many seconds into the future the path is optimized over, Δ_{tf} is the sensor sampling period, $N_m = T_p/\Delta_{tf}$ is the number of measurements that would be taken if the path were followed, and $\boldsymbol{p}_{t_c}$ is the current position of the agent. The knot points of the spline are $\boldsymbol{t}_k = (t_c, \ldots, t_c, t_c + j\Delta_{tk} \ldots, t_c + T_p, \ldots, t_c + T_p), j \in (1, 2, \ldots, N_k)$, where $\Delta_{tk} = T_p/N_k$ is the time spacing of the non-repeated knot points. This ensures the spline is defined on the time-interval $[t_c, t_c + T_p]$. Sampling the continuous constraints at $\boldsymbol{t}_s$ creates N_s evenly spaced samples on $[t_c, t_c + T_p]$ where Δ_{ts} is the time difference between samples.

The objective function is the sum of the utility (12) of future measurements. We find future measurements by sampling the B-spline trajectory at the sensor sampling frequency. The utility of each future measurement is calculated and summed as shown in (13a).

We choose to sample the continuous constraint functions (13b)–(13c) to create discrete constraints. Since we enforce the constraints at discrete time steps, it is possible to have constraint violations in between sampled points. However, the likelihood of this is low and can be reduced by choosing a large value for N_s. Our path planning problem has six constraints. The equality constraints ensure the starting position and initial velocity of the path are set to the current position and velocity of the agent. The next constraints ensure kinematic feasibility. The velocity constraint guarantees that the agent's velocity will not exceed the max speed v_{ub}, or dip below the minimum speed v_{lb}. These bounds are determined based on the capabilities of the agent. For a fixed-wing UAS, this can be used to ensure the commanded speed is below the agents' maximum speed and is above the stall speed. For our work, we assume $v_{lb} > 0$, the vehicles can only travel forward. The turn rate constraint, ensures the trajectory does not require unfeasible turn rates. The curvature constraint ensures that agents are not commanded to perform unfeasible sharp turns.

To perform the optimization, we use an interior point, gradient-based, nonlinear optimizer called IPOPT [14]. With gradient-based optimizers, all constraints and the objective function must be differentiable. Observations of (7), (8), (9), (10), and (13a) show that they are differentiable with respect to the B-spline control points. These Jacobians are shown in the following section.

Because future measurements affect the value of the objective function, it is important to re-plan paths when new measurements are available. We use an RH scheme to accomplish this. First, a path is planned T_p seconds into the future by optimizing (13). The first T_c seconds of the path are executed. The path is then re-planned using information from the new measurements.

D. Jacobians

We choose to use gradient-based optimization because it is often more efficient than gradient-free methods. Furthermore, gradient-free methods do not allow hard constraints (constraints must be added as penalties on the objective) [15]. To use a gradient-based optimizer the gradient of the objective function and Jacobians of the constraints are needed.

To compute the gradient and Jacobians we note the B-spline equation can be rearranged to a matrix representation given by $\boldsymbol{p}(t) = \Phi(t)C$, where $\Phi(t)$ is a matrix that encapsulates the B-spline basis functions and $C = [\boldsymbol{c}_1\ \boldsymbol{c}_2 \ldots\ \boldsymbol{c}_{Nc}]^\top$ are the control points. More information on how to calculate $\Phi(t)$ is

Authorized licensed use limited to: Brigham Young University. Downloaded on April 25,2025 at 17:24:34 UTC from IEEE Xplore. Restrictions apply.

given in [16]. From this we see that the derivative of a B-spline with respect to its control points is $\Phi(t)$.

To complete our optimization, we need the gradient of our objective function (13a) with respect to the B-spine control points (the optimization variables). We start by moving the derivative inside the summation resulting in $\partial F(\boldsymbol{p}(t))/\partial C = \sum_{i=1}^{N_m} \partial\Gamma(\boldsymbol{p}(t_c + i\Delta_{tf}))/\partial C$. The derivative of the utility function for a single point i is given by

$$\frac{\partial}{\partial C}\Gamma(\boldsymbol{p}(t_c + i\Delta_{tf})) = \alpha\frac{\partial}{\partial C}\sigma_{t_c}(\boldsymbol{p}(t_c + i\Delta_{tf})) + 2(1-\alpha)(h - \mu_{t_c}(\boldsymbol{p}(t_c + i\Delta_{tf})))\frac{\partial}{\partial C}\mu_{t_c}(\boldsymbol{p}(t_c + i\Delta_{tf})). \tag{14}$$

The derivative of the DSGPR output $\mu_{t_c}(\boldsymbol{p}(t_c + i\Delta_{tf}))$ and $\sigma_{t_c}(\boldsymbol{p}(t_c + i\Delta_{tf}))$ should also be computed. First, we compute the derivative of the mean value of DSGPR (given in (4)) with respect to the B-spline control points. The only value of (4) that varies with the control points is $K_{*\overline{U}}$. This is because as we change the B-spline we change the locations of the test points, which are evenly spaced along the B-spline, and this alters where we evaluate the DSGPR. The single test point that we are evaluating for the objective function at is $\boldsymbol{x}_{*i} = (\boldsymbol{p}(t_c + i\Delta_{tf}))$. This results in

$$\frac{\partial}{\partial C}\mu_{t_c}(\boldsymbol{p}(t_c + i\Delta_{tf})) = \frac{\partial}{\partial C}[K_{*\overline{U}}]K_{\overline{U}\overline{U}}^{-1}\overline{M}, \tag{15}$$

with

$$\frac{\partial}{\partial C}K_{*\overline{U}} = \left[\frac{\partial}{\partial C}\left[\kappa\left(\boldsymbol{p}(t_c + i\Delta_{tf}), \boldsymbol{x}_{u_k}\right)\right]\right]_{\boldsymbol{x}_{u_k}\in\overline{X}_{\overline{U}}}. \tag{16}$$

Next, we need the derivative of the kernel function;

$$\frac{\partial}{\partial C}\left[\kappa\left(\boldsymbol{p}(t_c + i\Delta_{tf}), \boldsymbol{x}_{u_k}\right)\right] = \frac{\partial}{\partial \boldsymbol{p}(t_c + i\Delta_{tf})}\left[\kappa\left(\boldsymbol{p}\left(t_c + i\Delta_{tf}\right), \boldsymbol{x}_{u_k}\right)\right]\frac{\partial}{\partial C} \times \left[\boldsymbol{p}\left(t_c + i\Delta_{tf}\right)\right]. \tag{17}$$

The kernel function needs to be differentiable with respect to its inputs for the gradient to exist. The derivative of $\boldsymbol{p}(t_c + i\Delta_{tf})$ in (17) is given by $\Phi(t_c + i\Delta_{tf})$.

The derivative of $\sigma_{t_c}(\boldsymbol{p}(t_c + i\Delta_{tf}))$ is similarly given by

$$\frac{\partial}{\partial C}\left[\sigma_{t_c}(\boldsymbol{p}(t_c + i\Delta_{tf}))\right] = 2\frac{\partial}{\partial C}[K_{*\overline{U}}]K_{\overline{U}\overline{U}}^{-1}\overline{\Lambda}K_{\overline{U}\overline{U}}^{-1}K_{\overline{U}*} - 2\frac{\partial}{\partial C}[K_{*\overline{U}}]K_{\overline{U}\overline{U}}^{-1}K_{\overline{U}*}, \tag{18}$$

where $\frac{\partial}{\partial C}[K_{*\overline{U}}]$ is given by (16). The position constraint Jacobian in (13b) is given by $\Phi(\boldsymbol{t}_s)$. The velocity constraint Jacobian can be found by differentiating (13b) with respect to C resulting in

$$\frac{\partial v(\boldsymbol{t}_s)}{\partial C} = \frac{\dot{\boldsymbol{p}}(\boldsymbol{t}_s)\dot{\Phi}(\boldsymbol{t}_s)^\top}{||\dot{\boldsymbol{p}}(\boldsymbol{t}_s)||_2}. \tag{19}$$

Similarly, the turn rate constraint Jacobian is given by (20). Finally, the Jacobian of the curvature constraint can be found using the quotient rule and (19) and (20) shown at the bottom of this page. We use the gradient of our objective function and the Jacobians of our constraints to efficiently execute our optimization algorithm.

E. Block Coordinate Ascent

The simplest approach to enable multi-agent path planning is to combine all the agents' trajectories into a single optimization problem. However, this increases the number of optimization design variables and constraints, making path planning prohibitively expensive. Combining agents' trajectories also requires a centralized processing unit with a reliable connection to all agents to perform the optimization. These communication constraints limit the size of the operational area and make path planning sensitive to communication failures. In this work, we provide a decentralized solution that improves performance and robustness.

We use BCA to achieve decentralized path planning. BCA iteratively maximizes a function by holding portions of the optimization variables constant while optimizing others [7]. BCA has been used to create decentralized multi-agent path planners in [17], [18]. In both, BCA was used to plan discrete paths, either by optimizing paths to discrete locations or by discretizing the trajectory into waypoints. Neither work addresses LSE path planning and to our knowledge, this is the first time BCA has been used to plan continuous trajectories parameterized with B-splines for multi-agent path planning.

We apply BCA by holding all other agents' paths constant while a single agent's path is optimized. After an agent optimizes its path, it transmits information to connected agents about its path, including "virtual" local models $(\widetilde{\boldsymbol{m}}^j, \widetilde{\Lambda}^j, \widetilde{X}_u^j)$ that incorporate agent j's past measurements and potential future measurements along their planned path. Using this information, the next agent plans its path and then transmits its path information. This process is repeated, looping through all agents until a convergence criterion is met.

Algorithm 1 outlines how we use BCA for multi-agent B-spline path planning for LSE. As part of DSGPR each agent stores a list of inducing points locations, mean values, and covariances it has received from other agents. These lists are $\overline{X}_{\overline{U}}^j$, $\overline{M}^j$, and $\overline{\Lambda}^j$ for the j^{th} agent. To store virtual path information agents will need another group of lists for the virtual inducing point locations, virtual mean values, and virtual covariances. For the j^{th} agent we will denote these as $\widetilde{X}_{\overline{U}}^j$, $\widetilde{M}^j$, and $\widetilde{\Lambda}^j$. As an input, our algorithm takes the actual and virtual lists as well as the current state and velocity of the agent. Note that because our algorithm and DSGPR are decentralized, each agent only needs its own lists, including the stored virtual lists of other agents, to perform path planning. The algorithm outputs a new path for each agent. This algorithm is triggered whenever any agent needs to re-plan its path ($t_e^j > T_c$).

$$\frac{\partial u(\boldsymbol{t}_s)}{\partial C} = \frac{||\dot{\boldsymbol{p}}(\boldsymbol{t}_s)||_2^2(\dot{\Phi}(\boldsymbol{t}_s) \times \ddot{\boldsymbol{p}}(\boldsymbol{t}_s) + \dot{\boldsymbol{p}}(\boldsymbol{t}_s) \times \ddot{\Phi}(\boldsymbol{t}_s)) - 2(\dot{\boldsymbol{p}}(\boldsymbol{t}_s) \times \ddot{\boldsymbol{p}}(\boldsymbol{t}_s))\dot{\boldsymbol{p}}(\boldsymbol{t}_s)\dot{\Phi}(\boldsymbol{t}_s)^\top}{||\dot{\boldsymbol{p}}(\boldsymbol{t}_s)||_2^4} \tag{20}$$

Authorized licensed use limited to: Brigham Young University. Downloaded on April 25,2025 at 17:24:34 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: BCA for B-Spline LSE Path Planning.

1: **Input:** $\overline{X}^j_{\overline{U}}, \overline{M}^j, \overline{\Lambda}^j, \widetilde{X}^j_{\overline{U}}, \widetilde{M}^j, \widetilde{\Lambda}^j, [x^j_1, y^j_1, \theta^j_1], v^j_1, \forall j \in [1, \ldots, N_a]$
2: **Output:** $\boldsymbol{p}^j(t)\ \forall j \in [1, \ldots, N_a]$
3: **for** $i \in [1, \ldots, N_B]$ **do**
4: **for** $j \in [1, \ldots, N_a]$ **do**
5: **if** $t^j_e > T_c$ **then**
6: $\boldsymbol{p}^j(t) =$ `optimizePath`$(\boldsymbol{p}^j_{t_c}, v^j_{t_c}, \theta^j_{t_c} \widetilde{X}^j_{\overline{U}}, \widetilde{M}^j, \widetilde{\Lambda}^j)$ {Eq. (13)}
7: $\widetilde{X}^j = (\boldsymbol{p}^j(t_c), \boldsymbol{p}^j(t_c + \Delta_{tf}), \ldots, \boldsymbol{p}^j(t_c + N_m \Delta_{tf}))$
8: $\widetilde{\boldsymbol{z}}^j = \mu_{t_c}(\widetilde{X}^j)$ {Eq. 4}
9: $\widetilde{X}^j_u = (\boldsymbol{p}^j(t_c), \boldsymbol{p}^j(t_c + \Delta_{tv}), \ldots, \boldsymbol{p}^j(t_c + N_v \Delta_{tv}))$
10: $\widetilde{X}^j_c = \widetilde{X}^j \cup \overline{X}^j, \widetilde{X}^j_{uc} = \widetilde{X}^j_u \cup \overline{X}^j_u, \widetilde{\boldsymbol{z}}^j_c = [\widetilde{\boldsymbol{z}}^{j\top}, \boldsymbol{z}^{j\top}]^\top$
11: $\widetilde{\boldsymbol{m}}^j_c, \widetilde{\Lambda}^j_c =$ `virtualLocalModel`$(\widetilde{X}^j_c, \widetilde{\boldsymbol{z}}^j_c, \widetilde{X}^j_{uc})$ {Eqs. (2), (3)}
12: `transmit`$(\widetilde{\boldsymbol{m}}^j_c, \widetilde{\Lambda}^j_c, \widetilde{X}^j_{uc})$
13: **if** $||\boldsymbol{p}^j_{t_c} - \boldsymbol{p}^q_{t_c}||_2 \leq r_c\ \forall q \in [1, \ldots, N_a]$ **then**
14: update $\widetilde{\boldsymbol{m}}^j_c, \widetilde{\Lambda}^j_c, \widetilde{X}^j_{uc}$ in $\widetilde{\Lambda}^i, \widetilde{M}^i, \widetilde{X}^i_{\overline{U}}$
15: **end if**
16: $t^j_e \leftarrow 0$
17: **end if**
18: **end for**
19: **end for**

The outer loop on line 3 shows the BCA iterations. We choose to use a fixed number of BCA iterations N_B; however, this could be replaced with different convergence criteria. For every BCA iteration, we iterate through all agents and check if t^j_e (the time elapsed since the j^{th} agent re-planned its path) is greater than T_c (the RH re-plan horizon). If the path needs to be re-planned the B-spline control points are found using a non-linear optimizer to solve (13) (line 6). The mean and covariance in the objective function are computed using DSGPR (4) and (5) and the agent's current lists of virtual information $\widetilde{X}^j_{\overline{U}}, \widetilde{M}^j, \widetilde{\Lambda}^j$. If an agent has not yet received a virtual model of another agent it uses the actual model, from $\overline{X}^j_{\overline{U}}, \overline{M}^j, \overline{\Lambda}^j$.

Then, virtual local models $\widetilde{\boldsymbol{m}}^j_c$ and $\widetilde{\Lambda}^j_c$ are created in lines 7–11. First, we sample $\boldsymbol{p}^j(t)$ at a frequency of $1/\Delta_{tf}$ to simulate future measurement locations $\widetilde{X}^j$ in line 7. The virtual measurements $\widetilde{\boldsymbol{z}}^j$ are computed as the output of the global model of DSGPR at the virtual measurement locations, $\widetilde{\boldsymbol{z}}^j = \mu_{t_c}(\widetilde{X}^j)$ from (4). Virtual inducing points are found by sampling the planned trajectory at a frequency of $1/\Delta_{tv}$, where $\Delta_{tv} = T_p/N_v$ and N_v is the number of virtual inducing points The virtual inducing points and measurements are then combined with the agent's actual measurements and inducing points in line 10. The agent computes the mean and covariance $(\widetilde{\boldsymbol{m}}^j_c, \widetilde{\Lambda}^j_c)$ of the combined virtual local models using (2) and (3). The mean, covariance, and inducing point locations of the agent's virtual model are then transmitted to all agents within communication range r_c in line 12. Every agent that receives the virtual local models updates their lists of virtual models (line 14).

TABLE I
B-SPLINE PATH OPTIMIZATION PARAMETERS

Parameter	Symbol	Algorithm	Value
Kernel signal variance	σ_f	DSGPR	1.0
Kernel length scale	λ	DSGPR	0.05
Measurement noise	σ_n	DSGPR	0.0001
High kernel significance factor	-	DSGPR	0.8
Low kernel significance factor	-	DSGPR	0.05
Re-cluster significance factor	-	DSGPR	0.6
Interval scaling factor	β	LSE	1.0
Accuracy parameter	ϵ	LSE	0.6
Number of test points	N_x	LSE	10000
Max/min turn rate	u_{lb}/u_{ub}	Path planner	$\pm$5.0 (rad/sec)
Minimum velocity	v_{lb}	Path planner	5.0 (m/sec)
Maximum velocity	v_{ub}	Path planner	10.0 (m/sec)
Maximum curvature	κ_{ub}	Path planner	0.5 (rad/m)
Tuning parameter	α	Path planner	0.9
Look ahead time	T_p	Path planner	10.0 (sec)
Re-plan time	T_c	Path planner	2.0 (sec)
BCA iterations	N_B	Path planner	2
Number of spline control points	N_c	Path planner	9
Number of constraint samples	N_s	Path planner	20
Sensor sampling frequency	f_s	Path planner	1.0 (Hz)

V. SIMULATION RESULTS

In this section, we evaluate the performance of our LSE path planning algorithm. We provide a baseline comparison with a complete coverage "lawnmower" path and a greedy planner. The "lawnmower" path planner divides the boundary space into rectangular cells and each agent performs a sweeping pattern in the cell. The greedy path planner selects the highest reward point according to the utility function given by (12) and travels towards it. For multiple agents, each subsequent agent selects the next highest reward point, provided it is a pre-defined distance away from all prior selected reward points. Specifically, the greedy algorithm is implemented as follows. To start all points in D are added to W (the current points for consideration). The first agent selects the highest reward point in W as its current waypoint (wp^1). All points within a distance d of the waypoint are removed from W and the next agent selects the highest reward point in the new W. When the j^{th} agent reaches a waypoint the points within a distance d of its current waypoint are added back to W for consideration. Then the highest reward point in W is selected. This process is repeated until the mission is over.

To simulate a field with multiple disjoint high- and low-concentration regions, we randomly place six Gaussian functions in the boundary space with randomly chosen covariances and positions. Gaussian zero-mean noise is added to each measurement to simulate sensor noise. We perform 100 Monte Carlo (MC) runs where the location and covariance of the Gaussian functions are randomly changed for each run. Agent dynamics are simulated by numerically integrating (1). The parameters used for DSGPR, LSE, and our path planner are listed in Table I.

To measure how well each path planner performs we use the f_1 score as was previously done in [1] and [2]. The f_1 score puts more weight on correctly classifying the high-concentration region than the low-concentration region and is

$$f_1 = \frac{T_p}{T_p + \frac{1}{2}(F_p + F_n)}, \tag{21}$$

where T_p is the number of true positives (the number of correctly labeled points in H_t), F_p is the number of false positives (the number of points that should be classified into L_t but are either classified in H_t or U_t) and F_n is the number of false negatives

Authorized licensed use limited to: Brigham Young University. Downloaded on April 25,2025 at 17:24:34 UTC from IEEE Xplore. Restrictions apply.

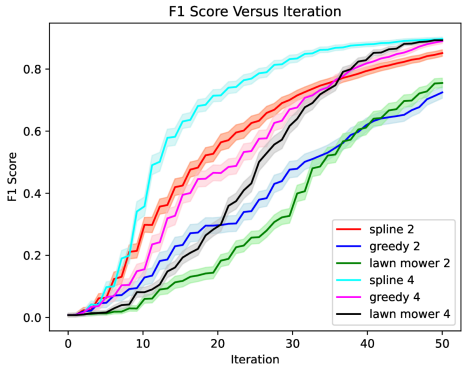

Fig. 1. B-spline path planner for runs with two and four agents. This plot shows the average f1 score versus iteration for 100 Monte Carlo runs. The 1-sigma bounds are also shown in a lighter shading. Our path planner is shown in red for 2 agents and cyan for 4 agents. The greedy and lawnmower path planners are also shown for both 2 and 4 agents.

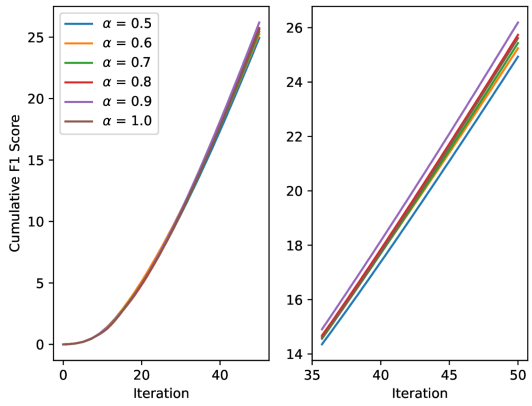

Fig. 2. Cumulative f_1 score is shown averaged across 100 MC runs while varying the α tuning parameter. The left plot shows all 50 iterations and the right plot shows the last 15 iterations where there is more variation.

(the number of points that should be classified in H_t but were actually classified into L_t or U_t).

Fig. 1 shows the mean f_1 score and its $1-\sigma$ bound versus the iteration for our path planner, the greedy, and "lawnmower" path planners for both two and four agents. Our path planner outperforms the greedy and complete coverage planner given the same number of agents. After the first few measurements are taken the average f_1 score of our path planner exceeds the greedy and "lawnmower" planners. This is because our path planner adapts its path based on incoming measurements and tends towards regions where more accurate estimates of the field are needed. Towards the end, when complete coverage is starting to occur, the complete coverage "lawnmower" path planner starts to catch up. As expected the four agent run increases the f_1 score faster because more measurements are being incorporated at each iteration.

The staircase-like steps shown in Fig. 1 are an artifact of the DSGPR algorithm, which uses a clustering algorithm to select the number and location of inducing points for the SGPR. When clustering occurs, accuracy improves. The clustering algorithm is spatially dependent triggering clustering when new areas are explored. During the initial iterations the whole boundary space is unexplored so re-clustering will occur at approximately the same iterations through all Monte Carlo runs. When more of the boundary space has been explored, re-clustering will not occur at the same iterations across the MC runs and the staircase-like steps are averaged out.

It took on average 1.47 seconds to perform the optimization in (13). This means that if there are N_{au} agents' paths that need to be updated (all agents where $t_e^j > T_c$), then the complete BCA optimization will take $1.47 N_{au} N_B$ seconds. We implemented all algorithms in un-optimized Python code and believe run-times will improve when using a higher-performance language. If paths are planned asynchronously, making sure only a small number of agents' paths need to be re-planned at a given iteration, our current optimization time is reasonable for real-time path planning.

We now illustrate the effect of the tuning parameter on the performance of our path planner. Fig. 2 shows a plot of the cumulative f_1 score averaged across 100 MC runs for different tuning parameter values ranging from $\alpha \in [0, 1]$. Low values of α (i.e. $\alpha \leq 0.5$) perform poorly because they are unable to explore the environment and are omitted from the figure. The plot shows that all α values perform similarly through the initial iterations. At around 30 iterations the performance becomes noticeably different. The cumulative f_1 score increases with increasing α values before peaking at a value of $\alpha = 0.9$ and then decreasing with $\alpha = 1.0$. This shows that a value of $\alpha = 0.9$ offers the best balance between exploration and exploitation of the boundary space. The ideal α value depends on the magnitude of the data being measured, the uncertainty of the measurements, and how fast the field changes spatially, which can vary for different applications.

VI. Hardware Results

We conducted hardware tests to validate our path planner using two TurtleBot robots, which is a type of unicycle robot that operates on the Robot Operating System (ROS). To model the TurtleBot's dynamics, we used (1). We use the TurtleBots to classify light and dark regions within an operating area. In our experiments, each TurtleBot was equipped with a BH1750 sensor, which measures light intensity in lux. To create distinct regions of high and low light intensity, we started with an unlit room and hung a flashlight pointing straight down. To navigate the paths generated by our algorithm, we implemented the controller presented in [19].

For our test we let each robot take 50 measurements at a sampling frequency of 1 Hz. We had a look ahead time $T_p = 20$ seconds and a re-plan time of 5 seconds. The boundary space was a 4×4 meter region. The DSGPR length scale was set to $\lambda = 10$. The kinematic constraints were set to $u_{lb}/u_{ub} = \pm 1.0$ rad/sec, $v_{lb} = 0.1$ m/sec, $v_{ub} = 0.25$ m/sec, and $k_{ub} = 6.5$ rad/m. The rest of the parameters remained the same the simulation tests. Fig. 3 shows the paths and measurement locations of each agent as well as the estimated high- and low-intensity regions. To show that our path planner was able to generate feasible paths, Fig. 4 shows the velocity $v(t)$, turn rate $u(t)$, and curvature $k(t)$ of the planned trajectories. From this plot, we can see that our path planner complied with the kinematic feasibility constraints. A video of our hardware results can be found at https://www.youtube.com/watch?v=7jVc9QdYnJs.

Authorized licensed use limited to: Brigham Young University. Downloaded on April 25,2025 at 17:24:34 UTC from IEEE Xplore. Restrictions apply.

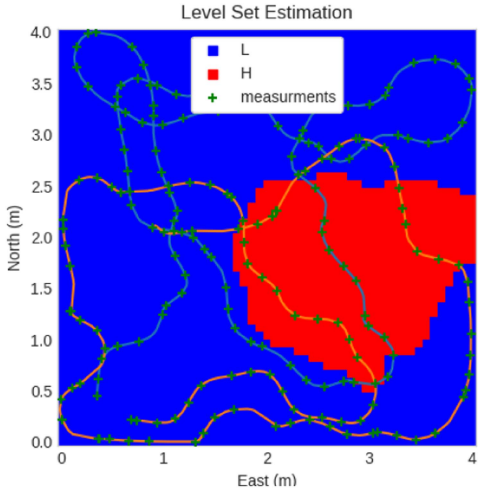

Fig. 3. Final LSE classifications are shown for two agents who took 100 measurements (green marks). The paths are light blue (Agent 0) and orange (Agent 1). The sets are red (high), blue (low), and white (unclassified).

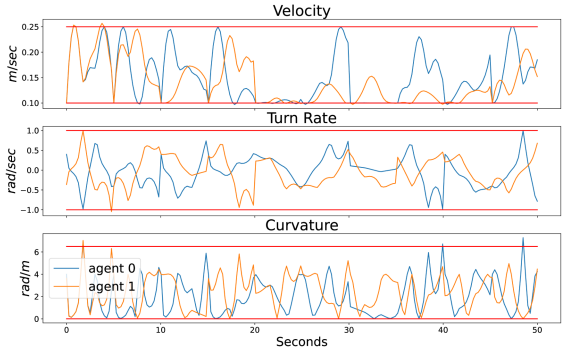

Fig. 4. Velocity, turn rate, and curvature for Agent 0 (blue) and Agent 1 (orange), with constraint bounds shown in red. These results show that the path planner generates kinematically feasible paths.

VII. Conclusion

In this letter, we presented a multi-agent informative path planner for level set estimation. We use B-splines to parameterize paths and optimize the paths based on a novel objective function for LSE. We use differential flatness to generate constraints that ensure kinematically feasible paths are planned. Decentralized multi-agent path planning is facilitated through the use of DSGPR and BCA optimization. We validated our method through simulation and hardware tests. We also show that our path planner provides kinematically feasible paths in hardware that allow agents to rapidly classify the operating area into into high- and low-concentrations sets using LSE. Future work includes implementing obstacle and inter-agent avoidance as well as extending our algorithm to work with more communication topologies. We anticipate that obstacle avoidance and inter-agent avoidance constraints could be implemented by removing obstacles from the region D and excluding the area around agents' paths from D during the optimization of other agents' paths.

References

[1] A. Gotovos, N. Casati, G. Hitz, and A. Krause, "Active learning for level set estimation," in *Proc. 23rd Int. Joint Conf. Artif. Intell.*, 2013, pp. 1344–1350.

[2] L. Bottarelli, M. Bicego, J. Blum, and A. Farinelli, "Orienteering-based informative path planning for environmental monitoring," *Eng. Appl. Artif. Intell.*, vol. 77, pp. 46–58, 2019.

[3] G. Hitz, E. Galceran, M.-È. Garneau, F. Pomerleau, and R. Siegwart, "Adaptive continuous-space informative path planning for online environmental monitoring," *J. Field Robot.*, vol. 34, no. 8, pp. 1427–1449, 2017.

[4] Y. Brouwer, A. Vale, and R. Ventura, "Informative path planner with exploration–exploitation trade-off for radiological surveys in non-convex scenarios," *Robot. Auton. Syst.*, vol. 136, 2021, Art. no. 103691.

[5] W. Luo, C. Nam, G. Kantor, and K. Sycara, "Distributed environmental modeling and adaptive sampling for multi-robot sensor coverage," in *Proc. 18th Int. Conf. Auton. Agents MultiAgent Syst.*, 2019, pp. 1488–1496.

[6] Y. Shi et al., "Adaptive informative sampling with environment partitioning for heterogeneous multi-robot systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 11718–11723.

[7] S. J. Wright, "Coordinate descent algorithms," *Math. Program.*, vol. 151, no. 1, pp. 3–34, 2015.

[8] T. Norton, G. Stagg, D. Ward, and C. K. Peterson, "Decentralized sparse gaussian process regression with event-triggered adaptive inducing points," *J. Intell. Robotic Syst.*, vol. 108, Aug. 2023, Art. no. 72.

[9] P. Panyakeow and M. Mesbahi, "Deconfliction algorithms for a pair of constant speed unmanned aerial vehicles," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 50, no. 1, pp. 456–476, Jan. 2014.

[10] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006.

[11] C. P. Tang, P. T. Miller, V. N. Krovi, J.-C. Ryu, and S. K. Agrawal, "Differential-flatness-based planning and control of a wheeled mobile manipulator–theory and experiment," *IEEE/ASME Trans. Mechatron.*, vol. 16, no. 4, pp. 768–773, Aug. 2011.

[12] D. Buccieri, D. Perritaz, P. Mullhaupt, Z.-P. Jiang, and D. Bonvin, "Velocity-scheduling control for a unicycle mobile robot: Theory and experiments," *IEEE Trans. Robot.*, vol. 25, no. 2, pp. 451–458, Apr. 2009.

[13] M. G. Cox, "The numerical evaluation of b-splines," *IMA J. Appl. Math.*, vol. 10, no. 2, pp. 134–149, 1972.

[14] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Math. Program.*, vol. 106, no. 1, pp. 25–57, 2006.

[15] J. R. Martins and A. Ning, *Engineering Design Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2021.

[16] K. Qin, "General matrix representations for B-splines," in *Proc. IEEE 6th Pacific Conf. Comput. Graph. Appl.*, 1998, pp. 37–43.

[17] W. Shi et al., "Multi-drone 3-D trajectory planning and scheduling in drone-assisted radio access networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 8145–8158, Aug. 2019.

[18] C. K. Peterson, D. W. Casbeer, S. G. Manyam, and S. Rasmussen, "Persistent intelligence, surveillance, and reconnaissance using multiple autonomous vehicles with asynchronous route updates," *IEEE Robot. Automat. Lett.*, vol. 5, no. 4, pp. 5550–5557, Oct. 2020.

[19] C. Guo, Z. Sun, Y. Chen, Y. Xie, S. Li, and H. Qian, "Trajectory tracking of unicycle-type robots with constraints," in *Proc. IEEE Int. Conf. Robot. Biomimetics*, 2018, pp. 1700–1705.

Authorized licensed use limited to: Brigham Young University. Downloaded on April 25,2025 at 17:24:34 UTC from IEEE Xplore. Restrictions apply.