Intermittent Deployment for Large-Scale Multi-Robot Forage Perception: Data Synthesis, Prediction, and Planning

Jun Liu, Murtaza Rangwala, Kulbir Singh Ahluwalia, Shayan Ghajar, Harnaik Dhami, Pratap Tokekar, *Member, IEEE*, Benjamin Tracy, and Ryan K. Williams, *Member, IEEE*

***Abstract*— Monitoring the health and vigor of grasslands is vital for informing management decisions to optimize rotational grazing in agriculture applications. To take advantage of forage resources and improve land productivity, we require knowledge of pastureland growth patterns that is simply unavailable at the state of the art. In this paper, we propose to deploy a team of robots to monitor the evolution of an unknown pastureland environment to fulfill the above goal. To monitor such an environment, which usually evolves slowly, we need to design a strategy for rapid assessment of the environment over large areas at a low cost. Thus, we propose an integrated pipeline comprising data synthesis, deep neural network training, and prediction along with a multi-robot deployment algorithm that monitors pasturelands *intermittently*. Specifically, using expert-informed agricultural data coupled with novel data synthesis in ROS Gazebo, we first propose a new neural network architecture to learn the spatiotemporal dynamics of the environment. Such predictions help us to understand pastureland growth patterns on *large scales* and make appropriate monitoring decisions for the future. Based on our predictions, we then design an intermittent multi-robot deployment policy for low-cost monitoring. Finally, we compare the proposed pipeline with other methods, from data synthesis to prediction and planning, to corroborate our pipeline's performance.**

***Note to Practitioners*—Pasturelands are an integral part of agricultural production in the United States. To take full advantage of the forage resource and avoid environmental degradation, pastureland must be managed optimally. This paper focuses on the question of how to deploy robot teams to sense and model physical processes over varying timescales. The goal of this work is to develop a new integrated pipeline for the long-term deployment of heterogeneous robot teams grounded in the problem of autonomous monitoring in precision grazing to improve land productivity. By using the proposed pipeline in grassland ecosystem management, we will have a better understanding of the physical environment while respecting energy budgets.**

***Index Terms*— Precision agriculture, intermittent deployment, planning, spatiotemporal prediction, deep learning.**

I. Introduction

GRASSLANDS provide many ecosystem services such as livestock production, wildlife habitat, water infiltration, and carbon sequestration [1]. Consequently, monitoring the health and vigor of grasslands is vital for informing management decisions to protect or optimize these ecosystem services [2]. Sward or canopy height data provide valuable insights into the productivity, habitat value, or maturity of grasslands. When sward height data are monitored over time, changes in sward height can indicate whether a grassland is deteriorating, maintaining its vigor, or becoming more productive. In agricultural systems, monitoring height data can inform decisions about the appropriate timing and intensity of grazing in order to meet economic and ecological goals.

Traditional methods of measuring aboveground height or aboveground biomass rely on labor-intensive methods [3]. For height, this necessitates measuring canopy height by hand using a meter stick or Robel pole [4]. Aboveground biomass measurements often consist of destructive harvest of forage. Samples are usually cut by hand from a quadrat or frame, bagged, then dried in a forced-air forage oven until reaching a constant weight, known as the dry matter. Depending on the size of the pasture or scope of the monitoring project, height and biomass sampling may involve dozens or hundreds of such samples to ensure that the collected data represents the entirety of the pasture or landscape being monitored.

Advancements in proximal sensing technologies can provide accurate measurements of height and biomass predictions faster and over larger areas than these labor-intensive traditional methods. However, regular field measurements from pastures through remote sensing methods such as Unmanned Aerial Vehicles (UAVs) are constrained by multiple factors such as limited spatial coverage and low frequency of UAV deployments. Moreover, adverse weather conditions and other

Manuscript received 4 August 2022; accepted 12 September 2022. Date of publication 18 October 2022; date of current version 5 January 2024. This article was recommended for publication by Associate Editor C. Park and Editor B. Vogel-Heuser upon evaluation of the reviewers' comments. This work was supported by the National Institute of Food and Agriculture under Grant 2018-67007-28380. *(Corresponding author: Jun Liu.)*

Jun Liu, Murtaza Rangwala, and Ryan K. Williams are with the Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061 USA (e-mail: junliu@vt.edu; murtazar@vt.edu; rywilli1@vt.edu).

Kulbir Singh Ahluwalia is with the Department of Computer Science, University of Illinois at Urbana–Champaign, Champaign, IL 61801 USA (e-mail: ksa5@illinois.edu).

Shayan Ghajar is with the Department of Crop and Soil Science, Oregon State University, Corvallis, OR 97331 USA (e-mail: ghajars@oregonstate.edu).

Harnaik Dhami and Pratap Tokekar are with the Department of Computer Science, University of Maryland, College Park, MD 20782 USA (e-mail: dhami@umd.edu; tokekar@umd.edu).

Benjamin Tracy is with the School of Plant and Environmental Sciences, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061 USA (e-mail: bftracy@vt.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TASE.2022.3211873.

Digital Object Identifier 10.1109/TASE.2022.3211873

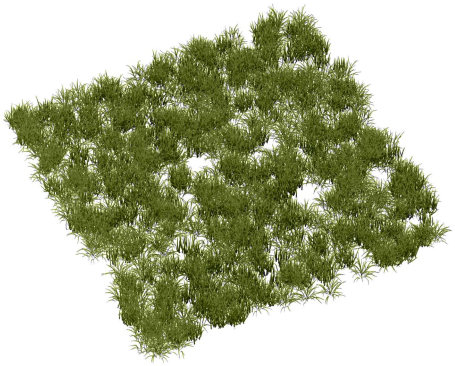


Fig. 1. A patch (2m × 2m) of the simulated pastureland with a density of 250 grass models per square meter.

resource constraints also contribute to limited deployment and consequently insufficient field measurements of the pasture required to plan grazing activities. In spite of these limitations, precision agriculture involving the use of UAVs to monitor the growth of crops is a promising approach to covering large areas in reasonable times. Analyzing point clouds obtained from the LiDAR attached to UAVs can help to determine the spatial distribution of plants and growth in different regions of the farm and consequently help the farmer focus their resources in regions where they are required the most. Developing prediction models for pastures that work within the limitation of UAV deployment schedules is a promising approach and helps alleviate resource-limited field measurements. With an intermittent deployment coupled with a tightly integrated prediction and planning model, this approach would generate maximum efficiency for livestock grazing and pasture recovery processes.

With the above motivation, this paper explores the use of autonomous robots to facilitate environmental monitoring for improving land productivity. At a high level, our proposed pipeline works as follows (Fig. 2). We first use historical data to synthesize a 3D dynamic field to simulate the spatiotemporal environmental process of the site that needs to be monitored (Section IV). For illustration, a small patch (2m × 2m) of the generated pasture is shown in Fig. 1, where the density is 250 grass models per square meter. We then use this data to train neural networks to learn the dynamics of this field (Section V). Then, an intermittent deployment policy is designed for multi-robot teams (UAVs in our case) using the future pasture height predictions while respecting system budgets (Section VI). After that, we simulate a pastureland environment in ROS Gazebo [5] to test the performance of the synthesized data (Section VII). Finally, we evaluate the performance of each aspect of our solution and compare it to competing methods (Section VIII).

In order to synthesize appropriate training data for high-quality predictions, historical data were generated using the expert-informed Agricultural Production Systems sIMulator (APSIM) Next Generation in Section IV. APSIM is designed to model long-term agricultural production in a variety of systems [6]. Using historical meteorological data from three sites in Iowa, APSIM simulated 30 years of tall fescue (Schedonorus arundinacea) pasture dry matter production for each site. Simulated pasture yield was then used to generate average pasture height data based on the equation reported by Schaefer and Lamb [7] describing the relationship between pasture green dry matter and LiDAR-measured pasture height. Based on the historical average pasture height data, we then use a Gaussian mixture model (GMM) to simulate the dynamics of this field over the desired monitoring horizon in Section IV.

In this paper, we aim to propose an entire pipeline for a precision agriculture application with the help of robots. The pipeline includes multiple parts: prediction, estimation, planning, evaluation, etc. Therefore, before conducting a real-world validation, we need to validate the effectiveness of the integration of different parts, as performing a real-world verification is a long process. Therefore, this paper will use a high-fidelity world that simulates the growth of a pastureland environment with the help of high-fidelity grass models and historical data.

Even with a sufficient training data set based on expert-informed historical data, developing an effective prediction model for estimating pasture growth is challenging due to changes incurred by the climate and the spatiotemporal characteristics of the growth. Previous studies for forecasting spatiotemporal dependencies have been based on conventional approaches. These methods require a complex and meticulous simulation of the physical environment for a particular application. Instead, learning-based models provide increased flexibility in tackling the difficult spatiotemporal sequence prediction problem that large-scale forage monitoring poses. To address this problem from a deep learning perspective, we use our proposed long short-term memory (LSTM) and convolutional neural network (CNN) based architecture [8] to model the pasture growth forecasting problem in Section V.

Finally, with high-quality pasture growth predictions, a multi-robot deployment policy can be designed to guide future field measurements, as detailed in Section VI. Our proposed deployment policy aims to maximize the sum of collected information (uncertainty) from the environment while considering the system's waiting penalties and energy constraints. In general, we formulate this deployment problem as a submodular maximization problem with matroid constraints [9]. The energy constraints will be formulated as matroid constraints, while the collected information and the waiting penalties will be part of our objective function. Additionally, as the deployment policies gather field measurements, the prediction model can be updated iteratively to generate more accurate estimates of future pasture growth. Through an optimized prediction and deployment model, we show that UAV deployments for pasture monitoring can be scheduled based on the prediction model instead of at regular intervals, effectively reducing the required number of deployments. This *need-based intelligent* deployment policy substantially reduces the cost of gathering field measurements and effectively allows resource-constrained enterprises to manage pasture grazing more effectively.

Beyond the forage monitoring problem, the proposed pipeline can be mapped to other large-scale monitoring problems for sufficiently slow spatiotemporal processes where intermittent deployment is reasonable. That is, the data syn-

thesis – neural network training/prediction – intermittent multi-robot deployment pipeline is quite general, given access to expert-informed data as a starting point. For example, in ocean monitoring applications [10], [11], we can use the proposed pipeline to generate and refine deployment policies given the availability of high-quality models of aquatic processes.

Contributions: In summary, the contributions of this work are as follows:

- We propose an integrated pipeline[1] to simulate and solve an important problem in the precision agricultural space, forage monitoring, including data synthesis, prediction, planning, and perception (Section III).
- We demonstrate how to exploit historical agricultural data to synthesize a pastureland environment in a manner that accommodates both neural network training and rapid prototyping in a simulation environment (Section IV).
- We demonstrate a scalable spatiotemporal learning architecture that can be used to integrate with an intermittent multi-robot planning strategy (Section V).
- We introduce a novel intermittent deployment policy that integrates neural network-based predictions while considering budgets to deploy robots for autonomous environmental monitoring (Section VI).
- We have built (and made available) a high-fidelity pastureland simulation environment in ROS Gazebo, allowing for rapid prototyping of pasture monitoring with LiDAR point clouds (Section VII).

A preliminary portion of this work appeared in [8] and [12]. In this work, we focus on building a general pipeline for precision agriculture applications, which first focuses on utilizing the proposed deep learning networks [8] to make predictions and then on how to utilize those predictions to make robotic deployments with different budget constraints.

II. Related Work

Predictive modeling of agricultural systems can provide helpful information on their long-term resilience and productivity. Predictive models have been developed for many crops and regions, tested extensively, and refined for increased predictive accuracy for commodities such as corn, soybeans, wheat, and rice, to name a few. As model accuracy increases, it can provide insights into crop responses to abiotic stressors such as climate change [13], [14], or identify optimal crop rotations between species over a period of years [15]. However, agricultural modeling of forage systems presents unique challenges compared to row cropping systems, as forage systems are often perennial rather than annual, the forage crop can be "harvested" multiple times per year by grazing livestock or cutting for hay, and pastures are typically multispecies ecosystems rather than monospecific crops with homogeneous phenotypes and physiology [16]. Several forage modeling programs have been developed despite the aforementioned challenges. These include—but are not limited to—the Simulation of Production and Utilization on Rangelands (SPUR) first developed on rangelands in the 1980s [17]; the GRAzing SImulation Model (GRASIM) developed in the late 1990s [18]; the Dairy Forage System Model (DAFOSYM) developed in the late 1980s and updated into the Integrated Farm System Model (IFSM) [19]; and the Agricultural Production Systems sIMulator (APSIM) Next Generation, which can model both row-cropping systems and pastures [20]. We select APSIM as the optimal modeling program for our research, as the program has been used previously for pasture modeling in a variety of contexts [21], [22]; provides daily timestep outputs, which fulfilled our need for fine-grained temporal data, and the program's modular nature allows for rapid customization of input parameters [14], [23].

The problem of predicting evolving pastureland over time can be tackled with conventional methods such as Gaussian processes (GPs) [24], which focus on stochastic Gaussian processes to model the regression for different pasture heights or observations. This method is also known as Kriging [24]. GPs are non-parametric methods by defining the internal relations between observations [25]. Similarly, Gaussian Markov Random Fields (GMRFs) are often used to model spatial environmental fields [24]. These conventional methods are suitable for problems with significant prior knowledge of the environment that needs to be monitored. The historical data used in this paper can give us a general trend of the average height change of the field. However, as we must make predictions on large scales, this prior is insufficient for us to generate a reasonable conventional model for the entire pastureland environment (not to mention the issues with computational scaling). Moreover, as the growth patterns may differ across the field, conventional models are not as flexible for modeling the heterogeneity of growth patterns spatially and temporally. We, therefore, sought to apply a neural network-based method to tackle this prediction problem when we have a large dataset for modeling the heterogeneity of the environment.

Specifically, we implement a framework that integrates a neural network-based encoder-decoder architecture to learn the historical data's underlying patterns over time for future predictions. The problem of predicting future pasture heights is analogous to video frame prediction [26], with the key challenge in our case lying in predicting the growth of pasture surfaces. In the deep learning-based prediction domain, sequence-to-sequence problems were originally introduced through recurrent neural networks (RNN) and long short-term memory (LSTM) models and provide a baseline for solving temporal forecasting problems that we encounter in this work [27], [28], [29]. To incorporate spatial features, prior works have generally used multiple architectures to consider spatial and temporal features separately by combining the autoencoder or a generative adversarial network (GAN) model with an RNN model [26], [30], [31]. Convolutional LSTMs (ConvLSTMs) [32] have been successfully used for spatiotemporal predictive learning and were originally proposed for precipitation nowcasting over radar images. Recent works have used the architecture for learning frame representations [33]. Motion-content network (MCNet) [34] uses an additional LSTM apart from the image encoder to

[1]https://github.com/precision-grazing/project

model the motion dynamics. The outputs of both the encoders are combined and fed to the decoder to predict frames. Recently, [35], [36] proposed ST-LSTM to learn structural information for spatiotemporal sequences and a new model structure PredRNN and PredRNN++ to explicitly decouple memory cells and improve the cross-layer interaction of memory states across different LSTMs. These methods help improve stronger spatial correlation and short-term dynamics for powerful modeling and prediction capabilities. We utilize the recent developments in ConvLSTM applications and propose a novel prediction architecture that is particularly effective in predicting the rapidly changing dynamics of the pasture over long horizons. We achieve this through the use of recurrent encoder-decoder networks based on ConvLSTMs over different resolutions of the pasture to effectively capture different features and dependencies of the pasture dynamics. Moreover, by using appropriate processing described in Section V over raw pasture data, our pasture terrain forecasting technique can scale to any terrain size.

Standard deep learning-based models do not capture model uncertainty. Unlike regression problems, where a model can output a predictive probability, we need to extrapolate prediction uncertainty from training data as we deal with a sequence-to-sequence prediction problem. Bayesian probability theory has been used in deep learning to deal with uncertainty [37], [38], [39], [40], where the weights of the neural network are defined as distributions. Bayesian neural networks (BNN) are more robust to over-fitting. However, they add significant computation complexity. Sampling-based and stochastic variational inference methods have been used to approximate Bayesian neural networks [41], [42], [43]. Similar to BNN, these methods incur high computational costs without additional benefits to improving accuracy. An alternative approach [44] with minimal computational and model complexity in deep learning models was proposed through the use of Dropouts [45]. The authors show that any neural network with arbitrary depth and non-linearities, modeled with dropouts behind every layer, is equivalent to the approximation of a probabilistic deep Gaussian process [46]. In our previous work [8], we integrate this concept of uncertainty estimation from dropout over our architecture to provide the necessary uncertainty maps for multi-robot monitoring from a typical machine learning perspective. Whereas in this work, our focus is on integrating a combinatorial optimization-based planner with proposed deep learning-based predictions [8] and performing a deep evaluation of the entire pipeline with realistic simulations. From the multi-robot deployment perspective, to estimate the evolving processes of pastureland environments more efficiently, we cannot deploy robots (UAVs) to collect observations frequently because it is not energy efficient. Those types of observations can be point clouds, heightmaps, sonar data, etc. Meanwhile, the deployment strategy should be generated based on environmental information instead of being artificially defined. This intermittent idea can also be found in other robotics applications. In [47], the robots in a team are designed to communicate intermittently while working together to explore an environment. In [48], the authors use time windows to model the availability of robots at different times in task allocation applications. Therefore, robots are not required to work continuously. The idea of intermittence is contrary to that of persistence, where robots are required to work continuously to fulfill different tasks [49]. In [50], the authors proposed a deep learning-based method for combining environmental prediction and path planning, where spatial path planning is the primary concern in the paper. In [51], the authors investigated a sampling-based path planning method for variance reduction. Similarly, in [52], the author used an adaptive sampling strategy for reducing entropy generated from GP modeling. In [53], a GMRF-based method was proposed for spatial predictions.

Since the monitored environment evolves spatially and temporally, we need to utilize the predicted environmental information to make a spatiotemporal deployment plan. In [54], we use a partially observable Markov decision process (POMDP) to model the dynamics of an environmental process. Then, a submodular objective function is applied to model the false alarm and delay cost. This method works only when the process models are known and can be modeled as POMDPs. In [12] and [55], we use non-parametric Gaussian processes [24] to model the dynamics of a monitored environmental process and then use mutual information as a metric to guide our deployments. Meanwhile, matroids are used to model the budget constants. Generally speaking, matroids [9], [56] can be used to model the independence in constraints, which can be found in many robotics applications, e.g., task allocation [57], deployment planning [58], [59], probabilistic security in multi-robot systems [60], etc. We will also use this tool to model the independence of different constraints in the deployment strategy in this work. In general, the environmental modeling methods used in our previous works are conventional GP-based methods, where a prior of the entire environment is needed to utilize historical data. While in this paper, the environment modeling method is a data-driven approach, which builds the dynamics of the environment through a historical dataset where an exact process model is not required. Moreover, the environment modeling and the deployment policy generation are highly connected in this paper. Finally, as the deployment policies gather more measurements, the proposed data-driven prediction model can be updated accordingly for better future predictions.

III. Preliminaries and Problem Formulation

A. Preliminaries

A core aspect of this paper is to optimize multi-robot deployment plans by utilizing historical agricultural data. We propose to tackle this problem from a combinatorial optimization perspective. To this end, we begin by reviewing the concepts related to our objective function and constraint modeling.

A set function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ is a function that maps any set $\mathcal{A} \subseteq \mathcal{V}$ into $\mathbb{R}$, where $\mathcal{V}$ is the finite discrete ground set.

Definition 1 ([9]): A set function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ is

- normalized, if $f(\emptyset) = 0$;
- monotone non-decreasing, if $f(\mathcal{A}) \leq f(\mathcal{B})$ when $\mathcal{A}, \mathcal{B} \subseteq \mathcal{V}$, and $\mathcal{A} \subseteq \mathcal{B}$;
- submodular, if $f(\mathcal{A} \cup \{v\}) - f(\mathcal{A}) \geq f(\mathcal{B} \cup \{v\}) - f(\mathcal{B})$ when $\mathcal{A}, \mathcal{B} \subseteq \mathcal{V}$, $\mathcal{A} \subseteq \mathcal{B}$, and $v \in \mathcal{V} \setminus \mathcal{B}$.

The property of *submodularity* is often described as having a diminishing returns property (as the above definition suggests), making it a natural option for modeling objective functions in robotics (e.g., sensor placement [61], set coverage [9], task allocation [62]). It is also convenient to work with the *marginal return* when an element v is added to $\mathcal{A}$, which is defined as $f(\{v\} \mid \mathcal{A}) \triangleq f(\mathcal{A} \cup \{v\}) - f(\mathcal{A})$. The submodularity level of a set function can be measured using curvature, a property of the set function itself. Curvature can be defined as follows.

Definition 2 ([63]): Let $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ be a monotone non-decreasing submodular function, we define the curvature of $f(\cdot)$ as

$$c_f \triangleq 1 - \min_{a \in \mathcal{A}} \frac{f(\mathcal{V}) - f(\mathcal{V} \setminus \{a\})}{f(\{a\})},$$

where $\mathcal{A} = \{a \in \mathcal{V} \mid f(\{a\}) > 0\}$, and $\mathcal{V}$ is the ground set.

It holds that $0 \leq c_f \leq 1$. If $c_f = 0$, then $f(\cdot)$ is a modular function and $f(\mathcal{A} \cup \{v\}) - f(\mathcal{A}) = f(\{v\}), \forall \mathcal{A} \subseteq \mathcal{V}, v \in \mathcal{V} \setminus \mathcal{A}$ when $\mathcal{V}$ is the ground set. If $c_f = 1$, then $f(\mathcal{A} \cup \{v\}) - f(\mathcal{A}) = 0$, where $\mathcal{A} \subseteq \mathcal{V}$ and $v \in \mathcal{V} \setminus \mathcal{A}$. This means adding v has no contribution to the function value $f(\cdot)$ if $\mathcal{A}$ is selected.

Next, we introduce the related concept of constraint modeling. Matroids generalize the idea of independence in set systems. Meanwhile, efficient sub-optimal solutions can be found when constraints are modeled by matroids [64].

Definition 3 ([56]): A matroid $\mathcal{M} = (\mathcal{V}, \mathcal{I})$ is a set system that contains a finite ground set $\mathcal{V}$ and a collection $\mathcal{I}$ of subsets of $\mathcal{V}$ with the following properties:

1) $\emptyset \in \mathcal{I}$;
2) If $\mathcal{A} \subseteq \mathcal{B} \in \mathcal{I}$, then $\mathcal{A} \in \mathcal{I}$;
3) If $\mathcal{A}, \mathcal{B} \in \mathcal{I}$ and $|\mathcal{B}| < |\mathcal{A}|$, there exists a $v \in \mathcal{A} \setminus \mathcal{B}$ such that $\mathcal{B} \cup \{v\} \in \mathcal{I}$.

The intersection of L matroids can be written as $\mathcal{M} = (\mathcal{V}, \mathcal{I})$, where $\mathcal{M}_i = (\mathcal{V}, \mathcal{I}_i)$ is the ith matroid and $\mathcal{I} = \bigcap_{i=1}^{L} \mathcal{I}_i$. The cardinality of this matroid intersection constraint is $|\mathcal{M}| = L$. A set belonging to a matroid means that this set should satisfy the matroid conditions. Examples of matroid constraint modeling in robotics can be found in task allocation [57], orienteering [65], action selection [66], etc.

B. Problem Formulation

Consider a spatiotemporal forage process evolving in a 2D pasture environment (which we model rigorously in the sequel). In simple terms, we are interested in determining forage height for any location in the pasture environment over a long time horizon by deploying multi-robot teams. To fulfill this goal, we divide our efforts into two tasks: forage process prediction and multi-robot planning. In the following, we outline the fundamentals of these tasks and conclude with a concrete problem statement.

1) Prediction: Assume that we are given historical 2D heightmaps $\mathbf{X}_t \in \mathbb{R}^{M \times N}$ of a pasture field at different times, where t is the associated time index, and M, N are the width and the length of the pasture heightmap, which correspond to the discretization resolution. This resolution will be used for both prediction and planning problems. Denote by $\mathcal{X} = \{\mathbf{X}_t \in \mathbb{R}^{M \times N} \mid t \in \mathcal{T}_x\}$ the historical dataset containing all the historical measurements, where $\mathcal{T}_x$ contains all the time indexes associated with each $\mathbf{X}_t \in \mathcal{X}$. The extracted height of the pasture in location (x, y) with respect to $\mathbf{X}_t$ is denoted by $\mathbf{X}_t(x, y)$. Our first goal is to train a neural network to predict future pasture heights $\bar{\mathbf{Y}}_t$ for the prediction horizon $\mathcal{T}_y$ using the historical dataset $\mathcal{X}$. This prediction process is modeled as

$$\left(\bar{\mathbf{Y}}_t, \bar{\sigma}_t^2\right) \leftarrow \Theta(\mathcal{X}, \mathbf{W}), \quad \forall t \in \mathcal{T}_y, \tag{1}$$

where Θ is our neural network model, $\mathbf{W}$ is the set of parameters of the model, $\bar{\mathbf{Y}}_t \in \mathbb{R}^{M \times N}$ is the predicted heightmap at t, and $\bar{\sigma}_t^2 \in \mathbb{R}^{M \times N}$ is the corresponding variance at t. Also, we denote by $\bar{\mathbf{Y}} = \left\{\bar{\mathbf{Y}}_t \in \mathbb{R}^{M \times N} \mid \forall t \in \mathcal{T}_y\right\}$ the prediction set that contains all of the predicted heightmaps for the prediction horizon $\mathcal{T}_y$, and denote by $\bar{\Sigma} = \left\{\bar{\sigma}_t^2 \in \mathbb{R}^{M \times N} \mid \forall t \in \mathcal{T}_y\right\}$ the corresponding variance set. The details will be specified in Section V.

2) Planning: Based on the predicted heightmaps $\bar{\mathbf{Y}}$ and variance maps $\bar{\Sigma}$, we seek a multi-robot deployment strategy to collect data, reinforce our predictions, and ultimately make better pastureland management decisions. To determine a deployment strategy, we first need to build the ground set $\mathcal{V}$, which contains all possible deployment decisions for robots over space and time. Consider a team of robots available to deploy over the time horizon $\mathcal{T}_y$, the prediction horizon. The family of all available locations is denoted by $\mathcal{P}$. Thus, we have $(x, y) \in \mathcal{P}$, and $|\mathcal{P}| = M \cdot N$ is the cardinality of the deployable locations. We also denote by $\mathcal{R}$ the set of indices of all robots. Each robot $r \in \mathcal{R}$ may have a different sensing ability, i.e., sensing noise. Therefore, the ground set $\mathcal{V}$ at time t, containing all available deployment choices, is as follows:

$$\mathcal{V}_t = \{(x, y, r, t) : \quad \forall (x, y) \in \mathcal{P}, r \in \mathcal{R}\}.$$

We interpret (x, y, r, t) as "location (x, y) is sensed by robot r at time t". To simplify the notation, we will use a 4-tuple $v = (x, y, r, t)$ to denote the deployment decision factor in the following. Finally, the ground set of the deployment problem is $\mathcal{V} = \bigcup_{t \in \mathcal{T}_y} \mathcal{V}_t$. The cardinality of the ground set is $|\mathcal{V}| = |\mathcal{P}| \cdot |\mathcal{R}| \cdot |\mathcal{T}_y|$. The associated predicted variance with respect to v at t is represented by $\bar{\sigma}_t^2(v), \forall v \in \mathcal{V}$. Since $\sigma_t^2(v)$ is the predicted variance map at time t for v, we can use $\bar{\sigma}_t^2(x, y) \in \mathbb{R}$ to denote the predicted variance that corresponds to location $(x, y) \in \mathcal{P}$ at time $t \in \mathcal{T}_y$. We want to determine a deployment policy set $\mathcal{S} \in \mathcal{V}$ to maximize the information we can get from the environment while respecting the system budgets. To this end, we denote by $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ the objective function and denote by $\mathcal{M}$ a set of sets defining the system's admissible deployment policies. The details will be specified in Section VI.

3) Problem Statement: With the basics of prediction and planning outlined, we now formalize the problem we solve in this paper.

Problem 1 (Intermittent Deployment): Consider a historical set $\mathcal{X} = \left\{\mathbf{X}_t \in \mathbb{R}^{M \times N} \mid \forall t \in \mathcal{T}_x\right\}$, containing heightmaps of a discretized $M \times N$ pasture over a time horizon $\mathcal{T}_x$. Let $\mathcal{V}$ be the ground set containing all possible multi-robot deployment factors $v = (x, y, r, t)$ over a time horizon $\mathcal{T}_y$. Assume further the existence of a predicted variance set $\bar{\Sigma} = \left\{\bar{\sigma}_t^2 \in \mathbb{R}^{M \times N} \mid \forall t \in \mathcal{T}_y\right\}$. The intermittent deployment problem

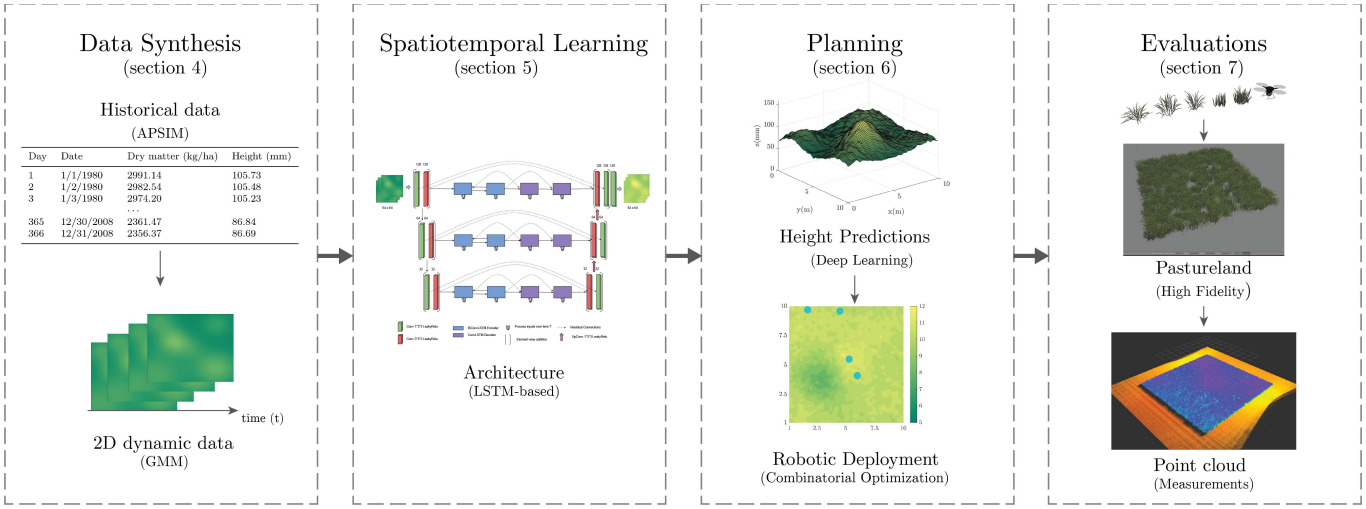


Fig. 2. A diagram of our overarching solution. Data synthesis is first used to simulate historical pasture data. Then spatiotemporal learning is used for neural network training and prediction based on historical data. Next, the planning aspect is used to make multi-robot deployment decisions for collecting new data. Finally, on the far right, we illustrate the high-fidelity simulation we have built for evaluating our pipeline.

is to maximize a set function $f(\cdot)$ by selecting appropriate deployment factors while respecting budget constraints. That is,

$$\begin{aligned} \underset{\mathcal{S}\subseteq\mathcal{V}}{\text{maximize}} \quad & f(\mathcal{S}) \\ \text{subject to} \quad & \mathcal{S} \in \mathcal{M}. \end{aligned}$$

By selecting the set $\mathcal{S}$ that maximizes the objective function, we can determine all deployment factors containing deployment locations and times to conduct deployments while maintaining efficiency.

Remark: This work focuses on predictions and planning perspectives, especially in spatiotemporal deployment location selection under limited energy conditions. We should note that a path planning problem is a subsequent problem of the proposed framework, and any of them can be integrated into the framework. Therefore, we focus on prediction and planning to set up a cornerstone for the subsequent problems.

The diagram of our proposed solution to the above problem is shown in Fig. 2. Our pipeline can be divided into the following aspects, which we detail in the sequel: data synthesis (Section IV), deep learning prediction (Section V), planning (Section VI), and evaluations (Section VII).

IV. Large-Scale Pasture Environment Synthesis

In this section, we focus on pasture environment synthesis. This process will be divided into two parts. First, we illustrate how to synthesize historical average pasture height data (Section IV-A). Based on this data, we then introduce how to utilize this data to create a dynamic pasture environment (Section IV-B).

A. Historical Data Preparation

'Historic' pasture data were generated using APSIM Next Generation's publicly available meteorological, soils, and pasture species modules. We selected three sites in Iowa due to the availability of meteorological data for each in APSIM's Met module as a result of prior research [67]. Meteorological data spanned 1979 to 2013 and included solar radiation (MJ/m^2), rain and snowfall, minimum and maximum temperature, atmospheric pressure, and day length. Soils selected in the module were fine-loamy, mixed, superactive, mesic Hapludolls common in Iowa, also available in APSIM's modules [67]. APSIM's tall fescue AgPasture module was used for the forage species [14]. Tall fescue was set to 1m rooting depth and initial belowground and aboveground biomass of 1000 kg/ha and 3000 kg/ha, respectively. The SoilOM module, which simulates soil organic matter processes, was set to 1000 kg/ha initial surface residue. Fertilizer application was simulated at 84 kg N/ha on January 1 and another 84 kg N/ha on August 15 each year in the form of nitrate (NO_3-N). The resulting simulated pasture yield was then used to generate average pasture height data using an equation reported by Schaefer and Lamb [7] describing the relationship between LiDAR-measured pasture height and pasture green dry matter, i.e., green aboveground biomass. In this paper, we denote by $\mathbf{h} \in \mathbb{R}^{T_x}$ the historical data, where $T_x = |\mathcal{T}_x|$ is the length of the historical dataset horizon $\mathcal{T}_x$, which is also defined in the problem formulation in Section III-B. We also denote by $\mathbf{h}_t \in \mathbb{R}$ the average pasture height data at time t in the historical dataset. Further information on the above APSIM modules' functions and processes may be found in the work of Li and colleagues [14].

B. Pasture Environment Data Synthesis

The generated average pasture height data is temporal data. When making predictions for different locations and developing spatiotemporal deployment strategies, we need both spatial and temporal historical data to know the growth patterns in different places of the pasture field. Therefore, this section focuses on synthesizing spatiotemporal data from the historical average pasture height data. In general, this process fits a spatiotemporal process to the temporal aspects of the historical data.

In this work, we use a dynamic Gaussian mixture model (GMM), a combination of Gaussian distributions, to simulate

the pasture evolution using historical data. The GMM used in this work is a discrete model as we need to make predictions for different days and make corresponding deployment decisions for different times and locations. We should note that there are many other ways to simulate a spatiotemporal process by using temporal data only, and we select GMM for the following reasons. GMM is a common tractable process representation, which allows us to adjust the model to match the actual field-evolving process. From expert inputs or manual field measurements, we can choose reasonable spatial parameters for the GMM to complement the temporal parameters coming from historical data, so the spatiotemporal model represents the pasture evolving process.

In general, we first use B components to build a dynamic GMM and then use the historical data $\mathbf{h}$ to adjust the generated model to match the historical information. The dynamic GMM is modeled as

$$\mathbf{G}_t(x, y) = \sum_{i=1}^{B} w_i(t) b_i(x, y) = \mathbf{w}(t)^\top \mathbf{b}(x, y), \tag{2}$$

where $(x, y) \in \mathbb{R}^2$ is a 2D location from the location set $\mathcal{P}$, basis $b_i(x, y) \in \mathbb{R}$ is the output of ith basis function in the location (x, y), B is the number of basis functions, and $\mathbf{G}_t(x, y)$ is the output of the GMM at time t and location (x, y). The weight $w_i(t) \in \mathbb{R}$ is associated with $b_i(x, y)$, where $t \in \mathcal{T}_x$. The ith basis function $b_i(x, y)$ is a Gaussian kernel function. That is,

$$b_i(x, y) = \exp\left[-\frac{\left[(x, y) - (b_{xi}, b_{yi})\right]^2}{2c_i^2} \right], \tag{3}$$

where $(b_{xi}, b_{yi}) \in \mathbb{R}^2$ is the 2D position of ith basis function $b_i(x, y)$, and c_i is the corresponding length-scale. Also, $\mathbf{w}(t) \in \mathbb{R}^B$ is the stacked weights at time t and $\mathbf{b}(x, y) \in \mathbb{R}^B$ is the stacked basis functions for the location (x, y). One way to generate a dynamic 3D smooth surface to simulate pasture is to change the weights of different basis functions smoothly at different times. To achieve this goal, we use a 1D Gaussian process [68] to model the change of each weight at different times for the entire time horizon $\mathcal{T}_x$. After this step, a dynamic GMM is fully built.

Next, we need to tune the offset between the generated surface and the historical data to make them match with each other. Specifically, we need to adjust the generated GMM surfaces at different times using the historical average height data. That is:

$$\mathbf{X}_t(x, y) = \mathbf{G}_t(x, y) + \mathbf{h}_t - \bar{\mathbf{G}}_t,$$

where $\mathbf{h}_t \in \mathbb{R}$ is the historical average height at time t, $\mathbf{G}_t(x, y)$ is the generated GMM data for location (x, y) at time t, and $\bar{\mathbf{G}}_t \in \mathbb{R}$ is the simulated average height of the field at that time, which is calculated by using $\bar{\mathbf{G}}_t = |\mathcal{P}|^{-1} \sum_{(x,y)\in\mathcal{P}} \mathbf{G}_t(x, y)$. The data $\mathbf{G}_t(x, y)$ is generated without incorporating our historical temporal data as we only focus on generating the dynamics of the field. To calculate the difference between our generated data and the actual historical temporal data, we can calculate the difference as $\mathbf{h}_t - \bar{\mathbf{G}}_t$, where $\mathbf{h}_t$ is the historical data at time t and $\bar{\mathbf{G}}_t$ is the average height of the generated 2D field. This difference is the offset between the average height of the generated surface and the historical average height. Then, based on this difference, we can tune the generated surface using this offset. After this series of operations, the spatiotemporal dataset $\mathcal{X} = \{\mathbf{X}_t \in \mathbb{R}^{M\times N} \mid t \in \mathcal{T}_x\}$ is fully constructed and can be used in the later learning and predictions.

To summarize the data synthesis process, we first synthesize the historical average data $\mathbf{h}$. Then, we build a dynamic GMM denoted by $\mathbf{G}$, and adjust this model to make the temporal property of $\mathbf{G}$ match the historical average data $\mathbf{h}$. Finally, the adjusted dataset is denoted by $\mathcal{X}$ that will be used in learning and predictions (below).

V. Spatiotemporal Learning and Prediction

In this section, we will be first focusing on learning the dynamics of the generated field using the generated dataset $\mathcal{X} = \{\mathbf{X}_t \in \mathbb{R}^{M\times N} \mid t \in \mathcal{T}_x\}$. Note that pre-processing will be applied before feeding $\mathcal{X}$ into our training networks. Then, we will use the learned dynamics to make predictions of the future field, which will serve as inputs for the deployment strategy planning (Section VI).

In general, our spatiotemporal learning network is modeled as

$$\left(\bar{\mathbf{Y}}_t, \bar{\sigma}_t^2\right) \leftarrow \Theta(\mathcal{X}, \mathbf{W}), \quad \forall t \in \mathcal{T}_y,$$

where Θ is our neural network model, $\mathbf{W}$ is the set of parameters of the model, $\bar{\mathbf{Y}}_t \in \mathbb{R}^{M\times N}$ is the predicted heightmap at t, and $\bar{\sigma}_t^2 \in \mathbb{R}^{M\times N}$ is the corresponding variance at t. Next, we give the details of our mean and variance predictions.

Since the prediction model will take as input a sequence of heightmaps to predict multi-step long-horizon future pasture heights, we need to reorganize $\mathbf{X}_t \in \mathcal{X}, \forall t \in \mathcal{T}_x$ to satisfy this requirement. Meanwhile, we need pre-processing to enable our proposed model to tackle different pasture sizes. We aim to reorganize all $\mathbf{X}_t \in \mathcal{X}, \forall t \in \mathcal{T}_x$ to form multiple training sequences. We define the ith training sequence as $\mathcal{X}_i = \left\{\mathbf{X}_t \in \mathbb{R}^{M\times N} \mid \forall t \in \mathcal{T}_i\right\}$ with $\mathcal{T}_i = \{i, i+\delta, \ldots, i+\alpha\delta\}$, where δ is the number of intervals before each input observation in the training sequence, α is the number of observations in $\mathcal{X}_i$, and $\mathcal{T}_i$ contains all the time index associated with every $\mathbf{X}_t \in \mathcal{X}_i$. Note that $\mathcal{T}_i \subseteq \mathcal{T}_x$. The complementary training label (ground truth) for an input sequence $\mathcal{X}_i$ is then defined as $\mathbf{Y}_i = \left\{\mathbf{Y}_t \in \mathbb{R}^{M\times N} \mid \forall t \in \mathcal{T}_y\right\}$, where $\mathcal{T}_y = \{i+(\alpha+1)\delta, i+(\alpha+2)\delta, \ldots, i+(2\alpha+1)\delta\}$ contains the expected prediction step at δ intervals. Therefore, $\alpha = |\mathcal{T}_i| = |\mathcal{T}_y|$. The effective length (horizon) of the input and output sequences are then calculated as $L = \delta \cdot \alpha$. Varying the number of strides δ in the sequences allows our prediction model to work with both short-term and long-term horizons. For example, given an input sequence of $\alpha = 15$ and a stride of $\delta = 4$, we have an effective observation length of 60 days. Finally, the neural network outputs a prediction sequence $\bar{\mathbf{Y}}_i = \{\bar{\mathbf{Y}}_t \in \mathbb{R}^{M\times N} \mid \forall t \in \mathcal{T}_y\}$ for each input sequence $\mathcal{X}_i$, where $\bar{\mathbf{Y}}_t$ is the prediction with respect to the ground truth $\mathbf{Y}_t$ at t.

Remark: For the scope of this paper, we train our networks using the same input and output observation interval δ and

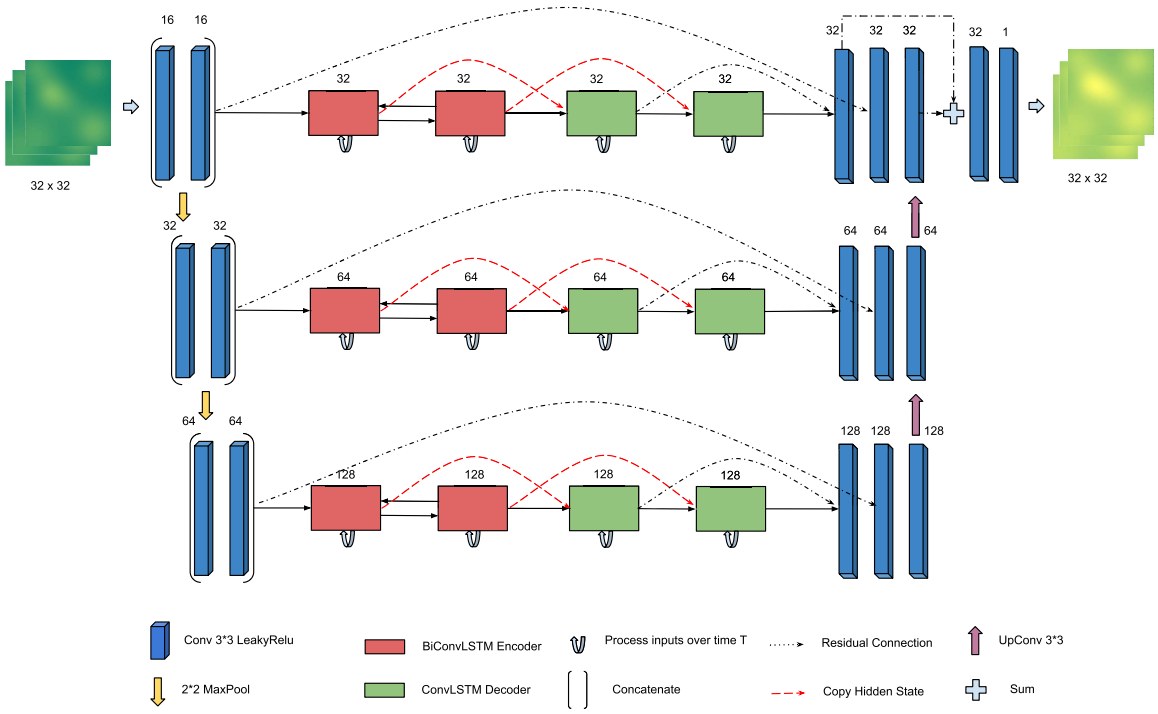


Fig. 3. Prediction neural network architecture is defined as an encoder-decoder. Initial feature maps across different resolutions are generated using Conv2D encoders and then are iteratively fed to the ConvLSTM encoders to encode the input observations. The ConvLSTM decoders then recursively generate representations for each interval in the output. These recurrent hidden representations across resolutions are then merged to generate the final prediction sequence. Residual connections are used to ease the regeneration of outputs from the generated hidden representations.

the same number of observation/measurements α. That is, $\alpha = |\mathcal{T}_x| = |\mathcal{T}_y|$. However, it is to be noted that the proposed network can be trained on varying sequence lengths without changing the architectural design. This is due to the inherent flexibility of an encoder-decoder design.

The high-level idea of the mean predictions is as follows. For each time t, we will use the averaged prediction $\bar{\mathbf{Y}}_t$ from several Monte Carlo (MC) predictions $\hat{\mathbf{Y}}_t^{(k)} \in \mathbb{R}^{M\times N}$ as the final predicted heightmap at t, where $\hat{\mathbf{Y}}_t^{(k)}$ is the kth MC prediction at time t. That is, $\hat{\mathbf{Y}}_t^{(k)} \leftarrow \Theta(\mathcal{X}, \mathbf{W}), \forall t \in \mathcal{T}_y$. The averaged mean prediction is given by

$$\bar{\mathbf{Y}}_t = \mathbb{E}_{\Pr(\hat{\mathbf{Y}}_t|\mathcal{X})}(\hat{\mathbf{Y}}_t) \approx \frac{1}{K}\sum_{i=1}^{K} \hat{\mathbf{Y}}_t^{(k)}, \quad \forall t \in \mathcal{T}_y. \tag{4}$$

Each MC prediction is a realization of the proposed network by using a different setting, which will be specified later. The number of sampled predictions is K for each heightmap prediction $\bar{\mathbf{Y}}_t, \forall t \in \mathcal{T}_y$.

Our network architecture is shown in Fig. 3. We adapt the ubiquitous U-Net Convolution Neural Network framework [69] with ConvLSTM cells as an encoder-decoder framework for making sequence predictions. We introduce a novel architecture where the model learns spatiotemporal dependencies for long-horizon predictions using a mixture of Convolutional Neural Networks (CNN) and ConvLSTM encoder-decoder layers.

We can now define the process that allows the architecture to capture spatiotemporal dependencies. As a first step, we define the ConvLSTM architecture [32] that forms a building block for our model. ConvLSTMs can be seen as a special case of LSTMs, by replacing the Hadamard product in LSTMs with convolution operators. The ConvLSTM block enables the network to recursively process a sequence of representations and update its hidden states $\mathbf{H}_t$ that encode the complete spatiotemporal representations for all $\mathcal{X}_i$'s. The key idea behind ConvLSTMs is its capability to *retain* information relevant to its prediction task and *forget* information over time that might be repetitive or unnecessary over the sequence. The operations within the ConvLSTM block are shown as follows:

$$\begin{aligned}
\mathbf{I}_t &= \varphi(\mathbf{W}_{xi} * \mathbf{X}_t^c + \mathbf{W}_{hi} * \mathbf{H}_{t-1} + \mathbf{W}_{ci} \circ \mathbf{C}_{t-1} + \mathbf{W}_i),\\
\mathbf{F}_t &= \varphi(\mathbf{W}_{xf} * \mathbf{X}_t^c + \mathbf{W}_{hf} * \mathbf{H}_{t-1} + \mathbf{W}_{cf} \circ \mathbf{C}_{t-1} + \mathbf{W}_f),\\
\mathbf{D}_t &= \mathbf{I}_t \circ \tanh(\mathbf{W}_{xc} * \mathbf{X}_t^c + \mathbf{W}_{hc} * \mathbf{H}_{t-1} + \mathbf{W}_c),\\
\mathbf{C}_t &= \mathbf{F}_t \circ \mathbf{C}_{t-1} + \mathbf{D}_t,\\
\mathbf{O}_t &= \varphi(\mathbf{W}_{xo} * \mathbf{X}_t^c + \mathbf{W}_{ho} * \mathbf{H}_{t-1} + \mathbf{W}_{co} \circ \mathbf{C}_t + \mathbf{W}_o),\\
\mathbf{H}_t &= \mathbf{O}_t \circ \tanh(\mathbf{C}_t),
\end{aligned}$$

where $*$ denotes the convolution operation, $\circ$ is the Hadamard product, $\mathbf{X}_t^c \in \mathbb{R}^{D\times W\times H}$ is the input of the ConvLSTM block at time t, D is the number of stacked heightmaps in the blocks, $\mathbf{H}_t \in \mathbb{R}^{D\times W\times H}$ is the hidden state and also the output of the block at t, $\mathbf{F}_t \in \mathbb{R}^{W\times H}$ is a gate that controls what information needs to be forgotten or retained for the next step, $\mathbf{O}_t \in \mathbb{R}^{W\times H}$ is an output gate that controls what information is passed on to the hidden state, $\mathbf{D}_t, \mathbf{C}_t \in \mathbb{R}^{D\times W\times H}$ are the temporary cell state and the cell state at time t that work to accumulate the information over the history of the sequence, $\varphi : \mathbb{R}^{W\times H} \mapsto \mathbb{R}^{W\times H}$ is a sigmoid function. Finally, $\mathbf{W}_\bullet \in \mathbb{R}^{W\times H}$ are the learnable weights and biases of the network.

Formally, we define the model uncertainty through dropouts in the neural network by sampling K different sets of parameters $\mathbf{W}_\bullet$. That is,

$$\bar{\sigma}_t^2 = \mathrm{Var}\left(\hat{\mathbf{Y}}_t^{(k)}\right), \quad \forall t \in \mathcal{T}_y.$$

This operation is equivalent to performing K stochastic forward passes with the dropout of weights enabled during inference and then averaging the results. In this work, we simulate the MC dropout sampling using the $\Pr = 0.4$ probability of dropping each weight set for stochastic inference during prediction. Through the use of dropouts between each layer in our network, both during training and testing time, we enable our encoder-decoder model to estimate the prediction set $\bar{\mathbf{Y}} = \{\bar{\mathbf{Y}}_t \in \mathbb{R}^{M\times N} \mid \forall t \in \mathcal{T}_y\}$ and the corresponding variance set $\bar{\Sigma} = \{\bar{\sigma}_t^2 \in \mathbb{R}^{M\times N} \mid \forall t \in \mathcal{T}_y\}$. The MC dropout method allows our model to generate the requisite prediction estimates and the uncertainty of its prediction in a computationally efficient process. Therefore, a more detailed network model showing our intermediate outputs, i.e., $\hat{\mathbf{Y}}_t$ and $\hat{\sigma}_t^2$, can be summarized as

$$\left(\bar{\mathbf{Y}}_t, \bar{\sigma}_t^2\right) \leftarrow \left(\hat{\mathbf{Y}}_t, \hat{\sigma}_t^2\right) \leftarrow \Theta(\mathcal{X}, \mathbf{W}), \quad \forall t \in \mathcal{T}_y.$$

We also refer the reader to [8] for more details about our mean and variance predictions.

After the spatiotemporal learning of the pasture, we are ready to make predictions for the horizon $\mathcal{T}_y$.

VI. Multi-Robot Intermittent Deployment

The goal of the deployment is to maximize the information we can get from the environment while respecting the budget. To this end, we model the objective function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ as follows:

$$f(\mathcal{S}) = \sum_{s\in\mathcal{S}} \left(\bar{\sigma}_t^2(s) \sum_{s'\in\mathcal{S}\setminus s} \frac{d(s,s')}{|\mathcal{S}\setminus s|}\right) - w_1(t - t_1), \tag{5}$$

where $\bar{\sigma}_t^2(s)$ is the prediction variance of the decision factor s at time t. Note that t_1 is the starting time index of the prediction horizon $\mathcal{T}_y$. The distance function $d(s, s')$ is the weighted Euclidean distance and time difference between s and s'. That is,

$$d(s,s') = w_2 \log(||(x,y) - (x',y')||) + w_3||t - t'||, \tag{6}$$

where $s = (x, y, r, t)$ and $s' = (x', y', r', t')$ are two different decision factors. And w_1 is the waiting penalty weight, w_2 is the weight for the physical distance, and w_3 is the weight for the time difference between s and $s' \in \mathcal{S} \setminus s$.

The objective function is a weighted sum of prediction variances. The nominator $\bar{\sigma}_t^2(s)$ is the prediction variance of the decision factor s at time t. The denominator is the sum of the weighted distances between s and $s' \in \mathcal{S} \setminus s$ for all $s \in \mathcal{S}$. Meanwhile, w_1 is the weight to penalize the waiting time. We can deploy robots using a deployment set $\mathcal{S}$ that maximizes the objective function $f(\cdot)$ to reduce the prediction uncertainty.

Since the deployment should be energy efficient, we have two constraints to model the deployment budgets. The first constraint is,

$$|\mathcal{S} \cap \mathcal{V}_t| \le \ell_t, \forall t \in \mathcal{T}_y. \tag{7}$$

This constraint (per-day budget) indicates that the number of deployments cannot be larger than ℓ_t at time t, where $\ell_t \in \mathbb{R}$.

Algorithm 1 The Algorithm for the Long-Term Pasture Prediction and Sensing Problem

Input: The inputs are as follows:
- The historical dataset $\mathcal{X} = \{\mathbf{X}_t \in \mathbb{R}^{M\times N} \mid \forall t \in \mathcal{T}_x\}$;
- The neural network $\Theta(\cdot)$;
- The deployment ground set $\mathcal{V}$;
- The objective function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$;
- The matroid intersection constraint $\mathcal{M} = (\mathcal{V}, \mathcal{I})$.

Output: The deployment strategy set $\mathcal{S}$.

1: **for** $t \in \mathcal{T}_y = \{\tau, \ldots, T\}$ **do**
2: $\quad \hat{\mathbf{Y}}_t^{(k)} \leftarrow \Theta(\mathcal{X}, \mathbf{W}), \forall k = 1, \ldots, K$;
3: $\quad \bar{\mathbf{Y}}_t \leftarrow \frac{1}{K}\sum_{k=1}^{K} \hat{\mathbf{Y}}_t^{(k)}$; ▷ Mean
4: $\quad \bar{\sigma}_t^2 \leftarrow \mathrm{Var}(\hat{\mathbf{Y}}_t^{(k)})$; ▷ Variance
5: $\quad \bar{\Sigma} \leftarrow \bar{\Sigma} \cup \{\bar{\sigma}_t^2\}$;
6: **end for**
7:
8: $\mathcal{S} \leftarrow \emptyset, \mathcal{Z} \leftarrow \mathcal{V}$;
9: **while** $\mathcal{Z} \neq \emptyset$ **do**
10: $\quad s \in \arg\max_{v\in\mathcal{V}\setminus\mathcal{Z}} f(\{v\} \mid \mathcal{S})$;
11: $\quad$ **if** $\mathcal{S} \cup \{s\} \in \mathcal{I}$ **then**
12: $\quad\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$;
13: $\quad$ **end if**
14: $\quad \mathcal{Z} \leftarrow \mathcal{Z} \cup \{s\}$;
15: **end while**
16: $\mathcal{S} \leftarrow$ deployment strategy;

The second constraint is,

$$\sum_{t\in\mathcal{T}_y} \mathbb{1}(|\mathcal{S} \cap \mathcal{V}_t|) \le \ell, \tag{8}$$

where $\mathbb{1}(\cdot)$ is an indicator function as

$$\mathbb{1}(|\mathcal{S} \cap \mathcal{V}_t|) = \begin{cases} 1, & \text{if } |\mathcal{S} \cap \mathcal{V}_t| \ge 1\ , \\ 0, & \text{if } |\mathcal{S} \cap \mathcal{V}_t| = 0. \end{cases}$$

This constraint (total budget) suggests that the total number of deployable days cannot be larger than ℓ, where $\ell \in \mathbb{R}$. Therefore, the problem formulation is

$$\begin{aligned} &\underset{\mathcal{S}\subseteq\mathcal{V}}{\text{maximize}}\ f(\mathcal{S}, \bar{\Sigma}) \\ &\text{subject to } |\mathcal{S} \cap \mathcal{V}_t| \le \ell_t, \quad \forall t \in \mathcal{T}_y, \\ &\qquad\qquad \sum_{t\in\mathcal{T}_y} \mathbb{1}(|\mathcal{S} \cap \mathcal{V}_t|) \le \ell. \end{aligned}$$

It has been shown that both constraints are matroidal [12], and we will use $\mathcal{M}_1 = (\mathcal{V}, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{V}, \mathcal{I}_2)$ to denote those two, where $\mathcal{M}_1, \mathcal{M}_2$ are matroids and $\mathcal{I}_1, \mathcal{I}_2$ are independent sets. To simplify the notation, we use $\mathcal{M} = (\mathcal{V}, \mathcal{I})$, where $\mathcal{I} = \mathcal{I}_1 \cap \mathcal{I}_2$, to denote the intersection of two constraints. Thus, $\mathcal{M}$ is a matroid intersection constraint and the cardinality is $|\mathcal{M}| = 2$.

Using the proposed architecture with ConvLSTM and residual connections, we have the predicted variance set $\bar{\Sigma} = \{\bar{\sigma}_t^2 \in \mathbb{R}^{M\times N} \mid \forall t \in \mathcal{T}_y\}$. Given the intermittent deployment problem (Problem 1) and the deployment ground set $\mathcal{V}$, we propose to solve the problem using Algorithm 1. From Line 1 to Line 6, we first use the proposed architecture to

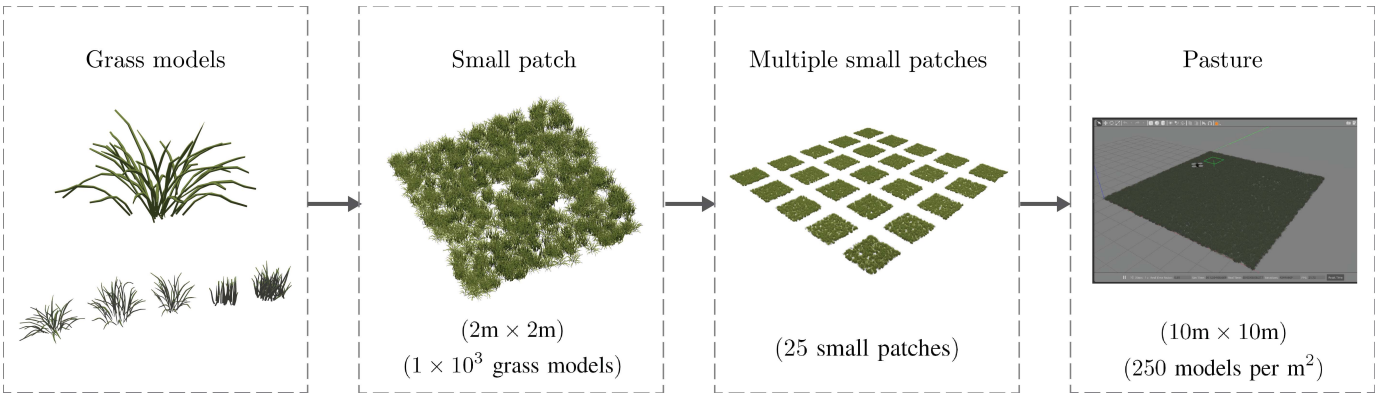


Fig. 4. A diagram of our pasture construction process.

predict the variance set $\bar{\Sigma}$ for the prediction time set $\mathcal{T}_y$. In the second part, the algorithm greedily selects all the available decision factors $v \in \mathcal{V} \setminus \mathcal{Z}$ based on the marginal gain $f(\{v\} \mid \mathcal{S})$, where $\mathcal{Z}$ is used to store all checked decision factors. The set $\mathcal{S}$ is the current solution of the deployment strategy and will be updated iteratively. Specifically, we initialize a set $\mathcal{Z}$ as $\mathcal{V}$. Then, in Line 10, we select one of the decision factors v that maximizes the marginal gain of the objective function $f(\{v\} \mid \mathcal{S})$, where $\mathcal{S}$ is the current solution of the problem and will be expanded as more decision factors are checked. In Line 11 to Line 13, we need to check if v satisfies the matroidal deployment constraint $\mathcal{M} = (\mathcal{V}, \mathcal{I})$. If so, v is added to the solution set $\mathcal{S}$ as $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$. Otherwise, the next round will be started. Meanwhile, $\mathcal{Z}$ is updated to store the checked decision factor v as $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{s\}$. The iteration will be finished when every decision factor in $\mathcal{V}$ is checked against the constraint $\mathcal{M} = (\mathcal{V}, \mathcal{I})$.

If we define the optimal deployment policy of the intermittent deployment problem as $\mathcal{S}^\star$ with respect to the predicted variance set $\bar{\Sigma}$, we then have the following result.

Theorem 1 ([63]): The optimality ratio of the greedy solution $\mathcal{S}$ generated by Algorithm 1 has the following performance:

$$f(\mathcal{S}) \geq \frac{1}{|\mathcal{M}| + c_f} f\left(\mathcal{S}^\star\right) = \frac{1}{2 + c_f} f\left(\mathcal{S}^\star\right),$$

where c_f is the curvature of $f(\cdot)$, $|\mathcal{M}| = 2$ is the cardinality of the matroid intersection constraint, and $\mathcal{S}^\star$ is an optimal solution.

The above result gives our problem a lower bound of algorithm Algorithm 1. Note that the curvature of the objective function $f(\cdot)$ can be evaluated by checking the contribution of every decision factor v from the ground set $\mathcal{V}$ as shown in Definition 2. Therefore, by using the proposed Algorithm 1, we can get an intermittent deployment policy using the proposed method while having a performance guarantee, as shown above.

Remark: The proposed pipeline can also be implemented in a receding horizon manner. That is, based on the proposed deployment policy $\mathcal{S}$, the new robot measurements from a series of deployments can be integrated into the historical dataset $\mathcal{X}$ to refine the learned network $\Theta(\cdot)$. Therefore, we can produce better predictions $(\bar{\mathbf{Y}}_t, \bar{\sigma}_t^2)$ and thus improved plans $\mathcal{S}$ for the future.

VII. Pasture Construction and Perception

When we have a synthesized pasture, we need a representation of what aerial robots with LiDAR would measure. Thus, based on our synthesized data, we simulate a realistic pasture environment and LiDAR measurements in Gazebo. This high-fidelity pastureland simulation environment will help us to understand the effectiveness of the simulated process. In this section, we will first focus on constructing pasture environments from the simulated data and then on the height estimation using LiDAR measurements.

A. Pasture Construction

In this work, we simulate a 10m × 10m pasture using 2.5×10^4 grass models. We set the size of the simulated pasture and the density like this when considering the computational complexity. First, we randomly pick 2.5×10^4 locations from this pastureland environment. We then assign a 3D grass model to each sampled location. The heights of grass models correspond to the heights at the same locations in the smooth 3D surfaces. To accelerate the simulation speed and lower the computational requirement, we divide the pasture into 25 small patches (2m × 2m per patch). In each patch, the density is 250 grass models per square meter with a total of 1×10^3 models per small patch. In this small patch, we use five species of plants to simulate different growth patterns as shown in Fig. 4. This selection will be validated later in the experimental section as the actual measurements look similar to our simulated environment. Each plant is spawned at the same randomly chosen coordinates throughout our simulation. We rescale a grass model in each dimension for each randomly selected location according to the desired height on the 3D surface. The flowchart of our pasture construction is shown in Fig. 4.

B. Pasture Perception

To estimate the height of the pasture, we use a UAV equipped with a LiDAR to collect point clouds over the pasture. Meanwhile, we need post-processing to remove noise after getting the point clouds. Those extra points are not part of the simulated field and need to be removed. To achieve this, we use crop box filters to remove the extra points and retain

the points of the 10m × 10m pasture as well as the points in the perimeter around it.

The height of a point cloud includes two parts: the height of the pasture and the height of the ground plane. To get the estimated height of the field, we use the mowed-down perimeter to estimate the ground plane. First, we have the following assumption of the ground of the environment.

Assumption 1 (Ground Plane): We assume that the ground of the pastureland is a plane in this paper. However, we will also introduce some methods that can be used to tackle other cases in the followings.

In the simulation, we use this assumption to facilitate the ground height estimation. However, other types of ground can also be integrated into our simulation framework, where we can use more sophisticated methods for ground surface regression.

In this work, we use the least squares method to compute the height of the ground by using perimeter points. The perimeter is all the points surrounding the target plot area. These perimeter points will be used to estimate the ground plane that is used for height estimation. An equation for a plane can be defined as shown below.

$$Ax + By + Cz + D = 0. \tag{9}$$

where A, B, C, and D are the parameters defining the plane. Without loss of generality, we assume $C = 1$.

We denote by $\mathbf{p}_i = [x_i, y_i, z_i]^\top \in \mathbb{R}^3$ the location of ith point in the perimeter. Since we are solving for a best-fit plane of multiple points, a least square form of our formulation can be written as

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ & \cdots & \\ x_P & y_P & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ D \end{bmatrix} = - \begin{bmatrix} z_1 \\ z_2 \\ \cdots \\ z_P \end{bmatrix},$$

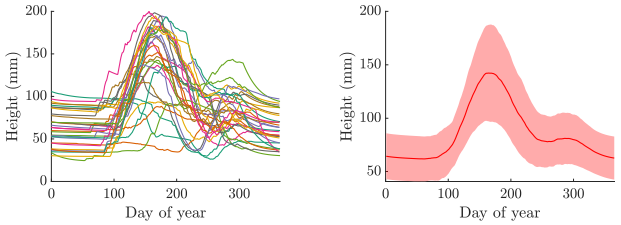
where P is the total number of points in the perimeter.

To perform the linear least squares test, we multiply a transpose of the left-most matrix on both sides. Note that we will use $\sum x_i x_i$ as $\sum_{i=1}^{P} x_i x_i$ for simplicity. This notation will also be applied to other relevant terms. Then the above equation is simplified as

$$\begin{bmatrix} \sum x_i x_i & \sum x_i y_i & \sum x_i \\ \sum y_i x_i & \sum y_i y_i & \sum y_i \\ \sum x_i & \sum y_i & P \end{bmatrix} \begin{bmatrix} A \\ B \\ D \end{bmatrix} = - \begin{bmatrix} \sum x_i z_i \\ \sum y_i z_i \\ \sum z_i \end{bmatrix}.$$

Defining all points, i.e., $\mathbf{p}_i = [x_i, y_i, z_i]^\top$, relative to the plane centroid sets the summations of the individual components to 0 and simplifies the above equations. The centroid of the plane is calculated by using $\mathbf{o} = P^{-1} \sum_{i=1}^{P} \mathbf{p}_i \in \mathbb{R}^3$. Note that $P \neq |\mathcal{P}|$. Then, all the points in the perimeter are updated as $[x_i, y_i, z_i]^\top \leftarrow [x_i, y_i, z_i]^\top - \mathbf{o}$. The above equation can then be simplified as

$$\begin{bmatrix} \sum x_i x_i & \sum x_i y_i & 0 \\ \sum y_i x_i & \sum y_i y_i & 0 \\ 0 & 0 & P \end{bmatrix} \begin{bmatrix} A \\ B \\ D \end{bmatrix} = - \begin{bmatrix} \sum x_i z_i \\ \sum y_i z_i \\ 0 \end{bmatrix}.$$



(a) Average height. (b) Mean & standard deviation.

Fig. 5. (a). The average height of the pastureland environment in 30 years (represented by 30 lines). The x-axis represents the day of the year. The y-axis represents the corresponding average height. (b). The mean and the standard deviation of each day's height are calculated using historical data.

From the last row, we observe that $D = 0$ since $P \neq 0$. Therefore, this equation can further be simplified as

$$\begin{bmatrix} \sum x_i x_i & \sum x_i y_i \\ \sum y_i x_i & \sum y_i y_i \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = - \begin{bmatrix} \sum x_i z_i \\ \sum y_i z_i \end{bmatrix}.$$

Using Cramer's rule, we get A and B as follows.

$$\begin{aligned} A &= \left(\sum y_i z_i \times \sum x_i y_i - \sum x_i z_i \times \sum y_i y_i \right) / \Delta, \\ B &= \left(\sum x_i y_i \times \sum x_i z_i - \sum x_i x_i \times \sum y_i z_i \right) / \Delta, \end{aligned} \tag{10}$$

where

$$\Delta = \sum x_i x_i \times \sum y_i y_i - \sum x_i y_i \times \sum x_i y_i.$$

Therefore, the ground plane is fully defined, and the actual height of the pastureland can be adjusted accordingly.

VIII. Evaluation

In this section, we demonstrate the results of each component in our pipeline. The historical data is generated using Matlab. The pasture is simulated by using Blender. The neural network training was conducted with a PyTorch backend on 2x AMD Epyc 7742 CPUs and a multi-GPU training regime with 8x Nvidia RTX 6000 GPUs.

A. Training Data Preparation

In this section, we focus on preparing training data. Based on historical meteorological data from a site in Iowa, we simulate 30 years of tall fescue pasture dry matter production. Specifically, we will use the first 28 years of data for training and reserve the last year's data for testing purposes. In Fig. 5, we demonstrate the statistics of the historical data. Fig. 5(a) shows the average height change of the simulated pastureland environment, where each line denotes the height change for different days in a year. We also calculate each day's mean and standard deviation using the historical data as shown in Fig. 5(b).

The simulated pasture yield was then used to generate average pasture height data based on the model in [7] describing the relationship between pasture green dry matter and LiDAR-measured pasture height for each day. Following the pastureland generation procedure described in Section IV-B, we generate a 2D pastureland environment over 28 years using a dynamic GMM. In the simulation, we use $B = 7$ basis functions. The random initial settings of the basis function

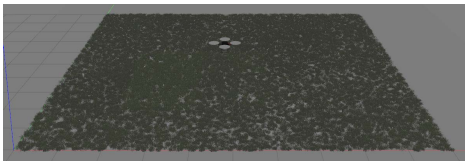


Fig. 6. A simulated pastureland 10m $\times$ 10m environment using 2.5×10^4 grass models.

TABLE I
THE LOCATIONS, LENGTH-SCALES, AND INITIAL WEIGHTS OF DIFFERENT BASIS FUNCTIONS

i	Location (x, y)	Length-scale c_i	Initial weight w_i
1	(5.0, 5.0)	0.13	4.17
2	(3.0, 4.0)	0.13	4.17
3	(2.0, 1.5)	0.15	2.50
4	(8.0, 8.0)	0.18	6.67
5	(8.0, 1.5)	0.13	3.33
6	(1.0, 1.0)	0.13	3.33
7	(1.0, 9.0)	0.25	4.17

location (x, y), the length-scale c_i, and the initial weight w_i of each basis function $b_i(x, y)$ are shown in Table I. Later on, in our experimental section (Section VIII-E), we can see that those parameter settings can help us to simulate pastureland environments that are very close to the real-world scenario.

B. Point Cloud Testing Data Preparation

In this section, we focus on preparing the testing data. To test the performance of proposed pipeline, we simulate an *actual* 10m $\times$ 10m pasture (Fig. 6) in Gazebo using the method proposed as illustrate in Fig. 4. The selected pasture size, i.e., 10m $\times$ 10m, is a result of careful consideration of the computational speed and complexity of the entire process. We will then collect data from this simulated *actual* environment to test the prediction and deployment performance.

To further exploit parallelization computation to improve the simulation speed, we divide the pasture into 25 small 2m $\times$ 2m patches. For each patch, we use 1×10^3 grass models (250 models per square meter). Finally, 25 small patches are attached to form one large 10m $\times$ 10m pastureland environment.

We randomly assign grass models and extrude the model in three dimensions to match the predefined height for the model in that location. As the swards of the grass grow with a curvature, we need to adjust the location of the base of each grass model, i.e., the location of each grass model at the ground plane to match the location of its sward at its highest point. This offset allows us to calculate the vertical height of each individual plant directly. Specifically, denote by $\gamma \in \mathbb{R}$ the scaling factor of a grass model, which is calculated by comparing the desired height of a grass model and the original height. We denote $\mathbf{m} = [m_x, m_y]^\top \in \mathbb{R}^2$ the original 2D location of the topmost point, and $\xi = [\xi_x, \xi_y]^\top \in \mathbb{R}^2$ the offset of the topmost point, where $\xi_x, \xi_y \in \mathbb{R}$ are the offset in x and y direction. We apply a shifting process as $\mathbf{m} \leftarrow \mathbf{m} - \gamma \cdot \xi$ to ensure that the topmost point is at the predefined location. After this adjustment, the 2D location of the topmost point of a grass model is transformed to the predefined 2D location.

Following the steps described in Section VII, we generate 15 pastureland environments in Gazebo using the historical data with a day interval of $\delta = 4$. An example illustrating the simulated pastureland environment (10m $\times$ 10m) is shown in Fig. 6. In this simulated pastureland environment, there are 2.5×10^4 grass models.

Next, we use a robot to collect a point cloud for each simulated environment. The simulated LiDAR has an inherent standard deviation of 4mm in its readings. We also assume the ground of the field is flat. Nevertheless, adding different terrains to simulate different types of ground is easy. During the simulations, the plant locations, pose, and species for a particular location in the pasture are fixed throughout the years to accelerate those processes. Finally, all the collected point clouds will be used as testing inputs.

C. Neural Network Training and Prediction

In this section, we focus on the following two different parts of our neural network.

- We first give details of our training settings based on training data generated from GMM, as stated in Section VIII-A.
- We then test the performance of the networks based on two different types of testing inputs.

Training: Based on the generated GMM data from Section VIII-A, the training sequences $\mathcal{X}_i$'s are generated by setting the number of measurements in each sequence as $\alpha = 15$. Also, the number of intervals is $\delta = 4$. We use early stopping, where training is stopped if validation loss does not improve for ten epochs, usually resulting in 30 training epochs.

Testing: We evaluate the performance of the prediction network in two different cases:

1) In the first testing case, we use one group of point cloud measurements as testing inputs and another group of point cloud measurements as testing outputs. This testing case can help us understand the theoretical prediction performance.
2) In the second testing case, we use two groups of GMM data as testing inputs and outputs. This testing case can help us understand the practical prediction performance.

1) Testing by using point cloud measurements: We note that due to the computationally intensive process of constructing the aforementioned pastures and their subsequent point cloud measurements, we only generate a single test case consisting of 30 days, where the data is split into 15 days as an input sequence to the prediction network, and the remaining 15 days are used to evaluate the prediction performance. The collected point clouds will be first converted to heightmaps of size 100 $\times$ 100 as the original point clouds are too large to be used as network prediction inputs. Meanwhile, we need to process those heightmaps since the measurements are noisy. In Fig. 8, we demonstrate a downsampled surface of the point cloud shown in Fig. 7. Specifically, after downsampling, the processing includes two steps: (a). a size of 3 $\times$ 3 median filtering; (b). a size of 3 $\times$ 3 flat convolution filtering. After processing, we see that the growth pattern of the pastureland is visible as shown in Fig. 9(b) when compared with the raw

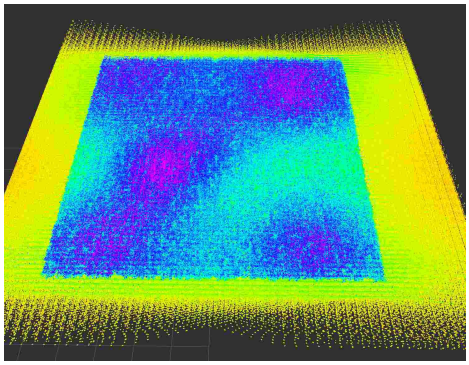


Fig. 7. The corresponding point cloud of the pasture (10m × 10m) shown in Fig. 6.

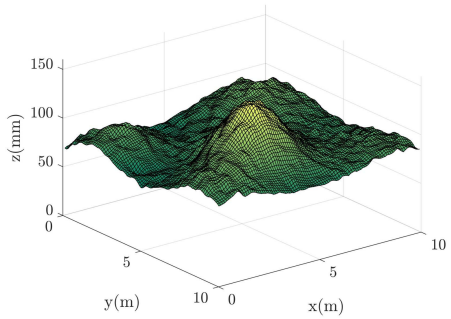
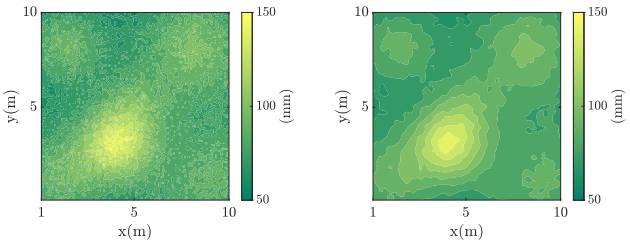


Fig. 8. The downsampled surface of the point cloud shown in Fig. 7.

map as shown in Fig. 9(a). Those processes make the neural networks' prediction of pastureland dynamics more efficient. Meanwhile, this processing only has a mild change on the original data, as can be seen from the histogram comparison in Fig. 10.

To test the prediction performance, we predict the heightmaps for another 15 days using a stride length of $\delta = \{1, 4\}$. The longest effective horizon for those two are $L = 15$ and $L = 60$ respectively. Then, we generate the corresponding point cloud measurement to check the testing performance. To estimate the uncertainties, we set the number of MC dropout samples as $K = 500$. In general, our network performance on average performs within a 12% error rate in the worst case when $\delta = 4$ and $L = 60$, and within a 5% error rate when data is available more frequently when $\delta = 1$ and $L = 15$. To further demonstrate the details of our long-time horizon prediction results, we show the 5th and 15th prediction results with their ground truth in Fig. 11. Note that the effective time horizons for those two predictions are $L = 40$ and 60 days. The uncertainty estimations in terms of standard deviations of the predictions run over $K = 500$ samples. We observe that the network has higher confidence at the highest pasture heights since it is easier to learn feature mappings due to more significant correlations within its neighborhood. The prediction error is lowest at the highest points in the pasture since the network learns to estimate the peak points with higher confidence.

In Fig. 12, we compare the ground truth and the corresponding predictions using all the predictions across the output sequence. From the result, we can see that most prediction-ground truth pairs lie in the 45 degree line, where we can observe that most predictions are close to the ground



(a) Before processing. (b) After processing.

Fig. 9. The downsampled heightmaps of the point cloud shown in Fig. 7.

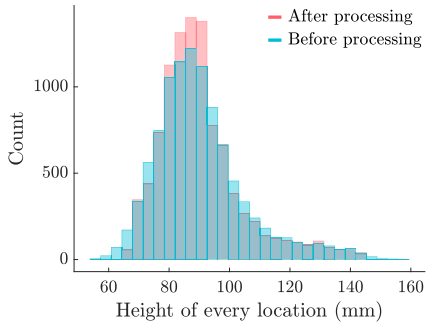


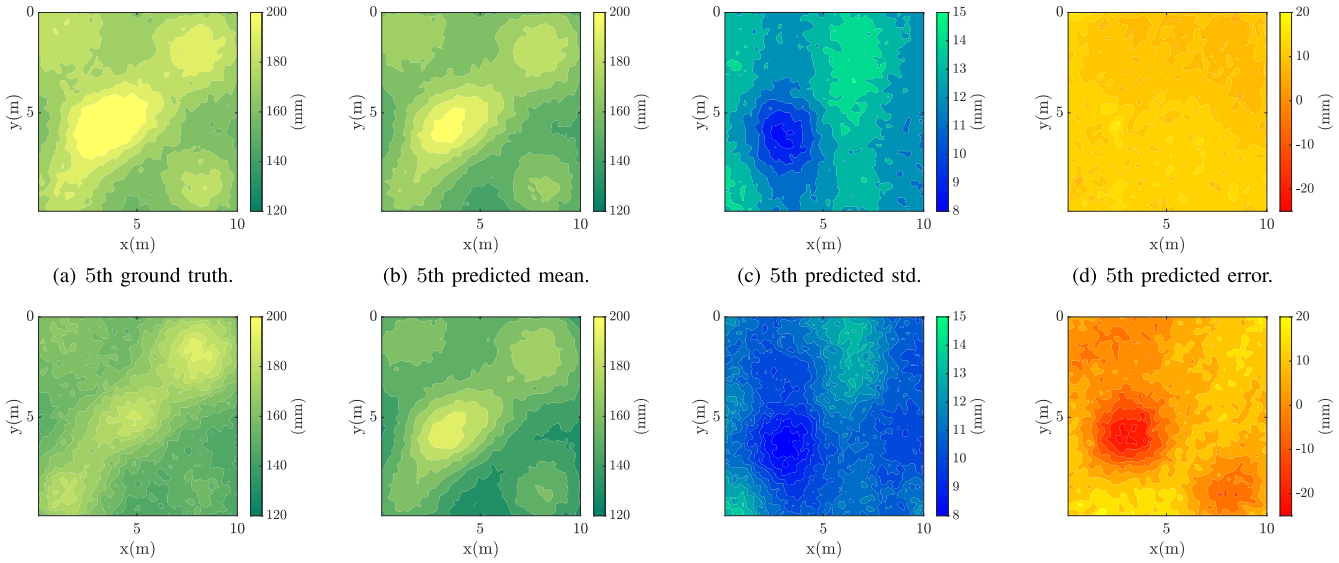
Fig. 10. The histogram comparison of the two heightmaps shown in Fig. 9.

truth. The prediction uncertainty of the model increases as the prediction error increases, as shown in Fig. 13, showing a clear correlation between the network's confidence against the prediction errors. This reiterates the suitability of the computationally efficient Bayesian approximation for uncertainty estimates, allowing a quick and efficient turnaround in prediction. This methodology reduces the need for expensive hardware for training and inference of the deep learning-based prediction model and allows even small industries or farms to tune and integrate the prediction systems into their workflow.

Remark: The network makes predictions from point cloud measurements in the above tests, and the original training dataset of pasture height maps generated from GMM is not used. That indicates the ability of our network to generalize beyond the simulated training data to real-world applications by using LIDAR-based measurements.

2) Testing by using GMM data: Next, we use Monte Carlo simulations to test the prediction performance using different GMM inputs. Note that the training model is unchanged in this case. We set $|\mathcal{T}_y| = \alpha = 15$. That is, we use a length of 15 heightmaps to predict another group of 15 heightmaps. Since we will test the performance using two years' data and the first 15 heightmaps can only be used as inputs, the number of distinct output prediction sequences is $H = (365 - |\mathcal{T}_y|) \cdot 2$. We should also note that K in (4) is the number of samples used for calculating one prediction $\bar{\mathbf{Y}}_t, \forall t \in \mathcal{T}_y$. We evaluate the performance of the neural network predictions with the following metrics: Root Mean Square Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Averaged Standard Deviation (ASTD).

$$\text{RMSE} = \sqrt{\frac{1}{H \cdot |\mathcal{T}_y|} \sum_{h=1}^{H} \sum_{t \in \mathcal{T}_y} \left(\mathbf{Y}_t^{(h)} - \bar{\mathbf{Y}}_t^{(h)} \right)^2},$$



(a) 5th ground truth. (b) 5th predicted mean. (c) 5th predicted std. (d) 5th predicted error.

(e) 15th ground truth. (f) 15th predicted mean. (g) 15th predicted std. (h) 15th predicted error.

Fig. 11. The comparison between deep learning predictions and the corresponding ground truth. The first row is for $\alpha = 5$ prediction (the effective horizon is 40 days), and the second row is for $\alpha = 15$ prediction (the effective horizon is 60 days). The comparisons include predicted mean, predicted standard deviation, and the prediction error for every location.

$$\text{MAE} = \frac{1}{H \cdot |\mathcal{T}_y|} \sum_{h=1}^{H} \sum_{t \in \mathcal{T}_y} \left| \mathbf{Y}_t^{(h)} - \bar{\mathbf{Y}}_t^{(h)} \right|,$$

$$\text{MAPE} = \frac{1}{H \cdot |\mathcal{T}_y|} \sum_{h=1}^{H} \sum_{t \in \mathcal{T}_y} \left| \frac{\mathbf{Y}_t^{(h)} - \bar{\mathbf{Y}}_t^{(h)}}{\mathbf{Y}_t^{(h)}} \right|,$$

$$\text{ASTD} = \sqrt{\frac{1}{H \cdot |\mathcal{T}_y|} \sum_{h=1}^{H} \sum_{t \in \mathcal{T}_y} \left(\bar{\sigma}_t^{(h)} \right)^2},$$

where H is the batch size of the output prediction. For simplicity, we omit the superscript h on $\mathcal{T}_y$ and note that the discrete time intervals t for the output predictions can differ for each batch sample.

The results of these metrics are reported in Table II, containing the results from the above two testings. These metrics quantify the performance of the prediction algorithm by aggregating the errors and uncertainty over the discretized values of the pasture.

D. Robotic Deployments Results

In this section, we use Monte Carlo simulations to test the performance of the proposed integrated pipeline. Specifically, we have two goals for the testing:

1) Deployment strategy comparison: we want to test the effectiveness of the proposed pipeline using the same neural network prediction results but with different deployment strategies.
2) Pipeline comparison: We want to test the effectiveness of the proposed pipeline, i.e., neural network prediction and intermittent deployment, with another often used pipeline.

In the followings, we will first specify the details of the comparison settings and then demonstrate the comparison results for each testing scenario.

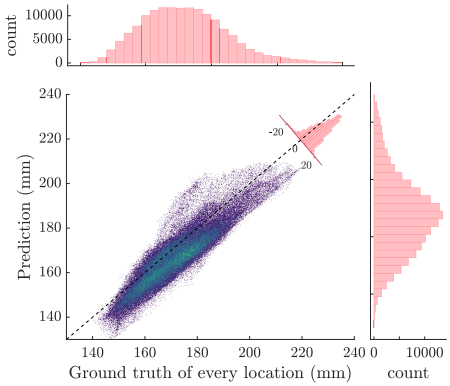


Fig. 12. The relationship between the ground truth and the corresponding prediction by using all the predictions and the ground truth. The x-axis represents the ground truth of every point, and the y-axis represents the corresponding prediction result. The marginal histograms show the statistics of the ground truth and the prediction independently.

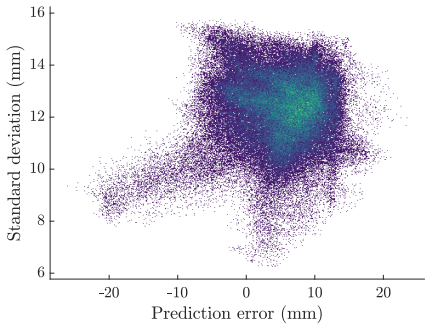


Fig. 13. The relationship between the prediction error and the corresponding prediction standard deviation of every location by using all the predictions.

1) Deployment Strategies Comparison: Based on the same deep learning prediction result, we compare the performance using the following deployment policies:

- *Intermittent*: the intermittent deployment policy, where the policy is generated through the method proposed

TABLE II

ACCURACY METRICS (IN mm) FOR DIFFERENT METHODS WITH AND WITHOUT UNCERTAINTY ESTIMATES (UE), WHERE THE STRIDE LENGTH IS $\delta = \{1, 4\}$ AND THE CARDINALITY OF HORIZONS SET IS $\alpha = 15$*

	(GMM Data)				(Point Cloud Data)			
Model	RMSE	MAE	MPAE	ASTD	RMSE	MAE	MPAE	ASTD
$\delta = 4$ + UE	19.25	13.42	11.91	9.27	**7.41**	**6.46**	**3.672**	12.39
$\delta = 1$ + UE	**6.91**	**5.12**	**4.65**	**8.31**	–	–	–	–
$\delta = 4$	24.65	18.79	15.62	–	19.31	18.07	10.58	–
$\delta = 1$	18.52	14.38	13.13	–	–	–	–	–

*(Since the tests without UE do not implement MC Dropout methods, no ASTD values are available. Additionally, due to high computation requirements for point cloud data, results for $\delta = 1$ are unavailable.)

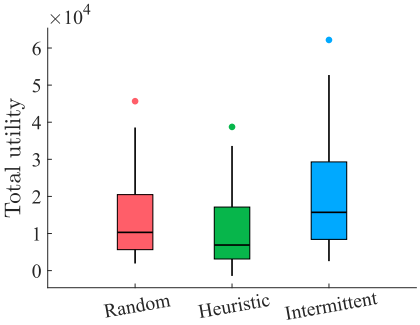
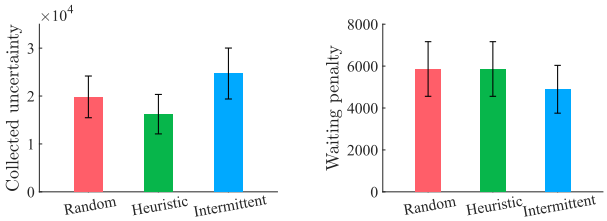


Fig. 14. The utility statistics of the three methods after running 50 trials. The proposed intermittent deployment method has the highest average utility.



(a) Collected uncertainty. (b) Waiting penalty.

Fig. 15. (a). The comparison of the collected uncertainties of three different methods, which is the first part of the objective function. (b). The waiting penalty of three different methods, which is the second part of the objective function.

in Section VI. We refer to this deployment policy as "Intermittent".

- *Heuristic*: a heuristic deployment policy, which is used to mimic how humans manually monitor pasturelands on spatial and temporal scales. On the temporal scale, the robots are deployed within a fixed time interval. For example, if the planning horizon is 9 days and the robot team can only be deployed 3 times, then those robots will be deployed at 1st day, 5th day, and 9th day. On the spatial scale, the deployment locations are evenly distributed across the pastureland. This spatial strategy simulates the case where each robot has the same coverage ability, and they are deployed to the locations where the overall coverage is maximized. We refer to this deployment policy as "Heuristic".
- *Random*: a random deployment policy, where both deployment times and locations are randomly selected from the ground set $\mathcal{V}$. We refer to this deployment policy as "Random".

We first evaluate the performance using the collected reward (the objective function value). Then, we collect samples based on the generated deployment policies. After that, we use the collected information from different methods to make 10 more predictions. We finally compare the prediction performance of those methods. Thus, those steps will help verify the proposed pipeline's effectiveness.

Based on the $\alpha = 15$ deep learning prediction results, we run 50 Monte Carlo simulations to generate different deployments under different settings. The settings are as follows. The maximum number of deployable days (total budget) ℓ is randomly sampled from the set $\{5, 6, \ldots, 12\}$. The number of maximum sampling points ℓ_t is sampled from the set $\{2^2, 3^2, \ldots, 8^2\}$. The cardinality of the planning horizon is $|\mathcal{T}_y| = 15$. We also assign each robot a random weight for the same traveling cost to simulate the heterogeneity. In each run, we generate one instance and then use the above three methods to compare the performance. Also, the number of robots is set to $\ell_t \cdot \ell$ for each instance. The weights for distance and time are $w_2 = 0.1, w_3 = 1$. And the weight for time penalty is $w_1 = 5$. The definitions of those parameters can be found in our problem formulation in Section III-B.

Then, we first compare the results of different deployment policies by comparing the collected reward of each method after running 50 trials. The result is shown in Fig. 14, where the proposed intermittent policy has the highest averaged utility. Then, in Fig. 15, we demonstrate the two different parts of the objective function. In Fig. 15(a), we show the collected uncertainty of three different deployment methods, which is the first part of the objective function. In Fig. 15(b), we compare the waiting penalty part in the objective function. Then, in Fig. 16, we set a high waiting penalty as $w_1 = 10$ while keeping w_2, w_3 the same as before. Again, we run other 50 trials to compare the performance. In Fig. 16, the values of different parts in the objective function prove that our proposed method can collect more rewards while having less waiting penalty. Similarly, in Fig. 17, we compare the result from each part of the objective function using three different deployment policies. This finishes up the first part of this comparison. In Fig. 18, we demonstrate a deployment result using one setting instance. In this result, we set the parameters as follows. The maximum number of deployments for each day is set to $\ell_t = 4$, and the maximum number of deployable days is set to $\ell = 3$.

Next, we use robots to collect data based on the generated deployment policies. The collected height information will be

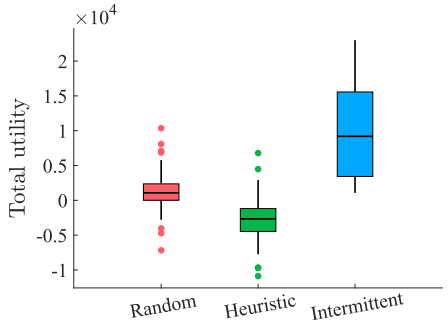
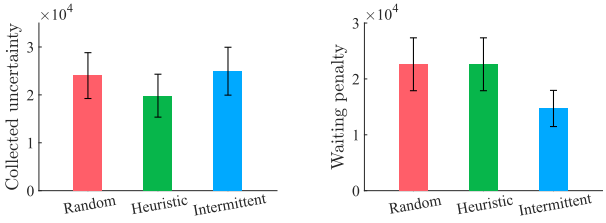


Fig. 16. The utility statistics of the three methods after running 50 trials when the waiting penalty weight w_1 is high.



(a) Collected uncertainty. (b) Waiting penalty.

Fig. 17. The utility/cost of different parts of our objective function when the waiting penalty weight w_1 is high.

TABLE III

THE AVERAGED PREDICTION COMPARISONS (IN mm) BETWEEN THE DEEP LEARNING (DL) BASED METHOD AND THE GAUSSIAN PROCESS (GP) FOR $\alpha = 5$ AND $\alpha = 15$

		(DL prediction)		(GP prediction)	
Time	Truth	Prediction	Error	Prediction	Error
$\alpha = 5$	181.53	169.82	**11.71**	149.87	31.66
$\alpha = 15$	164.42	158.83	**5.60**	117.08	47.34

used to update our knowledge of the environment. Finally, we compare the predictions from each method using this updated information. Note that white noise is also added to simulate measurement noise. Specifically, we use the collected information to make another 10 more predictions, i.e., from $\alpha = 1$ to $\alpha = 10$, with the size of interval of $\delta = 2$. For each prediction, we compare it with the ground truth. Then, the final averaged prediction error is shown in Fig. 19. The averaged prediction error is calculated by averaging the prediction errors from all locations. Since we run the simulation 50 trials, there are 50 different averaged prediction errors for each method shown in Fig. 19. After using the updated information from different methods, we see that the proposed pipeline has a lower average prediction error than the other two methods. That demonstrates the effectiveness of the proposed method.

2) Pipelines Comparison: In the paper, the deployment strategies are made based on the predicted variance map. However, as the deployment problem itself is an NP-hard combinatorial optimization problem, there is no simple way to get an optimal deployment strategy based on any prediction result. In the paper, we adopt a commonly used mutual information-based pipeline in the second comparison category and compare it with our proposed pipeline. Specifically, we will use a 2D Gaussian process (GP) to model the environment and make predictions. We then use the proposed intermittent deployment idea to make deployment strategies based on the different prediction results. Again, we finally

TABLE IV

THE STATISTICS OF THE AVERAGED PREDICTION ERRORS OF THE PROPOSED DEEP LEARNING (DL) BASED PIPELINE AND THE GAUSSIAN PROCESS (GP) BASED PIPELINE USING 50 TRIALS

	Mean (mm)	Std. (mm)
DL-based pipeline	**11.39**	**0.08**
GP-based pipeline	36.09	0.21

make another 10 future predictions after deployments to test the performance.

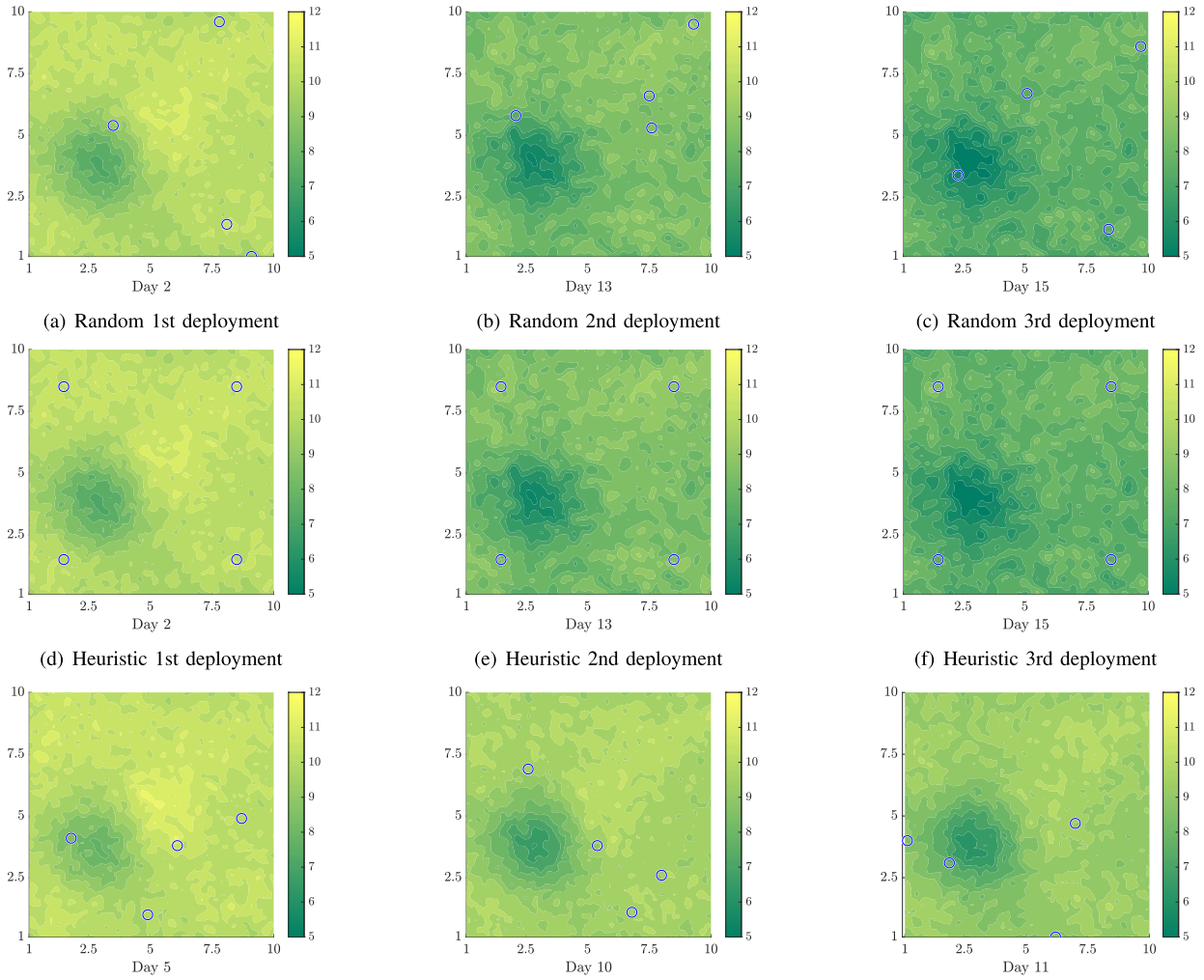
In this comparison, we first use GP to make predictions. The settings of this GP are as follows. The training dataset is from the point clouds generated in Gazebo as described in Section VII. We also need to convert those point clouds into heightmaps. We denote the training inputs of the GP by $@_t = [x, y, t] \in \mathbb{R}^3$. The training outputs are corresponding grass height of location (x, y) at time t. From the above 15 heightmaps, we randomly pick 1×10^3 points across all those heightmaps to form the training set. Since GP is a non-parametric method, we select kernels as follows. The mean kernel is defined using the historical mean data as shown in Fig. 5. The covariance kernel is a composite of a 2D Gaussian covariance kernel and a 1D linear covariance kernel with noise term. The Gaussian covariance kernel is similar to the one shown in (3). Then, those two covariance kernels are summed up to form the final covariance kernel. In Fig. 20, we show two prediction results when $\alpha = 5$ and $\alpha = 15$. We notice that the proposed deep learning-based method can maintain more details than the GP-based predictions. We also include the comparison details of those two comparisons in Table III. Note that the comparison is based on the averaged prediction errors.

Next, we use the proposed intermittent deployment method to select measurement locations based on the mutual information while respecting the proposed constraints (7) and (8). Note that the mutual information matrix is built on all the available locations at different times. The parameters and the settings are the same as described in the first comparison. We then run 50 trials using those settings to generate different deployment policies. Since the mutual information-based method requires a full covariance matrix for the deployment ground set $\mathcal{V}$, which is extremely computationally expensive, we choose to reduce the original 100×100 deployable locations to 10×10 deployable locations. Based on the deployment result of each trial, the training set is updated using the new measurements from the deployments. After that, we make predictions for the next 10 steps. Therefore, the performance comparison between the mutual information and GP-based pipeline and the proposed DP-based pipeline is based on the newly updated prediction results. In Fig. 21 and Fig. 22, we demonstrate the averaged prediction error of each pipeline. Also, the statistics of this comparison are shown in Table IV.

E. Perception Results

In this section, we demonstrate the result from a field experiment to illustrate the effectiveness of our simulated pastureland environment.

We experimented at Virginia Tech's Turfgrass Research Center to test our growth analysis algorithm. A 10m × 10m



(a) Random 1st deployment (b) Random 2nd deployment (c) Random 3rd deployment

(d) Heuristic 1st deployment (e) Heuristic 2nd deployment (f) Heuristic 3rd deployment

(g) Intermittent 1st deployment (h) Intermittent 2nd deployment (i) Intermittent 3rd deployment

Fig. 18. An example of deployment policies generated by different methods. (a)-(c). random, (d)-(f). heuristic, (g)-(i). intermittent. Those are predicted variance maps.

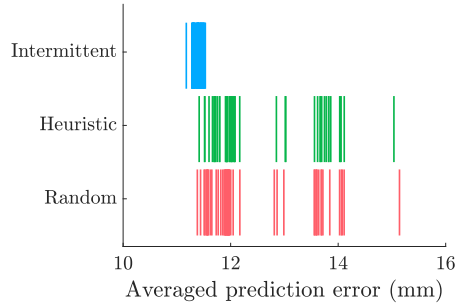
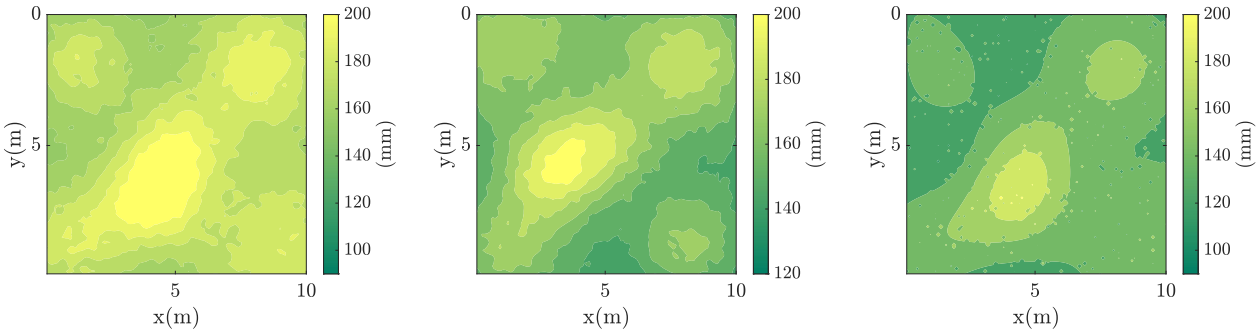


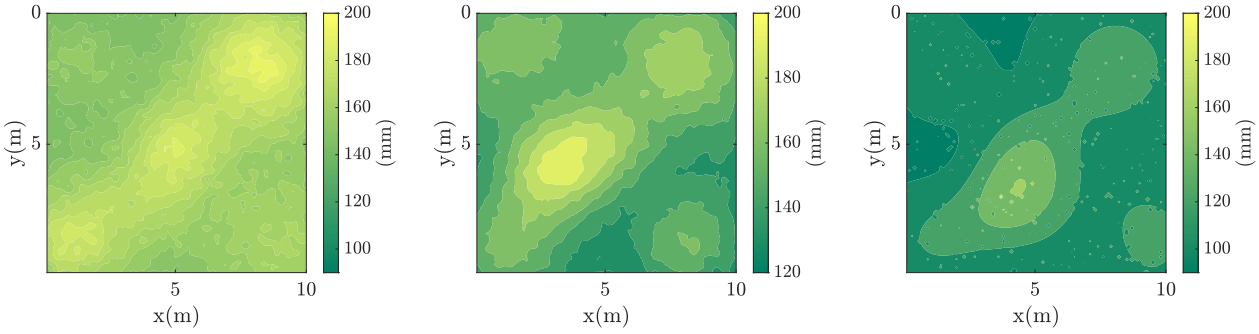
Fig. 19. The comparison of the averaged prediction error by calculating the mean of the errors at each location using 50 trials.

region of the Turfgrass center was reserved, and the grass was grown. It was untouched over time, but the perimeter around it was consistently mowed down. A DJI M600 UAV with a bottom-facing Velodyne VLP-16 3D-LiDAR, as shown in Fig. 23, was flown over the region. The localization information and LiDAR scans during flights were collected using an onboard NVIDIA Jetson TX2. This data was used to build a point cloud map of the region using a similar technique to the one used in the point cloud generation for the simulation discussed previously. These flights were conducted weekly to get temporal point cloud maps of the region. Manual measurements of the region were also collected. Nine spots throughout the region were used as the manual measurement points. These were averaged to get the manual measurements shown in Table V.

The distance between every point in the region's point cloud and the ground plane was computed to estimate the height of these points. This was done using the normal vector determined using the method described Section VII. Because the vast majority of LiDAR points were not at the very top of the grass, the height estimations using this method were under-estimations regardless of which percentile we looked at. This is shown in Table V, and the result is shown in Fig. 24. However, since growth is what we are looking into, we can look at the relative differences in the estimations between each flight. These differences are shown in Table VI with the first row showing the manually measured growth between sampling weeks. With perfect estimations, the percent difference between the manual measurements and estimated height would be 0%. However, our best results were found using the 99th percentile of growth estimates which had



(a) 5th ground truth. (b) 5th predicted using DL. (c) 5th prediction using GP.



(d) 15th ground truth. (e) 15th predicted using DL. (f) 15th prediction using GP.

Fig. 20. The comparison between the ground truth and the predictions by using deep learning (DL) based method and Gaussian process (GP) based method.

TABLE V
TURFGRASS EXPERIMENT HEIGHTS

	Week 1	Week 2	Week 3	Week 4
Hand Measurement	48.00 cm	68.17 cm	74.11 cm	69.04 cm
50th Percentile	3.97 cm	10.67 cm	12.17 cm	9.60 cm
75th Percentile	6.31 cm	16.01 cm	18.41 cm	14.60 cm
90th Percentile	8.73 cm	22.08 cm	26.11 cm	19.93 cm
95th Percentile	11.03 cm	27.48 cm	31.53 cm	24.49 cm
97.5th Percentile	13.43 cm	32.22 cm	36.64 cm	30.43 cm
99th Percentile	15.63 cm	36.43 cm	41.65 cm	36.17 cm
99.5th Percentile	16.70 cm	39.23 cm	44.29 cm	39.42 cm

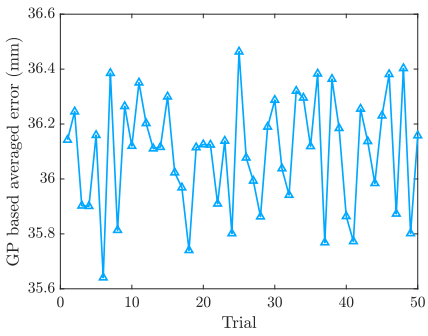


Fig. 21. The averaged prediction errors using Gaussian process (GP) and mutual information-based pipeline after 50 trials.

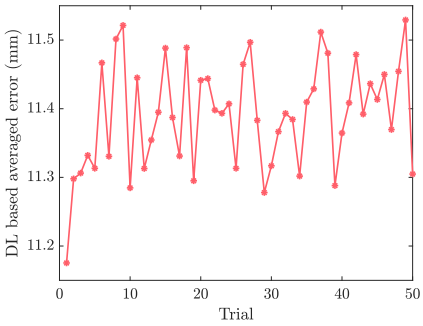


Fig. 22. The averaged prediction errors using the proposed deep learning (DL) based pipeline after 50 trials.

(a) (b)

Fig. 23. A UAV (DJI M600) mounted with a LiDAR (Velodyne VLP-16) and an onboard system (Nvidia Jetson TX2).

an average percent difference of 7.9% compared to manual measurements.

Those real-world experiments shown above validate our pasture simulation regime and thus our evaluation results. Meanwhile, the point cloud results from the real-world experiments are close to the results in our simulated world. All those results demonstrate the effectiveness of our proposed pipeline.

Finally, we mention some methods that can be used to tackle the case when the ground is not a plane.

A ground model can be created during the very first flight of the farm when it has had no growth. Flying the UAV over the entire farm, we could create a 3D point cloud map of the environment. Then, we could downsample the

TABLE VI
TURFGRASS EXPERIMENT GROWTH

	Week 1-2	Week 2-3	Week 3-4
Hand Measurement	20.17 cm	5.94 cm	-5.07 cm
50th Percentile	6.72 cm	1.49 cm	-2.58 cm
75th Percentile	9.70 cm	2.40 cm	-3.81 cm
90th Percentile	13.34 cm	4.03 cm	-6.18 cm
95th Percentile	16.45 cm	4.05 cm	-7.05 cm
97.5th Percentile	18.79 cm	4.40 cm	-6.21 cm
99th Percentile	20.80 cm	5.22 cm	-5.48 cm
99.5th Percentile	22.54 cm	5.05 cm	-4.87 cm

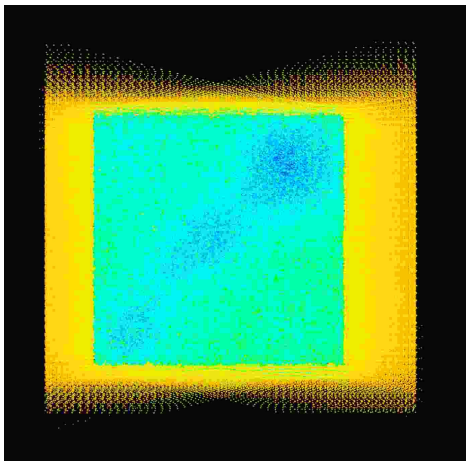


Fig. 24. Unprocessed point cloud of 10 × 10 meter Turfgrass plot with the perimeter. The UAV flew over the generated plot in the simulation. The data collected from the 3D LiDAR is shown in this figure. The yellow points are the perimeter around the plot, and the blue/green points are the points in the plot. With the plot being taller, the points registered were closer. The bluer the point, the closer it is.

environment map (using a voxel grid filter to average all points within a voxel) and create a Delaunay triangulation of this downsampled map. This triangulation gives us localized ground planes used for height estimations.

If there are parts of the environment that we know will always be trimmed and each part can be approximated as flat ground, we could use the proposed approach in Section VII to estimate both the height of crops and the height of the ground. In this case, the environment map is broken down into a grid where each grid cell covers a section of the trimmed area. Then, a localized ground plane model is found using the trimmed area to estimate the plane. With this, height estimates of non-ground points falling within the grid cell are measured using the localized ground plane.

Also, since we focus on growth (change in height), as long as the ground model is the same for each flight, we only need to focus on the relative changes. The relative growth of a "point" to a ground model will always be the same regardless of the ground model as long as that ground model does not change. Therefore, the measured growth will still be accurate even without a ground height estimation.

IX. CONCLUSION AND FUTURE WORK

In this work, we proposed an integrated pipeline that can be used for long-term, large-scale forage perception applications. From the perspective of simulation, we demonstrated how to simulate large pastureland environments reasonably fast using parallel processing. From the perspective of pasture prediction, we proposed a new deep learning architecture that can be used for long-term pasture predictions. From the perspective of perception, we demonstrated how to get accurate pasture height estimation through regression. From the perspective of autonomy, we combined predictions and an intermittent deployment policy to deploy robots with high accuracy while at a low cost.

This work resulted in novel approaches from the initial data-generating to the final deployment testing. The proposed pipeline offers a promising and cost-effective alternative to real-life experiments and can be used as a platform for other testings.

REFERENCES

[1] J. Bengtsson et al., "Grasslands—More important for ecosystem services than you might think," *Ecosphere*, vol. 10, no. 2, pp. 1–20, 2019.
[2] R. L. Kallenbach, "Describing the dynamic: Measuring and assessing the value of plants in the pasture," *Crop Sci.*, vol. 55, no. 6, pp. 2531–2539, Nov. 2015.
[3] J. Y. L. Tay, A. Erfmeier, and J. M. Kalwij, "Reaching new heights: Can drones replace current methods to study plant population dynamics?" *Plant Ecol.*, vol. 219, no. 10, pp. 1139–1150, Oct. 2018.
[4] K. R. Harmoney, K. J. Moore, J. R. George, E. C. Brummer, and J. R. Russell, "Determination of pasture biomass using four indirect methods," *Agronomy J.*, vol. 89, no. 4, pp. 665–672, Jul. 1997.
[5] *Gazebo Simulator*. [Online]. Available: http://gazebosim.org
[6] D. P. Holzworth et al., "APSIM—Evolution towards a new generation of agricultural systems simulation," *Environ. Model. Softw.*, vol. 62, pp. 327–350, Dec. 2014.
[7] M. T. Schaefer and D. W. Lamb, "A combination of plant NDVI and LiDAR measurements improve the estimation of pasture biomass in tall fescue (*Festuca arundinacea* var. Fletcher)," *Remote Sens.*, vol. 8, no. 2, p. 109, Feb. 2016.
[8] M. Rangwala et al., "DeepPaSTL: Spatio–temporal deep learning methods for predicting long-term pasture terrains using synthetic datasets," *Agronomy*, vol. 11, no. 11, p. 2245, Nov. 2021.
[9] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, vol. 24. Berlin, Germany: Springer, 2003.
[10] 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," *J. Field Robot.*, vol. 28, no. 5, pp. 714–741, 2011.
[11] S. McCammon et al., "Ocean front detection and tracking using a team of heterogeneous marine vehicles," *J. Field Robot.*, vol. 38, no. 6, pp. 854–881, Sep. 2021.
[12] J. Liu and R. K. Williams, "Submodular optimization for coupled task allocation and intermittent deployment problems," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3169–3176, Oct. 2019.
[13] S. Asseng et al., "Uncertainty in simulating wheat yields under climate change," *Nature Climate Change*, vol. 3, no. 9, pp. 827–832, Sep. 2013.
[14] F. Y. Li, P. C. D. Newton, and M. Lieffering, "Testing simulations of intra- and inter-annual variation in the plant production response to elevated CO_2 against measurements from an 11-year FACE experiment on grazed pasture," *Global Change Biol.*, vol. 20, no. 1, pp. 228–239, Aug. 2013.
[15] G. R. Balboa et al., "A systems-level yield gap assessment of maize-soybean rotation under high- and low-management inputs in the western U.S. corn belt using APSIM," *Agricult. Syst.*, vol. 174, pp. 145–154, Aug. 2019.
[16] R. M. Jones, R. J. Jones, and C. K. McDonald, "Some advantages of long-term grazing trials, with particular reference to changes in botanical composition," *Austral. J. Exp. Agricult.*, vol. 35, no. 7, pp. 1029–1038, 1995.
[17] M. S. Corson, R. H. Skinner, and C. A. Rotz, "Modification of the SPUR rangeland model to simulate species composition and pasture productivity in humid temperate regions," *Agricult. Syst.*, vol. 87, no. 2, pp. 169–191, Feb. 2006.
[18] R. H. Mohtar, T. Zhai, and X. Chen, "A world wide Web-based grazing simulation model (GRASIM)," *Comput. Electron. Agricult.*, vol. 29, no. 3, pp. 243–250, Dec. 2000.

[19] C. A. Rotz and U. S. Gupta, "DAFOSYM for windows: A teaching aid in forage system management," in *Proc. Amer. Forage Grassland Conf.*, 1996, pp. 1–2.

[20] D. Holzworth et al., "APSIM next generation: Overcoming challenges in modernising a farming systems model," *Environ. Model. Softw.*, vol. 103, pp. 43–51, May 2018.

[21] M. Ahmed et al., "APSIM next generation to model red clover under Nordic climate," in *Proc. Int. Crop Model. Symp.*, Feb. 2020, pp. 1–3.

[22] C. Bosi, P. C. Sentelhas, N. I. Huth, J. R. M. Pezzopane, M. P. Andreucci, and P. M. Santos, "APSIM-tropical pasture: A model for simulating perennial tropical grass growth and its parameterisation for palisade grass (*Brachiaria brizantha*)," *Agricult. Syst.*, vol. 184, Sep. 2020, Art. no. 102917.

[23] F. Y. Li, V. O. Snow, D. P. Holzworth, and I. R. Johnson, "Integration of a pasture model into APSIM," in *Proc. Austral. Agronomy Conf.*, 2010, pp. 6–10.

[24] C. K. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006, vol. 2, no. 3.

[25] J. Liu and R. K. Williams, "Data-driven models with expert influence: A hybrid approach to spatiotemporal process estimation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 2467–2473.

[26] M. Oliu, J. Selva, and S. Escalera, "Folded recurrent neural networks for future video prediction," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 716–731.

[27] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[28] A. Graves, "Generating sequences with recurrent neural networks," 2013, *arXiv:1308.0850*.

[29] M. Rangwala and R. Williams, "Learning multi-agent communication through structured attentive reasoning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 10088–10098.

[30] V. Michalski, R. Memisevic, and K. Konda, "Modeling deep temporal dependencies with recurrent grammar cells," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 27, 2014, pp. 1925–1933.

[31] N. Srivastava, E. Mansimov, and R. Salakhudinov, "Unsupervised learning of video representations using LSTMs," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 843–852.

[32] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-C. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 802–810.

[33] W. Lotter, G. Kreiman, and D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," 2016, *arXiv:1605.08104*.

[34] R. Villegas, J. Yang, S. Hong, X. Lin, and H. Lee, "Decomposing motion and content for natural video sequence prediction," 2017, *arXiv:1706.08033*.

[35] Y. Wang, M. Long, J. Wang, Z. Gao, and P. S. Yu, "PredRNN: Recurrent neural networks for predictive learning using spatiotemporal LSTMs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 879–888.

[36] Y. Wang, Z. Gao, M. Long, J. Wang, and S. Y. Philip, "PredRNN++: Towards a resolution of the deep-in-time dilemma in spatiotemporal predictive learning," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 5123–5132.

[37] R. M. Neal, *Bayesian Learning for Neural Networks*, vol. 118. New York, NY, USA: Springer, 2012.

[38] D. J. C. MacKay, "A practical Bayesian framework for backpropagation networks," *Neural Comput.*, vol. 4, no. 3, pp. 448–472, May 1992.

[39] P. Xu, J.-H. Cho, and A. Salado, "Expert opinion fusion framework using subjective logic for fault diagnosis," *IEEE Trans. Cybern.*, vol. 52, no. 6, pp. 1–12, Nov. 2020.

[40] P. Xu, A. Salado, and G. Xie, "A reinforcement learning approach to design verification strategies of engineered systems," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2020, pp. 3543–3550.

[41] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*.

[42] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 1278–1286.

[43] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *J. Mach. Learn. Res.*, vol. 14, no. 5, pp. 1–45, 2013.

[44] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1050–1059.

[45] P. Baldi and P. Sadowski, "Understanding dropout," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 26, 2013, pp. 2814–2822.

[46] A. Damianou and N. D. Lawrence, "Deep Gaussian processes," in *Proc. 16th Int. Conf. Artif. Intell. Statist.*, 2013, pp. 207–215.

[47] G. A. Hollinger and S. Singh, "Multirobot coordination with periodic connectivity: Theory and experiments," *IEEE Trans. Robot.*, vol. 28, no. 4, pp. 967–973, Aug. 2012.

[48] M. Gini, "Multi-robot allocation of tasks with temporal and ordering constraints," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 1–7.

[49] P. Tokekar and V. Kumar, "Visibility-based persistent monitoring with robot teams," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2015, pp. 3387–3394.

[50] T. Li, C. Wang, M. Q.-H. Meng, and C. W. De Silva, "Attention-driven active sensing with hybrid neural network for environmental field mapping," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 3, pp. 1–18, May 2021.

[51] G. A. Hollinger and G. S. Sukhatme, "Sampling-based robotic information gathering algorithms," *Int. J. Robot. Res.*, vol. 33, no. 9, pp. 1271–1287, Aug. 2014.

[52] L. V. Nguyen, S. Kodagoda, R. Ranasinghe, and G. Dissanayake, "Information-driven adaptive sampling strategy for mobile robotic wireless sensor network," *IEEE Trans. Control Syst. Technol.*, vol. 24, no. 1, pp. 372–379, Jan. 2015.

[53] Y. Xu, J. Choi, S. Dass, and R. Maiti, "Efficient Bayesian spatial prediction with mobile sensor networks using Gaussian Markov random fields," *Automatica*, vol. 49, pp. 3520–3530, Aug. 2013.

[54] J. Liu and R. K. Williams, "Optimal intermittent deployment and sensor selection for environmental sensing with multi-robot teams," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 1078–1083.

[55] J. Liu and R. K. Williams, "Monitoring over the long term: Intermittent deployment and sensing strategies for multi-robot teams," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 7733–7739.

[56] J. G. Oxley, *Matroid Theory*, vol. 3. New York, NY, USA: Oxford Univ. Press, 2006.

[57] R. K. Williams, A. Gasparri, and G. Ulivi, "Decentralized matroid optimization for topology constraints in multi-robot allocation problems," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 293–300.

[58] J. Liu and R. K. Williams, "Coupled temporal and spatial environment monitoring for multi-agent teams in precision farming," in *Proc. IEEE Conf. Control Technol. Appl. (CCTA)*, Aug. 2020, pp. 273–278.

[59] J. Liu and R. K. Williams, "A fast algorithm for robust action selection in multi-agent systems," 2022, *arXiv:2206.11824*.

[60] R. Wehbe and R. K. Williams, "Optimizing topologies for probabilistically secure multi-robot systems," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 6640–6646.

[61] A. Krause, A. Singh, and C. Guestrin, "Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies," *J. Mach. Learn. Res.*, vol. 9, no. 2, pp. 1–50, 2008.

[62] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Trans. Robot.*, vol. 25, no. 4, pp. 912–926, Aug. 2009.

[63] M. Conforti and G. Cornuéjols, "Submodular set functions, matroids and the greedy algorithm: Tight worst-case bounds and some generalizations of the Rado-Edmonds theorem," *Discretes Appl. Math.*, vol. 7, no. 3, pp. 251–274, 1984.

[64] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, "An analysis of approximations for maximizing submodular set functions—II," in *Polyhedral Combinatorics*. Berlin, Germany: Springer, 1978, pp. 73–87.

[65] S. Jorgensen, R. H. Chen, M. B. Milam, and M. Pavone, "The matroid team surviving orienteers problem: Constrained routing of heterogeneous teams with risky traversal," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 5622–5629.

[66] J. Liu, L. Zhou, P. Tokekar, and R. Williams, "Distributed resilient submodular action selection in adversarial environments," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 5832–5839, Jul. 2021.

[67] S. V. Archontoulis, F. E. Miguez, and K. J. Moore, "Evaluating APSIM maize, soil water, soil nitrogen, manure, and soil temperature modules in the midwestern United States," *Agronomy J.*, vol. 106, no. 3, pp. 1025–1040, May 2014.

[68] P. Hennig, "Animating samples from Gaussian distributions," Spemannstraße, Tübingen, Germany, Max Planck, Tech. Rep. 8, 2013.

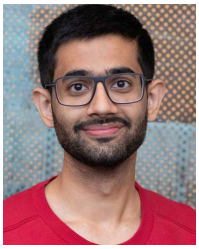
[69] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image. Comput. Comput. Assist. Interv.* Cham, Switzerland: Springer, 2015, pp. 234–241.

Jun Liu received the M.S. degree in electronic engineering from the Harbin Institute of Technology, China, in 2014. He is currently pursuing the Ph.D. degree in computer engineering with the Virginia Polytechnic Institute and State University, Blacksburg, VA, USA.

He was a Research Assistant at The Hong Kong Polytechnic University from 2014 to 2016. His research interests include multi-agent systems, dynamic networks, distributed systems, and mobile robotics.

Murtaza Rangwala received the bachelor's degree in electrical and electronics engineering from PES University, Bengaluru, India, and the M.S. degree in electrical engineering from Virginia Tech, Blacksburg, USA, where he is currently pursuing the Ph.D. degree in electrical engineering.

He comes from an automotive background, serving as an EE System Leader at Volvo Trucks and holds a patent in autonomous driving assistance systems. His current interests include multi-agent coordination and safe control policies using deep reinforcement learning.



Kulbir Singh Ahluwalia received the Bachelor of Technology degree in electrical engineering from the Punjab Engineering College, Chandigarh, India, in 2019, and the Master of Engineering degree in robotics from the University of Maryland, College Park, MD, USA, in 2021. He is currently pursuing the Ph.D. degree in computer science in the area of artificial intelligence with the University of Illinois at Urbana–Champaign.

His current research interests include natural language grounding and computer vision with applications for agricultural robots.

Shayan Ghajar received the M.S. degree in rangeland ecosystem science from Colorado State University in 2014, and the Ph.D. degree in crop and soil science from Virginia Tech in 2020.

His past research has included integrating native grasses and forbs into horse and beef pasture systems as well as using proximal sensing technologies for pasture monitoring. His current research interests include optimizing warm-season grazing options for Oregon, the potential use of remote and proximal sensing technologies in organic or low-input systems, and assessing the challenges faced by organic and low-input graziers with regard to pasture and forage management.

Harnaik Dhami received the B.S. and M.S. degrees in computer engineering from Virginia Tech in 2017 and 2019 respectively. He is currently pursuing the Ph.D. degree in computer science with the University of Maryland, College Park, MD, USA, supervised by Dr. Pratap Tokekar.

He was a member of the RAAS Laboratory. His research interests include applying computer vision techniques to solve problems in areas such as agriculture and infrastructure inspection through the use of drones and 3D LiDAR in both simulation and real world experiments.

Pratap Tokekar (Member, IEEE) received the Bachelor of Technology degree in electronics and telecommunication from the College of Engineering, Pune, India, in 2008, and the Ph.D. degree in computer science from the University of Minnesota in 2014.

From 2015 to 2019, he was an Assistant Professor at the Department of Electrical and Computer Engineering, Virginia Tech. Previously, he was a Post-Doctoral Researcher at the GRASP Laboratory, University of Pennsylvania. He is currently an Associate Professor with the Department of Computer Science and UMIACS, University of Maryland. He was a recipient of the NSF CAREER Award in 2020, Amazon Research Award in 2022, and CISE Research Initiation Initiative Award in 2016. He serves as an Associate Editor for the IEEE TRANSACTIONS ON ROBOTICS, and the ICRA and IROS Conference Editorial Board.

Benjamin Tracy received the B.S. degree from Rutgers University, the M.S. degree from Penn State University, and the Ph.D. degree from Syracuse University.

His research program focuses broadly on managing plant diversity in forage-livestock systems to improve their productivity. His research interests include the use of remote sensing technologies to estimate forage production and plant diversity in grasslands.

Ryan K. Williams (Member, IEEE) received the B.S. degree in computer engineering from the Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, in 2005, and the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 2014.

He is currently an Assistant Professor with the Electrical and Computer Engineering Department, Virginia Polytechnic Institute and State University, where he runs the Virginia Tech Laboratory for Coordination at Scale. His research interests include control, cooperation, and intelligence in distributed multiagent systems, topological methods in cooperative phenomena, and distributed algorithms for optimization, estimation, inference, and learning.

Dr. Williams was a recipient of the Viterbi Fellowship, the NSF CRII, and CAREER grants for young investigators. He was the Finalist for the Best Multirobot Paper at the 2017 IEEE International Conference on Robotics and Automation, and has been featured by various news outlets, including the L.A. Times.