Real-Time Learning of Signed Distance Function via Kernel Regression with Uncertainty Quantification

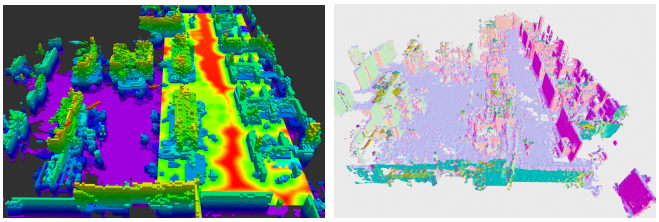
Zhirui Dai[1] Tianxing Fan[1] Mani Amani[1] Jaemin Seo[2]
Ki Myung Brian Lee[1] Hyondong Oh[2] Nikolay Atanasov[1]

***Abstract*—Accurate and efficient environment representation is crucial for robotic applications such as navigation, path planning, and manipulation. Signed distance function (SDF) has emerged as a powerful representation due to its ability to encode obstacle boundaries and provide distance information, useful for collision checking in motion planning and safety constraint specification in autonomous navigation. However, existing SDF learning methods face significant limitations: voxel-based approaches suffer from fixed resolution constraint and lack uncertainty quantification, neural network methods require high computational cost for training, and existing Gaussian process (GP) based methods struggle with scalability, sign estimation, and inconsistent uncertainty quantification. In this letter, we propose Kernel-SDF, which uses kernel regression to learn SDF with uncertainty quantification in real-time. Our approach consists of a surface estimation front-end that handles sensor noise and dynamic environment changes by using kernel regression in Hilbert space to learn the surface as a continuous occupancy field, and a back-end that learns accurate SDF using GPs, i.e., kernel regression in the functional space. Our method provides accurate SDF estimates, gradient predictions, uncertainty quantification, and online mesh construction at real-time rates. Evaluation results show that Kernel-SDF achieves superior accuracy compared to existing methods and outstanding real-time performance, making it suitable for various robotic applications requiring reliable environment representation with uncertainty awareness.**

I. Introduction

A good representation of the environment structure is crucial for many robotic applications, such as scene reconstruction, localization, collision checking, path planning, navigation, and manipulation. Such a representation should meet several requirements: 1) accuracy to reflect the environment structure, 2) computational efficiency to allow real-time processing, 3) scalability to incrementally handle large environments, 4) robustness to noise, 5) uncertainty quantification to support risk-aware decision making, 6) differentiability to support gradient-based optimization, 7) adaptability to dynamic environments, and 8) compatibility with different sensors.

Various representations have been proposed in the literature, including occupancy grids [1], point clouds [2]–[4], meshes [5], [6], and implicit functions such as signed distance function (SDF) [7]–[29], neural radiance fields (NeRF) [30], etc. Among these, SDF attracts much attention due to its ability to represent obstacle boundaries as its zero level set, which is useful for scene reconstruction [8] and localization [29], its distance information about the nearest obstacle, which is useful for collision checking and path planning [31], and its differentiability that is useful for gradient-based optimization in reconstruction [22], navigation [32], localization [29] etc.



(a) SDF visualization (b) Online reconstructed mesh

Fig. 1: An example of 3D SDF mapping using the proposed Kernel-SDF framework. (a) Visualization of the learned SDF as a heatmap, where warmer colors indicate higher SDF values (free space) and cooler colors indicate lower SDF values (occupied space), as well as the voxels (colored by height) that indicate the locations where local BHMs and GPs are instantiated. (b) The online mesh reconstructed in real-time from the surface estimation front-end of Kernel-SDF.

A wide range of methods have been proposed to learn distance functions from range sensors (LiDAR or depth camera). Voxel-based methods [7], [8], [11], [13], [18] mostly learn truncated SDF (TSDF), and some of them further estimate a discrete grid of SDF values [11], [13], [21], [25]. These voxel-based methods maintain a dense discrete approximation of TSDF or SDF with a fixed resolution, which suffers from the trade-off between accuracy and computational efficiency, and is lack of uncertainty quantification and differentiability. In contrast, Neural network (NN) based methods [12], [20], [22], [23], [26], [29] learn SDF as a continuous function, which is compact and achieves high-fidelity results. However, they are not suitable for online learning due to the high training cost to converge to a good solution and mostly learn TSDF.

On the other hand, Gaussian process (GP) [33] based frameworks [14], [15], [24], [27], [28] are non-parametric and capable of building accurate Euclidean distance function (EDF). However, previous GP-based methods suffer from several challenges: 1) high computational cost as the scale increases, 2) lack of robust distance sign estimation, and 3) inconsistency between the GP variance and the actual SDF prediction error. To address the first issue, Octree [15] or OpenVDB [28] are used to split the training data into smaller chunks for training multiple GPs. GMMGP [27] tries to recover the sign and quantify prediction error by using SDF prior obtained from hierarchical Gaussian mixture model and GP. However, the SDF prior used by GMMGP has larger errors as the query position moves away from the surface.

[1]Department of Electrical and Computer Engineering, University of California, San Diego, La Jolla, CA 92093 USA. Email: {zhdai,t2fan,kmblee,mamai5250,natanasov}@ucsd.edu

[2]Department of Mechanical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34051, Republic of Korea. Email: {jam.seo,h.oh}@kaist.ac.kr

In this letter, we propose a method that couples occupancy spatial hierarchical tree (e.g. quadtree for 2D or octree for 3D) with Bayesian Hilbert map (BHM) [34] and Gaussian process (GP) [33] to learn SDF that meets the mentioned requirements. We present an example of learning SDF with our method in Fig. 1, which shows the hierarchical tree structure for storing multiple BHMs and GPs, the reconstructed mesh, and the SDF prediction. Kernel-SDF is capable of incrementally building a differentiable SDF in real-time with range sensors (e.g. LiDAR or depth camera). Kernel-SDF consists of a surface estimation front-end and an SDF prediction back-end. The front-end uses multiple BHMs, which are kernel regression in Hilbert space, to estimate the surface as a continuous occupancy field, which provides robust sign estimation. Besides, marching algorithms are used to extract surface points from the occupancy field as training data for the back-end. The back-end uses multiple GPs, which are kernel regression in functional space, to learn SDF with uncertainty quantification. To enable real-time performance, our method employs a spatial hierarchical tree and priority queues for efficient storage and updates of both the BHMs and GPs.

Our method is capable of providing accurate SDF estimates, gradient prediction, uncertainty quantification, and online mesh construction at real-time rates. The contribution of this letter is as follows:

1) an open-source, real-time extensively-tested C++ implementation that can be used in various robotic applications, supporting 2D and 3D environments with interfaces for Python and ROS1/ROS2[1];
2) a method that couples BHM and GP with spatial hierarchical tree to build a differentiable SDF representation of the environment with uncertainty quantification;
3) a robust surface estimation front-end that supports both depth camera and LiDAR sensors, handling sensor noise and dynamic environment changes;
4) an efficient SDF prediction back-end that accurately learns SDF and predicts SDF gradients with uncertainty quantification.

II. Related Work

Existing methods for learning SDF related to this letter can be roughly categorized into three groups: voxel-based methods, neural network (NN) based methods, and Gaussian process (GP) based methods.

A. Voxel-based Methods

Many frameworks have been proposed to learn SDF with grids as the underlying data structure [7]. KinectFusion [8] and so on [9], [10], [18] propose to learn projective TSDF, which is inaccurate due to the projection. VoxField [19] learns non-projective TSDF, which is more accurate but still not suitable for tasks that require full distance information about the nearest obstacle. Voxblox [21] and VoxField [19] extend TSDF to SDF by breath-first-search (BFS) based integration from the occupied voxels. FIESTA [13] chooses to do the integral with occupancy map, but also reveals that estimated SDF values are still inaccurate due to the BFS integration. What's worse, the SDF estimates are maintained in a grid with fixed resolution, which makes it unscalable for larger environments and demand of higher resolution. VDBblox [21] and nvblox [25] try to address the scalability issue by using OpenVDB [35] or GPU-based voxel hashing. However, these voxel-based methods still cannot provide uncertainty quantification and are indifferentiable. Hence, we propose to use Gaussian process (GP) to learn SDF such that the resolution limitation is eliminated and differentiability is provided.

B. Neural Network Based Methods

Neural Network (NN) based methods become popular because they provide differentiability natively and have the potential to learn high-fidelity SDF thus better surface reconstruction, as shown by DeepSDF [12], which uses an MLP with latent features to learn SDF of objects. NN-based methods show various applications of SDF. NeuS [16] and NeuS2 [22] combine SDF with neural radiance fields (NeRF) [30] to learn SDF and rendering jointly. PIN-SLAM [26] and MISO [29] implement simultaneous localization and mapping (SLAM) based on learning global consistent SDF map with neural features. H2-Mapping [23] introduces voxel-based method to provide SDF prior for neural TSDF learning, which is co-optimized with volume rendering. However, most NN-based methods are far behind the real-time performance due to the high training cost. Although some works like H2-Mapping [23] and PIN-SLAM [26] manage to learn neural SDF online, they sacrifice the SDF accuracy very much. In constrast, Gaussian process (GP) is non-parametric so that is suitable for online learning.

C. Gaussian Process Based Methods

However, GP-based methods still face some challenges. One main challenge is the high computational cost as the scale increases due to the matrix inversion in the GP regression. GPIS [14] presents an online GP-based framework that splits the training data into smaller chunks with octree and trains multiple GPs for each chunk. However, the SDF estimation drops to the zero-mean prior quickly as the query position moves further away from obstacles. Log-GPIS [15] exploits the connection between the heat kernel and unsigned distance function (UDF), and improves GPIS by learning UDF in the log space, which predicts UDF accurately. However, the transformation to log space causes the lose of sign information, which is crucial for SDF, and distorts the variance estimation that the GP variance explodes when the query position is far away from the surface. Besides, the proposed surface estimation method that generates training data for GPs is prone to sensor noise and dynamic changes in the environment. VDB-GPDF [28] tries to improve the surface estimation by using OpenVDB-based TSDF fusion with locally trained Log-GPIS but does not address issues of sign prediction and uncertainty quantification. Thus, we propose our Kernel-SDF framework that aims to address all these issues. The BHM-based surface estimation front-end is capable of handling dynamic changes in the environment and sensor noise, and providing cleaner

[1]Available at https://github.com/ExistentialRobotics/erl_gp_sdf

surface estimation and sign prediction via the continuous occupancy field [34], [36]. Then, the back-end learns SDF with multiple GPs and softmin-based uncertainty estimation, which predict SDF and its gradient with uncertainty quantification.

III. PROBLEM STATEMENT

Consider a robot in an n-dimensional environment $\mathcal{O} \subset \mathbb{R}^n$ ($n = 2, 3$). The robot is equipped with a sensor, such as a LiDAR or a depth camera, that yields a stream of range measurements $\{\mathcal{S}_t\}_{t=1}^T = \{\mathbf{R}_t, \mathbf{o}_t, \mathcal{P}_t\}_{t=1}^T$, where $\mathbf{R}_t \in SO(n)$ and $\mathbf{o}_t \in \mathbb{R}^n$ are the orientation and position of the sensor at time t, and $\mathcal{P}_t$ the point cloud of measurements in the sensor frame. Our objective is to learn an SDF representation of the environment $\mathcal{O}$:

$$d(\mathbf{x}) = \begin{cases} \min_{\mathbf{y}\in\partial\mathcal{O}} \|\mathbf{x}-\mathbf{y}\|_2, & \mathbf{x} \notin \mathcal{O}, \\ -\min_{\mathbf{y}\in\partial\mathcal{O}} \|\mathbf{x}-\mathbf{y}\|_2, & \mathbf{x} \in \mathcal{O}. \end{cases} \tag{1}$$

SDFs have two important properties: 1) the object surface $\partial\mathcal{O}$ can be recovered as the zero-level set of the SDF; and 2) the gradient is of unit norm whenever it is differentiable:

$$d(\mathbf{x}) = 0 \ \forall \mathbf{x} \in \partial\mathcal{O}, \text{ and } \|\nabla d(\mathbf{x})\|_2 = 1 \text{ a.e. } \mathbf{x} \in \mathcal{O}. \tag{2}$$

As we mentioned in Sec. II-C, existing GP-based SDF learning methods face several challenges, including high computational cost as the scale increases, lack of robust distance sign estimation, and inconsistency between the GP variance and the actual SDF prediction error. However, as a non-parametric method, GP is suitable for online learning. Hence, in this letter, we propose Kernel-SDF to address these challenges and learn SDF in real-time with uncertainty quantification. As shown in Fig. 2, our framework consists of a surface estimation front-end and a SDF prediction back-end. The front-end, which is presented in Sec. V, takes in the sensor observation and estimates the surface points $\mathcal{D} = \{\mathbf{x}_i \in \partial\mathcal{O}\}_{i=1}^N$, which are then used as training data for the GPs . Then, the back-end utilizes the GPs to estimate the SDF, the SDF gradient, and the SDF variance, which is described in Sec. IV.

IV. SDF PREDICTION BACKEND

To yield an SDF prediction, the SDF prediction backend utilizes the surface points with variance $\mathcal{D}_{\text{surf}} = \{\mathbf{x}_i, \sigma^2_{\mathbf{x}_i}\}_{i=1}^N$ estimated by the surface estimation front-end, which will be described later in Sec. V. Given that the occupancy field by the Bayesian Hilbert map (BHM) front-end provides a robust sign prediction, in this section, we focus on learning the unsigned distance function (UDF) from $\mathcal{D}_{\text{surf}}$ with GPs and estimating its uncertainty from the surface point uncertainty.

Same as the main ideas of [15], [24] that it is easier to learn a surrogate function f as a GP, such that the unsigned distance function $|\hat{d}|$ is obtained through a nonlinear transform $|\hat{d}(\mathbf{x})| = r(\hat{f}(\mathbf{x}))$. Such a surrogate function should be monotonically decreasing with respect to the distance $|d(\mathbf{x})|$ to the nearest surface point, such that $f \to 0$ as $|d(\mathbf{x})| \to \infty$ and $f \to 1$ as $|d(\mathbf{x})| \to 0$, which is consistent with zero-mean prior GP regression that as the query point goes far away from the training set, the posterior mean goes to zero.

TABLE I: Kernel functions and their nonlinear transforms.

Kernel	$k_S(r)$	$r(\hat{f})$
RBF	$\exp\left(-\frac{r^2}{2l^2}\right)$	$\sqrt{-2l^2 \log \hat{f}}$
Matérn 3/2	$\left(1+\frac{\sqrt{3}r}{l}\right)\exp\left(-\frac{\sqrt{3}r}{l}\right)$	$-\frac{l}{\sqrt{3}}\log \hat{f}$

Hence, given the sign prediction $\text{sign}(\mathbf{x})$ based on the occupancy field from the front-end, we can obtain the SDF prediction and its gradient from the surrogate function as:

$$\begin{aligned} \hat{d}(\mathbf{x}) &= \text{sign}(\mathbf{x}) r(\hat{f}(\mathbf{x})), \\ \nabla\hat{d}(\mathbf{x}) &= -\text{sign}(\mathbf{x}) \nabla\hat{f}(\mathbf{x}) / \|\nabla\hat{f}(\mathbf{x})\|_2. \end{aligned} \tag{3}$$

A. Regression of Surrogate

The surrogate function $\hat{f}$ is predicted from the surface point observations by modeling it as a log-GP $f \sim GP(0, k(\mathbf{x}, \mathbf{x}'))$ with a stationary kernel $k(\mathbf{x}, \mathbf{x}') = k_S(\|\mathbf{x} - \mathbf{x}'\|)$ having observations $\mathbf{y} = k_S(\mathbf{0}) = \mathbf{1}$ on the surface points:

$$\begin{aligned} \begin{bmatrix} \hat{f}(\mathbf{x}_*) \\ \nabla\hat{f}(\mathbf{x}_*) \end{bmatrix} &= \begin{bmatrix} \mathbf{k}_*^\top \\ \nabla_{\mathbf{x}_*}^\top \mathbf{k}_* \end{bmatrix} (\mathbf{K} + \mathbf{\Sigma}_y)^{-1} \mathbf{1}, \\ \mathbb{V}[\hat{f}(\mathbf{x}_*)] &= k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top (\mathbf{K} + \mathbf{\Sigma}_y)^{-1} \mathbf{k}_*, \end{aligned} \tag{4}$$

where $\mathbf{k}_* \in \mathbb{R}^N$ and $\mathbf{K} \in \mathbb{R}^{N\times N}$ are calculated by:

$$k_{*,i} = k(\mathbf{x}_i, \mathbf{x}_*) \quad 1 \le i \le N, \tag{5}$$

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) \quad 1 \le i \le N, 1 \le j \le N. \tag{6}$$

We call it log-GP because the observations are set to be 1 on the surface points, which corresponds to $\log(1) = 0$ in the log-space of distance function.

B. Choice of Nonlinear Transform

The nonlinear transform r is given by the inverse of the stationary kernel as $r(\hat{f}) = k_S^{-1}(\hat{f})$. Table I shows the nonlinear transforms for commonly used kernels. For RBF and Matérn 3/2 kernels, it can be shown that this choice of nonlinear transform coincides with Varadhan's distance formula, and hence asymptotically approximates the true distance function with smaller lengthscale [15], [37]. Even in other cases, strong empirical performance can be achieved [24].

C. Uncertainty Quantification

However, the resulting variance $\mathbb{V}[\hat{d}]$ based on the first-order approximation of this nonlinear transform does not reflect the actual estimation error correctly.

Consider a function $\mathbf{y} = f(\mathbf{x}), \mathbf{x} \in \mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$ and its Taylor expansion $\mathbf{y} \approx f(\boldsymbol{\mu}) + \nabla_{\mathbf{x}} f(\boldsymbol{\mu})^\top (\mathbf{x} - \boldsymbol{\mu})$, the covariance of $\mathbf{y}$ can be approximated as

$$\text{Cov}[\mathbf{y}] \approx \nabla_{\mathbf{x}} f(\boldsymbol{\mu})^\top \mathbf{\Sigma} \nabla_{\mathbf{x}} f(\boldsymbol{\mu}). \tag{7}$$

Therefore, to calculate SDF variance at $\mathbf{x}_*$, one idea is to approximate $\mathbb{V}[\hat{d}]$ via Taylor expansion of the nonlinear transform r, which is

$$\mathbb{V}[\hat{d}] \approx \left(dr/d\hat{f}\right)^2 \mathbb{V}[\hat{f}]. \tag{8}$$

However, when $\mathbf{x}_*$ is further away from the training set $\mathcal{D}_{\text{surf}}$, $\mathbb{V}[\hat{f}] \to 1$, and $|dr/d\hat{f}| \to \infty$ such that $\mathbb{V}[\hat{d}] \to \infty$. This approximation does not provide a useful estimation variance that

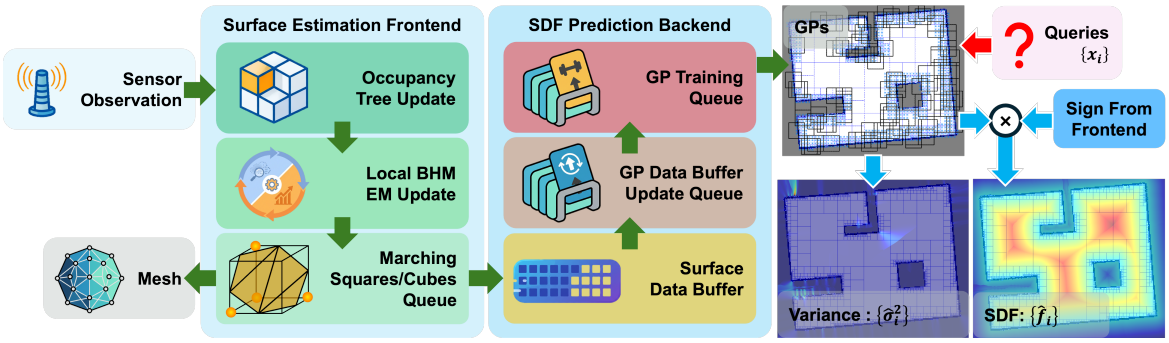


Fig. 2: Overview of the proposed Kernel-SDF framework. The front-end estimates surface points and normals from sensor observations using Bayesian Hilbert Map (BHM) and marching squares/cubes algorithm. The estimated surface points are then used as training data for the GPs in the back-end. The back-end learns the SDF using multiple GPs, which predict the SDF and its gradient with uncertainty quantification computed via softmin-based fusion.

reflects the actual estimation error correctly. Thus, although the input uncertainty can be propagated to the output uncertainty via GP, it does not work well in the log-GP framework for SDF estimation.

In Appendix IX-C, we show that $r(\hat{f}) = k_S^{-1}(\hat{f})$ is approximately a type of soft-min function of the distances to the surface points such that it can accurately estimate the shortest distance, which leads to $r(\hat{f})$ presented in Table I.

Based on this observation, we can estimate the SDF variance by propagating the uncertainty in the surface point locations to the uncertainty in the distance approximation via the soft-min formulation. First, we approximate the shortest distance with soft-min and the training set $\mathcal{D}_{\text{surf}} = \{\mathbf{x}_i, \sigma^2_{\mathbf{x}_i}\}_{i=1}^N$ as:

$$\begin{aligned} |d(\mathbf{x}_*)| &\approx h(\mathbf{x}_*, \{\mathbf{x}_i\}_{i=1}^N) = \mathbf{s}^\top \mathbf{z}, \\ z_i &= \|\mathbf{x}_i - \mathbf{x}_*\|_2 \quad s_i = \frac{\exp(-\alpha z_i)}{\sum_i^N \exp(-\alpha z_i)}, \alpha > 0. \end{aligned} \tag{9}$$

Then, assuming deterministic sign and united variance $\mathbb{V}_i = \sigma^2_{\mathbf{x}_i}$ for each dimension of $\mathbf{x}_i$, we can calculate the variance of $d(\mathbf{x}_*)$ and $\nabla d(\mathbf{x}_*)$ approximately by

$$\mathbb{V}[d(\mathbf{x}_*)] \approx \sum_{i=1}^N \|\nabla_{\mathbf{x}_i} h\|_2^2 \mathbb{V}_i, \tag{10}$$

$$\mathbb{V}[\nabla_k d(\mathbf{x}_*)] \approx \sum_{i=1}^N \left\|\nabla_{\mathbf{x}_i} g_k(\mathbf{x}_*, \{\mathbf{x}_i\}_{i=1}^N)\right\|_2^2 \mathbb{V}_i, \tag{11}$$

where $\nabla_{\mathbf{x}_i} h = -s_i(\alpha \mathbf{s}^\top \mathbf{z} - \alpha z_i + 1)(\mathbf{x}_* - \mathbf{x}_i)/z_i$, $\mathbf{g}(\mathbf{x}_*, \{\mathbf{x}_i\}_{i=1}^N) = \nabla_{\mathbf{x}_*} h(\mathbf{x}_*, \{\mathbf{x}_i\}_{i=1}^N)$. For more details, please check Appendix IX-D. The merit of this approximation is that the uncertainty in the surface point locations are directly propagated to the uncertainty in the distance approximation.

V. Surface Estimation Frontend

The role of the surface estimation front-end is to provide reliable estimates of on-surface points, their uncertainty, and optionally the corresponding normals. We use the Bayesian hilbert map (BHM) [34] for this purpose, although other front-ends may also be used.

BHM represents a continuous occupancy field $P(y \mid \mathbf{x})$ by using a set of M hinge points $\tilde{\mathbf{X}}$ with weights $\mathbf{w}$ to do kernel regression in the Hilbert space:

$$P(y \mid \mathbf{x}, \mathbf{w}) = \sigma\big((2y-1)\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x})\big), \tag{12}$$

where $\boldsymbol{\phi}(\mathbf{x}) = [k(\mathbf{x}, \tilde{\mathbf{x}}_1), \cdots, k(\mathbf{x}, \tilde{\mathbf{x}}_M), 1]^\top$ is a vector of kernel values, $k(\cdot,\cdot)$ is RBF kernel, and σ is the sigmoid function. A probabilistic estimate of the weight $\mathbf{w}$ is maintained as a Gaussian random variable $\mathbf{w} \sim P(\mathbf{w}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, so that the occupancy probability is given by:

$$P(y \mid \mathbf{x}) = \int P(y \mid \mathbf{x}, \mathbf{w}) P(\mathbf{w}) d\mathbf{w}. \tag{13}$$

A. BHM Update

To create or update the BHM, a dataset $\mathcal{D}_{\text{BHM}} = \{\mathbf{x}_k, y_k\}_{k=1}^K$ is generated from the sensor measurement $\mathcal{S}_t = (\mathbf{R}_t, \mathbf{o}_t, \mathcal{P}_t)$ by sampling occupied samples as $(\mathbf{R}_t\mathbf{p}_i + \mathbf{o}_t, 1)$ and free-space samples as $(\lambda\mathbf{R}_t\mathbf{p}_i + \mathbf{o}_t, 0)$ for $\lambda \sim \mathcal{U}(0,1)$ and $\forall \mathbf{p}_i \in \mathcal{P}_t$. Given the dataset $\mathcal{D}_{\text{BHM}}$, the update is performed using an expectation-maximization (EM) iteration by forming a lower bound around an output ξ_n. The E-step updates the mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ of $\mathbf{w}$ as:

$$\boldsymbol{\Sigma}_t^{-1} = \boldsymbol{\Sigma}_{t-1}^{-1} + \sum_{k=1}^K \left|\frac{1/2 - \sigma(\xi_k)}{\xi_k}\right| \boldsymbol{\phi}_k \boldsymbol{\phi}_k^\top, \tag{14}$$

$$\boldsymbol{\mu}_t = \boldsymbol{\Sigma}_t \left[\boldsymbol{\Sigma}_{t-1}^{-1}\boldsymbol{\mu}_{t-1} + \sum_{k=1}^K (y_k - 1/2)\boldsymbol{\phi}_k\right], \tag{15}$$

and the M-step updating the lowerbound parameter ξ as:

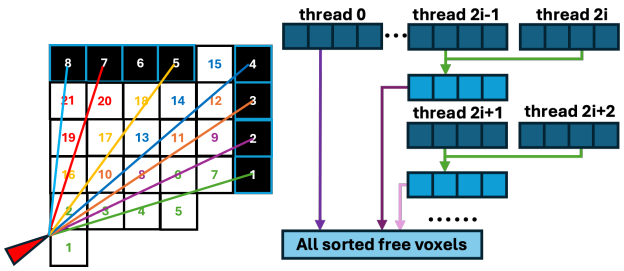
$$\xi_k = \sqrt{\boldsymbol{\phi}_k^\top \boldsymbol{\Sigma}_t \boldsymbol{\phi}_k + \left(\boldsymbol{\phi}_k^\top \boldsymbol{\mu}_t\right)^2} \tag{16}$$

with ξ_k emperically initialized to 0 or 1 for all k. $\boldsymbol{\Sigma}$ is initialized to $10^5\mathbf{I}$ and $\boldsymbol{\mu}$ is set to $\mu\mathbf{1}, \mu > 0$, indicating a prior of being occupied everywhere. For readers interested in the detailed derivation, please refer to Appendix IX-A.

B. Sign Prediction and Surface Point Extraction

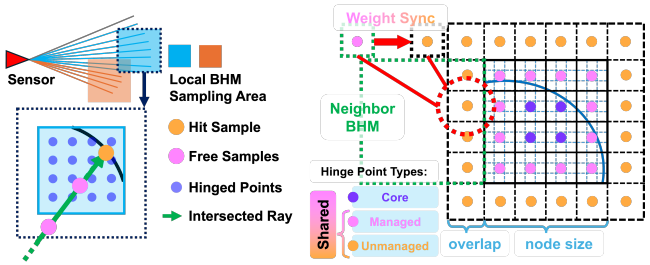
Given the estimated BHM parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, we can predict the occupancy probability efficiently as presented in Appendix IX-B1. However, for this work, we only need to know the log-odds of occupancy at a query point $\mathbf{x}_*$ as $l_{\text{BHM}}(\mathbf{x}_*) = \boldsymbol{\mu}^\top \boldsymbol{\phi}_*$, which can be used to predict the sign at $\mathbf{x}_*$ as:

$$\text{sign}(\mathbf{x}_*) = \begin{cases} +1, & l_{\text{BHM}}(\mathbf{x}_*) < \epsilon, \\ -1, & \text{otherwise}, \end{cases} \tag{17}$$



(a) Voxel update order (b) Free voxel parallel generation

Fig. 3: Optimization applied to construct the tree efficiently. (a) we update the free/occupied voxels in a sorted order that follows the ray order, which has a higher CPU cache hit rate. (b) to generate the proposed ordering, a parallelism with stride-2 reduction is used, where each thread computes voxels to update for its batch of rays (in parallel), then each odd thread $(2i-1)$ merges results from its paired even thread $(2i)$ (parallel stride-2 reduction), and finally all the results from the odd threads are merged into thread 0 one by one (sequential).



(a) Sample generation for BHMs (b) Weight sync between BHMs

Fig. 4: (a) Sample generation for BHMs. For each ray that hits within or intersects with the sampling region of a BHM, we generate occupied and free samples for updating the BHM. (b) Weight sync between neighboring BHMs. The hinged points near the boundary of a BHM are also hinged points of its neighboring BHMs. Hence, when updating the BHM parameters, we sync the weights of these shared hinged points between neighboring BHMs to guarantee the consistency of occupancy prediction.

where $\epsilon \in \mathbb{R}$ is the decision boundary that can be learned from data online with $\mathcal{S}_t$ using exponential moving average:

$$\epsilon \leftarrow (1-\alpha)\epsilon + \frac{\alpha}{K}\sum_{k=1}^{K} \boldsymbol{\mu}_t^\top \boldsymbol{\phi}_k, \quad \alpha \in (0,1], \tag{18}$$

where K is the number of hit samples in $\mathcal{S}_t$. More discussion about the sign prediction can be found in Appendix IX-B2.

To recover the surface points $\{\mathbf{x}_i\}_i$, we run marching squares/cubes algorithm [38] on a grid of log-odds values predicted by the BHM to extract the ϵ-level set. The marching process is performed only at hit grid cells and their neighbor cells to improve efficiency. For each extracted surface point $\mathbf{x}_i$, we compute its uncertainty as:

$$\sigma^2_{\mathbf{x}_i} = \beta |l_{\text{BHM}}(\mathbf{x}_i) - \epsilon|, \tag{19}$$

where $\beta > 0$ is a hyperparameter and ϵ is the surface logodds learned online as (18). Optionally, we can also compute the surface normal at $\mathbf{x}_i$ from the gradient of $l_{\text{BHM}}(\mathbf{x})$ as shown in Appendix IX-B3.

VI. Hierarchical Storage of Occupancy and SDF

The computational complexity of kernel-based continuous surface mapping methods scales quadratically (2D) or cubically (3D) as the size of the environment grows. To mitigate this, we propose a hierarchical data structure built on occupancy quadtrees/octrees to subdivide the workspace, in order to bound the computational complexity. The main idea is to build local submaps and GPs that correspond to occupied nodes at particular depths of the tree. And for free space, we only need to store the occupancy log-odds values for sign prediction in the nodes without building local submaps, which significantly reduces the number of local submaps and the overall computational complexity.

A. Octree Construction

We define an occupancy spatial hierarchical tree as a rooted tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ of max depth D_{tree} and resolution r. Each voxel $v \in \mathcal{V}$ at depth d is indexed by its discrete coords $\mathbf{k} \in \mathbb{Z}^n_{\geq 0}$, centered at $\mathbf{c}(v) = s_v \lfloor (\mathbf{k} - 2^{D_{\text{tree}}-1})/2^{D_{\text{tree}}-d} \rfloor + 0.5 s_v$ with size $s_v = r 2^{D_{\text{tree}}-d}$, and stores a log-odds value $l_{\text{tree}}(v)$. Therefore, at position $\mathbf{x}_*$, we obtain the log-odds by:

$$l(\mathbf{x}_*) = \begin{cases} l_{\text{BHM}}(\mathbf{x}_*), & \exists \mathbf{c}, \|\mathbf{c} - \mathbf{x}_*\|_\infty < r 2^{D_{\text{tree}} - D_{\text{BHM}} - 1}, \\ l_{\text{tree}}(v_*), & \text{otherwise}, \end{cases} \tag{20}$$

where $\mathbf{c}$ is the center of a local BHM and v_* is the leaf voxel that contains $\mathbf{x}_*$.

It is critical to make the tree construction with occupancy update efficient and fast. Based on Octomap [1], we extend the octree implementation to a quadtree implementation for 2D specifically. Further, the implementation is deeply optimized to make sure the occupancy update is fast. As shown in Fig. 3a, we update the voxels in a sorted along-ray order, which is critical to make the update more efficient because when a free voxel is updated, its neighboring voxels are more likely to be updated. This locality of voxel update is essential to achieve a higher CPU cache hit rate for better performance. While the order of occupied voxels is obvious, we implement an algorithm demonstrated in Fig. 3b to efficiently generate the free voxels in the sorted order.

B. Management of Local BHMs

For each occupied voxel at depth $D_{\text{BHM}} < D_{\text{tree}}$, we build a local BHM for surface estimation. Each local BHM only covers a small area near the surface, which significantly reduces the computational complexity compared to building a global BHM for the whole environment. We define a local BHM $\mathcal{B}$ as a tuple $\mathcal{B} = (\boldsymbol{\mu}, \boldsymbol{\Sigma}, \{\tilde{\mathbf{x}}_i\}_{i=1}^M, \mathbf{c}, s_{\mathcal{B}})$, where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the BHM parameters, $\{\tilde{\mathbf{x}}_i\}_{i=1}^M$ are the hinged points, $\mathbf{c} \in \mathbb{R}^n$ is the center position, and $s_{\mathcal{B}} \in \mathbb{R}_{>0}$ is the size of the BHM. The $M = m^n$ hinged points are uniformly placed in the BHM with spacing $\Delta = s_{\mathcal{B}}/m$.

While the splitting of the environment into multiple local BHMs significantly reduces the computational complexity, it also brings the issue of maintaining the consistency between neighboring BHMs. To address this, we propose to overlap neighboring BHMs by the same margin Δ as hinge point

spacing, so that the surface near the boundary of a BHM can be better captured and synced with neighbors. So, the map size is set to $s_{\mathcal{B}} = r2^{D_{\text{tree}}-D_{\text{BHM}}} + 2\Delta$. To ensure accurate updates of weights near the boundary, we consider a sampling area of size αs centered at $\mathbf{c}$ with $\alpha > 1$ when generating the dataset $\mathcal{D}_{\text{BHM}}$ from the sensor measurement $\mathcal{S}_t$ for updating a BHM, as shown in Fig. 4a.

The overlapping regions between BHMs must be consistent. Therefore, we propose a weight sync strategy between neighboring BHMs. Three different cases must be considered, as shown in Fig. 4b: (1) core weights that are not shared with any neighboring BHMs, (2) managed weights that are shared with neighboring BHMs but owned by the current BHM, and (3) unmanaged weights that are shared with neighboring BHMs but owned by the neighboring BHMs. When updating the BHM parameters, we sync the managed weights to the corresponding unmanaged weights of neighboring BHMs to guarantee the consistency of occupancy prediction. In addition to improving the consistency, this weight sync strategy also helps the convergence of BHMs since the learned surface geometry is broadcasted to neighboring BHMs.

C. Management of Local Log-GPs

In addition to creating local BHM for the occupied voxels at depth D_{BHM}, we also train the corresponding local GP for SDF prediction according to Sec. IV. Similarly, the splitting of a single global GP into multiple local GPs significantly reduces the computational complexity. To guarantee the consistency of SDF prediction between neighboring GPs, we overlap the bounding boxes for collecting surface points for training the local GPs by setting the box size to $s_{GP} = \beta s_{\mathcal{B}}, \beta > \alpha$. With the consistency of prediction between neighboring GPs, we can simply fuse the SDF predictions from the K nearest multiple GPs by taking the minimum absolute value as:

$$\hat{d}(\mathbf{x}_*) = \text{sign}(\mathbf{x}_*) \min_k r\left(\hat{f}_k\right), \tag{21}$$

where $\hat{f}_k$ is the log-distance prediction from the k-th GP. And the corresponding uncertainty is computed with the selected GP's training set $\mathcal{D}_{k_*}^{\text{surf}}$ according to Sec. IV-C. To efficiently find the GPs for fusion, we maintain a Kd-tree of the positions of all local GPs, for which the position is defined as the running meaning of surface points used for training the GP. Then, at a query point $\mathbf{x}_*$, we can efficiently find the K nearest GPs for fusion.

D. Priority Based Update

To get real-time performance, the update and test of BHMs and GPs must be done in an efficient manner. The main idea is to delay the surface extraction from BHMs, the update of GP training data buffers, and the training of GPs until necessary, so that we can avoid unnecessary computation for BHMs and GPs. To make sure the system runs in real-time but also remains responsive to the latest observation and query without sacarificing accuracy, we introduce three priority queues for managing the update of BHMs and GPs, as shown in Fig. 2. Basically, frequently queried GPs get higher priority to be updated and re-trained. These GPs' data buffers thus get updated first. Besides, BHMs in the queue with the oldest marching request timestamps are processed first because they are more likely to be unchanged in the near future. Due to the limited space, we leave the details in Appendix IX-E

VII. Experiments

In this section, we compare Kernel-SDF with four baselines: Voxblox [11], FIESTA [13], iSDF [20], and VDB-GPDF [28]. We quantitatively evaluate the quality of mesh reconstruction and the accuracy of SDF prediction on three datasets: the Replica dataset [39], the Cow and Lady dataset [11], and the Newer College dataset [40]. We use the axial noise model $\sigma = kz^2(k = 0.0025)$ [41] to add per pixel noise to the synthesized depth images of the Replica dataset. For the detailed experiment setup and metric definition, please refer to Appendix IX-F. We first compare the mesh reconstruction quality of all methods in Sec. VII-A. Then, we compare SDF prediction accuracy in Sec. VII-B. Finally, we dicuss the computational time of all methods (Sec. VII-C), and verify the consistency between the SDF error and the SDF variance estimated by our method in Sec. VII-D. In addition, we demonstrate a real-world robot path planning application using Kernel-SDF in Sec. VII-F.

A. Reconstruction Accuracy

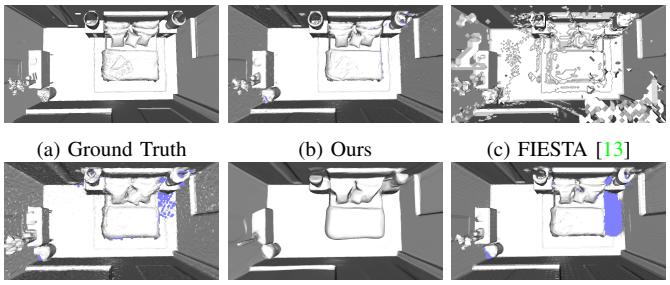
As shown in Figures 5, 6, and 7, our method produces high-quality mesh reconstructions that closely resemble the ground truth models across all three datasets. In contrast, the baseline methods exhibit various artifacts and inaccuracies in their reconstructions. For instance, when tested on the Replica dataset with relatively low noise, Voxblox [11], FIESTA [13] and VDB-GPDF [28] tend to produce block-like artifacts due to their voxel-based representations, while iSDF shows over-smooth mesh results. However, as the sensor noise increases in real world datasets like Cow and Lady and Newer College, the performance of these baselines degrades significantly, resulting in incomplete or distorted meshes. For example, due to the discrete occupancy grid used by FIESTA, the extracted meshes have low completeness, missing fine details. Voxblox and VDB-GPDF also have the similar issues due to their reliance on regular grids for the near surface representation. iSDF, while shows better noise robustness, still produces over-smoothed meshes that lack detail. We also quantitatively evaluate the mesh reconstruction quality and summarize the results in Table II. The results show that our method achieves the best or the second-best performance across almost all metrics and datasets. Overall, our method consistently outperforms these baselines by effectively handling sensor noise and preserving fine details in the reconstructed meshes.

B. SDF Accuracy

We evaluate the SDF prediction accuracy of all methods using the Mean Absolute Error (MAE) metric for both SDF values and SDF gradients. We further categorize the evaluation points into three regions: "All" points in the environment,

TABLE II: Mesh reconstruction metrics on the Replica dataset [39], the Cow and Lady dataset (C.L. for the scene and Cow for the cow only) [11], and the Newer College (N.C.) dataset [40]. When computing the ratio, $\delta = 5$cm, except for the Newer College dataset, $\delta = 20$cm. The best and second best results are bold and underlined, respectively.

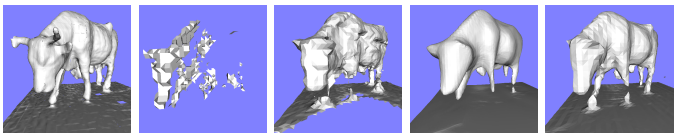
Metric	Method	room 0	room 1	room 2	office 0	office 1	office 2	office 3	office 4	C.L.	Cow	N.C.
F1 Score [$< \delta$]% ↑	Ours	**95.81**	**97.33**	**97.47**	**98.26**	**96.60**	**95.17**	**93.52**	**94.54**	**83.62**	**92.81**	**75.85**
	FIESTA	76.71	76.49	78.15	77.64	82.43	75.45	79.73	78.19	19.37	18.03	54.73
	Voxblox	92.00	94.77	92.64	93.72	94.23	89.50	86.58	89.94	70.30	79.38	54.14
	iSDF	87.85	85.11	90.64	89.80	91.37	87.53	87.52	91.21	78.33	91.84	70.15
	VDB-GPDF	60.06	57.20	58.45	60.99	56.47	58.00	54.78	56.92	78.37	91.08	61.94
Recall [$< \delta$]% ↑	Ours	**92.96**	**95.53**	**95.43**	**97.53**	**95.54**	**92.17**	**90.13**	90.07	**97.49**	**98.96**	75.06
	FIESTA	77.56	78.33	81.04	81.33	82.00	77.17	81.15	78.09	19.89	17.27	52.60
	Voxblox	88.17	91.81	90.79	92.18	90.62	85.67	82.21	84.94	72.71	78.03	56.64
	iSDF	82.88	77.54	88.64	85.51	86.30	83.30	85.46	**90.12**	82.70	93.47	**86.24**
	VDB-GPDF	58.03	55.50	57.29	60.30	55.36	56.55	51.47	55.12	80.73	91.61	62.44
Precision [$< \delta$]% ↑	Ours	**98.85**	**99.21**	**99.61**	**98.99**	97.67	**98.36**	**97.18**	**99.47**	73.21	87.37	**76.65**
	FIESTA	75.87	74.73	75.47	74.28	82.85	73.81	78.36	78.28	18.87	18.86	57.03
	Voxblox	96.18	97.94	94.56	95.31	**98.13**	93.69	91.44	95.55	68.05	80.77	51.84
	iSDF	93.45	94.33	92.74	94.56	97.08	92.21	89.68	92.33	74.40	90.27	59.12
	VDB-GPDF	62.24	59.01	59.66	61.69	57.63	59.52	58.53	58.85	**76.14**	**90.56**	61.45
Completion Ratio [$< \delta$]% ↑	Ours	**92.44**	**95.35**	**95.22**	**97.50**	**95.43**	**91.63**	**89.33**	89.01	**93.50**	**97.28**	74.63
	FIESTA	78.05	79.32	82.34	82.94	81.82	78.16	81.80	78.03	23.99	9.67	48.61
	Voxblox	87.09	91.26	90.41	91.92	89.85	84.33	80.21	83.06	74.46	77.26	60.31
	iSDF	80.69	72.68	88.12	83.98	84.59	81.51	84.74	**89.88**	84.43	93.69	**90.57**
	VDB-GPDF	54.98	52.69	55.53	59.39	53.53	54.26	44.82	52.08	81.83	91.70	63.03
Completion [cm] ↓	Ours	**2.84**	2.35	2.33	2.07	2.26	**3.02**	**2.99**	3.50	**2.58**	2.57	14.23
	FIESTA	3.98	3.16	2.88	2.73	4.71	4.40	3.69	3.98	18.68	25.77	71.87
	Voxblox	3.69	**2.16**	**2.31**	**1.71**	**2.07**	3.52	4.37	4.38	4.44	5.77	21.30
	iSDF	5.63	8.72	2.62	4.14	3.98	5.62	3.03	**2.23**	3.12	**2.24**	**9.75**
	VDB-GPDF	5.52	4.82	3.87	3.52	4.12	4.80	13.78	5.07	3.68	3.05	40.57
Accuracy [cm] ↓	Ours	1.95	1.81	1.93	1.89	1.86	1.97	**1.96**	2.00	4.69	3.24	**13.99**
	FIESTA	3.57	3.19	3.18	3.43	3.11	4.23	3.51	3.22	26.67	16.58	21.02
	Voxblox	**1.74**	**1.14**	**1.50**	**1.22**	**0.81**	**1.68**	2.16	**1.67**	4.98	3.64	25.44
	iSDF	2.04	1.60	1.52	1.64	1.11	1.89	2.30	1.83	**4.66**	**2.75**	31.66
	VDB-GPDF	3.42	3.34	3.17	3.30	3.55	3.32	3.55	3.37	4.94	3.28	18.56
Chamfer-L1 [cm] ↓	Ours	**2.40**	2.08	2.13	1.98	2.06	**2.49**	**2.48**	2.75	**3.64**	2.91	**14.11**
	FIESTA	3.77	3.17	3.03	3.08	3.91	4.31	3.60	3.60	22.68	21.17	46.45
	Voxblox	2.72	**1.65**	**1.90**	**1.47**	**1.44**	2.60	3.26	3.02	4.71	4.70	23.37
	iSDF	3.83	5.16	2.07	2.89	2.55	3.75	2.67	**2.03**	3.89	**2.49**	20.71
	VDB-GPDF	4.47	4.08	3.52	3.41	3.83	4.06	8.67	4.22	4.31	3.17	29.57



(a) Ground Truth (b) Ours (c) FIESTA [13]

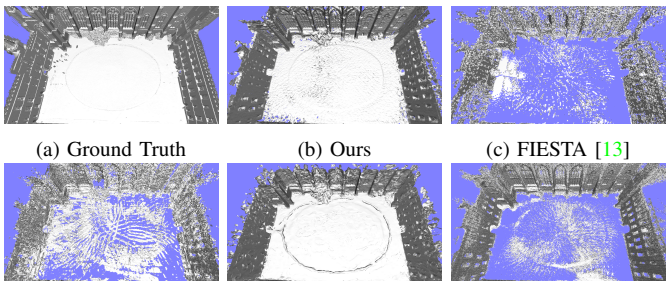
(d) Voxblox [11] (e) iSDF [20] (f) VDB-GPDF [28]

Fig. 5: Qualitative comparison of mesh reconstruction on the Replica dataset [39]. Our method achieves the best visual quality, accuracy and completeness compared to the baselines. For example, the textures of the bed sheets are better preserved in our reconstruction.



Ours FIESTA Voxblox iSDF VDB-GPDF

Fig. 6: Qualitative comparison of mesh reconstruction of the cow in Cow&Lady dataset [11]. Our method successfully recovers the detailed geometry of the cow, such as the horns, ears, and legs, which are missed in the baselines' reconstructions.

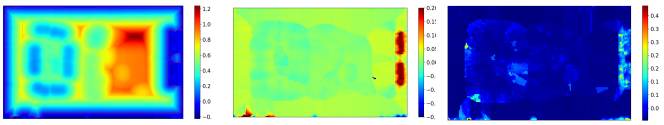


(a) Ground Truth (b) Ours (c) FIESTA [13]

(d) Voxblox [11] (e) iSDF [20] (f) VDB-GPDF [28]

Fig. 7: Qualitative comparison of mesh reconstruction on the Newer College dataset [40].

"Near" points within 0.2m of the surface, and "Far" points beyond 0.2m from the surface. To exclude the influence of sign due to occupancy misclassification (e.g. by FIESTA), we compute the SDF MAE using the absolute values of the predicted and ground truth SDFs. Tables III summarizes the SDF prediction results across all datasets. Our method consistently achieves the lowest SDF MAE and gradient MAE across almost all datasets and regions. Although FIESTA shows competitive accuracy performance on the Replica dataset, its accuracy drops significantly on the Cow and Lady and Newer College datasets due to sensor noise. Voxblox and iSDF generally exhibit higher SDF MAE and gradient MAE, indicating less accurate distance field predictions due to their voxel-based representations and smoothing effects. VDB-GPDF performs better than Voxblox and iSDF. However, its gradient MAE is the highest among all methods, indicating poor gradient estimation. In the "Near" region,



(a) Predicted SDF (b) SDF Error (c) SDF Variance

Fig. 8: Demonstration of the consistency between the SDF error and the SDF variance estimated by our method on a slice of Replica office3 [39]. (a) shows the predicted SDF values on the slice, (b) shows the absolute SDF error compared to the ground truth, and (c) shows the estimated SDF variance. We can observe that regions with higher SDF error correspond to regions with higher estimated variance, indicating that our method effectively captures uncertainty in SDF predictions.

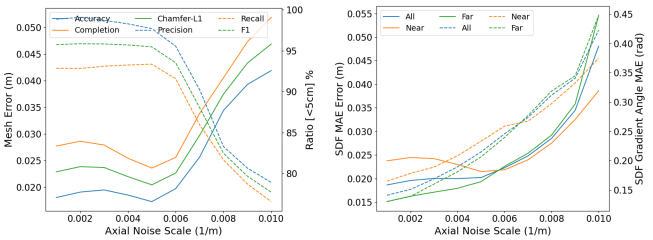
SDF accuracy is particularly important for applications such as robot navigation and manipulation, where precise distance estimates to nearby surfaces are crucial. In the "Far" region, accurate SDF predictions help in understanding the overall environment structure and planning long-range paths. Besides, accurate gradient estimates are helpful for motion planning algorithms that rely on gradient information to navigate around obstacles. Our method's superior performance in both SDF values and gradients demonstrates its effectiveness in capturing the underlying geometry of the environment.

C. Computation Efficiency

We assess the computational efficiency of all methods by measuring the average time taken for SDF updates per frame and SDF prediction at 1k positions during online operation. Table IV shows the timing results across all datasets. Our method consistently achieves real-time performance with average update times at round 150ms per frame at different scene scales, making it suitable for online applications. Although Voxblox runs fast in building the TSDF map, its SDF integration is very slow due to the large number of voxels that need to be updated. FIESTA runs the SDF integration the fastest by using priority queues to reduce the number of voxel updates. However, as we have shown in Sec. VII-A and VII-B, its accuracy is not satisfactory in noisy environments. iSDF is also slow because it needs multiple iterations to converge for each frame and takes time to backward propagate in order to compute the gradients analytically. VDB-GPDF is the most similar to our method in terms of the GP-based SDF representation. However, due to its reliance on the VDB structure for spatial partitioning and storage, its update time is significantly higher than ours. Overall, our method strikes a good balance between computational efficiency and accuracy, making it a practical choice for real-time SDF mapping in robotics applications.

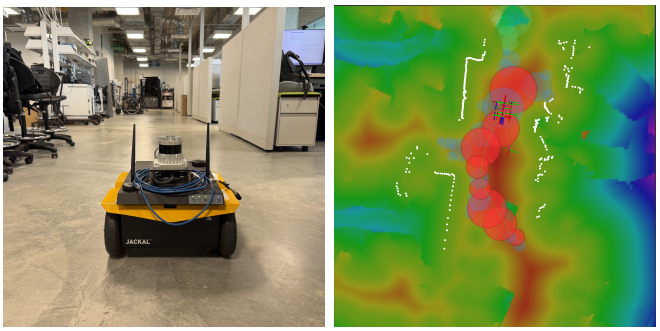
D. Error-Uncertainty Consistency

One of the key advantages of our method is its ability to quantify uncertainty in SDF predictions through the estimated variance. To validate the consistency between the predicted SDF variance and the actual SDF error, we show the SDF error and variance on a slice of the Replica office3 dataset [39] in Fig. 8. We observe that regions with higher SDF error correspond to regions with higher estimated variance, indicating that our method effectively captures uncertainty in SDF predictions. This consistency is crucial for applications such as robot navigation and manipulation, where understanding the confidence in distance estimates can inform decision-making and risk assessment.



(a) Mesh Metrics vs. Sensor Noise (b) SDF Metrics vs. Sensor Noise

Fig. 9: Ablation study on the effect of sensor noise on (a) mesh reconstruction metrics and (b) SDF metrics on the Replica room0 dataset [39]. The results demonstrate that our method maintains robust performance even as sensor noise increases, with only a gradual degradation in both mesh quality and SDF accuracy.



(a) Jackal robot in the environment (b) SDF and safe bubble cover

Fig. 10: Using Kernel-SDF to enable safe bubble cover [42] navigation. (a) A Clearpath Jackal robot equipped with a LiDAR sensor navigating in a laboratory environment. (b) Visualization of the real-time SDF as a heatmap, the safe bubble cover (union of red circles) generated from Kernel-SDF, and the reference trajectory (green line).

E. Ablation Study on Sensor Noise Handling

To evaluate the effectiveness of our method in handling sensor noise, we conduct an ablation study by comparing the metrics with different sensor noise levels on Replica room0 dataset [39]. We vary the noise level by adjusting the axial noise parameter k in the noise model $\sigma = kz^2$ and keep other algorithm parameters fixed. Fig. 9 shows the results of mesh reconstruction metrics (Fig. 9a) and SDF metrics (Fig. 9b) as the sensor noise increases. The results demonstrate that our method maintains robust performance even as sensor noise increases, with only a gradual degradation in both mesh quality and SDF accuracy. This indicates that our surface estimation front-end using Bayesian Hilbert Maps effectively mitigates the impact of sensor noise, leading to reliable SDF mapping in noisy environments.

TABLE III: SDF reconstruction metrics on the Replica dataset [39], the Cow and Lady dataset (C.L. for the scene and Cow for the cow only) [11], and the Newer College (N.C.) dataset [40].

Metric	Region	Method	room 0	room 1	room 2	office 0	office 1	office 2	office 3	office 4	C.L.	Cow	N.C.
SDF MAE [cm] ↓	All	Ours	**1.994**	**1.676**	**1.719**	**1.617**	**1.715**	**1.947**	**1.850**	**2.068**	4.699	**2.547**	24.009
		FIESTA	2.175	2.364	2.418	2.662	2.461	2.234	2.344	2.378	15.344	15.092	37.244
		Voxblox	3.128	2.539	2.733	3.017	2.725	3.470	3.740	3.077	6.297	4.000	62.917
		iSDF	3.745	4.230	4.259	3.890	4.226	5.094	4.245	4.147	7.327	4.258	64.304
		VDB-GPDF	2.925	2.900	2.778	2.902	2.968	2.861	3.176	2.929	**4.683**	3.056	**21.783**
	Near	Ours	2.454	**1.955**	**1.940**	**1.732**	**1.909**	2.264	2.253	2.695	**3.427**	**2.513**	10.998
		FIESTA	**2.121**	2.046	2.040	2.233	1.926	**1.848**	**1.974**	**2.077**	10.591	13.582	32.209
		Voxblox	2.780	2.153	2.233	2.196	2.172	2.852	3.409	2.959	4.287	3.404	19.634
		iSDF	3.649	3.632	3.295	3.206	3.646	3.821	3.397	3.301	4.258	3.595	**6.540**
		VDB-GPDF	2.847	2.924	2.802	2.827	2.936	2.770	3.136	2.870	3.938	3.025	20.693
	Far	Ours	**1.676**	**1.454**	**1.565**	**1.521**	**1.524**	**1.716**	**1.531**	**1.645**	**5.336**	**2.574**	25.214
		FIESTA	2.205	2.574	2.656	2.961	2.898	2.461	2.563	2.545	17.753	16.331	37.744
		Voxblox	3.368	2.845	3.081	3.711	3.270	3.918	4.002	3.158	7.338	4.471	67.322
		iSDF	3.811	4.705	4.932	4.466	4.797	6.016	4.918	4.718	8.864	4.802	69.652
		VDB-GPDF	3.486	2.620	2.606	3.778	3.585	3.472	3.511	3.320	7.066	3.272	**22.739**
Grad. MAE [rad] ↓	All	Ours	**0.159**	**0.138**	**0.166**	**0.161**	**0.140**	**0.166**	**0.161**	**0.161**	0.501	**0.295**	**0.311**
		FIESTA	0.220	0.207	0.184	0.216	0.250	0.198	0.224	0.180	1.141	1.093	0.498
		Voxblox	0.230	0.195	0.222	0.251	0.190	0.271	0.279	0.232	**0.477**	0.338	0.777
		iSDF	0.358	0.200	0.339	0.276	0.279	0.286	0.292	0.346	0.569	0.400	0.667
		VDB-GPDF	0.998	1.051	1.022	1.050	1.165	0.998	1.032	0.980	1.066	0.982	1.176
	Near	Ours	**0.181**	**0.171**	**0.174**	**0.201**	0.180	**0.189**	**0.198**	**0.180**	**0.671**	0.397	**0.934**
		FIESTA	0.304	0.276	0.230	0.313	0.323	0.253	0.300	0.232	1.329	1.224	1.398
		Voxblox	0.282	0.183	0.204	0.259	**0.179**	0.301	0.336	0.299	0.679	0.466	1.435
		iSDF	0.290	0.176	0.247	0.229	0.206	0.222	0.257	0.277	0.679	**0.395**	1.309
		VDB-GPDF	1.070	1.101	1.088	1.099	1.193	1.075	1.093	1.057	1.116	1.030	1.413
	Far	Ours	**0.149**	**0.121**	0.162	**0.141**	**0.115**	**0.155**	**0.143**	**0.153**	0.439	**0.230**	**0.288**
		FIESTA	0.182	0.171	**0.162**	0.166	0.203	0.171	0.189	0.158	1.050	0.985	0.412
		Voxblox	0.208	0.201	0.230	0.246	0.198	0.257	0.256	0.205	**0.406**	0.258	0.720
		iSDF	0.389	0.214	0.389	0.303	0.330	0.319	0.309	0.377	0.528	0.403	0.632
		VDB-GPDF	0.613	0.622	0.627	0.643	0.762	0.603	0.684	0.590	0.907	0.644	0.968

TABLE IV: Timing metrics. Results are reported as the average per-frame processing time (FPT) and prediction time for 1k points (QT-1k) on the Replica room0 [39], the Cow and Lady dataset [11] and the Newer College dataset [40]. The best and second best results are bold and underlined, respectively. Timing was measured on a system with an Intel i9-14900K CPU and an NVIDIA RTX 3090 GPU.

Method	Replica		Cow & Lady		Newer College	
	FPT (s)	QT-1k (ms)	FPT (s)	QT-1k (ms)	FPT (s)	QT-1k (ms)
Ours	0.19	2.80	0.11	4.43	**0.17**	3.04
VDB-GPDF	0.68	10.66	0.10	8.29	0.26	10.16
Voxblox	21.23	103.81	4.48	99.42	94.01	85.20
iSDF	2.49	0.29	0.84	0.27	1.38	0.28
FIESTA	**0.05**	**0.05**	**0.02**	**0.03**	0.28	**0.04**

F. Path Planning Case Study through Kernel-SDF

One practical application of accurate, uncertainty-quantified and real-time SDF mapping is in robot path planning. In this case study, we demonstrate how Kernel-SDF can be utilized to enable safe navigation for a mobile robot using the bubble cover method [42]. In this experiment, we demonstrate the use of fast, accurate SDF mapping for robot path planning. Lee et al. [42] showed that an SDF can be used to construct a safe bubble cover, where each bubble is centered at a point $x \in \mathbb{R}^n$ and has the radius equal to the distance-field value at that point minus a safty margin, which could be a function of the SDF uncertainty. Using sampling-based planning methods, we can recover a connected chain, defined as the union of overlapping bubbles, that contains both the start and goal positions. Once this chain is obtained, planning can be performed safely within the free space it defines. Fig. 10a shows a Clearpath Jackal operating in a laboratory environment with its LiDAR enabled. Fig. 10b visualizes the real-time SDF values as a heatmap, the resulting safe bubble cover (union of the red and blue circles), and the reference trajectory as the green line.

VIII. Conclusion

In this work, we presented Kernel-SDF, a novel framework for real-time SDF learning using kernel regression with uncertainty quantification. By leveraging Bayesian Hilbert Maps for surface estimation and Gaussian Process regression for SDF prediction, our method achieves high accuracy and efficiency in reconstructing the environment as a signed distance function. The proposed hierarchical tree structure for managing multiple local BHMs and GPs enables scalable and real-time performance, making it suitable for online applications in robotics. And the uncertainty quantification provided by our method allows for better risk assessment in downstream tasks such as path planning and obstacle avoidance. Extensive experiments on synthetic and real-world datasets demonstrate that Kernel-SDF outperforms existing state-of-the-art methods in terms of reconstruction quality and prediction accuracy with good balance of computational speed. Besides, the BHM-based surface estimation properly handles sensor noise and dynamic environment changes, further enhancing the robustness of our approach.

References

[1] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees," *Autonomous Robots*, 2013.

[2] J. Behley and C. Stachniss, "Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments," *Robotics: Science and Systems XIV*, 2018.

[3] F. Lu, G. Chen, Y. Liu, Y. Zhan, Z. Li, D. Tao, and C. Jiang, "Sparse-to-Dense Matching Network for Large-Scale LiDAR Point Cloud Registration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[4] Y. Zeng, J. Hou, Q. Zhang, S. Ren, and W. Wang, "Dynamic 3D Point Cloud Sequences as 2D Videos," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[5] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang, "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images," in *ECCV*, 2018.

[6] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[7] B. Curless and M. Levoy, "A Volumetric Method for Building Complex Models from Range Images," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH, 1996.

[8] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *IEEE International Symposium on Mixed and Augmented Reality*, 2011.

[9] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3D Reconstruction at Scale using Voxel Hashing," *ACM Trans. Graph.*, 2013.

[10] O. Kähler, V. Adrian Prisacariu, C. Yuheng Ren, X. Sun, P. Torr, and D. Murray, "Very High Frame Rate Volumetric Integration of Depth Images on Mobile Devices," *IEEE Transactions on Visualization and Computer Graphics*, 2015.

[11] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[12] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[13] L. Han, F. Gao, B. Zhou, and S. Shen, "FIESTA: Fast Incremental Euclidean Distance Fields for Online Motion Planning of Aerial Robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[14] B. Lee, C. Zhang, Z. Huang, and D. D. Lee, "Online Continuous Mapping using Gaussian Process Implicit Surfaces," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019.

[15] L. Wu, K. M. B. Lee, L. Liu, and T. Vidal-Calleja, "Faithful Euclidean Distance Field From Log-Gaussian Process Implicit Surfaces," *IEEE Robotics and Automation Letters*, 2021.

[16] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang, "NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction," in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, 2021.

[17] T. Takikawa, J. Litalien, K. Yin, K. Kreis, C. Loop, D. Nowrouzezahrai, A. Jacobson, M. McGuire, and S. Fidler, "Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[18] I. Vizzo, T. Guadagnino, J. Behley, and C. Stachniss, "VDBFusion: Flexible and Efficient TSDF Integration of Range Sensor Data," *Sensors*, 2022.

[19] Y. Pan, Y. Kompis, L. Bartolomei, R. Mascaro, C. Stachniss, and M. Chli, "Voxfield: Non-Projective Signed Distance Fields for Online Planning and 3D Reconstruction," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[20] J. Ortiz, A. Clegg, J. Dong, E. Sucar, D. Novotny, M. Zollhoefer, and M. Mukadam, "iSDF: Real-Time Neural Signed Distance Fields for Robot Perception," in *Robotics: Science and Systems (RSS)*, 2022.

[21] Y. Bai, Z. Miao, X. Wang, Y. Liu, H. Wang, and Y. Wang, "VDBblox: Accurate and Efficient Distance Fields for Path Planning and Mesh Reconstruction," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[22] Y. Wang, Q. Han, M. Habermann, K. Daniilidis, C. Theobalt, and L. Liu, "NeuS2: Fast Learning of Neural Implicit Surfaces for Multi-view Reconstruction," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[23] C. Jiang, H. Zhang, P. Liu, Z. Yu, H. Cheng, B. Zhou, and S. Shen, "H2-Mapping: Real-time Dense Mapping Using Hierarchical Hybrid Representation," *IEEE Robotics and Automation Letters*, 2023.

[24] C. Le Gentil, O.-L. Ouabi, L. Wu, C. Pradalier, and T. Vidal-Calleja, "Accurate Gaussian-Process-Based Distance Fields With Applications to Echolocation and Mapping," *IEEE Robotics and Automation Letters*, 2024.

[25] A. Millane, H. Oleynikova, E. Wirbel, R. Steiner, V. Ramasamy, D. Tingdahl, and R. Siegwart, "nvblox: GPU-Accelerated Incremental Signed Distance Field Mapping," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[26] Y. Pan, X. Zhong, L. Wiesmann, T. Posewsky, J. Behley, and C. Stachniss, "PIN-SLAM: LiDAR SLAM Using a Point-Based Implicit Neural Representation for Achieving Global Map Consistency," *IEEE Transactions on Robotics*, 2024.

[27] Q. Zou and M. Sester, "3D Uncertain Implicit Surface Mapping Using GMM and GP," *IEEE Robotics and Automation Letters*, 2024.

[28] L. Wu, C. Le Gentil, and T. Vidal-Calleja, "VDB-GPDF: Online Gaussian Process Distance Field With VDB Structure," *IEEE Robotics and Automation Letters*, 2025.

[29] Y. Tian, H. Cao, S. Kim, and N. Atanasov, "MISO: Multiresolution Submap Optimization for Efficient Globally Consistent Neural Implicit Reconstruction," 2025.

[30] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis," in *European Conference on Computer Vision (ECCV)*, 2020.

[31] K. M. B. Lee, Z. Dai, C. L. Gentil, L. Wu, N. Atanasov, and T. Vidal-Calleja, "Safe Bubble Cover for Motion Planning on Distance Fields," 2024.

[32] K. Long, Y. Yi, Z. Dai, S. Herbert, J. Cortés, and N. Atanasov, "Sensor-based distributionally robust control for safe robot navigation in dynamic environments," *The International Journal of Robotics Research*, 2025.

[33] O. Williams and A. Fitzgibbon, "Gaussian Process Implicit Surfaces," in *Gaussian Processes in Practice*, 2006.

[34] R. Senanayake and F. Ramos, "Bayesian Hilbert Maps for Dynamic Continuous Occupancy Mapping," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017.

[35] K. Museth, "VDB: High-resolution Sparse Volumes with Dynamic Topology," *ACM Trans. Graph.*, 2013.

[36] T. S. Jaakkola and M. I. Jordan, "A Variational Approach to Bayesian Logistic Regression Models and their Extensions," in *Proceedings of the Sixth International Workshop on Artificial Intelligence and Statistics*, 1997.

[37] S. R. S. Varadhan, "On the Behavior of the Fundamental Solution of the Heat Equation with Variable Coefficients," *Communications on Pure and Applied Mathematics*, 1967.

[38] W. E. Lorensen and H. E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '87, 1987.

[39] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J. J. Engel, R. Mur-Artal, C. Ren, S. Verma, A. Clarkson, M. Yan, B. Budge, Y. Yan, X. Pan, J. Yon, Y. Zou, K. Leon, N. Carter, J. Briales, T. Gillingham, E. Mueggler, L. Pesqueira, M. Savva, D. Batra, H. M. Strasdat, R. D. Nardi, M. Goesele, S. Lovegrove, and R. Newcombe, "The Replica Dataset: A Digital Replica of Indoor Spaces," *arXiv preprint arXiv:1906.05797*, 2019.

[40] L. Zhang, M. Camurri, D. Wisth, and M. Fallon, "Multi-Camera LiDAR Inertial Extension to the Newer College Dataset," 2021.

[41] M. S. Ahn, H. Chae, D. Noh, H. Nam, and D. Hong, "Analysis and noise modeling of the intel realsense d435 for mobile robots," in *16th International Conference on Ubiquitous Robots (UR)*.

[42] K. M. B. Lee, Z. Dai, C. L. Gentil, L. Wu, N. Atanasov, and T. Vidal-Calleja, "Safe Bubble Cover for Motion Planning on Distance Fields," 2024.

IX. Appendix

A. EM Algorithm for Updating BHM

Readers can refer to [34], [36] for more details about the Bayesian Hilbert Map (BHM). Here, we provide a brief derivation of the EM algorithm for updating the BHM model parameters, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.

Given a dataset $\mathcal{D} = \{(\mathbf{x}_k, y_k)\}_{k=1}^K$ of K generated samples at time t, our goal is to update the weights $\mathbf{w}$ of the model such that we get the posterior distribution:

$$P(\mathbf{w}|\mathcal{D}) = \frac{P(\mathcal{D}|\mathbf{w})P(\mathbf{w})}{P(\mathcal{D})}, \tag{22}$$

where $P(\mathbf{w})$ is the prior distribution of the weights, and the likelihood is given by:

$$\begin{aligned} P(\mathcal{D}|\mathbf{w}) &= \prod_{k=1}^{K} P(y_k|\mathbf{x}_k, \mathbf{w}) \\ &= \prod_{k=1}^{K} \sigma(-y_k \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_k)), \quad y_k \in \{-1, 1\}. \end{aligned} \tag{23}$$

The marginal likelihood $P(\mathcal{D})$ is intractable, so we resort to approximate inference methods. Our goal to maximize the log marginal likelihood:

$$\begin{aligned} \log P(\mathcal{D}) &= \log \int P(\mathcal{D}|\mathbf{w})P(\mathbf{w})d\mathbf{w} \\ &= \log \int Q(\mathbf{w}) \frac{P(\mathcal{D}|\mathbf{w})P(\mathbf{w})}{Q(\mathbf{w})} d\mathbf{w} \\ &\geq \int Q(\mathbf{w}) \log \frac{P(\mathcal{D}|\mathbf{w})P(\mathbf{w})}{Q(\mathbf{w})} d\mathbf{w}, \\ &\quad \uparrow \text{Jensen's inequality} \end{aligned} \tag{24}$$

where $Q(\mathbf{w})$ is an approximate posterior distribution of the weights.

Since $P(\mathcal{D}|\mathbf{w})$ is product of sigmoids that contains the weights $\mathbf{w}$ inside, we use the variational bound [36] to get a looser lower bound:

$$\sigma(r) \geq \sigma(\xi) \exp\left(\frac{r - \xi}{2} + \lambda(\xi)\left(r^2 - \xi^2\right) \right),$$

where ξ is a variational parameter and $\lambda(\xi) = \frac{1}{2\xi}\left(\frac{1}{2} - \sigma(\xi)\right)$.

Using the idea of Sequential BHM EM update [34], we assume a Gaussian prior $P(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1})$ and a Gaussian approximate posterior $Q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ for the weights. Hence, we can write the variational lower bound of the log marginal likelihood as:

$$\begin{aligned} \mathcal{L} &= \mathbb{E}_Q\left[\log P(\mathcal{D}|\mathbf{w})\right] + \mathbb{E}_Q\left[\log P(\mathbf{w})\right] - \mathbb{E}_Q\left[\log Q(\mathbf{w})\right] \\ &\geq \sum_{k=1}^{K} \mathbb{E}_Q\left[\log \sigma(-y_k \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_k))\right] \\ &\quad + \mathbb{E}_Q\left[\log P(\mathbf{w})\right] - \mathbb{E}_Q\left[\log Q(\mathbf{w})\right] \\ &\geq \sum_{k=1}^{K} \left(\log \sigma(\xi_k) - \frac{\xi_k}{2} + \frac{\mathbb{E}_Q[r_k]}{2} + \lambda(\xi_k)\left(\mathbb{E}_Q\left[r_k^2\right] - \xi_k^2\right) \right) \\ &\quad + \mathbb{E}_Q\left[\log P(\mathbf{w})\right] - \mathbb{E}_Q\left[\log Q(\mathbf{w})\right] \\ &= \sum_{k=1}^{K} \left(\log \sigma(\xi_k) - \frac{\xi_k}{2} - \xi_k^2 \lambda(\xi_k) + (y_k - \frac{1}{2})\boldsymbol{\mu}_t^\top \boldsymbol{\phi}(\mathbf{x}_k) \right) \\ &\quad + \sum_{k=1}^{K} \left(\lambda(\xi_k) \boldsymbol{\phi}^\top(\mathbf{x}_k) \left(\boldsymbol{\Sigma}_t + \boldsymbol{\mu}_t \boldsymbol{\mu}_t^\top\right) \boldsymbol{\phi}(\mathbf{x}_k) \right) \\ &\quad + \frac{1}{2} \operatorname{tr}\left(\left(\boldsymbol{\Sigma}_t^{-1} - \boldsymbol{\Sigma}_{t-1}^{-1}\right) \boldsymbol{\Sigma}_t \right) + \frac{1}{2} \log \frac{|\boldsymbol{\Sigma}_t|}{|\boldsymbol{\Sigma}_{t-1}|} \\ &\quad - \frac{1}{2} \left(\boldsymbol{\mu}_t - \boldsymbol{\mu}_{t-1}\right)^\top \boldsymbol{\Sigma}_{t-1}^{-1} \left(\boldsymbol{\mu}_t - \boldsymbol{\mu}_{t-1}\right), \end{aligned}$$

where $r_k = y_k \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_k), y_k \in \{0, 1\}$.

To maximize the lower bound $\mathcal{L}$, we take derivatives with respect to the variational parameters $\{\xi_k\}_{k=1}^K$ and $\boldsymbol{\mu}_t$, and set them to zero. This results in the following update equations:

a) E-Step: $\partial\mathcal{L}/\partial\boldsymbol{\mu}_t = 0$ leads to

$$\begin{aligned} \boldsymbol{\mu}_t &= \boldsymbol{\Sigma}_t \left(\boldsymbol{\Sigma}_{t-1}^{-1} \boldsymbol{\mu}_{t-1} + \sum_{k=1}^{K} \left(y_k - \frac{1}{2}\right) \boldsymbol{\phi}(\mathbf{x}_k) \right), \\ \boldsymbol{\Sigma}_t^{-1} &= \boldsymbol{\Sigma}_{t-1}^{-1} + \sum_{k=1}^{K} \left| \frac{1/2 - \sigma(\xi_k)}{\xi_k} \right| \boldsymbol{\phi}(\mathbf{x}_k) \boldsymbol{\phi}^\top(\mathbf{x}_k), \end{aligned}$$

which are the same as (14) and (15).

b) M-Step: $\partial\mathcal{L}/\partial\xi_k = 0$ leads to

$$\xi_k = \sqrt{\boldsymbol{\phi}^\top(\mathbf{x}_k) \left(\boldsymbol{\Sigma}_t + \boldsymbol{\mu}_t \boldsymbol{\mu}_t^\top\right) \boldsymbol{\phi}(\mathbf{x}_k)}, \quad \forall k = 1, \ldots, K,$$

which is the same as (16). Note that $\xi_{k,t} = \xi_k$, where t is omitted for simplicity. We suggest initialize ξ_k to 0.0 or 1.0 for all k (Empirically, both work well).

Practical Optimization for the Implementation: For each generated dataset $\mathcal{D}$, usually 1 to 2 EM iterations are sufficient for convergence. To make the algorithm more efficient, we set $\phi_m(\mathbf{x}_k) = 0$ if its value is smaller than a threshold (e.g., 10^{-3}) so that we can speed up the computation by the sparsity of $\boldsymbol{\phi}(\mathbf{x}_k)$. Besides, the update of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ needs matrix inversion of $\boldsymbol{\Sigma}_t^{-1}$, which is handled by Cholesky decomposition for efficiency and numerical stability.

B. Approximation of BHM Prediction

1) **Mean Occupied Probability:** With enough iterations, we get the converged $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, and predict the occupancy probability at $\mathbf{x}_*$ by (12):

$$g(\mathbf{x}_*, \mathbf{w}) = P(y = 1 \mid \mathbf{x}_*, \mathbf{w}) = \sigma(\mathbf{w}^\top \boldsymbol{\phi}_*), \tag{25}$$

where $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $\boldsymbol{\phi}_* = \boldsymbol{\phi}(\mathbf{x}_*)$. Let $g_*(\mathbf{w}) = g(\mathbf{x}_*, \mathbf{w})$. Then, the mean of $g_*(\mathbf{w})$ is

$$\mathbb{E}[g_*(\mathbf{w})] = P(y = 1 \mid \mathbf{x}_*) = \int \sigma\left(\mathbf{w}^\top \boldsymbol{\phi}_*\right) P(\mathbf{w}) d\mathbf{w}. \tag{26}$$

However, we cannot compute the above integral analytically. One way of approximation is to collect N samples of $\mathbf{w}$ from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, so that

$$\mathbb{E}[g_*(\mathbf{w})] = P(y = 1 \mid \mathbf{x}_*) \approx \frac{1}{N} \sum_{i=1}^{N} \sigma(\mathbf{w}_i^\top \boldsymbol{\phi}_*). \tag{27}$$

A faster and more accurate approximation is

$$\mathbb{E}[g_*(\mathbf{w})] \approx \sigma(h), \quad h = \frac{\boldsymbol{\mu}^\top \boldsymbol{\phi}_*}{\sqrt{1 + \frac{\pi}{8} \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma} \boldsymbol{\phi}_*}}. \tag{28}$$

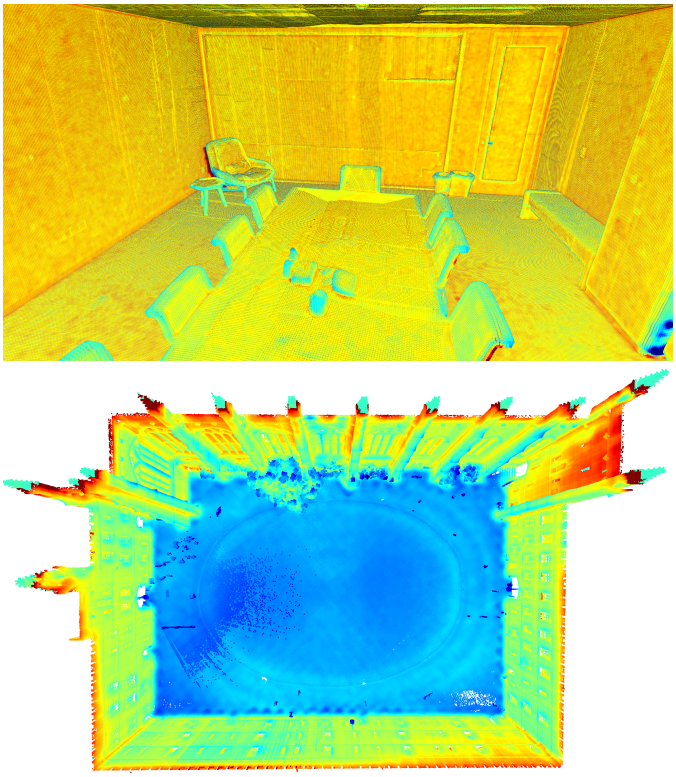


Fig. 11: Surface points colored by their log-odds values from BHM.

Proof. We approximate the sigmoid function $\sigma(x)$ with the cumulative distribution function of the normal distribution, $\Phi(\lambda x)$, where $\lambda = \sqrt{\pi/8}$. Therefore,

$$
\begin{aligned}
&P(y=1 \mid \mathbf{x}_*) \\
&\approx \int \Phi(\lambda \mathbf{w}^\top \boldsymbol{\phi}_*) P(\mathbf{w}) d\mathbf{w} = \int \Phi(\lambda u) P(u) du \\
&= \int \Phi\left(\frac{v + \boldsymbol{\mu}^\top \boldsymbol{\phi}_* / \sqrt{\boldsymbol{\phi}_*^\top \boldsymbol{\Sigma} \boldsymbol{\phi}_*}}{\left(\lambda \sqrt{\boldsymbol{\phi}_*^\top \boldsymbol{\Sigma} \boldsymbol{\phi}_*}\right)^{-1}} \right) P(v) dv \\
&= \Phi\left(\frac{\boldsymbol{\mu}^\top \boldsymbol{\phi}_* / \sqrt{\boldsymbol{\phi}_*^\top \boldsymbol{\Sigma} \boldsymbol{\phi}_*}}{\sqrt{1 + \left(\lambda \sqrt{\boldsymbol{\phi}_*^\top \boldsymbol{\Sigma} \boldsymbol{\phi}_*}\right)^{-2}}} \right) \\
&= \Phi\left(\frac{\boldsymbol{\mu}^\top \boldsymbol{\phi}_*}{\sqrt{\lambda^{-2} + \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma} \boldsymbol{\phi}_*}} \right) \approx \sigma\left(\frac{\boldsymbol{\mu}^\top \boldsymbol{\phi}_*}{\sqrt{1 + \lambda^2 \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma} \boldsymbol{\phi}_*}} \right) \\
&= \sigma\left(\frac{\boldsymbol{\mu}^\top \boldsymbol{\phi}_*}{\sqrt{1 + \frac{\pi}{8} \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma} \boldsymbol{\phi}_*}} \right),
\end{aligned}
$$

where $u = \mathbf{w}^\top \boldsymbol{\phi}_* \sim \mathcal{N}\left(\boldsymbol{\mu}^\top \boldsymbol{\phi}_*, \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma} \boldsymbol{\phi}_*\right)$, $v = \frac{u - \boldsymbol{\mu}^\top \boldsymbol{\phi}_*}{\sqrt{\boldsymbol{\phi}_*^\top \boldsymbol{\Sigma} \boldsymbol{\phi}_*}} \sim \mathcal{N}(0,1)$. And the following integral is used

$$
\int \Phi\left(\frac{w-a}{b}\right) P(w) dw = \Phi\left(\frac{-a}{\sqrt{1+b^2}}\right). \tag{29}
$$

To prove the above equation, let $X \sim \mathcal{N}(a, b^2)$ and $Y \sim \mathcal{N}(0,1)$ be independent random variables. Then,

$$
P(X \leq Y \mid Y = w) = P(X \leq w) = \Phi\left(\frac{w-a}{b}\right), \tag{30}
$$

$$
\begin{aligned}
P(X \leq Y) &= \int P(X \leq Y \mid Y = w) P(w) dw \\
&= \int \Phi\left(\frac{w-a}{b}\right) P(w) dw.
\end{aligned} \tag{31}
$$

Since $P(X \leq Y) = P(X - Y \leq 0)$ and $X - Y \sim \mathcal{N}(a, b^2+1)$, $P(X \leq Y) = \Phi\left(\frac{-a}{\sqrt{1+b^2}}\right)$. Therefore, (29) holds. □

When it converges, $\det(\boldsymbol{\Sigma}) \approx 0$, we have

$$
\mathbb{E}[g_*(\mathbf{w})] \approx \sigma\left(\boldsymbol{\mu}^\top \boldsymbol{\phi}_*\right), \tag{32}
$$

where $\boldsymbol{\mu}^\top \boldsymbol{\phi}_*$ is the log-odds of occupancy at $\mathbf{x}_*$.

2) **Sign Prediction with BHM:** When it comes to classify occupancy for the sign prediction, we can use the following decision rule:

$$
\operatorname{sign}(\mathbf{x}_*) = \begin{cases} +1, & \boldsymbol{\mu}^\top \boldsymbol{\phi}_* < \epsilon, \\ -1, & \text{otherwise}, \end{cases}
$$

where $\epsilon \in \mathbb{R}$ is the decision boundary that can be learned from data online with $\mathcal{S}_t$ using exponential moving average:

$$
\epsilon_t \leftarrow (1-\alpha)\epsilon_{t-1} + \frac{\alpha}{K} \sum_{k=1}^{K} \boldsymbol{\mu}_t^\top \boldsymbol{\phi}(\mathbf{x}_k), \quad \alpha \in (0,1],
$$

where K is the number of hit surface samples in $\mathcal{S}_t$ and $\mathbf{x}_k$ is in the map frame.

Ideally, ϵ should be 0.0 if the sampled dataset $\mathcal{D}$ is balanced and the BHM model converges. However, in practice, as shown in Fig. 11 for occupied samples $(\mathbf{x}, 1)$, $\mathbf{x}$ is biased towards the surface, which means ϵ should be a small positive value. In addition, the number of occupied and free samples are affected by the sensor's field of view and its viewing direction relative to the surface, which may cause much more free samples, leading to a negative ϵ. Hence, we suggest learning ϵ online.

3) **Gradient and Surface Normal Prediction:** We can also derive the gradient of (28) as

$$
\begin{aligned}
&\nabla \mathbb{E}[g_*(\mathbf{w})] = \sigma(h)\sigma(-h) \nabla_{\mathbf{x}} \boldsymbol{\phi}_* \nabla_{\boldsymbol{\phi}_*} h, \\
&\nabla_{\boldsymbol{\phi}_*} h = \frac{1}{\sqrt{1 + \frac{\pi}{8} \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma} \boldsymbol{\phi}_*}} \left(\boldsymbol{\mu} - \frac{\frac{\pi}{8} \boldsymbol{\mu}^\top \boldsymbol{\phi}_* \boldsymbol{\Sigma} \boldsymbol{\phi}_*}{1 + \frac{\pi}{8} \boldsymbol{\phi}_*^\top \boldsymbol{\Sigma} \boldsymbol{\phi}_*} \right),
\end{aligned} \tag{33}
$$

where $\nabla_{\mathbf{x}} \boldsymbol{\phi}_* = -2\gamma \mathbf{X} \operatorname{diag}(\boldsymbol{\phi}_*)$, $\mathbf{X} = [\mathbf{x}_* - \tilde{\mathbf{x}}_1, \cdots, \mathbf{x}_* - \tilde{\mathbf{x}}_M, \mathbf{0}]$ for the RBF kernel and M hinge points $\{\tilde{\mathbf{x}}_i\}_{i=1}^M$.

When $\det(\boldsymbol{\Sigma}) \approx 0$, the gradient of the occupancy probability at $\mathbf{x}_*$ is approximated as

$$
\mathbf{v}_* \approx \sigma(\boldsymbol{\mu}^\top \boldsymbol{\phi}_*) \sigma(-\boldsymbol{\mu}^\top \boldsymbol{\phi}_*) \nabla_{\mathbf{x}} \boldsymbol{\phi}_* \boldsymbol{\mu}. \tag{34}
$$

The surface normal at $\mathbf{x}_*$ is the opposite direction of $\mathbf{v}_*$. Since we do not care about the magnitude of $\mathbf{v}_*$ when calculate the surface normal, we have

$$
\mathbf{n}_* = -\frac{\mathbf{z}(\boldsymbol{\mu})}{\|\mathbf{z}(\boldsymbol{\mu})\|_2}, \quad \mathbf{z}(\boldsymbol{\mu}) = \nabla_{\mathbf{x}} \boldsymbol{\phi}_* \boldsymbol{\mu}. \tag{35}
$$

C. Connection between Log-GP and Softmin

Assume we have only two points in the dataset $\{\mathbf{x}_1, \mathbf{x}_2\}$ and there is no noise $\sigma^2 = 0$ so that the posterior of log distance according to (4) is:

$$f_* = \begin{bmatrix} k(\mathbf{x}_*, \mathbf{x}_1) \\ k(\mathbf{x}_*, \mathbf{x}_2) \end{bmatrix}^\top \begin{bmatrix} 1 & k(\mathbf{x}_1, \mathbf{x}_2) \\ k(\mathbf{x}_1, \mathbf{x}_2) & 1 \end{bmatrix}^{-1} \begin{bmatrix} \alpha \\ \alpha \end{bmatrix}, \quad (36)$$

where $\alpha = 1$ when $\mathbf{x}_1$ and $\mathbf{x}_2$ are surface points. Simplifying the above equation leads to:

$$f_* = \alpha \frac{k(\mathbf{x}_*, \mathbf{x}_1) + k(\mathbf{x}_*, \mathbf{x}_2)}{1 + k(\mathbf{x}_1, \mathbf{x}_2)}, \quad (37)$$

so that

$$\log f_* = \log \alpha + \log(1 + k_{12}) + \log(k_{*1} + k_{*2}), \quad (38)$$

where $k_{12} = k(\mathbf{x}_1, \mathbf{x}_2)$, $k_{*1} = k(\mathbf{x}_*, \mathbf{x}_1)$, and $k_{*2} = k(\mathbf{x}_*, \mathbf{x}_2)$. When $\alpha = 1$ and the kernel scale is small enough, $k_{12} \approx 0$, we have

$$\log f_* \approx \log(k_{*1} + k_{*2}) \approx \log\left(\min(k_{*1}, k_{*2})\right), \quad (39)$$

which shows that the log-GP posterior mean is approximately a kind of softmin function when the kernel is exponential and the scale is small enough.

Note that if $\alpha \neq 1$, i.e. $\mathbf{x}_1$ and $\mathbf{x}_2$ are not surface points, we will not obtain (39) and the log-GP posterior mean will not be the approximation of the unsigned distance function. Also, to get a better approximation, we need to ensure that the kernel scale is small enough so that $k_{12} \approx 0$ and the surface point closest to $\mathbf{x}_*$ dominants the sum of the kernel evaluations.

However, it is challenging to set a small kernel scale for the log-GP in practice, because a small kernel scale leads to small $k(\mathbf{x}_*, \mathbf{x}_i)$ for all $\mathbf{x}_i$ that are not very close to $\mathbf{x}_*$, which causes underflow issues in the implementation. To address this issue, we propose to rewrite (4) as:

$$\begin{bmatrix} \hat{f}'(\mathbf{x}_*) \\ \nabla \hat{f}'(\mathbf{x}_*) \end{bmatrix} = \begin{bmatrix} \mathbf{k'}_*^\top \\ \nabla_{\mathbf{x}_*}^\top \mathbf{k'}_* \end{bmatrix} (\mathbf{K} + \mathbf{\Sigma}_y)^{-1} \mathbf{1}, \quad (40)$$

where $\mathbf{k}'_* \in \mathbb{R}^N$ is calculated by:

$$k'_{*,i} = k'(\mathbf{x}_i, \mathbf{x}_*) = \gamma k_S(\|\mathbf{x}_i - \mathbf{x}_*\|_2) \quad 1 \leq i \leq N. \quad (41)$$

Here, $\gamma \in \mathbb{R}^+$ is a scaling factor that can be tuned to avoid underflow issues when using small kernel scales. In practice, we set $\gamma = \exp\left(d^2/(2l^2)\right)$ for the RBF kernel or $\gamma = \exp\left(\sqrt{3}d/l\right)$ for the Matérn 3/2 kernel, where $d = \|\mathbf{x}_* - \mathbf{x}_1\|_2$. γ should be merged into the exponential term of the kernel function to avoid underflow caused by the exponent. Considering that surface points in $\mathcal{D}_{\text{surf}}$ are usually close to each other, this setting can effectively avoid underflow issues. With this modification, the inverse transformation $r(\hat{f}')$ should also be changed accordingly. For the RBF kernel, we have:

$$r(\hat{f}') = \sqrt{-2l^2\left(\log \hat{f}' - \log \gamma\right)} = \sqrt{-2l^2 \log \hat{f}' + d^2}. \quad (42)$$

For the Matérn 3/2 kernel, we have:

$$r(\hat{f}') = -\frac{l}{\sqrt{3}}\left(\log \hat{f}' - \log \gamma\right) = -\frac{l}{\sqrt{3}} \log \hat{f}' + d. \quad (43)$$

D. SDF Gradient Variance Calculation

Based on the softmin approximation of (9), we can obtain the approximation of the UDF gradient:

$$\begin{aligned} \nabla d(\mathbf{x}_*) &\approx \nabla_{\mathbf{x}_*} h(\mathbf{x}_*, \{\mathbf{x}_i\}_{i=1}^N) = g(\mathbf{x}_*, \{\mathbf{x}_i\}_{i=1}^N) \\ &= \mathbf{V}\left(\alpha \mathbf{s}\mathbf{s}^\top \mathbf{z} - \alpha \operatorname{diag}(\mathbf{s})\mathbf{z} + \mathbf{s}\right), \\ \mathbf{V} &= \left[\mathbf{v}_1 \cdots \mathbf{v}_N\right], \mathbf{v}_i = \frac{\mathbf{x}_* - \mathbf{x}_i}{z_i}. \end{aligned} \quad (44)$$

Therefore, assuming isotropic variance $\mathbb{V}_i$ for each dimension of $\mathbf{x}_i$, we can calculate the variance of $\nabla d(\mathbf{x}_*)$ approximately by Taylor expansion and get:

$$\mathbb{V}[\nabla_k d(\mathbf{x}_*)] \approx \sum_{i=1}^N \left\|\nabla_{\mathbf{x}_i} g_k(\mathbf{x}_*, \{\mathbf{x}_i\}_{i=1}^N)\right\|_2^2 \mathbb{V}_i, 1 \leq k \leq n, \quad (45)$$

where

$$\begin{aligned} &\nabla_{\mathbf{x}_i} g(\mathbf{x}_*, \{\mathbf{x}_i\}_{i=1}^N) = \\ &\alpha \mathbf{v}_i \left(l_i\left(\mathbf{v}_i - \mathbf{V}\mathbf{s}\right) + s_i(\mathbf{v}_i - \mathbf{g})\right)^\top + \frac{l_i}{z_i}\left(\mathbf{v}_i \mathbf{v}_i^\top - \mathbf{I}\right), \\ &l_i = s_i(\alpha \mathbf{s}^\top \mathbf{z} - \alpha z_i + 1). \end{aligned} \quad (46)$$

E. Priority Based Update

In our Kernel-SDF framework, several operations take significant time, including BHM EM update, BHM marching, GP training data collection, and GP training. However, not all operations need to be performed at every time step. Therefore, we introduce a priority-based update scheme to efficiently allocate computational resources to the most needed updates.

To maintain the responsiveness of the system to the latest sensor data, BHM EM updates are performed at every time step for all BHMs that have new sensor data. Newly created BHMs also need to be marched immediately to get surface points for its corresponding GP training data buffer update. For all GP training operations, we delay them until they are required for SDF prediction.

For other operations, we use priority queues to determine the order of updates based on their necessity. In order to make the system also responsive to the latest query trends (e.g., the down-stream planner may focus on a specific area), each GP has a query counter c_0 that records how many times the GP is requested for SDF prediction. The counter helps to reshape the priority of GP training and data buffer updates. When a GP is trained, its query counter is reduced by half. And the query counter is capped to a maximum value (10000) to avoid over-prioritization.

1) **Marching Priority Queue:** Note that a single EM update of a BHM only affects a small portion of the surface points. Especially for well-updated BHMs, the surface point changes are even smaller. Hence, we first introduce a priority queue for marching BHMs, which orders the BHMs by the latest timestamp $t_{\mathcal{B}}$ when a BHM is requested to update surface estimation. The BHM with the oldest timestamp (smaller $t_{\mathcal{B}}$) in the queue is processed first because it is more likely to be unchanged in the near future:

$$s_{\text{march}} = t_{\mathcal{B}} \downarrow, \quad (47)$$

where $\downarrow$ indicates smaller values have higher priority.

2) **GP Data Buffer Update Queue:** After marching a BHM, its corresponding GP training data buffer needs to be updated with the new surface points. However, it also takes time to get significant changes of $\mathcal{D}_{\text{surf}}$ for updating the GP. And $\mathcal{D}_{\text{surf}}$ is not required until the GP needs to be trained. Therefore, we introduce a priority queue that orders the GPs by the following score:

$$s_{\text{buffer update}} = c_1(1 + \eta_1 c_0) \uparrow, \tag{48}$$

where c_1 is the number of times the GP data buffer is marked to be updated since the last update, η_1 is a weight parameter, and $\uparrow$ indicates larger values have higher priority. c_1 is increased by 1 each time the GP is marked to update its data buffer after marching its corresponding BHM and reset to 0 after the data buffer is updated. For newly created GPs, we initialize c_1 to $c_1^{\max} \exp(-\gamma \|\mathbf{c}_{GP} - \mathbf{o}_t\|_2)$, where $\mathbf{c}_{GP}$ is the center of the GP training data collection bounding box, $\mathbf{o}_t$ is the current sensor origin, and $\gamma > 0$ is a decay parameter. This initialization gives higher priority to GPs closer to the sensor origin.

3) **GP Training Queue:** Similar to the GP data buffer update, GP training is not required until SDF prediction is requested. In addition, training a GP only makes sense when its training data buffer has significant changes. Therefore, we introduce another priority queue that orders the GPs by the following score:

$$s_{\text{GP train}} = c_2(1 + \eta_2 c_0) \uparrow, \tag{49}$$

where c_2 is the number of times the GP is marked to be trained since the last training, η_2 is a weight parameter, and $\uparrow$ indicates larger values have higher priority. c_2 is increased by c_1 each time the GP's buffer is updated and reset to 0 after the GP is trained. When the GP is trained, c_0 is also halved in case the GP is over-prioritized for future updates as the query trend may change.

F. Experiment Details

1) Parameter Settings: For Kernel-SDF, we set the octree resolution to 0.08m, 0.05m and 0.7m for the Replica, Cow&Lady, and Newer College datasets, respectively. The local BHMs are placed at the depth $D_{\text{tree}} - 1$, which means the local BHM bounding box size is twice the octree voxel size. The number of hinge points per axis is set to 7 for the Replica and Cow&Lady datasets, and 9 for the Newer College dataset. The BHM kernel scale l is set to 0.016m, 0.013m and 0.11m for the three datasets, respectively. For the GP, we use RBF kernel and set $\lambda = \frac{1}{2l^2}$ to 500, 300, and 250 for the three datasets, respectively. For other parameters, please refer to our released code.

For the baselines, we use the default parameters provided in their released code except for necessary modifications to adapt to the datasets. For example, iSDF [20] only supports image input, so we convert the LiDAR scans of the Newer College dataset to depth images using the camera intrinsic parameters that give the same field of view as the LiDAR. We set the voxel size of Voxblox and FIESTA to be 0.05m for the Replica and Cow&Lady datasets, and 0.2m for the Newer College dataset to get a better trade-off between accuracy and efficiency.

For our method and VDB-GPDF [28], a mesh is available from the algorithm directly. For Voxblox [11], FIESTA [13], and iSDF [20], we extract the mesh using the marching cubes algorithm [38] with the voxel size as the marching cubes resolution, except for iSDF, where we use a resolution of 0.02m for the Replica and Cow&Lady datasets and 0.1m for the Newer College dataset.

2) Mesh Metrics: For Replica dataset, we sample 2 million points from the ground truth mesh and the reconstructed mesh, respectively. For Cow&Lady dataset, we sample 54k points from the ground truth point cloud and the reconstructed mesh, respectively. For Newer College dataset, we sample 2 million points from the ground truth point cloud and the reconstructed mesh, respectively.

We compute precision, recall, and F1 score with a threshold of 0.05m for the Replica and Cow&Lady datasets, and 0.2m for the Newer College dataset. A point is considered to be correctly reconstructed if its distance to the other point cloud is less than the threshold.

Besides, we compute the accuracy and the completion, which are defined as:

$$\text{Accuracy} = \frac{1}{|\mathcal{P}|} \sum_{\mathbf{p} \in \mathcal{P}} \min_{\mathbf{q} \in \mathcal{Q}} \|\mathbf{p} - \mathbf{q}\|_2, \tag{50}$$

$$\text{Completion} = \frac{1}{|\mathcal{Q}|} \sum_{\mathbf{q} \in \mathcal{Q}} \min_{\mathbf{p} \in \mathcal{P}} \|\mathbf{p} - \mathbf{q}\|_2, \tag{51}$$

where $\mathcal{P}$ is the reconstructed point cloud and $\mathcal{Q}$ is the ground truth point cloud.

We also choose to use Chamfer-L1 distance in place of the standard Chamfer distance for more intuitive interpretation of the error in meters, which is the average of accuracy and completion.

3) SDF Metrics: For SDF evaluation, we query each method with a regular grid of points covering the bounding box of the ground truth mesh or point cloud. The grid resolution is set to 0.05m for the Replica and Cow&Lady datasets, and 0.2m for the Newer College dataset. Prediction may fail for Voxblox and FIESTA at some query points because they rely on the eight voxel corners to interpolate the SDF value. In such cases, we exclude the failed points when calculating the metrics for Voxblox and FIESTA.

4) Time Metrics: While it is straightforward to measure the prediction time for all methods, measuring the update time is more complex due to the different update strategies employed by each method. For a fair comparison, we measure the average update time per scan for all methods. Specifically, we record the total time taken to process all scans in a dataset (but exclude time of data loading, logging and etc.) and divide it by the number of scans to obtain the average update time per scan. This approach provides a consistent basis for comparing the efficiency of each method's update process.