UNIVERSITY OF CALIFORNIA SAN DIEGO

Machine learning techniques for shape reconstruction and metric-semantic mapping

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Electrical Engineering (Machine Learning and Data Science)

by

Ehsan Zobeidi

Committee in charge:

Professor Nikolay Atanasov, Chair
Professor Manmohan Chandraker
Professor Truong Nguyen
Professor Hao Su

2023

University of California San Diego

2023

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Nikolay Atanasov for his support as the adviser and the chair of my committee. During my PhD, I could work on interesting problems, think on the novel ideas and investigate interesting approaches.

I would like to thank my family for their support. I want to thank my parents for being with me, passionately, for each step of my life. I want to thank my mother to let me share my happiness and sadness with her. I want to thank my father for being my father, my teacher, my supporter and my best friend through my life.

Chapter 2, in full, is a reprint of the material as it appears in ”Ehsan Zobeidi, Alec Koppel and Nikolay Atanasov. Dense Incremental Metric-Semantic Mapping for Multiagent Systems via Sparse Gaussian Process Regression. IEEE Transactions on Robotics, vol. 38, no. 5, pp. 3133-3153, Oct. 2022”. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in full, is a reprint of the material as it appears in ”Ehsan Zobeidi and Nikolay Atanasov. A deep signed directional distance function for object shape representation. arXiv preprint arXiv:2107.11024 (2021)”. The dissertation author was the primary investigator and author of this paper.

VITA

2017	Bachelor of Science in Electrical Engineering (Communications), Sharif University of Technology
2017	Bachelor of Science in Computer Science, Sharif University of Technology
2020	Master of Science in Electrical Engineering (Machine Learning and Data Science), University of California San Diego
2023	Doctor of Philosophy in Electrical Engineering (Machine Learning and Data Science), University of California San Diego

ABSTRACT OF THE DISSERTATION

Machine learning techniques for shape reconstruction and metric-semantic mapping

by

Ehsan Zobeidi

Doctor of Philosophy in Electrical Engineering (Machine Learning and Data Science)

University of California San Diego, 2023

Professor Nikolay Atanasov, Chair

These days, robots are contributing to many aspects of our lives and this role is growing. One of the fundamental problems in robotics is shape representation and mapping, that is necessary for most robotic applications. In this regard, from a stream of lidar scans or RGB-D images we model an object shape or map an environment. One of the main challenges in this regard is an appropriate shape representation with appropriate ray-tracing speed and accuracy. There are many other challenges for mapping like being 3D, continuous, probabilistic, online, containing semantic information, and the possibility of the mapping algorithm to be distributed. In this thesis we propose a novel mapping algorithm based on existing shape representation signed distance function and Gaussian Processes which is able to reconstruct an online, continuous, probabilistic 3-D representations of the geometric surfaces and semantic classes in the environments. Then we extend it to a distributed mapping algorithm that is able to be perform in a network of robots. Additionally, we propose a novel shape representation signed directional distance function (SDDF) that measures signed distance from a query location to a set surface in a desired viewing direction. The main advantage of SDDF, compared to existing shape representations, is that ray tracing can be performed as a look-up operation through a single function call. Additionally, we propose a deep neural network model for SDDF shape learning. A key property of our DeepSDDF model is that it encodes by construction that distance to the surface decreases linearly along the viewing direction. This ensures that the model represents only valid SDDF functions and allows reduction in the input dimensionality and analytical confidence in the distance prediction accuracy even far away from the surface. We show that our DeepSDDF model can represent entire object categories and interpolate or complete shapes from partial views.

Chapter 1

Introduction

Nowadays, robots have various applications in our life like construction, agriculture, healthcare, environment monitoring, transportation and etc. We are now at the time that we can expect the robots to do a complete tasks like surgery, delivery, driving etc. These tasks should be done such that it is fast, efficient, in distributed way, in human understandable terms and most importantly safe for humans. However, these tasks could be broken into more basic and fundamental sub-tasks. One of these fundamental sub-tasks, is shape representation and mapping. Usually the data that we receive, like RGB-D, has 2D structure. However since the robot is working in 3D space, it is essential to provide 3D shape representation and mapping. Usually the data that we receive has discrete structure for example it is either color or depth on the pixels. However, when a robot wants to accomplish a task it needs ray-tracing which requires continuous shape re[presentation and mapping. In practice we are interested in mapping a large environment or learn the object shapes over a very huge class of data sets, all require our methods to be scalable. Usually we have noise in the process of data accumulation and data processing, as a result this is very important to have probabilistic estimate of the map. It helps with realizing how much we are confident about a part of our map, or from which parts we require more observations. If the robots are performing tasks in a priory unknown environment, they should be able to do the mapping online, so they can build the map as they observe the environment and immediately they could be able to perform their tasks. The tasks a robot is assigned to require a semantic understanding of the environment, making semantic mapping very important to accomplish theses tasks. For example, in order to tell a robot go to a room and bring a cup, it should realize different parts of a map including door, cup, table, etc. Usually for a large environment it is more efficient, faster and cheaper to be done by a group of robots in collaboration rather than by a single huge and expensive robot. As a result, one of challenges of mapping is how to provide a distributed mapping algorithm. In this thesis, we have two main works in the first one we use already existing shape representation Signed Distance Function (SDF) and Gaussian Processes to propose a mapping method that addresses aforementioned challenges, and then we propose a novel shape representation Signed Directional Distance Function (SDDF) which has specifications that makes it more appropriate for robotics applications.

1.1 Gaussian Process Mapping

We develop an online probabilistic metric-semantic mapping approach for mobile robot teams relying on streaming RGB-D observations. The generated maps contain full continuous distributional information about the geometric surfaces and semantic labels (e.g., chair, table, wall). Our approach is based on online Gaussian Process (GP) training and inference, and avoids the complexity of GP classification by regressing a truncated signed distance function (TSDF) of the regions occupied by different semantic classes. Online regression is enabled through a sparse pseudo-point approximation of the GP posterior. To scale to large environments, we further consider spatial domain partitioning via a hierarchical tree structure with overlapping leaves. An extension to a multi-robot setting is developed by having each robot execute its own online measurement update and then combine its posterior parameters via local weighted geometric averaging with those of its neighbors. This yields a distributed information processing architecture in which the GP map estimates of all robots converge to a common map of the environment while relying only on local one-hop communication. Our experiments demonstrate the effectiveness of the probabilistic metric-semantic mapping technique in 2-D and 3-D environments in both single and multi-robot settings and in comparison to a deep TSDF neural network approach.

1.2 Signed Directional Distance Function

Object shape and scene surface representations play a key role in enabling autonomous robot mapping, navigation, and manipulation. Mesh, point cloud, signed distance function (SDF), occupancy, and density representations are among the most popular for shape reconstruction. They offer different advantages, such as interpretable visualization for meshes, continuous surface representation for SDFs, and continuous decision boundary for occupancy models. The first contribution of this paper is a new shape representation, called signed directional distance function (SDDF), that measures signed distance from a query location to a set surface in a desired viewing direction. The main advantage of SDDF, compared to existing shape representations, is that distance at any desired view can be obtained as a look-up operation through a single function call. Compared to SDF, our formulation removes post-processing steps like surface extraction or sphere tracing and allows extremely efficient ray tracing, which is crucial for real-time robotics applications. Recent results demonstrate impressive performance of deep neural networks for shape learning, including DeepSDF for SDF, Occupancy networks for occupancy, AtlasNet for meshes, and NeRF for density. Our second contribution is a deep neural network model for SDDF shape learning. A key property of our DeepSDDF model is that it encodes by construction that distance to the surface decreases linearly along the viewing direction. This ensures that the model represents only valid SDDF functions and allows reduction in the input dimensionality and analytical confidence in the distance prediction accuracy even far away from the surface. Similar to DeepSDF, we show that DeepSDDF can model whole object categories and interpolate or complete shapes from partial views.

Chapter 2

Dense Incremental Metric-Semantic Mapping for Multiagent Systems via Sparse Gaussian Process Regression

Autonomous robot systems navigating and executing complex tasks in real-world environments require an understanding of 3-D geometry and semantic context. This chapter develops a probabilistic metric-semantic mapping algorithm, using streaming distance and semantic category observations (see Fig. 2.1), to reconstruct geometric surfaces and their semantic identity (e.g., chairs, tables, doors). To support collaboration among multiple robots, we also consider a distributed setting in which each robot observes the environment locally, with its onboard sensors, and communicates with the other robots to arrive at a common map.

We develop on a map representation which models geometric surfaces implicitly as the zero level-set of a TSDF function [6, 7, 8]. TSDF surface representations have gained popularity due to their high accuracy (compared to regular, adaptive, or sparse grid representations [9, 10]) and ability to directly provide distance and gradient information (compared to explicit mesh representations [11]) useful for specification of safety and visibility constraints. Classification of the geometric surfaces into semantic categories is also necessary to support context understanding and robot task specification [12, 13, 14, 15].

We propose a multi-class TSDF inference approach based on GP regression [16, 17]. GP inference techniques for mapping [18, 19, 20, 21] have key advantages, including uncertainty

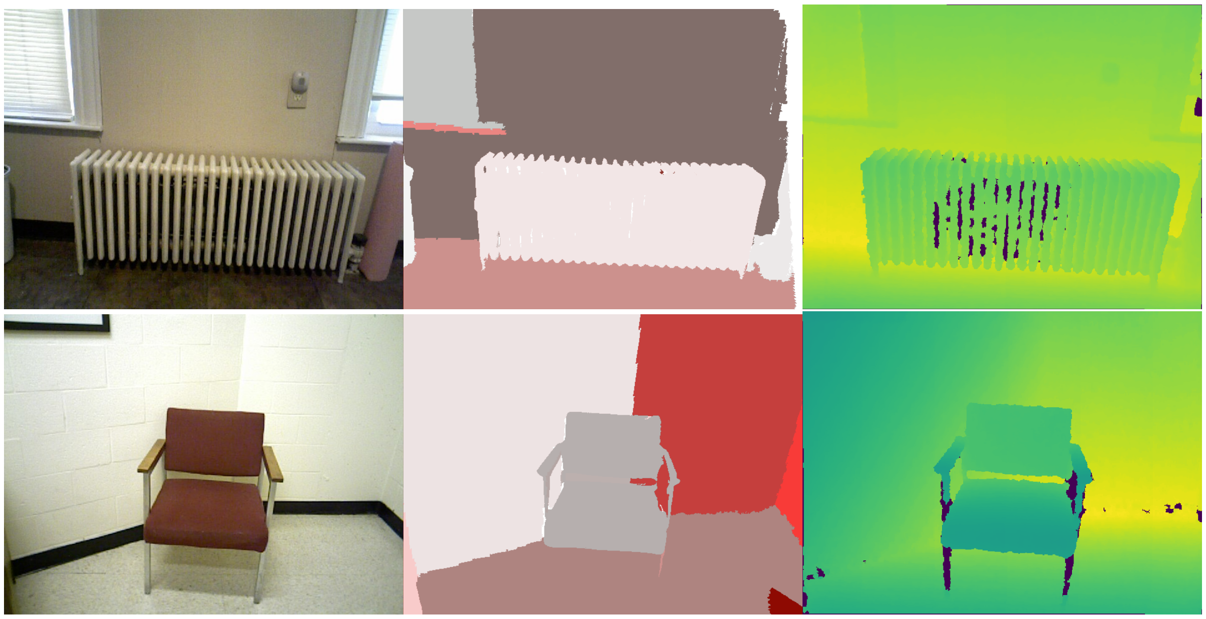


Figure 2.1. RGB images (first column), segmented images (second column), and depth images (third column) used by the proposed approach for online construction of dense metric-semantic maps.

quantification and resolution-free representation of the environment. Map uncertainty quantification is important for motion planning, as it captures sensing errors during collision checking, and for autonomous exploration, as the sensor motion can be planned to reduce uncertainty. In this paper specifically, uncertainty information enables us to formulate 3-D semantic segmentation as a regression problem by comparing the signed distance estimates of different classes. As discussed in more detail below, a regression formulation is conducive to more computationally efficient algorithms than classification for 3-D metric-semantic mapping. The uncertainty information is also needed to weight the map estimates of different robots appropriately when merging them across multiple robots to obtain a consistent joint map of the environment.

Our work contributes to the family of GP mapping algorithms by considering TSDF regression and multi-category classification, instead of binary occupancy mapping. Range sensors, such as LiDARs and depth cameras, do not provide direct TSDF observations because they measure distance in a specific viewing direction rather than to the nearest obstacle surface. To obtain TSDF training examples, we triangulate each depth image into a local mesh surface and measure the distance to it from a set of 3-D locations. We use semantic segmentation to divide the TSDF samples and train separate GPs for each object category. While the GPs are trained separately, all of their posteriors are taken into account when deciding the nearest surface to and the semantic categories of query points. This loosely coupled formulation is possible because segmentation errors in the sensor observations are related to physical proximity of objects of different classes in 3-D space. Hence, (discrete) semantic segmentation errors can be transferred into (continuous) distance measurement errors, allowing us to use GP regression with efficient closed-form incremental updates. In online mapping applications training needs to done incrementally and efficiently. Relying on GP regression with closed-form updates instead of GP classification with iterative approximation has a significant impact on the computational efficiency of our approach.

Efficient incremental training for GP classification [22, 23, 24, 25] is more challenging because the non-Gaussian likelihood of the segmentation data is not conjugate with the GP prior and makes the integral needed for posterior normalization analytically intractable. Exciting recent developments enable scalable variational inference [23] and conjugate GP classification using latent variable augmentation [24], and offer a promising alternative to our approach.

Onboard sensors provide repeated observations of the same scene. While this redundancy is important for mitigating measurement noise, the amount of training data keeps growing over time. Hence, an important consideration is to build maps whose memory and computation requirements are determined by the underlying structure of the environment, rather than the number of observations. While GP training scales cubically with the number training examples, there are various ways to address this bottleneck [26, 27, 28, 29]. We observe that, in our setting, the data can be compressed significantly through averaging before GP training and, notably, this does not affect the posterior TSDF distribution. The remaining training pairs are used as *pseudo-points* [26] to support the continuous GP representation with a finite set of parameters. To reduce the complexity in large maps further, one might consider local kriging, decomposing the spatial domain into subdomains and making predictions at a test location using only the pseudo-points contained within the subdomain. Choosing independent subdomains, however, leads to discontinuities of the predicted TSDF function at the subdomain boundaries. Ensemble methods that construct multiple local estimators and use a weighted combination of their predictions include Bayesian committee machines [30, 31], sparse probabilistic regression [32], or infinite mixtures Gaussian process experts [33]. These techniques avoid the discontinuities of local kriging but their computation cost is still significant for online training. Inspired by the adaptive occupancy representation of Octomap [9], we propose a hierarchical tree structure that decomposes the environment into overlapping subdomains, which prevents discontinuities in the GP posterior. Combining the data compression and hierarchical tree decomposition ideas allows our method to generate dense metric-semantic surfaces and, yet, remain efficient even in large environments.

Finally, we provide a distributed formulation of our TSDF GP regression, enabling multiple robots to collaboratively build a common metric-semantic map. Our distributed inference approach is inspired by probabilistic consensus techniques [34, 35]. We generalize those techniques to enable distributed function approximation instead of fixed-dimension parameter estimation. Each robot updates a local GP pseudo-point approximation and synchronizes its pseudo-point statistics with its one-hop communication neighbors. The number of pseudo-points maintained by a robot is increasing as the robot explores new regions of the environment online. We prove that the local GP estimates of each individual robot converge in *finite time* to the same GP posterior that would have been obtained by a central server using all observations obtained from all robots.

This paper improves the theoretical development for the single-robot setting in our prior work [36] and extends the approach to a decentralized multi-robot setting by introducing a novel approach for distributed incremental sparse GP regression with theoretical guarantees for consistent estimation. The main **contributions** of this work are to:

- develop an scalable incremental GP training and inference algorithm utilizing lossless data compression into a sparse set of pseudo-points (Sec. 2.3.2) and pseudo-point decomposition into a hierarchical tree structure (Sec. 2.3.4),

- achieve 3-D probabilistic metric-semantic mapping from streaming sensor data using the GP algorithm (Sec. 2.4),
- propose a new distributed algorithm for GP regression for directed communication graphs and prove its convergence to the same posterior distribution as centralized GP regression (Sec. 2.5),
- enable a robot team to collaboratively build a common metric-semantic map using local observations and one-hop communication (Sec. 2.6, Sec. 2.7, Sec. 2.8).

Our approach is demonstrated in simulated and real-world datasets and may be used either offline, with all sensory data provided in advance, or online, processing distance and semantic category observations incrementally as they arrive. We compare our incremental GP approach with a state-of-the-art neural network approach for TSDF reconstruction, called Implicit Geometric Regularization (IGR) [1].

2.1 Background

Various representations have been proposed for occupancy or geometric surface estimation from range or depth measurements. Occupancy grid mapping [37] discretizes the environment into a regular voxel grid and estimates the occupancy probability of each voxel independently. A dense voxel representation quickly becomes infeasible for large domains and adaptive resolution data structures, such as an octree, are necessary [9, 38]. While accurate maps may also be constructed using point cloud [39, 40] or surfel [41, 42] representations, such sparse maps do not easily support collision and visibility checking for motion and manipulation planning. Recent work is considering explicit polygonal mesh [43, 44, 11] and implicit signed distance function [45, 46, 47, 48] models. We focus our review on TSDF techniques as they are most closely related to our work.

The seminal work of Curless et al. [6] emphasized the representation power of TSDF and showed that dense surface modeling can be done incrementally using range images. KinectFusion [45] achieved online TSDF mapping and RGB-D camera pose estimation by storing weighted TSDF values in a voxel grid and performing multi-scale iterative closest point (ICP) alignment between the predicted surface and the depth images. Niessner et al. [10] demonstrated that TSDF mapping can be achieved without regular or hierarchical grid data structures by hashing TSDF values only at voxels near the surfaces. These three works inspired a lot of subsequent research, allowing mapping of large environments [49], real-time operation without GPU acceleration [50, 51], map correction upon loop closure [52, 53], and semantic category inference [54]. Bylow et al. [55] propose a direct minimization of TSDF projective depth error instead of relaying on explicit data association or downsampling as in ICP. TSDF maps are accurate and collision checking in them is essentially a look-up operation, prompting their use as an alternative to occupancy grids for robot motion planning and collision checking [56, 48]. Voxblox [47] incrementally builds a (non-truncated) Euclidean signed distance field (ESDF), applying a wavefront algorithm to the hashed TSDF values. Fiesta [48] improves the ESDF construction by introducing two independent queues for inserting and deleting obstacles. Saulnier et al. [57] show that weights of the TSDF values arise as the variance of a Kalman filter and may be used as an uncertainty measure for autonomous exploration and active TSDF mapping.

Most TSDF mapping techniques, however, forgo probabilistic representations in the interest of scalability. Gaussian process (GP) inference has been used to capture correlation in binary occupancy mapping. O'Callaghan et al. [18] is among the first works to apply GP regression to infer a latent occupancy function using data from a range sensor. The GP posterior is squashed to a binary observation model a posteriori to recover occupancy likelihood. The resulting probabilistic least-squares method is more efficient than GP classification but still scales cubically with the amount of training data. To address this, several works [31, 19, 58, 59] rely on sparse kernels to perform separate GP regressions with small subsets of the training data and Bayesian Committee Machines (BCM) to fuse the separate estimates into a full probabilistic occupancy map. Ramos et al. [60, 61] proposed fast kernel approximations to project the occupancy data into a Hilbert space where a logistic regression classifier can distinguish occupied and free space. This idea has been extended to dynamic maps [62, 63] as well as into a variational autoencoder formulation [64] that compresses the local spatial information into a latent low-dimensional feature representation and then decodes it to infer the occupancy of a scene. Guo and Atanasov [65] showed that using a regular grid discretization of the latent function and a decomposable radial kernel leads to special structure of the kernel matrix (kronecker product of Toeplitz matrices) that allows linear time and memory representation of the occupancy distribution.

Augmenting occupancy representations with object and surface category information is an important extension, allowing improved situational awareness and complex mission specification for robots. Several works [13, 66, 67, 68, 69] employ conditional random fields (CRFs) to capture semantic information. Vineet et al. [66] provide incremental reconstruction and semantic segmentation of outdoor environments using a hash-based voxel map and a mean-field inference algorithm for densely-connected CRFs. Grinvald et al. [54] reconstruct individual object shapes from multi-view segmented images and assemble the estimates in a voxelized TSDF map. Gan et al. [70] propose a continuous-space multi-class mapping approach, which relies on a Dirichlet class prior, a Categorical observation likelihood, and Bayesian kernel inference to extrapolate the class likelihoods to continuous space. Rosinol et al. [11] provide a modern perception library combining the state of the art in geometric and semantic understanding. Zheng et al. [71] incorporate spatial information across multiple levels of abstraction and form a probability distribution over semantic attributes and geometric representations of places using TopoNet, a deep sum-product neural network. Wang et al. [8] propose a fully convolutional neural network for semantic 3-D reconstruction that takes an octree of TSDFs fused from different camera views as input and generates a semantically labeled octree as output. IGR [1] uses a deep fully connected network with skip connections and softplus nonlinearity to map a 3D point and a latent shape vector of an object category to the signed distance from that point to the object surface. The authors observe that the norm of the gradient of an SDF function should be 1 everywhere and incorporate this in the training loss. Instead of single-object reconstruction, recent deep learning methods have focused on complete scene reconstruction using distance fields, radiance fields, and multi-view stereo [72, 73, 74, 75]. These techniques have shown impressive performance but currently need all training data at once and provide only most likely estimates instead of complete posterior probability distributions. In contrast, online mapping applications require incremental update and expansion of the reconstructed scene as new data arrives. Also, uncertainty quantification is needed to support safe navigation, autonomous exploration, and collaborative multi-robot mapping.

In many applications, mapping may be performed by a team of collaborating robots. Relying on centralized estimation has limitations related to the communication, computation, and storage requirements of collecting all robot measurements and map estimates at a central server. Developing distributed techniques that allow local inference and storage at each robot, communication over few-hop neighborhoods, and consensus among the robot estimates is important. Techniques extending consensus [76] to distributed probabilistic estimation [77, 78, 79, 34] are closely related to our work. These works show that distributed estimation of a finite-dimensional parameter is consistent when the probability density functions maintained by different nodes are averaged over one-hop neighborhoods in strongly connected di-graphs. We extend these results to distributed probabilistic estimation functions relying on local averaging of sparse (pseudo-point) GP distributions. Specific to cooperative semantic mapping, Choudhary et al. [80] develop distributed pose-graph optimization algorithms based on successive and Jacobi over-relaxation to split the computation among the robots. Koch et al. [81] develop a parallel multi-threaded implementation for cooperative 2-D SDF mapping. Lajoie et al. [82] propose a distributed SLAM approach with peer-to-peer communication that rejects spurious inter-robot loop closures using pairwise consistent measurement sets.

2.2 Problem Formulation

Consider a team of n robots operating in an unknown environment, represented by two disjoint sets $\mathscr{O} \subset \mathbb{R}^3$ and $\mathscr{F} \subset \mathbb{R}^3$, comprising obstacles and free space, respectively. The obstacle region is a pairwise disjoint union, $\mathscr{O} = \cup_{l=1}^{\mathscr{C}} \mathscr{O}_l$, of $\mathscr{C}$ closed sets, each denoting the region occupied by object instances from the same semantic class. For example, $\mathscr{O}_1$ may be the space occupied by all chairs, while $\mathscr{O}_2$ may be the space occupied by all tables.

Each robot is equipped with a sensor, such as a lidar scanner or an RGB-D camera, that provides distance and class observations of the objects in its vicinity. We assume that the position $\mathbf{p}_t^i \in \mathbb{R}^3$ and orientation $\mathbf{R}_t^i \in SO(3)$ of each sensor $i \in \mathscr{V}$ at time step t are known, e.g., from a localization algorithm running onboard the robots. We model a sensor observation as a set of unit-vector rays $\{\eta_k^i \in \mathbf{R}^3 | \|\eta_k^i\| = 1\}$, e.g., corresponding to lidar scan rays or RGB-D image pixels. At time t, the k-th sensor ray of robot i, starts at position $\mathbf{p}_t^i$ and has direction $\mathbf{R}_t^i \eta_k^i$. Each ray measures the distance to and semantic class of the object that it intersects with first. In practice, the class measurements are obtained from a semantic segmentation algorithm (e.g., [83]), applied to the RGB image or lidar scan (see Fig. 2.1), while the distance measurements are provided either as a transformation of the depth image or directly from the lidar scan.

Definition 1. *A* sensor observation *of robot i at time t is a collection of distance $\lambda_{t,k}^i \in \mathbb{R}_{\geq 0}$ and object class $c_{t,k}^i \in \{1, ..., \mathscr{C}\}$ measurements acquired from position $\mathbf{p}_t^i$ along the sensor rays $\mathbf{R}_t^i \eta_k^i$.*

Given sensor poses $\mathbf{p}_t^i$, $\mathbf{R}_t^i$ and streaming observations $\lambda_{t,k}^i$, $c_{t,k}^i$ for $t = 1, 2, \ldots$, the main objective of this work is to construct a 3-D metric-semantic map of the observed environment incrementally by estimating the object class sets $\mathscr{O}_l$. We use an implicit TSDF representation of the sets $\mathscr{O}_l$.

Definition 2. *The* truncated signed distance function *(TSDF) $f_l(\mathbf{x})$ of object class $\mathscr{O}_l$ is the*

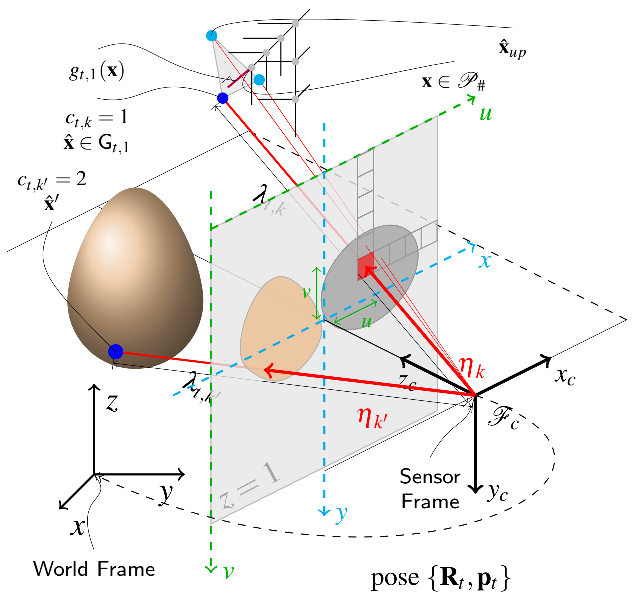


Figure 2.2. Sensor observation at time t showing the distance $\lambda_{t,k}$, $\lambda_{t,k'}$ and class $c_{t,k}$, $c_{t,k'}$ measurements obtained along sensors rays η_k, η'_k when a camera sensor is at position $\mathbf{p}_t$ with orientation $\mathbf{R}_t$. The pseudo-points $\mathscr{P}_\#$ (see Sec. 2.4.1) close to the observed surface are shown in gray.

signed distance from $\mathbf{x}$ *to the boundary* $\partial\mathscr{O}_l$*, truncated to a maximum of* $\bar{d} \geq 0$*:*

$$f_l(\mathbf{x}) := \begin{cases} -\min\left(d(\mathbf{x}, \partial\mathscr{O}_l), \bar{d}\right) & \textit{if } \mathbf{x} \in \mathscr{O}_l \\ \min\left(d(\mathbf{x}, \partial\mathscr{O}_l), \bar{d}\right) & \textit{if } \mathbf{x} \notin \mathscr{O}_l, \end{cases} \tag{2.1}$$

$$d(\mathbf{x}, \partial\mathscr{O}_l) := \inf_{y \in \partial\mathscr{O}_l} \|x - y\|.$$

As we explained in Sec. 2.4, we model the effect of noise from various sources as Gaussian noise on the TSDF values. We develop incremental sparse Gaussian Process regression to maintain distributions $\mathscr{GP}(\mu^i_{t,l}(\mathbf{x}), k^i_{t,l}(\mathbf{x}, \mathbf{x}'))$ over the TSDF functions $f_l(\mathbf{x})$ in (3.1) at each robot i, conditioned on the sensor observations $\left\{\lambda^i_{\tau,k}, c^i_{\tau,k}\right\}$ up to time t.

In Sec. 2.3, we review a sparse pseudo-point formulation of GP regression and introduce lossless compression and hierarchical decomposition of the training data to acheive incremental and scalable training. In Sec. 2.4, we apply our general GP regression algorithm to the metric-semantic mapping problem, discussing construction of GP training data from the sensor measurements and semantic class prediction based on the TSDF function distributions of the different object classes. Next, we extend our approach from a centralized single-robot to a distributed multi-robot formulation. We develop new techniques for distributed incremental sparse GP regression in Sec. 2.5 and apply them to the collaborative semantic TSDF mapping problem in Sec. 2.6. Our method allows each robot to update its own TSDF GP model using local sensor observations and one-hop communication with its neighbors, yet guarantees theoretically that the individual GP models converge to the same posterior distribution as centralized GP regression.

2.3 Data Compression and Decomposition for Incremental Sparse Gaussian Process Regression

This section reviews sparse Gaussian Process regression and introduces a new approach for compressing and decomposing training data acquired by repeated observation of the same locations. The latter is typical when an onboard robot sensor observes the same environment multiple times as discussed in Sec. 2.4.1. When repeated observations are present, our data compression allows training a pseudo-point GP model with much fewer samples, yet provably generates the same GP posterior that would have been computed using the full uncompressed training set. The pseudo-point GP model and data compression allow us to design an efficient incremental GP algorithm that updates the posterior with sequential data instead of recomputing it from scratch. To handle large datasets, we introduce a hierarchical tree structure of pseudo-points such that separate GPs may be trained efficiently within each tree leaf.

2.3.1 Background on Sparse GP Regression

A Gaussian Process is a set of random variables such that the joint distribution of any finite subset of them is Gaussian. A GP-distributed function $f(\mathbf{x}) \sim \mathscr{GP}(\mu_0(\mathbf{x}), k_0(\mathbf{x}, \mathbf{x}'))$ is defined by a mean function $\mu_0(\mathbf{x})$ and a covariance (kernel) function $k_0(\mathbf{x}, \mathbf{x}')$. The mean and covariance are such that for any finite set $\mathscr{X} = \{\mathbf{x}_j\}_j$, the random vector $f(\mathscr{X}) := \left[f(\mathbf{x}_1), \ldots, f(\mathbf{x}_{|\mathscr{X}|})\right]^\top \in \mathbb{R}^{|\mathscr{X}|}$ has mean with j-th element $\mu_0(\mathbf{x}_j)$ and covariance matrix with (j,l)-th element $k_0(\mathbf{x}_j, \mathbf{x}_l)$ for $j,l = 1, \ldots, |\mathscr{X}|$. To avoid introducing additional symbols, we use the notation $f(\mathscr{X})$ to mean the application of the vector function $f(\mathbf{x})$ to each element of the set $\mathscr{X}$. Given a training set $\mathscr{D} = \{(\mathbf{x}_j, y_j)\}_{j=1}^{|\mathscr{X}|}$, generated according to $y_j = f(\mathbf{x}_j) + \eta_j$ with independent Gaussian noise $\eta_j \sim \mathscr{N}(0, \sigma^2)$, the posterior distribution of the random function $f(\mathbf{x})$ can be obtained from the joint distribution of the value $f(\mathbf{x})$ at an arbitrary location $\mathbf{x}$ and the random vector $\mathbf{y} := \left[y_1, \ldots, y_{|\mathscr{X}|}\right]^\top$ of measurements. In detail, the joint distribution is:

$$\begin{bmatrix} f(\mathbf{x}) \\ \mathbf{y} \end{bmatrix} \sim \mathscr{N}\left(\begin{bmatrix} \mu_0(\mathbf{x}) \\ \mu_0(\mathscr{X}) \end{bmatrix}, \begin{bmatrix} k_0(\mathbf{x}, \mathbf{x}) & k_0(\mathbf{x}, \mathscr{X}) \\ k_0(\mathscr{X}, \mathbf{x}) & k_0(\mathscr{X}, \mathscr{X}) + \sigma^2 I \end{bmatrix}\right),$$

while the corresponding conditional distribution $f(\mathbf{x})|\mathscr{X}, \mathbf{y} \sim \mathscr{GP}(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$ has mean and covariance functions [16]:

$$\begin{aligned} \mu(\mathbf{x}) &:= \mu_0(\mathbf{x}) + k_0(\mathbf{x}, \mathscr{X})(k_0(\mathscr{X}, \mathscr{X}) + \sigma^2 I)^{-1}(\mathbf{y} - \mu_0(\mathscr{X})), \\ k(\mathbf{x}, \mathbf{x}') &:= k_0(\mathbf{x}, \mathbf{x}') - k_0(\mathbf{x}, \mathscr{X})(k_0(\mathscr{X}, \mathscr{X}) + \sigma^2 I)^{-1} k_0(\mathscr{X}, \mathbf{x}'). \end{aligned} \tag{2.2}$$

Computing the GP posterior has cubic complexity in the number of observations $|\mathscr{X}|$ due to the matrix inversion in (2.2).

Inspired by Snelson and Ghahramani [26], we introduce a sparse approximation to the GP posterior in (2.2) using a set of *pseudo-points* $\mathscr{P} \subset \mathscr{D}$ whose number is $|\mathscr{P}| \ll |\mathscr{X}|$. Maintaining a GP distribution only over the pseudo-points is sufficient to obtain a high-fidelity approximation of the true GP posterior, assuming that the training data are independent samples drawn from the pseudo-point GP. The key idea is to first determine the distribution $\mathscr{N}(\mu, \Sigma)$ of $\mathbf{f} := f(\mathscr{P})$

conditioned on $\mathscr{X}$, $\mathbf{y}$ according to (2.2):

$$\begin{aligned}\mu &:= \mu_0(\mathscr{P}) + k_0(\mathscr{P},\mathscr{X})(k_0(\mathscr{X},\mathscr{X}) + \sigma^2 I)^{-1}(\mathbf{y} - \mu_0(\mathscr{X})) \\ &= \mu_0(\mathscr{P}) + k_0(\mathscr{P},\mathscr{P})\left(k_0(\mathscr{P},\mathscr{P}) + \Gamma\right)^{-1}\gamma \\ \Sigma &:= k_0(\mathscr{P},\mathscr{P}) - k_0(\mathscr{P},\mathscr{X})\left(k_0(\mathscr{X},\mathscr{X}) + \sigma^2 I\right)^{-1} k_0(\mathscr{X},\mathscr{P}), \\ &= k_0(\mathscr{P},\mathscr{P})\left(k_0(\mathscr{P},\mathscr{P}) + \Gamma\right)^{-1} k_0(\mathscr{P},\mathscr{P})\end{aligned} \tag{2.3}$$

where:

$$\begin{aligned}\Gamma &:= k_0(\mathscr{P},\mathscr{X})\left(\Lambda + \sigma^2 I\right)^{-1} k_0(\mathscr{X},\mathscr{P}) \\ \Lambda &:= k_0(\mathscr{X},\mathscr{X}) - k_0(\mathscr{X},\mathscr{P}) k_0(\mathscr{P},\mathscr{P})^{-1} k_0(\mathscr{P},\mathscr{X}) \\ \gamma &:= k_0(\mathscr{P},\mathscr{X})\left(\Lambda + \sigma^2 I\right)^{-1}(\mathbf{y} - \mu_0(\mathscr{X}))\end{aligned}$$

Using the definitions of information matrix $\Omega := \Sigma^{-1}$ and information mean $\omega := \Omega\mu$, we can equivalently write:

$$\begin{aligned}\omega &= \Omega\mu_0(\mathscr{P}) + k_0(\mathscr{P},\mathscr{P})^{-1}\gamma, \\ \Omega &= k_0(\mathscr{P},\mathscr{P})^{-1}\left(k_0(\mathscr{P},\mathscr{P}) + \Gamma\right) k_0(\mathscr{P},\mathscr{P})^{-1}.\end{aligned} \tag{2.4}$$

Then, the posterior density of $f(\mathbf{x})$ conditioned on $\mathscr{X}, \mathbf{y}$ is:

$$p(f(\mathbf{x})|\mathscr{X},\mathbf{y}) = \int p(f(\mathbf{x})|\mathbf{f}) p(\mathbf{f}|\mathscr{X},\mathbf{y}) d\mathbf{f} \tag{2.5}$$

which is a GP with mean and covariance functions:

$$\begin{aligned}\mu(\mathbf{x}) &= \mu_0(\mathbf{x}) + k_0(\mathbf{x},\mathscr{P})k_0(\mathscr{P},\mathscr{P})^{-1}\left(\Omega^{-1}\omega - \mu_0(\mathscr{P})\right)\\ k(\mathbf{x},\mathbf{x}') &= k_0(\mathbf{x},\mathscr{P})k_0(\mathscr{P},\mathscr{P})^{-1}\Omega^{-1}k_0(\mathscr{P},\mathscr{P})^{-1}k_0(\mathscr{P},\mathbf{x}')\\ &\quad + k_0(\mathbf{x},\mathbf{x}') - k_0(\mathbf{x},\mathscr{P})k_0(\mathscr{P},\mathscr{P})^{-1}k_0(\mathscr{P},\mathbf{x}'). \end{aligned} \tag{2.6}$$

If we assume that conditioned on $\mathscr{P}$, the measurements y_j are generated independently, i.e., Λ is approximated by a diagonal matrix with elements

$$\lambda(\mathbf{x}_j) := k_0(\mathbf{x}_j,\mathbf{x}_j) - k_0(\mathbf{x}_j,\mathscr{P})k_0(\mathscr{P},\mathscr{P})^{-1}k_0(\mathscr{P},\mathbf{x}_j),$$

then the complexity of computing μ, Σ in (2.3) (training) and $\mu(\mathbf{x})$, $k(\mathbf{x},\mathbf{x}')$ in (2.6) (testing) are $O(|\mathscr{P}|^2|\mathscr{X}|+|\mathscr{P}|^3)$ and $O(|\mathscr{P}|^2)$, respectively, instead of $O(|\mathscr{X}|^3)$ and $O(|\mathscr{X}|^2)$ without pseudo-points in (2.2). The use of pseudo-points leads to significant computational savings when $|\mathscr{P}| \ll |\mathscr{X}|$. To select the locations and values of the pseudo-points $\mathscr{P}$, the main approach in the literature is to perform iterative optimization to maximally approximate the training data [32, 84, 21]. However, iterative optimization is expensive in an incremental or distributed setting, which is the focus of this paper. Instead, we exploit the structure of the TSDF reconstruction problem to select pseudo-points close to the end points of the senor rays (observed surface) on a latent grid over the environment. This allows selecting pseudo-points and computing the GP posterior very efficiently. We assume that the kernel parameters are optimized offline and focus on online computation of the terms in (2.6), needed for prediction.

2.3.2 Repeated Input Data Compression

Next, we detail a way to obtain additional savings in terms of data storage requirements. Specifically, if the training data $\mathscr{D} = (\mathscr{X},\mathbf{y})$ contains repeated observations from the same locations, i.e., the points in $\mathscr{X}$ are not unique, then the GP training complexity can be reduced from cubic in $|\mathscr{X}|$ to cubic in the number of distinct points in $\mathscr{X}$. We formalize this in the following proposition, which establishes that the GP posterior is unchanged if we compress the observations in $\mathbf{y}$ obtained from the same locations in $\mathscr{X}$. Repeated observations are meaningful when there is measurement noise, i.e., $\sigma > 0$. In this case the matrix $k_0(\mathscr{X},\mathscr{X}) + \sigma^2 I$ in (2.2) is never singular.

Proposition 1. *Consider $f(\mathbf{x}) \sim \mathscr{GP}(\mu_0(\mathbf{x}), k_0(\mathbf{x},\mathbf{x}'))$. Let:*

$$\mathscr{X} = \{\mathbf{x}_1\ ,\dots,\mathbf{x}_1\ \ ,\mathbf{x}_2\ ,\dots,\mathbf{x}_2\ \ ,\dots,\mathbf{x}_n\ ,\dots,\mathbf{x}_n\ \ \}$$
$$\mathbf{y} = [y_{1,1},\dots,y_{1,m_1},y_{2,1},\dots,y_{2,m_2},\dots,y_{n,1},\dots,y_{n,m_n}]^\top$$

be data generated from the model $y_{i,j} = f(\mathbf{x}_i) + \eta_{i,j}$ with $\eta_{i,j} \sim \mathscr{N}(0,\sigma^2)$ for $i = 1,\dots,n$ and $j = 1,\dots,m_i$. Let:

$$\mathscr{P} = \{\mathbf{x}_1,\dots,\mathbf{x}_n\},\ \zeta = \left[\frac{1}{m_1}\sum_{j=1}^{m_1} y_{1,j},\dots,\frac{1}{m_n}\sum_{j=1}^{m_n} y_{n,j}\right]^\top \tag{2.7}$$

be a compressed version of the data generated from $f(\mathbf{x}_i)$ with noise $\hat{\eta}_i \sim \mathscr{N}(0,\frac{\sigma^2}{m_i})$. Then, $f(\mathbf{x})|\mathscr{X},\mathbf{y}$ and $f(\mathbf{x})|\mathscr{P},\zeta$ have the same Gaussian Process distribution $\mathscr{GP}(\mu(\mathbf{x}),k(\mathbf{x},\mathbf{x}'))$ with:

$$\begin{aligned}\mu(\mathbf{x}) &= \mu_0(\mathbf{x}) + k_0(\mathbf{x},\mathscr{P})Z(\zeta - \mu_0(\mathscr{P})),\\ k(\mathbf{x},\mathbf{x}') &= k_0(\mathbf{x},\mathbf{x}') - k_0(\mathbf{x},\mathscr{P})Zk_0(\mathscr{P},\mathbf{x}'),\end{aligned} \tag{2.8}$$

where $Z^{-1} := k_0(\mathscr{P},\mathscr{P}) + \sigma^2\,\mathrm{diag}(\mathbf{m})^{-1}$ and $\mathbf{m}$ is a vector with elements m_i.

Proof. The distribution of $f(\mathbf{x})|\mathscr{X},\mathbf{y}$ is provided in (2.2). Using the data $\mathscr{P}$, ζ, instead of $\mathscr{X}$, $\mathbf{y}$, to compute the posterior GP distribution of $f(\mathbf{x})$, according to (2.2), leads to the expression in (2.8). We need to show that (2.2) and (2.8) are equal given the relationship between $\mathscr{X}$, $\mathbf{y}$ and $\mathscr{P}$, ζ in (2.7). Let E be a binary matrix defined such that $k_0(\mathscr{X},\mathbf{x}) = Ek_0(\mathscr{P},\mathbf{x})$. Note that $k_0(\mathscr{X},\mathscr{X}) = Ek_0(\mathscr{P},\mathscr{P})E^\top$, $k_0(\mathbf{x},\mathscr{X}) = k_0(\mathbf{x},\mathscr{P})E^\top$, $E^\top E = \mathrm{diag}(\mathbf{m})$, and $\zeta =$

$(E^\top E)^{-1}E^\top \mathbf{y}$. Using these expressions in (2.2) leads to:

$$\begin{aligned}
\mu(\mathbf{x}) &= \mu_0(\mathbf{x}) + \\
&\quad k_0(\mathbf{x},\mathscr{P})E^\top(Ek_0(\mathscr{P},\mathscr{P})E^\top + \sigma^2 I)^{-1}(\mathbf{y} - E\mu_0(\mathscr{P})), \\
k(\mathbf{x},\mathbf{x}') &= k_0(\mathbf{x},\mathbf{x}') - \qquad\qquad (2.9)\\
&\quad k_0(\mathbf{x},\mathscr{P})E^\top(Ek_0(\mathscr{P},\mathscr{P})E^\top + \sigma^2 I)^{-1}Ek_0(\mathscr{P},\mathbf{x}').
\end{aligned}$$

An application of the matrix inversion lemma followed by algebraic manipulation shows that

$$E^\top(Ek_0(\mathscr{P},\mathscr{P})E^\top + \sigma^2 I)^{-1} = \left(k_0(\mathscr{P},\mathscr{P}) + \sigma^2(E^\top E)^{-1}\right)^{-1}(E^\top E)^{-1}E^\top = Z(E^\top E)^{-1}E^\top.$$

Replacing this and $\zeta = (E^\top E)^{-1}E^\top \mathbf{y}$ in (2.9) shows that the GP distributions of $f(\mathbf{x})|\mathscr{X},\mathbf{y}$ and $f(\mathbf{x})|\mathscr{P},\zeta$ are equal.

$\square$

Prop. 1 allows us to summarize a training set $\mathscr{X}$, $\mathbf{y}$ by keeping the distinct points $\mathscr{P} \subset \mathscr{X}$ as well as the average observation value $\zeta(\mathbf{p})$ and number of times $m(\mathbf{p})$ that each point $\mathbf{p} \in \mathscr{P}$ has been observed. Given these statistics, the mean function $\mu(\mathbf{x})$ and covariance function $k(\mathbf{x},\mathbf{x}')$ of the posterior GP can be obtained according to (2.8) with $\zeta := \zeta(\mathscr{P})$ and $\mathbf{m} := m(\mathscr{P})$. The number of observations $\mathbf{m}$ determines the value of the matrix Z in (2.9) and correctly scales the influence of the points that are observed more frequently. When the training points $\mathscr{X}$ contain many repetitions, the subset $\mathscr{P}$ of distinct points is a natural choice of pseudo-points (Sec. 2.3.1). In this case, Prop. 1 shows that the GP posterior obtained from training with $\mathscr{P}$ is *exactly equal* to the GP posterior obtained from training with $\mathscr{X}$. This is illustrated in Fig. 2.3, where a dataset with repeated observations is used for GP regression of $\sin(x)$ with or without the compression of Prop. 1. Our result is related to a general distribution-to-distribution regression formulation in [85]. It is interesting to consider whether this connection can allow removing the assumption that the training data contains observations from the same locations, while keeping the simplicity

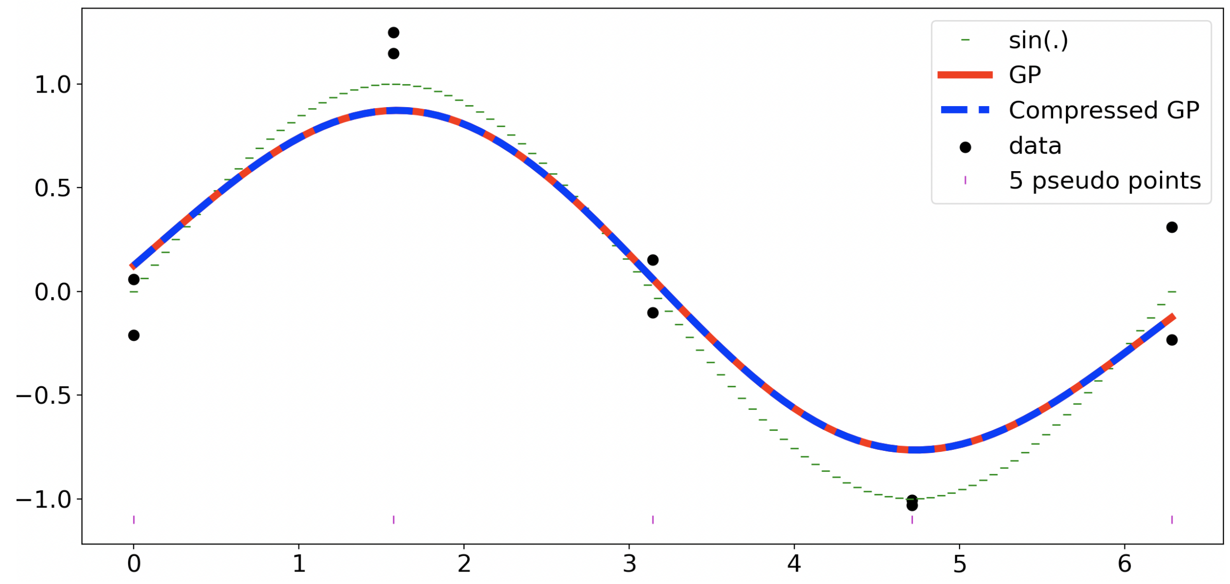


Figure 2.3. Noisy data (red) obtained from a sine function (teal) at 5 equidistant pseudo-points (magenta). Two measurements are obtained at each pseudo-point. GP regression using all data is equivalent to GP regression with a compressed dataset described in Prop. 1.

of computing summary statistics in (2.7).

The raw sensor data in the mapping problem does not directly satisfy the assumption of Prop. 1. In Sec. 2.4.1, we first construct a set of pseudo-points with TSDF values computed from the sensor data and only then apply Prop. 1 to compute the GP posterior efficiently. We exploit this compression technique for efficient incremental GP training since the same pseudo-point locations are observed multiple times.

2.3.3 Incremental Compressed Sparse GP Regression

Suppose now that, instead of a single training set $\mathscr{D}$, the data are provided sequentially, i.e., an additional dataset $\tilde{\mathscr{D}}_t$ of points $\tilde{\mathscr{X}}_t$ with labels $\tilde{\mathbf{y}}_t$ is provided at each time step t. The cumulative data up to time t are $\mathscr{D}_t := \cup_{\tau=1}^{t} \tilde{\mathscr{D}}_\tau$. Based on Prop. 1, we can define an incrementally growing set of pseudo-points $\mathscr{P}_t$ with associated number of observations $m_t(\mathbf{p})$ and average observation $\zeta_t(\mathbf{p})$ for $\mathbf{p} \in \mathscr{P}_t$ and observation precision Z_t. We show how to update these statistics when a new dataset $\tilde{\mathscr{D}}_{t+1} = (\tilde{\mathscr{X}}_{t+1}, \tilde{\mathbf{y}}_{t+1})$ arrives at time $t+1$. Let $\tilde{\mathscr{P}}_{t+1}$ be the set of unique points in $\tilde{\mathscr{X}}_{t+1}$ with number of observations $\tilde{m}_{t+1}(\mathbf{p})$ and average observation $\tilde{\zeta}_{t+1}(\mathbf{p})$

for $\mathbf{p} \in \tilde{\mathscr{P}}_{t+1}$. The update of $\mathscr{P}_t$, $m_t(\mathbf{p})$ and $\zeta_t(\mathbf{p})$ is:

$$\begin{aligned}
\mathscr{P}_{t+1} &= \mathscr{P}_t \cup \tilde{\mathscr{P}}_{t+1} \\
m_{t+1}(\mathbf{p}) &= \begin{cases} m_t(\mathbf{p}) + \tilde{m}_{t+1}(\mathbf{p}), & \text{if } \mathbf{p} \in \mathscr{P}_t, \\ \tilde{m}_{t+1}(\mathbf{p}), & \text{else,} \end{cases} \\
\zeta_{t+1}(\mathbf{p}) &= \begin{cases} \frac{m_t(\mathbf{p})\zeta_t(\mathbf{p}) + \tilde{m}_{t+1}(\mathbf{p})\tilde{\zeta}_{t+1}(\mathbf{p})}{m_{t+1}(\mathbf{p})}, & \text{if } \mathbf{p} \in \mathscr{P}_t, \\ \tilde{\zeta}_{t+1}(\mathbf{p}), & \text{else.} \end{cases}
\end{aligned} \tag{2.10}$$

To update the observation precision Z_t, first consider the existing pseudo-points $\mathscr{P}_t$. Let l be the index of $\mathbf{p} \in \mathscr{P}_t$ in Z_t. Define $\varepsilon_l := \sigma^2 \left(\frac{1}{m_{t+1}(\mathbf{p})} - \frac{1}{m_t(\mathbf{p})} \right)$, $B_0 := Z_t$, and for $l = 1, \ldots, |\mathscr{P}_t|$:

$$B_{l+1} = \left(B_l^{-1} + \varepsilon_l \mathbf{e}_l \mathbf{e}_l^\top \right)^{-1} = B_l - \frac{B_l \mathbf{e}_l \mathbf{e}_l^\top B_l}{\frac{1}{\varepsilon_l} + \mathbf{e}_l^\top B_l \mathbf{e}_l}. \tag{2.11}$$

This update is applied only to $\tilde{\mathscr{P}}_{t+1} \cap \mathscr{P}_t$ because the rest of the pseudo-points have $\varepsilon_l = 0$ and, hence, $B_{l+1} = B_l$. With some abuse of notation, let $B := B_{|\mathscr{P}_t|}$ be the observation precision after all $\mathbf{p} \in \mathscr{P}_t$ have been updated. Finally, we update B by introducing the pseudo-points $\tilde{\mathscr{P}}_{t+1} \setminus \mathscr{P}_t$ that have been observed for the first time:

$$Z_{t+1} = \begin{bmatrix} B^{-1} & C \\ C^\top & D \end{bmatrix}^{-1} = \begin{bmatrix} B + BCSC^\top B & -BCS \\ -SC^\top B & S \end{bmatrix}, \tag{2.12}$$

where $C := k_0(\mathscr{P}_t, \tilde{\mathscr{P}}_{t+1} \setminus \mathscr{P}_t)$, $D := k_0(\tilde{\mathscr{P}}_{t+1} \setminus \mathscr{P}_t, \tilde{\mathscr{P}}_{t+1} \setminus \mathscr{P}_t) + \sigma^2 \operatorname{diag}(\tilde{m}_{t+1}(\tilde{\mathscr{P}}_{t+1} \setminus \mathscr{P}_t))^{-1}$, and $S := (D - C^\top BC)^{-1}$. The equality in (2.12) follows from the block matrix inversion lemma, which relates the blocks of a matrix inverse to the inverse of their Schur complement [86, Ch. 9.1]. By recursively tracking these matrix inverses, the posterior update can be executed efficiently every time a new observation arrives with complexity that is cubic in the number of new distinct points. This is a significant improvement over naïve GP training.

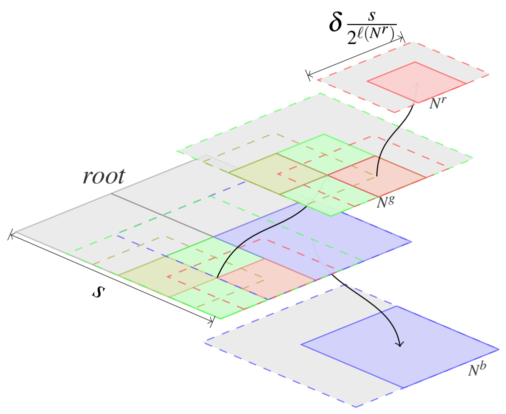


Figure 2.4. Illustration of a hierarchical tree data structure, containing two pseudo-points (blue and cyan) in two dimensions. The support regions $\mathscr{S}(\cdot)$ and test regions $\mathscr{T}(\cdot)$ of three nodes N^r, N^g, N^b are shown as dashed and filled areas with red, green, and blue color, respectively. No pseudo-points are contained in the test region $\mathscr{T}(N^g)$ (filled green) of node N^g but two pseudo-points are in its support region $\mathscr{S}(N^g)$ (dashed green). In this example, the maximum number of allowable pseudo-points for each region is $max(N) = 1$, so node N^g is split into the red (N^r) and yellow (not labeled) regions. The cyan pseudo-point belongs to both $\mathscr{P}_t(N^b)$ and $\mathscr{P}_t(N^r)$.

2.3.4 Hierarchical Tree Structure

Even after compressing the training data to a set of distinct pseudo-points using Prop. 1, the GP training complexity still scales cubically with the number of pseudo-points. In the mapping problem, the correlation between two sufficiently distant points is negligible. To ensure that online training is possible for large datasets, we develop a hierarchical tree data structure that decomposes the sample space into overlapping regions to store the pseudo-points. Our data structure is similar to an octree [9] but the regions associated with the tree nodes overlap. We train separate GPs in each region, which is efficient since the maximum number of pseudo-points per region is fixed. The resolution of the pseudo-points and the kernel parameters are fixed and are not different in different levels of the tree. The region overlap serves to eliminate discontinuities in the resulting GP estimate. The GPs associated with different regions are not statically independent because they may share some of the same pseudo-points and associated measurements. However, to enable efficient inference, we approximate the joint GP model over all pseudo-points with separately trained GP models, each using training data only from one region. At test time, the value of a query point is inferred using only the parameters of the corresponding region according to (2.8). The overlapping regions are illustrated in Fig. 2.4.

Formally, a hierarchical tree structure of pseudo-points in d dimensions is a tree data structure such that each internal node has 2^d children. Each node N is associated with a spatial region. The root is associated with a hypercube with side length $s > 0$, which is recursively subdivided into 2^d overlapping regions by the child nodes. Each node N maintains the following information:

1. $\ell(N) \geq 0$: level of N in the tree, starting from 0 at the root node.
2. $ctr(N) \in \mathbb{R}^d$: center of the region associated with N.
3. $\mathscr{S}(N) := \{\mathbf{x} \in \mathbb{R}^d \mid \|\mathbf{x} - ctr(N)\|_\infty \leq \delta \frac{s}{2^{\ell(N)+1}}\}$: support region of N with $\delta > 1$.
4. $\mathscr{T}(N) := \{\mathbf{x} \in \mathbb{R}^d \mid \|\mathbf{x} - ctr(N)\|_\infty \leq \frac{s}{2^{\ell(N)+1}}\}$: test region of N.
5. $\mathscr{P}(N) \subseteq \mathscr{S}(N) \cap \mathscr{P}_\#$: set of pseudo-points assigned to this node
6. $max(N)$: node N splits into 2^d children if the number of observed pseudo-points $\mathscr{P}(N)$ exceeds $max(N)$
7. $children(N)$: empty set if N is a leaf and, otherwise, a set of 2^d nodes at level $\ell(N)+1$ with centers in $\{ctr(N) + \mathbf{c} \mid c_i \in \{-\frac{s}{2^{\ell(N)+1}}, +\frac{s}{2^{\ell(N)+1}}\}\}$.

The pseudo-points $\mathscr{P}_t$ observed up to time t are stored in the hierarchical tree structure. The points assigned to node N at time t are $\mathscr{P}_t(N) := \mathscr{P}_t \cap \mathscr{S}(N)$. The pseudo-points $\mathscr{P}_t(N)$ of each leaf node N are used to train a separate GP. At time step t, prediction for test points in the region $\mathscr{T}(N)$ is performed by the GP associated with node N.

2.4 Probabilistic Metric-Semantic Mapping

In this section, we address the single-robot mapping problem using the incremental GP regression theory developed in Sec. 2.3. For simplicity, we suppress the robot index superscript i. We apply GP regression to estimate the TSDF $f_l(\mathbf{x})$ of each semantic class l. Since the sensor measurements $\{\lambda_{t,k}, c_{t,k}\}$ are not direct samples from the TSDFs, in Sec. 2.4.1 we transform them into training sets $\tilde{\mathscr{D}}_{t,l}$, suitable for updating the GP distributions of $f_l(\mathbf{x})$. In Sec. 2.4.2, we apply incremental updates to GPs for each class and discuss how to predict the semantic class labels on the surfaces of the implicitly estimated object sets $\mathscr{O}_l$.

2.4.1 Training Set Construction

The class measurements allow us to associate the sensor data with particular semantic classes, while the distance measurements allow us to estimate the points where the sensor rays hit the object sets $\mathscr{O}_l$. We define the following point sets for each detected semantic class at time t:

$$\mathsf{G}_{t,l} = \{\hat{\mathbf{x}} \in \mathbb{R}^3 \,\big|\, \hat{\mathbf{x}} = \lambda_{t,k}\mathbf{R}_t\eta_k + \mathbf{p}_t \text{ and } c_{t,k} = l\}. \tag{2.13}$$

The values $f_l(\hat{\mathbf{x}})$ of the TSDFs are close to zero at points $\hat{\mathbf{x}} \in \mathsf{G}_{t,l}$ because the sensor rays hit an object surface close to these locations.

As shown in Prop. 1, the complexity of online GP training can be improved by forcing the training data to repeatedly come from a finite set of points. We choose a grid discretization $\mathscr{P}_\#$ of the environment $\mathscr{O} \cup \mathscr{F}$ and construct a training set by selecting points $\mathbf{x} \in \mathscr{P}_\#$, that are at most $\varepsilon > 0$ away from the points $\hat{\mathbf{x}} \in \mathsf{G}_{t,l}$, and approximating their TSDF values $f_l(\mathbf{x}) \approx g_{t,l}(\mathbf{x})$ (see Fig. 2.2). Precisely, the training data sets are constructed at time t as:

$$\tilde{\mathscr{D}}_{t,l} = \{(\mathbf{x}, g_{t,l}(\mathbf{x}))|\mathbf{x} \in \mathscr{P}_\#, \exists \hat{\mathbf{x}} \in \mathsf{G}_{t,l} \text{ s.t. } ||\mathbf{x} - \hat{\mathbf{x}}||_2 \le \varepsilon\}. \tag{2.14}$$

In the case of a camera sensor, the TSDF value $g_{t,l}(\mathbf{x})$ of a pseudo-point $\mathbf{x}$ is obtained by projecting $\mathbf{x}$ to the image plane and approximating its distance from the distance values of nearby pixels. In detail, suppose η_k is the unit vector corresponding to the pixel closest to the projection of $\mathbf{x}$ (red pixel in Fig. 2.2) and let $\hat{\mathbf{x}} \in \mathsf{G}_{t,l}$ be the coordinates of its ray endpoint (blue point in Fig. 2.2). Let $\hat{\mathbf{x}}_{right}$ and $\hat{\mathbf{x}}_{up}$ (two cyan points in Fig. 2.2) be the ray endpoints of two adjacent pixels. Then, $g_{t,l}(\mathbf{x})$ is the signed distance from $\mathbf{x}$ to the plane defined by $\hat{\mathbf{x}}$, $\hat{\mathbf{x}}_{right}$, and $\hat{\mathbf{x}}_{up}$:

$$\begin{gathered} g_{t,l}(\mathbf{x}) := \mathbf{n}^\top(\mathbf{x}-\hat{\mathbf{x}}), \quad \mathbf{n} := \operatorname{sign}(\mathbf{q}^\top(\mathbf{p}_t-\hat{\mathbf{x}}))\mathbf{q}, \\ \mathbf{q} = \frac{(\hat{\mathbf{x}}_{right}-\hat{\mathbf{x}}) \times (\hat{\mathbf{x}}_{up}-\hat{\mathbf{x}})}{\|(\hat{\mathbf{x}}_{right}-\hat{\mathbf{x}}) \times (\hat{\mathbf{x}}_{up}-\hat{\mathbf{x}})\|}, \end{gathered} \tag{2.15}$$

where $\mathbf{q}$ is the normal of the plane and the signed distance from $\mathbf{p}_t$ to the plane is positive because the sensor is known to be outside of the object set $\mathscr{O}_l$. Approximating the pseudo-point SDF values by measuring the distances to triangles formed by the nearest pixels means that the SDF values may be larger or smaller than the ground-truth SDF and using a Gaussian noise model is reasonable. The pseudo-point locations in the world frame are known since the sensor position $\mathbf{p}_t$ and orientation $\mathbf{R}_t$ are assumed known in this paper. If sensor pose uncertainty is considered, the Gaussian noise model for the measured SDF values needs to be modified to account for localization errors.

In the experiments, we discretize the workspace as a grid $\mathscr{P}_\#$ with resolution *voxel size*. Given a sensor ray end point $\hat{\mathbf{x}}$ in (2.13), instead of a sphere (circle) with radius ε, we choose a cubic (square) region of pseudo-points from $\mathscr{P}_\#$ around $\hat{\mathbf{x}}$. This is illustrated in Fig. 2.5. The cubic region is parameterized by the number of pseudo-points on its the edge, called $frame\ size$. This parameter is such that $(frame\ size - 1) \times voxel\ size \geq 2\varepsilon$. We do not use ε but only $frame\ size$ as one of the hyperparameters of our method.

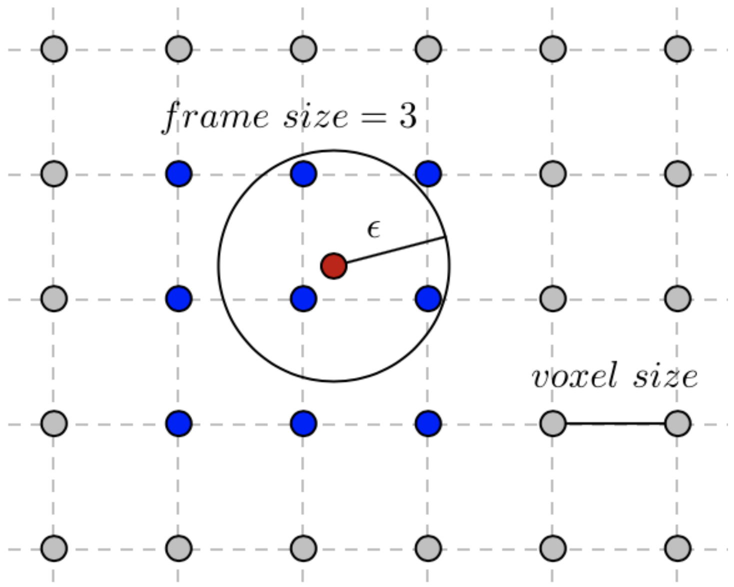


Figure 2.5. Illustration of pseudo-point selection. For each sensor ray end point (red), we select pseudo-points (blue) from a regular grid $\mathscr{P}_{\#}$ (gray) according to (2.14). For efficiency, instead of choosing points within radius ε, we choose a rectangular region of $frame\ size = 3$ such that $(frame\ size - 1) \times voxel\ size \geq 2\varepsilon$.

2.4.2 TSDF Mapping and Semantic Class Prediction

Given the transformed TSDF training data $\tilde{\mathscr{D}}_{t,l}$ obtained from the sensor measurement $\{\lambda_{t,k}, c_{t,k}\}_k$ at time t, we update the GP distribution of the TSDF $f_l(\mathbf{x})$ for each class l using the approach in Sec. 2.3.3. At time t, the new data are $\tilde{\mathscr{D}}_{t,l} = (\tilde{\mathscr{X}}_{t,l}, \tilde{\mathbf{y}}_{t,l})$, the new pseudo-points are $\tilde{\mathscr{P}}_{t,l} = \tilde{\mathscr{X}}_{t,l} \setminus \mathscr{P}_{t-1,l}$, and we update $\mathscr{P}_{t,l}$, $\zeta_{t,l}$, $m_{t,l}$ via (2.10). If online prediction is required, we can also update the precision matrix $Z_{t,l}$ using (2.11) and (2.12). Once we have the GPs of all classes updated, and can predict the TSDF at any query point according to (2.8). Next, we discuss how to predict the semantic class labels on the surfaces of the implicitly estimated object sets $\mathscr{O}_l$.

Semantic segmentation algorithms applied to camera images may produce incorrect pixel-level classification. This leads to some observations $\lambda_{t,k}$, $c_{t,k}$ being incorrectly included into the training set $\tilde{\mathscr{D}}_{t,l}$ of a different semantic class. We model the classification error as one of the sources of noise in the measured TSDF. This is motivated by the following observations.

First, the GP models for different classes are trained using the TSDF values of the pseudo-points rather than the sensor ray endpoints, which semantic segmentation is based on. A pseudo-point has TSDF values for all semantic classes so it should not be considered to belong to a specific class. The association between pseudo-points and ray endpoints is based on the TSDF value so a misclassification of a ray endpoint may be interpreted as noise on the measured TSDF value. Second, the misclassification probability is not uniform across the semantic categories. Rather, it is larger when two objects of different classes are close to each other in 3-D space. In the region around the object surfaces, the pseudo-points will have very small and very similar TSDF values. To predict the correct semantic class, we compare the likelihoods of the different classes at surface points using the posterior GP distributions of the TSDFs $f_l(\mathbf{x})$.

Proposition 2. *Let $\mathcal{GP}(\mu_{t,l}(\mathbf{x}), k_{t,l}(\mathbf{x},\mathbf{x}'))$ be the distributions of the truncated signed distance functions $f_l(\mathbf{x})$ at time t, determined according to* (2.8). *Consider an arbitrary point $\mathbf{x} \in \partial\mathcal{O}$ on the surface of the obstacle set, i.e., $\mathbf{x}$ is such that $f_l(\mathbf{x}) = 0$ for some class $l \in \{1,\dots,\mathcal{C}\}$. Then, the probability that the true class label of $\mathbf{x}$ is $c \in \{1,\dots,\mathcal{C}\}$ is:*

$$\mathbb{P}\left(\arg\min_l |f_l(\mathbf{x})| = c \;\middle|\; \min_l |f_l(\mathbf{x})| = 0\right) = \frac{\frac{1}{\sigma_{t,c}(\mathbf{x})}\phi\left(\frac{\mu_{t,c}(\mathbf{x})}{\sigma_{t,c}(\mathbf{x})}\right)}{\sum_l \frac{1}{\sigma_{t,l}(\mathbf{x})}\phi\left(\frac{\mu_{t,l}(\mathbf{x})}{\sigma_{t,l}(\mathbf{x})}\right)},$$

where $\phi(\cdot)$ is the probability density function of the standard normal distribution and $\sigma_{t,l}(\mathbf{x}) := \sqrt{k_{t,l}(\mathbf{x},\mathbf{x})}$.

Proof. Let $\mathbf{x}$ be an arbitrary (test) point. Denote the probability that closest surface at $\mathbf{x}$ is of class c and distance $|z|$ away by $l_c(z) := \mathbb{P}\left(\arg\min_l |f_l(\mathbf{x})| = c \text{ and } \min_l |f_l(\mathbf{x})| \le |z|\right)$. Since $\mathbb{P}\left(\min_l |f_l(\mathbf{x})| \le |z|\right) = \sum_l l_l(z)$, we have:

$$\mathbb{P}\left(\arg\min_l |f_l(\mathbf{x})| = c \;\middle|\; \min_l |f_l(\mathbf{x})| \le |z|\right) = \frac{l_c(z)}{\sum_l l_l(z)}.$$

The term we are interested in computing is $\lim_{z\to 0} \frac{l_c(z)}{\sum_l l_l(z)}$. Define $\mu_l := \mu_{t,l}(\mathbf{x})$ and $\sigma_l :=$

$\sigma_{t,l}(\mathbf{x})$ for $l = 1,\dots,\mathscr{C}$. The GP distribution of f_l stipulates that its value at $\mathbf{x}$ has a density function $p(z) = \frac{1}{\sigma_l}\phi\big(\frac{z-\mu_l}{\sigma_l}\big)$. Hence, $\mathbb{P}(|f_l(\mathbf{x})| \geq z) = 1 - \Phi(\frac{|z|-\mu_l}{\sigma_l}) + \Phi(\frac{-|z|-\mu_l}{\sigma_l})$. Note that $l_c(z)$ corresponds to the probability that $|f_c(\mathbf{x})| \leq |f_l(\mathbf{x})|$ for all l. Since all f_l are independent of each other:

$$l_c(z) = \frac{1}{\sigma_c}\int_{-z}^{z}\phi\big(\frac{\zeta-\mu_c}{\sigma_c}\big)\prod_{l\neq c}\Big(1-\Phi\big(\frac{|\zeta|-\mu_l}{\sigma_l}\big)+\Phi\big(\frac{-|\zeta|-\mu_l}{\sigma_l}\big)\Big)d\zeta$$

The claim is concluded by $\lim\limits_{z\to 0}\frac{l_c(z)}{2z} = \frac{1}{\sigma_c}\phi\big(\frac{-\mu_c}{\sigma_c}\big)$. □

The class distribution for an arbitrary point $\mathbf{x} \in \mathscr{O} \cup \mathscr{F}$, not lying on an object surface, may also be obtained, as shown in the proof of Prop. 2 but is both less efficient to compute and rarely needed in practice.

2.5 Distributed Incremental Sparse GP Regression

In this section, we develop a distributed version of the incremental sparse GP regression in Sec. 2.3. Suppose that the n robots communicate over a network, represented as a strongly connected directed graph $G = (\mathscr{V},\mathscr{E})$ with vertices $\mathscr{V} := \{1,...,n\}$ and edges $\mathscr{E} \subseteq \mathscr{V}\times\mathscr{V}$. An edge $(i,j) \in \mathscr{E}$ from robot i to robot j exists if the two robots can communicate. The robots directly connected to robot i are called *neighbors* and will be denoted by $\mathscr{N}e_i = \{j \in \mathscr{V} \mid (i,j) \in \mathscr{E}\}$. Let $W \in \mathbb{R}^{n\times n}$ be a weighted adjacency matrix such that $W_{i,j} > 0$ if $j \in \mathscr{N}e_i$ and $W_{i,j} = 0$, otherwise. Assume that W is a row-stochastic nonnegative and primitive matrix [76] and, hence, has a stationary distribution. The stationary distribution π is specified by the left eigenvector of W associated with the eigenvalue 1 and satisfies $\sum_{i=1}^{n}\pi_i = 1$. This weight matrix construction is common in consensus and distributed gradient descent algorithms [76, 87, 88]. Relying on consensus results for switching networks [76, 89, 90], our results may be generalized to time-varying graphs assuming that the graph sequence is uniformly strongly connected, i.e., there exists an integer $T > 0$ such that the union of the edges over any time interval of length T is strongly connected. However, we leave such an extension for future work.

Each robot $i \in \mathscr{V}$ receives its own local observations $\tilde{\mathscr{D}}_t^i = (\tilde{\mathscr{X}}_t^i, \tilde{\mathbf{y}}_t^i)$ at time t and extracts newly observed pseudo-points $\tilde{\mathscr{P}}_t^i$, with associated number of observations $\tilde{\mathbf{m}}_t^i$ and average values $\tilde{\zeta}_t^i$, as detailed in Sec. 2.3.3. This information is used to update the complete set of pseudo-points $\mathscr{P}_t^i$ observed up to time t, along with the number of observations $\mathbf{m}_t^i$ and average values ζ_t^i, according to (2.10). These parameters $\Theta_t^i := \left\{\mathscr{P}_t^i, \mathbf{m}_t^i, \zeta_t^i\right\}$, maintained by robot i, define a complete GP distribution for the function $f(\mathbf{x})$, with mean and covariance functions in (2.8).

While each robot can estimate $f(\mathbf{x})$ individually, we consider how the robots may exchange information to estimate $f(\mathbf{x})$ collaboratively. We observe that the continuous-space GP distribution of $f(\mathbf{x})$ is induced by the statistics $\mathbf{m}_t^i$, ζ_t^i of the pseudo-points $\mathscr{P}_t^i$. Hence, if the robots exchange information about and agree on these finite-dimensional parameters, then the corresponding GP distributions of $f(\mathbf{x})$ at each robot will agree. Our *main innovation* is a distributed algorithm for updating the sparse GP parameters of one robot using the parameters of its one-hop neighbors' distributions. While existing results apply to fixed finite-dimensional parameter estimation, our approach considers function estimation with an infinite-dimensional GP distribution, updated via consensus over an incrementally growing set of pseudo-point parameters.

To gain intuition about the construction of consensus schemes over GP posteriors, we first review distributed Kalman filtering for fixed-dimensional parameter estimation in Sec. 2.5.1. Then, we use the connection between the GP posterior induced by pseudo-points and the joint Gaussian distribution over the pseudo-points described from Sec. 2.3.1 to develop distributed GP regression in Sec. 2.5.2. In Sec. 2.5.3, we prove that the distributed algorithm converges to the same posterior GP distribution as a centralized sparse GP regression that uses the observations from all robots. Finally, in Sec. 2.5.4, we provide an approach to label the messages that the robots exchange in order to avoid repeated communication of the same information.

2.5.1 Distributed Kalman Filtering

Suppose that the robots aim to estimate a fixed (finite-dimensional) vector $\mathbf{f}$ cooperatively using local observations $\mathbf{y}_t^i$, generated according to a linear Gaussian model:

$$\mathbf{y}_t^i = H^i \mathbf{f} + \eta_t^i, \qquad \eta_t^i \sim \mathcal{N}(0, V^i). \tag{2.16}$$

Assume that the observations $\mathbf{y}_t^i$ received by robot i are independent over time and from the observations of all other robots. Assume also that the graph G is connected and that $\mathbf{f}$ is observable if one has access to the observations received by all robots, i.e., the matrix $\begin{bmatrix} H^1 & \cdots & H^n \end{bmatrix}$ has rank equal to the dimension of $\mathbf{f}$. Since individual observations $\mathbf{y}_t^i$ alone may be insufficient to estimate $\mathbf{f}$, the robots need to exchange information. We suppose that each robot starts with a prior probability density function $p_0^i(\mathbf{f})$ over the unknown vector $\mathbf{f}$ and updates it over time, relying on its local observations $\mathbf{y}_t^i$ as well as communication with one-hop neighbors in G.

Rahnama Rad and Tahbaz-Saleh [77] developed a consistent distributed estimation algorithm, in which each agent i uses standard Bayesian updates with its local observations $\mathbf{y}_{t+1}^i$ but, instead of its own prior p_t^i, each agent uses a weighted geometric average of its neighbors' priors:

$$p_{t+1}^i(\mathbf{f}) \propto p^i(\mathbf{y}_{t+1}^i|\mathbf{f}) \prod_{i=1}^{n} (p_t^i(\mathbf{f}))^{W_{ij}}, \tag{2.17}$$

where $p^i(\mathbf{y}_{t+1}^i|\mathbf{f})$ is an observation model, such as (2.16), that should satisfy certain regularity conditions [77]. Atanasov et al. [78] showed that if the prior distributions p_0^i are Gaussian and the observation models are linear Gaussian as in (2.16), the resulting distributed Kalman filter is mean-square consistent (the estimates $\arg\max_{\mathbf{f}} p_t^i(\mathbf{f})$ of all agents i converge in mean square to the true $\mathbf{f}$). Specifically, if the priors are $\mathbf{f} \sim \mathcal{N}(\mu_0^i, \Sigma_0^i)$ with information matrix $\Omega_0^i := (\Sigma_0^i)^{-1}$

and information mean $\omega_0^i := \Omega_0^i \mu_0^i$, the Gaussian version of the distributed estimator in (2.17) is:

$$\begin{aligned}\omega_{t+1}^i &= \sum_{i=1}^n W_{ij}\omega_t^j + H^{i\top} V^{i^{-1}} \mathbf{y}_{t+1}^i \\ \Omega_{t+1}^i &= \sum_{j=1}^n W_{ij}\Omega_t^j + H^{i\top} V^{i^{-1}} H^i\end{aligned} \tag{2.18}$$

because geometric averaging and Bayesian updates with Gaussian densities lead to a Gaussian posterior density [78]. The relationship between geometric means being used for belief propagation in (2.17) and weighted averaging via mixing matrix W forms the conceptual basis for message passing in the more general GP posterior inference setting which we detail next.

2.5.2 Distributed Incremental Sparse GP Regression

The distributed estimation algorithm in (2.18) does not directly apply to GP regression because the estimation target $f(\mathbf{x})$ is infinite-dimensional. However, the sparse GP regression, described in Sec. 2.3, relies on a finite (albeit incrementally growing) set of pseudo-points $\mathscr{P}_t$, and we show that it is possible to obtain distributed incremental sparse GP regression based on (2.18). As discussed earlier, each robot i maintains parameters $\Theta_t^i := \left\{\mathscr{P}_t^i, \mathbf{m}_t^i, \zeta_t^i\right\}$ based on its local observations $\tilde{\mathscr{D}}_t^i = (\tilde{\mathscr{X}}_t^i, \tilde{\mathbf{y}}_t^i)$. Our key idea is to perform weighted geometric averaging over local posteriors, which translates to simple weighted averaging of the means and covariances in (2.3) of f at a finite set of points $\mathscr{Q} \supseteq \mathscr{P}_t^i$, which will be specified precisely below. The parameters Θ_t^i induce a GP distribution over f in (2.8), which in turn provides a Gaussian probability density function $p_t^i(\mathbf{f}) := p(\mathbf{f}|\Theta_t^i)$ over the vector $\mathbf{f} := f(\mathscr{Q})$ with mean $\mu_t^i(\mathscr{Q})$ and covariance $\Sigma_t^i(\mathscr{Q})$, obtained from (2.8). In order to derive decentralized updates for GPs akin to (2.18), we first present the iterative updates associated with the robots' local posteriors in terms of their information mean and information matrix corresponding to the mean and covariance of $p_t^i(\mathbf{f})$.

Lemma 1. *The information mean $\omega_t^i(\mathscr{Q}) := \Omega_t^i(\mathscr{Q})\mu_t^i(\mathscr{Q})$ and information matrix $\Omega_t^i(\mathscr{Q}) :=$*

$(\Sigma_t^i(\mathcal{Q}))^{-1}$ *of the Gaussian probability density function* $p_t^i(\mathbf{f}) := p(\mathbf{f}|\Theta_t^i)$ *of* $\mathbf{f} := f(\mathcal{Q})$ *with parameters* $\Theta_t^i := \left\{\mathcal{P}_t^i, \mathbf{m}_t^i, \zeta_t^i\right\}$ *are:*

$$\begin{aligned}\omega_t^i(\mathcal{Q}) &= k_0^i(\mathcal{Q},\mathcal{Q})^{-1}\mu_0^i(\mathcal{Q}) + \sigma^{-2}\operatorname{diag}(m_t^i(\mathcal{Q}))\zeta_t^i(\mathcal{Q}) \\ \Omega_t^i(\mathcal{Q}) &= k_0^i(\mathcal{Q},\mathcal{Q})^{-1} + \sigma^{-2}\operatorname{diag}(m_t^i(\mathcal{Q})),\end{aligned} \tag{2.19}$$

where, similar to Sec. 2.3.3, $m_t^i(\mathbf{p})$ *and* $\zeta_t^i(\mathbf{p})$ *denote the number of observations and average observation, respectively, for* $\mathbf{p} \in \mathcal{P}_t^i$ *and their domains have been extended to* $\mathcal{Q} \supseteq \mathcal{P}_t^i$ *by defining* $m_t^i(\mathbf{q}) = \zeta_t^i(\mathbf{q}) = 0$ *for* $\mathbf{q} \in \mathcal{Q} \setminus \mathcal{P}_t^i$*.*

Proof. Similar to the proof of Prop. 1, let E be a binary matrix such that $k_0^i(\mathcal{P}_t^i,\mathbf{x}) = Ek_0^i(\mathcal{Q},\mathbf{x})$, i.e., E selects the points from the superset $\mathcal{Q}$ which correspond to $\mathcal{P}_t^i$. Note that $k_0^i(\mathcal{Q},\mathcal{P}_t^i) = k_0^i(\mathcal{Q},\mathcal{Q})E^\top$, $k_0^i(\mathcal{P}_t^i,\mathcal{Q}) = Ek_0^i(\mathcal{Q},\mathcal{Q})$, and $k_0^i(\mathcal{P}_t^i,\mathcal{P}_t^i) = Ek_0^i(\mathcal{Q},\mathcal{Q})E^\top$. The expression for $\Omega_t^i(\mathcal{Q})$ follows from the matrix inversion lemma applied to the covariance matrix $\Sigma_t^i(\mathcal{Q})$ and noting that $E^\top\operatorname{diag}(\mathbf{m}_t^i)E = \operatorname{diag}(m_t^i(\mathcal{Q}))$. Then, note that:

$$\begin{aligned}&\Omega_t^i(\mathcal{Q})k_0^i(\mathcal{Q},\mathcal{P}_t^i)Z_t^i \\ &\quad= \left(I + \sigma^{-2}E^\top\operatorname{diag}(\mathbf{m}_t^i)Ek_0^i(\mathcal{Q},\mathcal{Q})\right)E^\top Z_t^i \\ &\quad= \sigma^{-2}E^\top\operatorname{diag}(\mathbf{m}_t^i)(Z_t^i)^{-1}Z_t^i = \sigma^{-2}E^\top\operatorname{diag}(\mathbf{m}_t^i).\end{aligned}$$

Thus, the information mean is:

$$\begin{aligned}\omega_t^i(\mathcal{Q}) &= \Omega_t^i(\mathcal{Q})\left(\mu_0^i(\mathcal{Q}) + k_0^i(\mathcal{Q},\mathcal{P}_t^i)Z_t^i\left(\zeta_t^i - \mu_0^i(\mathcal{P}_t^i)\right)\right) \\ &= \Omega_t^i(\mathcal{Q})\mu_0^i(\mathcal{Q}) + \sigma^{-2}E^\top\operatorname{diag}(\mathbf{m}_t^i)\left(\zeta_t^i - \mu_0^i(\mathcal{P}_t^i)\right) \\ &= k_0^i(\mathcal{Q},\mathcal{Q})^{-1}\mu_0^i(\mathcal{Q}) + \sigma^{-2}E^\top\operatorname{diag}(\mathbf{m}_t^i)\zeta_t^i \\ &= k_0^i(\mathcal{Q},\mathcal{Q})^{-1}\mu_0^i(\mathcal{Q}) + \sigma^{-2}\operatorname{diag}(m_t^i(\mathcal{Q}))\zeta_t^i(\mathcal{Q}).\end{aligned}$$ □

With the expression for the parametric updates associated with the posterior inference defined by observations acquired locally at robot i only, we next detail how to augment this update with neighboring robots' information.

Distributed updates with a fixed pseudo-point set

To begin, suppose that the pseudo-point sets are fixed across all robots, i.e., $\mathscr{P} \equiv \mathscr{P}_t^i$, and the local observations $\tilde{\mathscr{D}}_{t+1}^i = (\tilde{\mathscr{X}}_{t+1}^i, \tilde{\mathbf{y}}_{t+1}^i)$ satisfy $\tilde{\mathscr{X}}_{t+1}^i \subseteq \mathscr{P}$ for all t, i. Then, the information means and matrices in (2.19) have equal dimensions across the robots. By defining $H_{t+1}^i := k_0^i(\tilde{\mathscr{X}}_{t+1}^i, \mathscr{P}) k_0^i(\mathscr{P}, \mathscr{P})^{-1}$, $\omega_t^i := \omega_t^i(\mathscr{P})$, and $\Omega_t^i := \Omega_t^i(\mathscr{P})$ we can apply the update in (2.18) directly. The information means and matrices have a simple structure, and, similar to (2.10), it is sufficient to track only the number of observations $\mathbf{m}_t^i$ and the average observations ζ_t^i over time:

$$\begin{aligned} \omega_{t+1}^i &= \sum_{i=1}^n W_{ij} \omega_0^j + \frac{1}{\sigma^2} \sum_{i=1}^n W_{ij} \operatorname{diag}(\mathbf{m}_t^j) \zeta_t^j + \frac{1}{\sigma^2} \operatorname{diag}(\tilde{\mathbf{m}}_{t+1}^i) \tilde{\zeta}_{t+1}^i \\ \Omega_{t+1}^i &= \sum_{j=1}^n W_{ij} \Omega_0^j + \frac{1}{\sigma^2} \sum_{i=1}^n W_{ij} \operatorname{diag}(\mathbf{m}_t^j) + \frac{1}{\sigma^2} \operatorname{diag}(\tilde{\mathbf{m}}_{t+1}^i), \end{aligned} \tag{2.20}$$

where $\tilde{\mathbf{m}}_{t+1}^i$ and $\tilde{\zeta}_{t+1}^i$ are the number of new observations and new observation averages received by robot i of the pseudo-points $\mathscr{P}$ at time $t+1$. We consider the case with incrementally growing pseudo-point sets that are potentially different across the robots before presenting the final distributed update equations for $\mathbf{m}_t^i$ and ζ_t^i. This is the focus of the following subsection.

Distributed updates with dynamic pseudo-point sets

Consider the general case where each robot maintains its own pseudo-point set $\mathscr{P}_t^i$ and the observations $\tilde{\mathscr{D}}_{t+1}^i = (\tilde{\mathscr{X}}_{t+1}^i, \tilde{\mathbf{y}}_{t+1}^i)$ may introduce new pseudo-points $\tilde{\mathscr{P}}_{t+1}^i \not\subseteq \mathscr{P}_t^i$. Our key observation is that the parameters Θ_t^i induce a GP distribution over the whole function f and, hence, can be used to obtain a Gaussian distribution over a pseudo-point set that is larger than $\mathscr{P}_t^i$ according to (2.19):

$$\mathscr{P}_{t+1}^i = \bigcup_{j \in \mathscr{N}e_i \cup \{i\}} \mathscr{P}_t^j \cup \tilde{\mathscr{P}}_{t+1}^i. \tag{2.21}$$

Note that the structure of the information mean and information matrix in (2.19) was derived for an arbitrary pseudo-points set $\mathscr{Q}$. First, without considering the observations $\tilde{\mathscr{D}}^i_{t+1}$, we let $\mathscr{Q} = \mathscr{P}^i_{t+1}$ and calculate $\omega^j_t(\mathscr{P}^i_{t+1})$, $\Omega^j_t(\mathscr{P}^i_{t+1})$. Then, the distributed averaging in (2.18) can be performed over the information means and information matrices in (2.19) with $\mathscr{Q} = \mathscr{P}^i_{t+1}$ and $H^i_{t+1} := k^i_0(\tilde{\mathscr{X}}^i_{t+1}, \mathscr{P}^i_{t+1}) k^i_0(\mathscr{P}^i_{t+1}, \mathscr{P}^i_{t+1})^{-1}$:

$$\begin{aligned}
\omega^i_{t+1}(\mathscr{P}^i_{t+1}) &= \sum_{i=1}^{n} W_{ij}\omega^j_t(\mathscr{P}^i_{t+1}) + H^{i\top}_{t+1}(\sigma^2 I)^{-1}\tilde{\mathbf{y}}^i_{t+1}, \\
\Omega^i_{t+1}(\mathscr{P}^i_{t+1}) &= \sum_{i=1}^{n} W_{ij}\Omega^j_t(\mathscr{P}^i_{t+1}) + H^{i\top}_{t+1}(\sigma^2 I)^{-1}H^i_{t+1}.
\end{aligned} \tag{2.22}$$

We may rewrite the preceding expressions in terms of the number of observations $m^i_{t+1}(\mathbf{p})$ and average observations $\zeta^i_{t+1}(\mathbf{p})$ for any $\mathbf{p} \in \mathscr{P}^i_{t+1}$, akin to (2.10), by following the steps in (2.20) for the dynamic pseudo-point case, leading to:

$$\begin{aligned}
m^i_{t+1}(\mathbf{p}) &= \sum_{j \in \mathscr{N}e_i \cup \{i\}} W_{ij} m^j_t(\mathbf{p}) + \tilde{m}^i_{t+1}(\mathbf{p}), \\
\zeta^i_{t+1}(\mathbf{p}) &= \frac{\sum_{j \in \mathscr{N}e_i \cup \{i\}} W_{ij} m^j_t(\mathbf{p})\zeta^j_t(\mathbf{p}) + \tilde{m}^i_{t+1}(\mathbf{p})\tilde{\zeta}^i_{t+1}(\mathbf{p})}{m^i_{t+1}(\mathbf{p})}.
\end{aligned} \tag{2.23}$$

With the updates for robot i in terms of its local observations and message passing with its neighbors $\mathscr{N}e_i$ specified, we shift in the following subsection to establishing its statistical properties.

2.5.3 Theoretical Guarantee for Consistent Estimation

We show that the proposed distributed incremental sparse GP regression defined by (2.21), (2.23), and (2.8) converges to a centralized sparse GP regression, which uses the observation data $\cup_t \cup_i \tilde{\mathscr{D}}^i_t$ from all robots. At each time step t, the centralized estimator receives data $\cup_i \tilde{\mathscr{D}}^i_t$, and, as discussed in Sec. 2.3.3, updates a global set of pseudo-points $\mathscr{P}^{ctr}_t$, the number of times $m^{ctr}_t(\mathbf{p})$ each pseudo-point $\mathbf{p} \in \mathscr{P}^{ctr}_t$ has been observed, and the average observation $\zeta^{ctr}_t(\mathbf{p})$ of $\mathbf{p} \in \mathscr{P}^{ctr}_t$.

In order to show that the GP maintained by each robot i eventually agrees with the centralized GP, the centralized estimator should also be affected by the Perron weight matrix W. If $W = \frac{1}{n}\mathbf{1}\mathbf{1}^\top$, the information provided by different robots is equally credible and the centralized estimator can use the combined set of observations $\cup_i \tilde{\mathscr{D}}_t^i$ directly. If, however, the left eigenvector π of W is not $\mathbf{1}$, then its elements π_i specify different credibility for the different robots. More precisely, the centralized estimator should treat the measurements $\tilde{\mathscr{D}}_t^i$ of robot i as if they were generated with noise variance σ^2/π_i, instead of the true noise variance σ^2. This is equivalent to scaling the number of observations $\tilde{m}_t^i$ provided by robot i by its "credibility" π_i, leading to the following update for the centralized sparse GP regression parameters:

$$\begin{aligned}
\mathscr{P}_{t+1}^{ctr} &= \cup_{i=1}^n \tilde{\mathscr{P}}_{t+1}^i \cup \mathscr{P}_t^{ctr},\\
m_{t+1}^{ctr}(\mathbf{p}) &= m_t^{ctr}(\mathbf{p}) + \sum_{i=1}^n \pi_i \tilde{m}_{t+1}^i(\mathbf{p}),\\
\zeta_{t+1}^{ctr}(\mathbf{p}) &= \frac{m_t^{ctr}(\mathbf{p})\zeta_t^{ctr}(\mathbf{p}) + \sum_{i=1}^n \pi_i \tilde{m}_{t+1}^i(\mathbf{p})\tilde{\zeta}_{t+1}^i(\mathbf{p})}{m_{t+1}^{ctr}(\mathbf{p})},
\end{aligned} \tag{2.24}$$

for all $\mathbf{p} \in \mathscr{P}_{t+1}^{ctr}$. The next result shows that the individual GP distributions maintained by each robot using the distributed updates in (2.23) converge to the centralized GP distribution determined by the parameters above.

Proposition 3. *Let $\tilde{\mathscr{D}}_t^i = (\tilde{\mathscr{X}}_t^i, \tilde{\mathbf{y}}_t^i)$ be the data received by robot i at time t, associated with pseudo-points $\tilde{\mathscr{P}}_t^i \subset \mathscr{P}_\#$ and number of observations $\tilde{m}_t^i(\mathbf{p})$ and average observation $\tilde{\zeta}_t^i(\mathbf{p})$ for $\mathbf{p} \in \mathscr{P}_\#$. If the data streaming stops at some time $T < \infty$, then as $t \to \infty$, the distributions $\mathscr{GP}(\mu_t^i(\mathbf{x}), k_t^i(\mathbf{x},\mathbf{x}'))$ maintained by each robot i, specified according to* (2.8) *with parameters $\mathscr{P}_t^i$, $m_t^i(\mathbf{p})$, $\zeta_t^i(\mathbf{p})$ in* (2.21) *and* (2.23) *converge to the distribution $\mathscr{GP}(\mu_t^{ctr}(\mathbf{x}), k_t^{ctr}(\mathbf{x},\mathbf{x}'))$ of the centralized estimator with parameters $\mathscr{P}_t^{ctr}$, $m_t^{ctr}(\mathbf{p})$, $\zeta_t^{ctr}(\mathbf{p})$ in* (2.24)*, i.e., $|\mu_t^i(\mathbf{x}) - \mu_t^{ctr}(\mathbf{x})| \to 0$ and $|k_t^i(\mathbf{x},\mathbf{x}') - k_t^{ctr}(\mathbf{x},\mathbf{x}')| \to 0$ almost surely for all $i \in \mathscr{V}$, $\mathbf{x},\mathbf{x}'$.*

Proof. Since the distributions $\mathscr{GP}(\mu_t^i(\mathbf{x}), k_t^i(\mathbf{x},\mathbf{x}'))$ and $\mathscr{GP}(\mu_t^{ctr}(\mathbf{x}), k_t^{ctr}(\mathbf{x},\mathbf{x}'))$ are completely determined by the parameters $\mathscr{P}_t^i$, $m_t^i(\mathbf{p})$, $\zeta_t^i(\mathbf{p})$ and $\mathscr{P}_t^{ctr}$, $m_t^{ctr}(\mathbf{p})$, $\zeta_t^{ctr}(\mathbf{p})$, respectively, it is sufficient to show that $|m_t^i(\mathbf{p}) - m_t^{ctr}(\mathbf{p})| \to 0$ and $|\zeta_t^i(\mathbf{p}) - \zeta_t^{ctr}(\mathbf{p})| \to 0$ for all $i \in \mathscr{V}$, $\mathbf{p} \in \mathscr{P}_\#$. Let $\mathbf{p} \in \mathscr{P}_\#$ be arbitrary and note that $m_0^i(\mathbf{p}) = m_0^{ctr}(\mathbf{p}) = 0$ and $\zeta_0^i(\mathbf{p}) = \zeta_0^{ctr}(\mathbf{p}) = 0$ since no pseudo-points have been observed initially. Expand (2.24) recursively to obtain $m_t^{ctr}(\mathbf{p})$ and $\zeta_t^{ctr}(\mathbf{p})$ in terms of the observation statistics:

$$\begin{aligned} m_t^{ctr}(\mathbf{p}) &= \sum_{\tau=0}^{t} \sum_{i=1}^{n} \pi_i \tilde{m}_\tau^i(\mathbf{p}), \\ \zeta_t^{ctr}(\mathbf{p}) &= \frac{1}{m_t^{ctr}(\mathbf{p})} \sum_{\tau=0}^{t} \sum_{i=1}^{n} \pi_i \tilde{m}_\tau^i(\mathbf{p}) \tilde{\zeta}_\tau^i(\mathbf{p}). \end{aligned} \tag{2.25}$$

Similarly, expand (2.23) to obtain $m_t^i(\mathbf{p})$ and $\zeta_t^i(\mathbf{p})$ in terms of the observation statistics:

$$\begin{aligned} m_t^i(\mathbf{p}) &= \sum_{\tau=0}^{t} \sum_{j=1}^{n} \left[W^{t-\tau}\right]_{ij} \tilde{m}_\tau^j(\mathbf{p}), \\ \zeta_t^i(\mathbf{p}) &= \frac{1}{m_t^i(\mathbf{p})} \sum_{\tau=0}^{t} \sum_{j=1}^{n} \left[W^{t-\tau}\right]_{ij} \tilde{m}_\tau^j(\mathbf{p}) \tilde{\zeta}_\tau^j(\mathbf{p}), \end{aligned} \tag{2.26}$$

where the weights $[W^{t-\tau}]_{ij}$ appear since the data $\tilde{m}_\tau^j(\mathbf{p})$ and $\tilde{\zeta}_\tau^j(\mathbf{p})$ propagate through the network with weight matrix W and reach robot i via all paths of length $t-\tau$. Alternatively, (2.26) can be viewed as the solution of the discrete-time linear time-invariant system in (2.23) with transition matrix $\Phi(t,\tau) = W^{t-\tau}$, $t \geq \tau$. Since the data collection stops at some finite time T, $\tilde{m}_t^i(\mathbf{p}) = \tilde{\zeta}_t^i(\mathbf{p}) = 0$ for all $t > T$, $i \in \mathscr{V}$. The convergence of (2.26) to (2.25) is concluded from the fact that $[W^t]_{ij} \to \pi_j > 0$ since W is a row-stochastic nonnegative and primitive matrix. $\square$

Prop. 3 is a similar result to [77, Thm. 3], where it is shown that, if the weight matrix W is doubly stochastic, a distributed parameter estimator is as efficient as any centralized parameter estimator. However, Prop. 3 applies to distributed function estimation using an incrementally growing set of parameters and re-weights the observations used by the centralized estimator via the stationary distribution π of W to ensure convergence even when W is not doubly stochastic.

2.5.4 Echoless Distributed GP Regression

The distributed pseudo-point update we derived in (2.23) is not efficient for two reasons. First, convergence to the central GP estimate is guaranteed only in the limit, as $t \to \infty$ (Prop. 3). Second, every time robots exchange messages, all information they have must be sent. This is inefficient as may be seen in the proof of Prop. 3, the observations are exchanged an infinite number of times (echos in the network). To address these limitations, we label the communication messages with the list of robots that have already received them and show that convergence to the centralized estimate can, in fact, be achieved in finite time. Let $\tilde{\Theta}_t^i := \{\tilde{\mathscr{P}}_t^i, \tilde{m}_t^i(\tilde{\mathscr{P}}_t^i), \tilde{\zeta}_t^i(\tilde{\mathscr{P}}_t^i), \ell_t^i\}$ define a communication package which contains the new observations $\tilde{\mathscr{P}}_t^i$, $\tilde{m}_t^i(\tilde{\mathscr{P}}_t^i)$, $\tilde{\zeta}_t^i(\tilde{\mathscr{P}}_t^i)$ of robot i at time t as well as a list of robots ℓ_t^i that have already received this package. The list ℓ_t^i is initialized at time t with $\{i\}$. For each robot i, define also a set of packages $\mathscr{B}_{t+1}^i$ that the robot should use at time t to update its GP parameters. The set $\mathscr{B}_t^i$ from the previous time step contains old packages that robot i should transmit to its neighbors. Inspired by the similarity of (2.25) and (2.26), we propose a distributed protocol which ensures that:

- each package, which contains the processed observations of robot i at time t, visits each robot once rather than echoing in the network, relying on ℓ_t^i to keep track of visited robots,
- convergence to the centralized GP distribution is achieved in finite and minimum time by using the *stationary distribution* (left eigenvector) π of W as the coefficient in (2.26).

The distributed parameter update for robot i at time t is:

$$\begin{aligned}
\mathscr{B}^i_{t+1} &= \bigcup_{\tilde{\Theta}^j_\tau \in \mathscr{B}^r_t, r \in \mathscr{N}e_i, i \notin \ell^j_\tau} \tilde{\Theta}^j_\tau \cup \tilde{\Theta}^i_{t+1}, \\
\ell^j_\tau &= \ell^j_\tau \cup \{i\} \text{ for all } \tilde{\Theta}^j_\tau \in \mathscr{B}^i_{t+1}, \\
\mathscr{P}^i_{t+1} &= \bigcup_{\tilde{\Theta}^j_\tau \in \mathscr{B}^i_{t+1}} \mathscr{P}^j_\tau \cup \mathscr{P}^i_t, \\
m^i_{t+1}(\mathbf{p}) &= m^i_t(\mathbf{p}) + \sum_{\tilde{\Theta}^j_\tau \in \mathscr{B}^i_{t+1}} \pi_j \tilde{m}^j_\tau(\mathbf{p}), \\
\zeta^i_{t+1}(\mathbf{p}) &= \frac{m^i_t(\mathbf{p})\zeta^i_t(\mathbf{p}) + \sum_{\tilde{\Theta}^j_\tau \in \mathscr{B}^i_{t+1}} \pi_j \tilde{m}^j_\tau(\mathbf{p}) \tilde{\zeta}^j_\tau(\mathbf{p})}{m^i_{t+1}(\mathbf{p})}.
\end{aligned} \tag{2.27}$$

We prove that this distributed update rule converges in finite time to the centralized GP distribution. Compared with (2.23), the distributed update in (2.27) is able to achieve finite-time convergence because it uses the weights π from the stationary distribution of W right away, instead of processing the same information an infinite number of times to determine π. Moreover, (2.23) stipulates that two robots should exchange all of their information at each time step, which is very inefficient in practice. The messages in (2.27) allow the robots to exchange only the latest information and guarantee that each observation reaches each robot once.

Proposition 4. *Let $\tilde{\mathscr{D}}^i_t = (\tilde{\mathscr{X}}^i_t, \tilde{\mathbf{y}}^i_t)$ be the data received by robot i at time t, associated with pseudo-points $\tilde{\mathscr{P}}^i_t \subset \mathscr{P}_\#$ and number of observations $\tilde{m}^i_t(\mathbf{p})$ and average observation $\tilde{\zeta}^i_t(\mathbf{p})$ for $\mathbf{p} \in \mathscr{P}_\#$. If the data streaming stops at some time $T < \infty$, then at time $t = T + n - 1$, the distributions $\mathscr{GP}(\mu^i_t(\mathbf{x}), k^i_t(\mathbf{x},\mathbf{x}'))$ maintained by each robot i, specified according to* (2.8) *with parameters in* (2.27) *are exactly equal to the distribution $\mathscr{GP}(\mu^{ctr}_t(\mathbf{x}), k^{ctr}_t(\mathbf{x},\mathbf{x}'))$ of the centralized estimator with parameters in* (2.24)*, i.e., $\mu^i_t(\mathbf{x}) = \mu^{ctr}_t(\mathbf{x})$ and $k^i_t(\mathbf{x},\mathbf{x}') = k^{ctr}_t(\mathbf{x},\mathbf{x}')$ almost surely for all $i \in \mathscr{V}$, $\mathbf{x}, \mathbf{x}'$.*

Proof. As in the proof of Prop. 3, it is sufficient to show that at $t = T + n - 1$, $m^i_t(\mathbf{p}) = m^{ctr}_t(\mathbf{p})$ and $\zeta^i_t(\mathbf{p}) = \zeta^{ctr}_t(\mathbf{p})$ for all $i \in \mathscr{V}$, $\mathbf{p} \in \mathscr{P}_\#$. As before, we express $m^i_t(\mathbf{p})$ and $\zeta^i_t(\mathbf{p})$ in terms of

$\tilde{m}_\tau^j(\mathbf{p})$ and $\tilde{\zeta}_\tau^j(\mathbf{p})$ for arbitrary $\mathbf{p} \in \mathscr{P}_\#$ and $\tau \leq t$. The key is to decide whether package $\tilde{\Theta}_\tau^j$ is received by robot i. Since the package exchanges are happening based on the communication graph structure, the elements of $W^{t-\tau}$ determine which robots have received a package released at time τ by time t. Precisely, if $[W^{t-\tau}]_{ij} > 0$, then robot i has received package $\tilde{\Theta}_\tau^j$ by time t and otherwise, if $[W^{t-\tau}]_{ij} = 0$, it has not received it. Let $\operatorname{sign}(x)$ denote the sign of a scalar x with $\operatorname{sign}(0) = 0$. Expanding (2.27) recursively leads to:

$$
\begin{aligned}
m_t^i(\mathbf{p}) &= \sum_{\tau=0}^{t} \sum_{i=1}^{n} \operatorname{sign}(\left[W^{t-\tau}\right]_{ij}) \pi_j \tilde{m}_\tau^j(\mathbf{p}) \\
\zeta_t^i(\mathbf{p}) &= \frac{1}{m_t^i(\mathbf{p})} \sum_{\tau=0}^{t} \sum_{j=1}^{n} \operatorname{sign}(\left[W^{t-\tau}\right]_{ij}) \pi_j \tilde{m}_\tau^j(\mathbf{p}) \tilde{\zeta}_\tau^j(\mathbf{p})
\end{aligned}
\tag{2.28}
$$

Since the data collection stops at some finite time T, $\tilde{m}_\tau^i(\mathbf{p}) = \tilde{\zeta}_\tau^i(\mathbf{p}) = 0$ for all $\tau > T$, $i \in \mathscr{V}$. Comparing (2.26) and (2.25), equality of $\mu_t^i(\mathbf{x})$ and $\mu_t^{ctr}(\mathbf{x})$ and $k_t^i(\mathbf{x}, \mathbf{x}')$ and $k_t^{ctr}(\mathbf{x}, \mathbf{x}')$ at $t = T + n - 1$ is concluded by the fact that $\left[W^{n-1}\right]_{ij} > 0$ because the network is connected. $\square$

To ensure that each package is received by each robot once, we assumed that the package keeps a list of visited robots. Since a package may be received by several robots at the same time, the robots should keep a list of received packages. More precisely, each package should be labeled based on the robot that observed it and the time slot the package is observed. Then, each robot keeps the list of received packages and always removes the packages with time label earlier than $n - 1$.

2.6 Distributed Metric-Semantic Mapping

We apply the distributed GP regression technique developed in Sec. 2.5 to the multi-robot metric-semantic TSDF mapping problem. Each robot i receives local distance and class observations $\left\{\lambda_{t+1,k}^i, c_{t+1,k}^i\right\}$, which are transformed using the procedure in Sec. 2.4.1 into training data sets $\tilde{\mathscr{D}}_{t+1,l}^i = \left(\tilde{\mathscr{X}}_{t+1,l}^i, \tilde{\mathbf{y}}_{t+1,l}^i\right)$ for estimating the TSDFs $\{f_l(\mathbf{x})\}$ of the different object classes. Each dataset $\tilde{\mathscr{D}}_{t+1,l}^i$ is compressed into a set of pseudo-points $\tilde{\mathscr{P}}_{t+1,l}^i$ with

Algorithm 1. Distributed Metric-Semantic Mapping

```
1:  **Routine for robot $i$ at time step $\tau$**:
2:  **Input**: sensor observation: $\{\lambda^i_{\tau,k}, c^i_{\tau,k}\}_k$
3:  Construct training set via Sec. 2.4.1:
        $$\{\lambda^i_{\tau,k}, c^i_{\tau,k}\}_k \to \{\tilde{\mathscr{D}}^i_{\tau,l}\}_l \to \{\tilde{\Theta}^i_{\tau,l}\}_l$$
4:  Receive packages $\mathscr{B}^j_\tau$ from neighbors $j \in \mathscr{N}e_i$
5:  Update the pseudo-point parameters via (2.27)
6:  Update the GP tree data structure for each class $l$ via Sec. 2.3.4
7:  **for** each leaf $N$ of each tree **do**
8:      Update precision matrix $Z^i_{\tau,l}(N)$ in Sec. 2.4.2 via (2.11), (2.12)
9:  **if** TSDF and class evaluation for a set of points is requested **then**
10:     Evaluate the class of each point via Prop. 2 in Sec. 2.4.2
11:     Evaluate the TSDF of each point via (2.8)
12:     **if** mesh reconstruction is requested **then**
13:         Apply Marching Cubes [91] to the TSDF values
```

associated number of observations $\tilde{m}^i_{t+1,l}(\mathbf{p})$ and average observation $\tilde{\zeta}^i_{t+1,l}(\mathbf{p})$ for $\mathbf{p} \in \tilde{\mathscr{P}}^i_{t+1,l}$. Each robot maintains a separate GP $\mathscr{GP}(\mu^i_{t,l}(\mathbf{x}), k^i_{t,l}(\mathbf{x},\mathbf{x}')$ for each class TSDF $f_l(\mathbf{x})$. In the multi-robot case, the GP distributions of robot i are updated simultaneously and separately for all classes using the new class-specific observation data $\tilde{\mathscr{P}}^i_{t+1,l}$, $\tilde{m}^i_{t+1,l}(\tilde{\mathscr{P}}^i_{t+1,l})$, $\tilde{\zeta}^i_{t+1,l}(\tilde{\mathscr{P}}^i_{t+1,l})$ as well as information from the neighboring robots in the form of class-specific packages $\mathscr{B}^i_{t+1,l}$ as described in (2.27). To make the GP models scalable to large environments, we organize the pseudo-points $\mathscr{P}^i_{t,l}$ for each robot i and class l in a hierarchical tree data structure, as in Sec. 2.3.4, and predict the class of a query point via the method in Sec. 2.4.2. Prop. 4 guarantees that the local TSDF GPs at each robot converge to a common GP, which is equivalent to the one that would be obtained by centralized sparse GP regression. Moreover, when the streaming of new observations stops, the convergence happens in finite time as soon as each observation is received by each robot exactly once. There is no unnecessary communication in the network. The algorithm is summarized in Alg. 1.

2.7 Evaluation using 2-D Simulated Data

In this section, we evaluate our semantic TSDF mapping approach in 2-D simulated environments. We first demonstrate the qualitative and quantitative performance of the single-robot approach of Sec. 2.4. Then, we report results for the multi-robot approach of Sec. 2.6 using three robots to map the same environment collaboratively. In all experiments, we use an isotropic sparse Matérn kernel ($\nu = 3/2$) [31]. Since a TSDF value of zero indicates an object surface, while an unknown environment is predominantly empty, we use a constant GP prior equal to the truncation value, $\mu_{0,l}^{i}(\mathbf{x}) = \bar{d} > 0$.

2.7.1 Single-Robot 2-D Evaluation

We generate random 2-D environments (see Fig. 2.6) and robot trajectories by sampling poses sequentially and keeping the ones that are in free space. Observations are obtained along the robot trajectories using a simulated distance-class sensor. We apply our incremental sparse GP regression method to obtain a probabilistic TSDF map and compare it with the ground truth TSDF.

TSDF Accuracy

A sample environment from our 2-D simulation with ground-truth and reconstructed TSDF and boundaries is shown in Fig. 2.6. Our method provides continuous probabilistic TSDF estimates. The choice of $frame\ size$ is dependent on the desired truncation value for the SDF reconstruction. Larger $frame\ size$ allows estimating larger truncation values but incurs additional computation cost. The precision and resilience to measurement noise of our method are evaluated in Fig. 2.7. The test points are chosen from a grid with resolution $0.5 \times voxel\ size$ within the truncation distance from the ground-truth object boundaries.

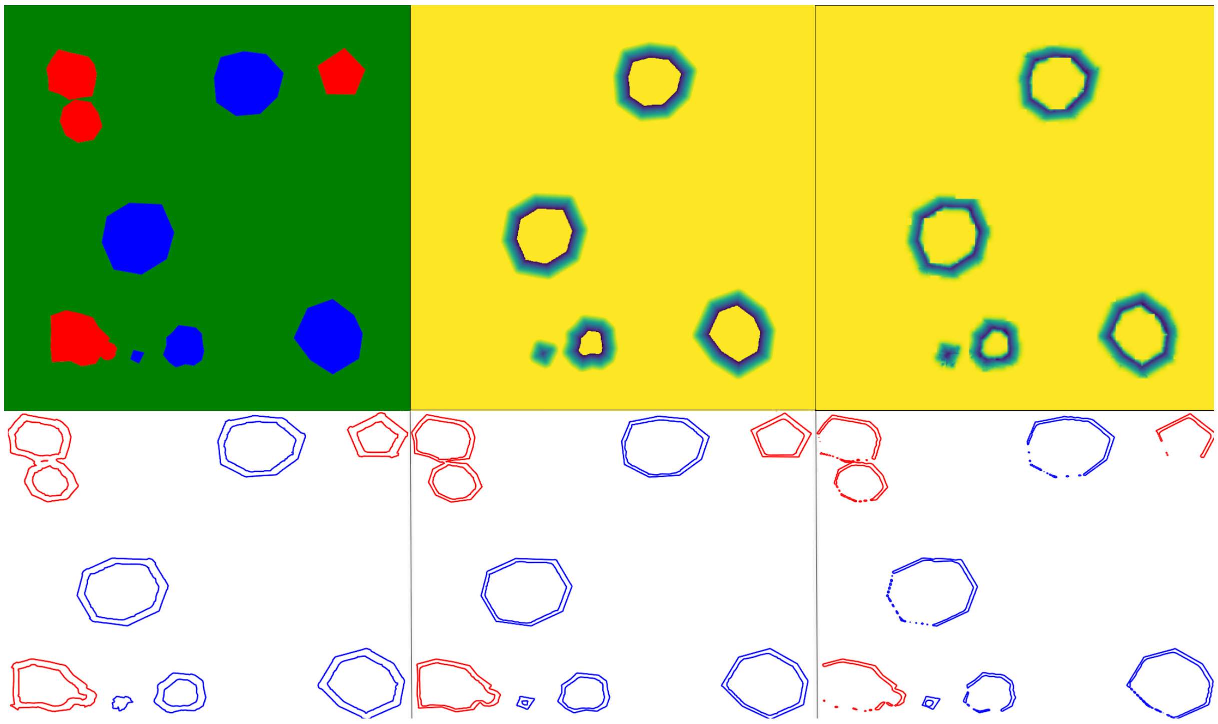


Figure 2.6. Ground-truth 2-D simulated environment (top left) with two object classes (red, blue), ground-truth TSDF for the blue class (top middle), and reconstructed TSDF with $frame\ size = 10$ (top right). The reconstructed TSDF boundaries are shown for three different $frame\ size$ parameters on the bottom row: 10 (bottom left), 3 (bottom middle), 2 (bottom right). Sharp edges are captured better with $frame\ size$ 3 vs. 10 but using $frame\ size$ less that 3 caused missing parts at the boundaries.

Classification Accuracy

We evaluate the average precision and recall of our posterior classification over 50 random 2-D maps. In each map, we pick uniformly distributed random points along the obstacle boundaries, and calculate the SDF error and the class-detection accuracy. Since the values are symmetric for binary classification, we present the average precision and recall over the two classes in Fig. 2.7. The figure shows that the misclassification rate, precision, recall, and SDF error are not very sensitive to class error probability. The misclassification rate is the ratio of all to the misclassified test points. The SDF error is the average absolute value difference between the estimated and ground-truth SDF values. We report normalized SDF error: $\frac{\text{SDF error}}{voxel\ size}$. Fig. 2.8 investigates the effect of the parameters of our algorithm on misclassification rate,

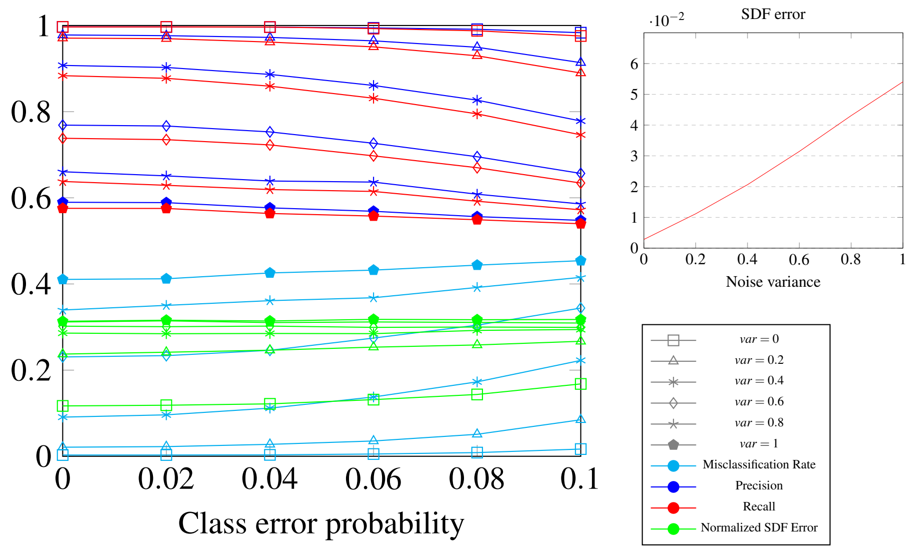


Figure 2.7. Misclassification Rate, Precision, Recall, and Normalized SDF Error for different class error probability and distance noise variance. The top right plot shows the average SDF error over 10 random 40×40 maps with a 100 random observations each, with $voxel\ size = 0.1$, $max(N) = 100$, $\delta = 1.2$.

normalized SDF error, False Discovery Rate (FDR := 1 − Precision), and False Negative Rate (FNR := 1 − Recall). We see that the misclassification rate, FNR, and FDR respond similarly to parameter variations.

Increasing the maximum number of pseudo-points per support region, $max(N)$, in the hierarchical tree structure improves the (normalized) SDF error. The improvement is significant at first but after a certain support region size, even exponential increases in $max(N)$ do not significantly affect the SDF error. The geometric prediction improvement caused by an initial increase in $max(N)$, improves the classification measures at first. The classification noise is iid and, hence, the training data have a Bernoulli misclassification probability and are roughly uniformly distributed in space. Further increase of $max(N)$ results in larger pseudo-point region being used for GP training. Since the misclassified samples are more and not concentrated, the classification measures deteriorate slightly with the increase of $max(N)$. Nevertheless, our approach remains accurate in the classification errors are low in all cases. Increasing δ has a similar effect on all performance measures. Increasing the GP noise variance σ^2 improves all the measures at first but then worsens them. A correct choice of σ^2 is critical to the method but affects the misclassification rate smoothly so as long as the value of σ^2 is in the right ballpark, choosing the optimal σ^2 is not critical.

2.7.2 Multi-Robot 2-D Evaluation

Next, we evaluate the distributed GP regression in a three-robot simulation and investigate the convergence of the local GP estimates of each robot to a centralized GP estimate. We use the same random polygonal 2-D environments with two object classes but this time generate trajectories for three different robots (see Fig. 2.9). The robots communicate with each other over a graph with a fixed weight matrix:

$$W = \begin{bmatrix} 0.5 & 0.25 & 0.25 \\ 0.25 & 0.75 & 0 \\ 0.25 & 0 & 0.75 \end{bmatrix}. \tag{2.29}$$

The GP regression parameters at each robot are the same as the defaults in Sec. 2.7.1. To verify Prop. 4 empirically, we compare the mean absolute error (MAE) between the GP prediction of an individual robot i and the centralized estimator ctr using all observations as described in Sec. 2.5.3. Specifically, at each t, we consider all classes l and associated pseudo-points $\mathscr{P}^{ctr}_{t,l}$ that have been observed by the centralized estimator and calculate the mean MAE as:

$$MAE_t = \frac{1}{L_t|\mathscr{P}^{ctr}_{t,l}|}\sum_{\ell}\sum_{\mathbf{p}\in\mathscr{P}^{ctr}_{t,l}}\left|\mu^{i}_{t,l}(\mathbf{p}) - \mu^{ctr}_{t,l}(\mathbf{p})\right|, \tag{2.30}$$

where L_t is the number of observed object classes by time t. The variance MAE is computed equivalently to (2.30) with $\mu^{i}_{t,l}(\mathbf{p})$ and $\mu^{ctr}_{t,l}(\mathbf{p})$ replaced by $k^{i}_{t,l}(\mathbf{p},\mathbf{p})$ and $k^{ctr}_{t,l}(\mathbf{p},\mathbf{p})$.

Fig. 2.9 shows the final reconstructions of one robot and the centralized estimator. As expected, the final reconstructions are identical and convergence happens in finite time. The behavior of the mean and variance MAE curves is similar. This is expected because the distance between the local and centralized GP parameters is due to unobserved information rather than stochastic noise. We see that the MAE curves approach 0 quickly. Several peaks are observed in the curves when new sections of the environment that are not visible to robot *i* are observed by another robot in the network. The new information disseminates in the network and the MAE curves approach zero again.

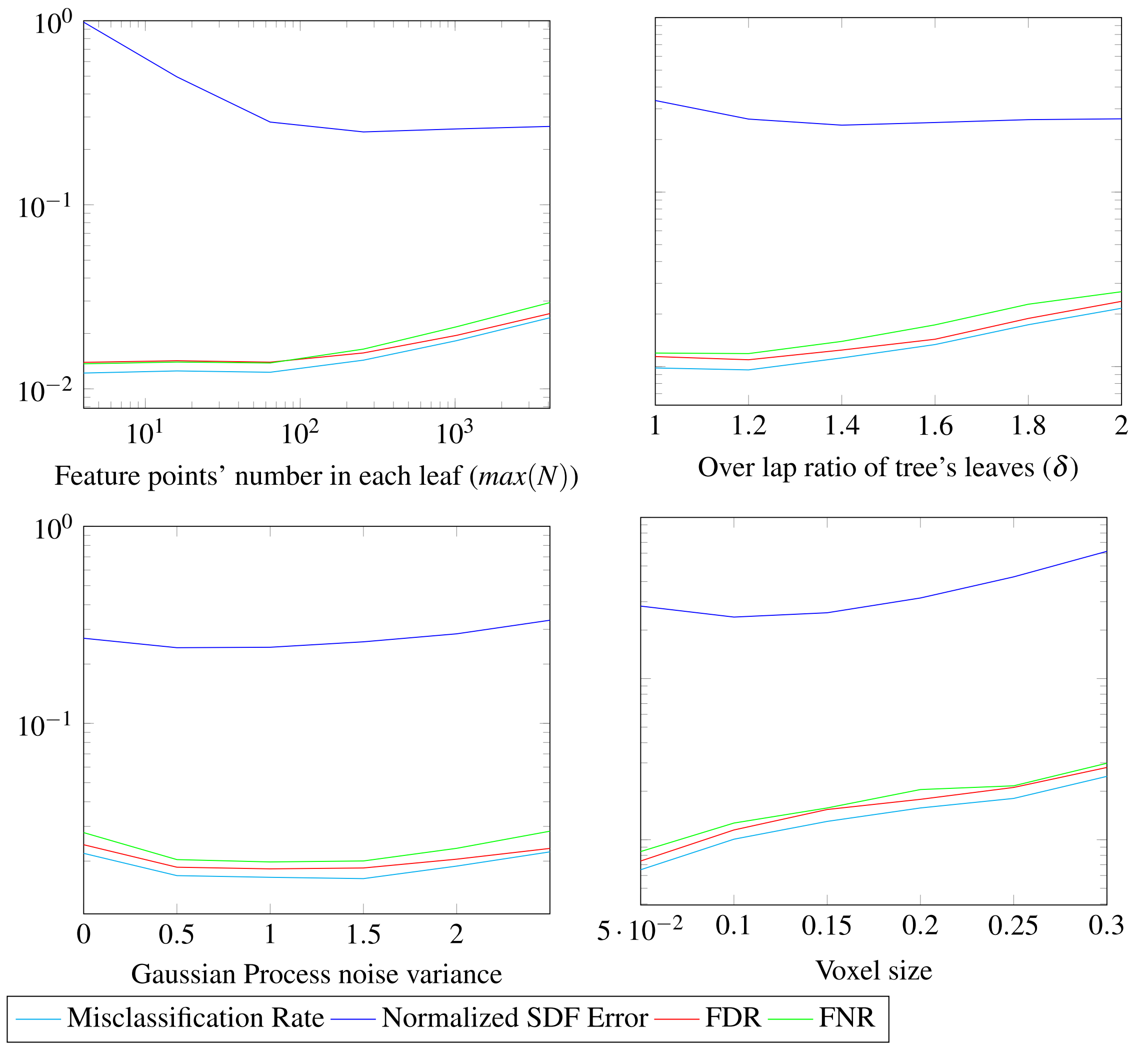


Figure 2.8. Misclassification rate, normalized SDF error, False Discovery Rate (FDR), and False Negative Rate (FNR) as a function of the number of pseudo-points per tree support region ($max(N)$), support region overlap ratio (δ), GP noise variance σ^2, and workspace discretization (*voxel size*). The default parameter values are $\delta = 1.5$, $max(N) = 100$, $\sigma^2 = 1$, *voxel size* $= 0.1$. Class and distance measurements with class error probability of 0.05 and distance noise variance 0.5 are obtained from 100 random observations in each of 50 random 2-D maps. Test points are selected within a threshold of 0.05 from the ground truth class boundaries.

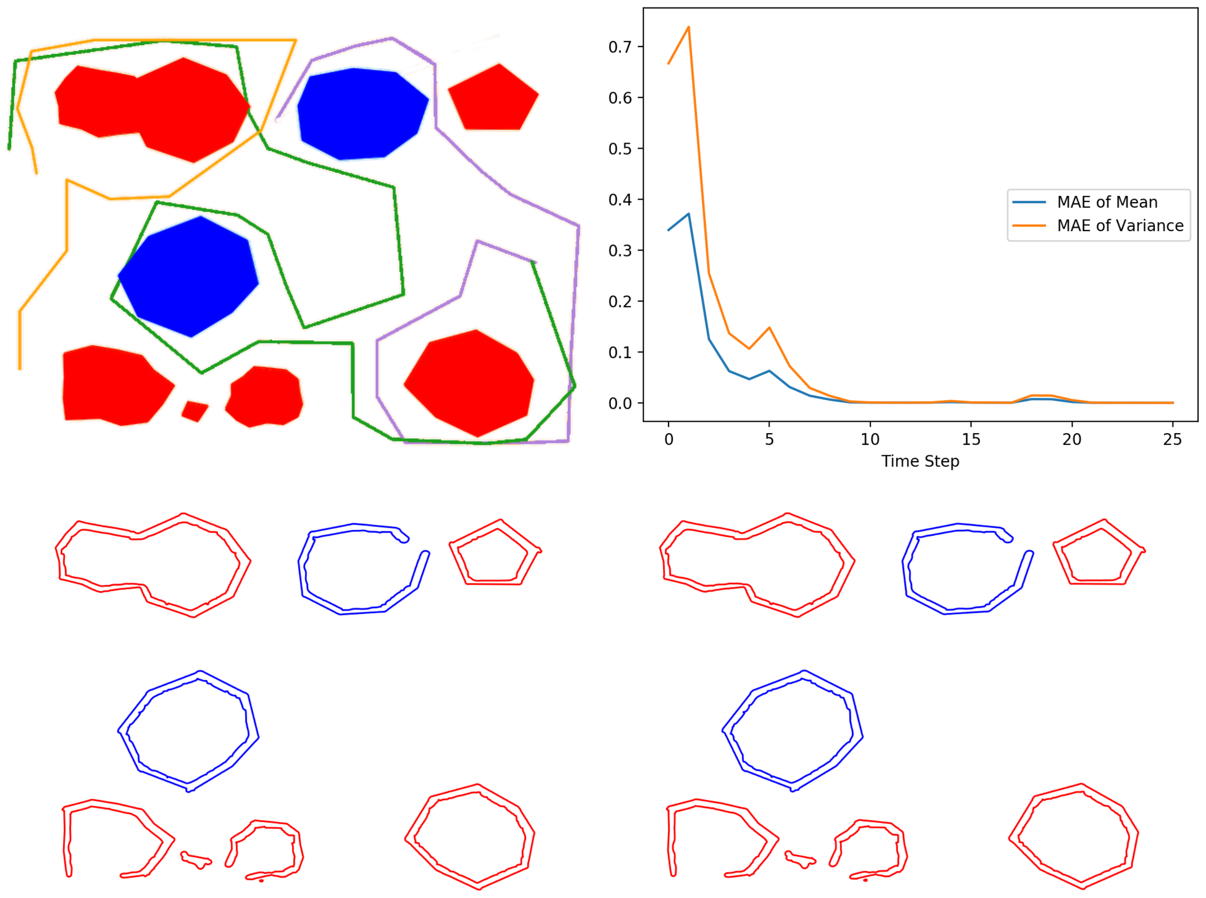


Figure 2.9. The orange robot (bottom right) are shown. As expected, due to Prop. 2, the centralized and individual robot reconstructions are identical. This is verified quantitatively in the GP mean and variance mean absolute error (MAE) plot (top right). The initial GP parameters for each robot and object class were $\mu^i_{0,l}(\mathbf{x}) = 0.5, k^i_{0,l}(\mathbf{x},\mathbf{x}) = 1$.

2.8 Evaluation using 3-D Real Data

In this section, we evaluate our approach using real RGB-D data from physical 3-D environments. As before, we use an isotropic sparse Matérn kernel ($v = 3/2$) and a grid of potential pseudo-points $\mathscr{P}_{\#}$ with resolution *voxel size*. Given a query point $\hat{\mathbf{x}}$, we choose a cubic region around it such that $(frame\ size - 1) \times voxel\ size \geq 2 \times \varepsilon$ to construct the training data in (2.13). All points from $\mathscr{P}_{\#}$ that lie in the cubic region are chosen as pseudo-points associated with $\hat{\mathbf{x}}$.

2.8.1 Single-Robot 3-D Evaluation

We compare our method to the incremental Euclidean signed distance mapping method Fiesta [48] on the Cow and Lady dataset [47]. We also demonstrate 3-D semantic reconstruction on the SceneNN dataset [92].

Cow and Lady Dataset

The reconstruction of the Cow and Lady dataset with 829 depth images and known camera trajectory by the single-robot TSDF GP regression of Sec. 2.4 is shown in Fig. 2.13. A triangular mesh is extracted from the mean TSDF prediction using the Marching Cubes algorithm [93]. The reconstruction time and error with respect to the ground-truth scene point cloud are reported in Fig. 2.10. The error of Fiesta with default parameters is shown as well. Similar to the 2-D simulations, increasing the maximum number of pseudo-points $max(N)$ per tree support region improves the SDF error of our approach. The improvement is significant at first and less pronounced afterwards. Conversely, the computation time decreases at first because the number of leaves in the hierarchical tree structure decreases and then increases afterwards as the GP covariance matrices get larger. Increasing δ leads to an insignificant improvement in the SDF error at the expense of a significant reconstruction time increase. Increasing the GP noise variance improves the SDF error at first (especially when the error is close to zero) but worsens it afterwards without significant impact on time. As *voxel size* varies, our method outperforms

Fiesta noticeably.

SceneNN Dataset

We evaluate the classification accuracy of our method on the SceneNN dataset in Fig. 2.11. The GP posterior is evaluated on a test grid with resolution $0.5 \times \textit{voxel size}$. The test points with posterior variance less than a threshold are used to reconstruct a triangular mesh via Marching Cubes [93]. We use Prop. 2 for classification. The effect of the different parameters is illustrated in Fig. 2.11. Increasing $max(N)$ improves both classification and TSDF reconstruction results. The improvement after $max(N) = 100$ is negligible but time increases significantly. Increasing δ improves the TSDF reconstruction significantly at first. After $\delta = 1.4$, the improvement is negligible. As seen in the 2-D simulations, choosing a correct magnitude for the GP noise variance σ^2 is very important for both the classification and TSDF reconstruction but choosing the optimal value for σ^2 is not critical. Our method provides continuous TSDF estimates but this does not mean that the zero-level set of the estimated TSDF is necessarily continuous. A positive TSDF prediction by the GP means that the corresponding location is empty. Choosing the truncation value as the prior GP mean results in assuming that space is empty by default. The presence of pseudo-points, generated around observed surfaces, is what introduces occupied space in the GP estimation. For example, the grid-like empty spots in Fig. 2.11, resulting when the parameter choices are suboptimal, are due to the choice of pseudo-points on a regular grid.

2.8.2 Parameter Selection

This section discusses parameter selection for our algorithm, based on the results in Sec. 2.8.1. With parameter selection there is a trade-off between accuracy and computational efficiency. For the hierarchical tree structure, as observed in Sec. 2.8.1, it is critical to select the maximum number of pseudo-points per node as $max(N) > 100$ and the size of the node support region as $\delta > 1.4$ to ensure continuous surface representation, robustness to noise, and sufficient accuracy of the decomposition into separate GPs. The larger these two parameters are, however,

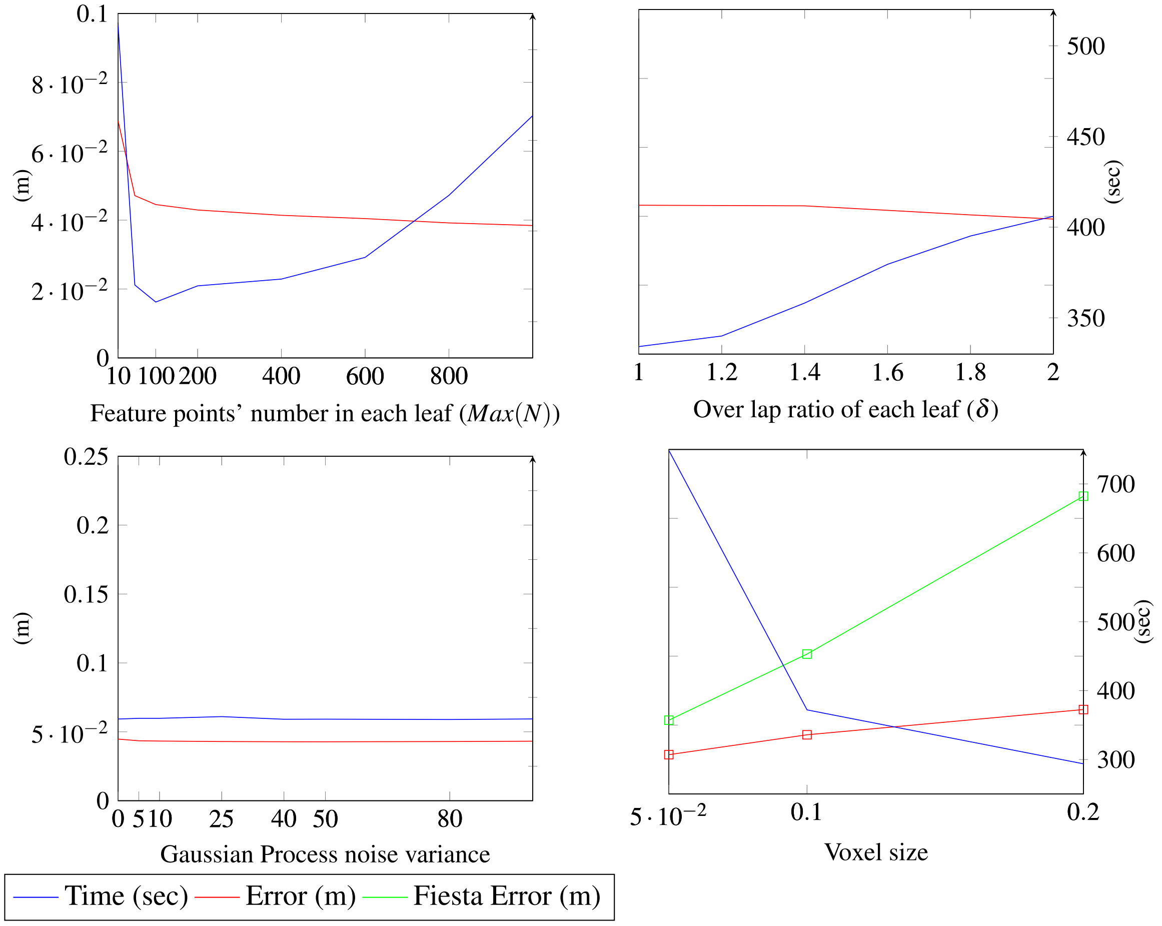
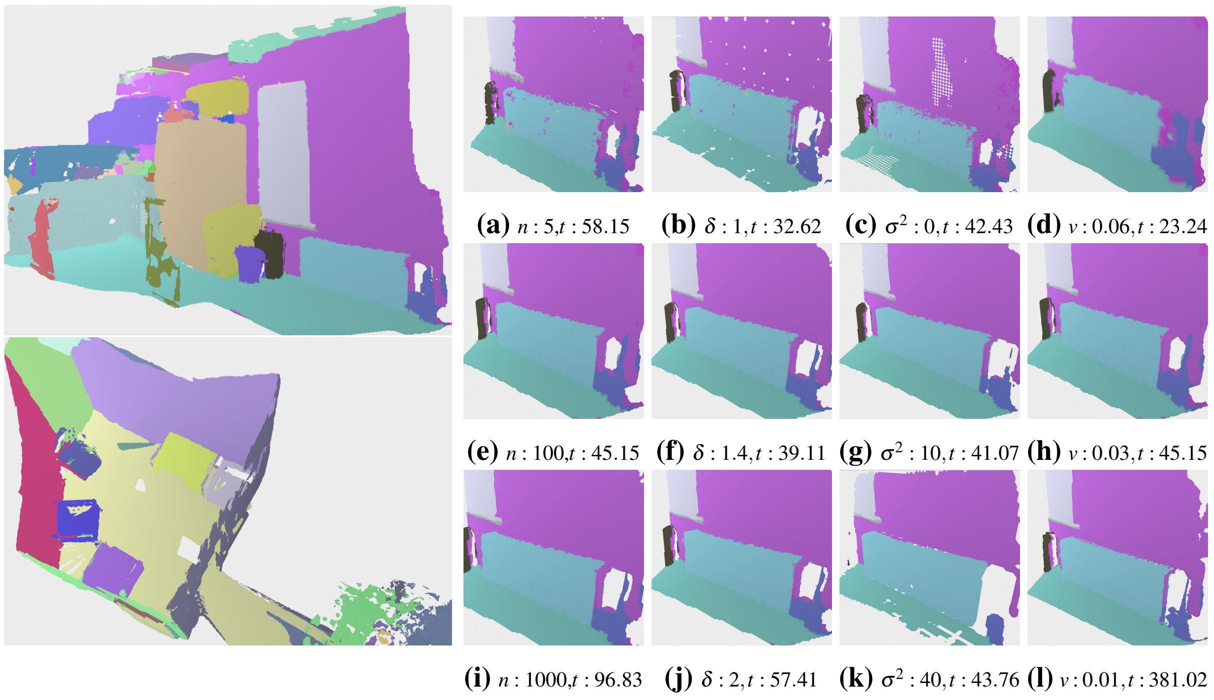


Figure 2.10. Evaluation of the SDF reconstruction time (sec) and error (m) of our incremental sparse GP regression algorithm on the Cow and Lady dataset [47] and in comparison with Fiesta [48]. The errors are evaluated with respect to the ground-truth scene point cloud provided by the dataset. Training is done with 829 depth images and known camera trajectory. The default parameters for our algorithm are $max(N) = 200$, $\delta = 1.5$, $\sigma^2 = 25$, $voxel\ size = 0.1$, $frame\ size = 5$, and SDF truncation value $3 \times voxel\ size$.

the more computationally expensive our algorithm becomes. Our experiments suggest optimal choices around $\delta = 1.5$ and $max(N) = 200$.

The noise parameter σ^2 of the GP model depends on the measurement data. For the RGB-D data used in our experiments $\sigma^2 = 3$ was a good choice. The *voxel size* parameter depends on the level of detail that needs to be captured in the reconstruction, with smaller values increasing the accuracy but also the computational complexity. In our experiments, $voxel\ size = 0.03$ was a good choice. A trade-off between accuracy and computation time is associated with the choice of $frame\ size$, which determines how many pseudo-points are added per sensor ray (Fig. 2.5).



(a) $n:5, t:58.15$ **(b)** $\delta:1, t:32.62$ **(c)** $\sigma^2:0, t:42.43$ **(d)** $v:0.06, t:23.24$

(e) $n:100, t:45.15$ **(f)** $\delta:1.4, t:39.11$ **(g)** $\sigma^2:10, t:41.07$ **(h)** $v:0.03, t:45.15$

(i) $n:1000, t:96.83$ **(j)** $\delta:2, t:57.41$ **(k)** $\sigma^2:40, t:43.76$ **(l)** $v:0.01, t:381.02$

Figure 2.11. Single-robot reconstructions of sequence 255 (top left), containing 2450 RGB-D images and 85 semantic categories (in random colors), and sequence 011 (bottom left), containing 3700 RGB-D images and 61 semantic categories (in random colors), of the SceneNN dataset [92]. The incremental sparse GP TSDF mapping process took 1040.41 sec. for sequence 255 and 1885.72 sec. for sequence 011. The following default parameters were used for the hierarchical tree structure: $\delta = 1.5$, $n = max(N) = 100$ and the GP training: $\sigma^2 = 3$, $v = voxel\ size = 0.03$, $frame\ size = 2$. On the right we see the effect of these parameters (t is time in seconds) on the metric-semantic reconstruction over 140 RGB-D images.

While $frame\ size = 2$ is very efficient and captures fine details, as discussed in Fig. 2.6 and Sec. 2.8.3, it might miss surface boundaries. Choosing $frame\ size = 3$ alleviates this issue while keeping the approach efficient. If TSDF reconstruction accuracy close to the surface, in addition to the surface extraction itself, is important, then $frame\ size \geq 5$ is a good choice (Fig. 2.10).

2.8.3 Comparison with Deep TSDF Reconstruction

This section compares our sparse GP approach to IGR [1], a deep learning approach for SDF reconstruction introduced in Sec. 2.1. We evaluated the two approaches on five car instances from the ShapeNet dataset [2]. Both methods were trained using 100 depth images of size 512×512, obtained from a sphere around the object with camera orientation facing

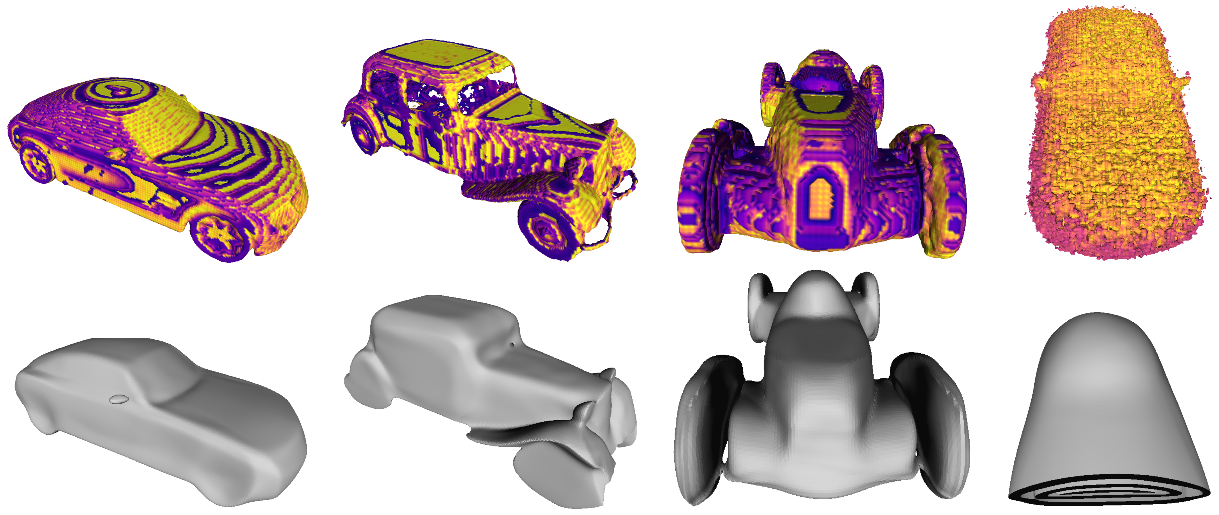


Figure 2.12. Reconstruction of car instances from the ShapeNet dataset using our incremental GP approach (top row) and IGR [1] (bottom row). The color indicates the reconstruction variance provided by our model, ranging from cold (low confidence) to hot (high confidence). The first three car instances are reconstructed from RGB-D images with zero measurement noise, while the last instance is obtained with noise standard deviation of 0.05.

the object. To make the measurement noise independent of the object scale, each element of the depth image was multiplied with Gaussian noise with mean 1 and three levels of standard deviation: 0 (no noise), 0.025 (low noise), and 0.05 (high noise). Our method was trained on a single CPU with parameters: $\sigma^2 = 3$, $voxel\ size = 0.03$, $frame\ size = 2$, and tree support regions with $\delta = 1.5$ and $max(N) = 200$. IGR was trained without a latent shape vector (single shape estimation) on an Nvidia GTX 1080 Ti GPU with initial learning rate 0.005 and decay factor of 2 every 200 steps for $2k$ iterations. Mesh reconstruction from the SDF values was performed using Marching Cubes [91] with the same resolution of $voxel\ size = 0.015$ for both methods.

Qualitative results for three of the object instances are shown in Fig. 2.12. The color patterns observed in the variance prediction of our method correspond to convex object parts which more visible (so the uncertainty is lower) from various camera poses. The meshes in ShapeNet are combinations of flat faces and the object surfaces are not smooth. Quantitative results for noise evaluation are shown in Table 2.1. In the absence of noise IGR is more accurate but the difference is minor. Examining the qualitative results in Fig. 2.12 suggests that our

Table 2.1. Quantitative comparison between our approach and IGR [1] for TSDF reconstruction, averaged over 5 car instances from ShapeNet [2] and evaluated using the metrics from [94]. The metrics are computed after the reconstructed meshes are normalized in a unit-length bounding box.

Method	Noise st. dev.	Chamfer-L_2	Chamfer-L_1	Accuracy	Completeness
Ours	0	0.0010	0.0091	0.0078	0.0105
IGR		0.0007	0.0071	0.0072	0.0070
Ours	0.025	0.0007	0.0104	0.0114	0.0093
IGR		0.0014	0.0262	0.0269	0.0255
Ours	0.05	0.0013	0.0198	0.0248	0.0148
IGR		0.0072	0.0677	0.0863	0.0492

method is able to capture fine details more precisely. As the measurement noise increases, our method's reconstruction accuracy remains robust while IGR's accuracy deteriorates. Fig. 2.12 shows that with large measurement noise the reconstruction from IGR may sometimes fail, while our method is still able to capture the overall object shape.

The total training time per instance for IGR, including mesh reconstruction, was 783.27 sec. Timing results for our method are provided in Table 2.2. Our method was able to process depth images for training set construction and incremental GP training at roughly 6.2 Hz. In robotics applications, such as autonomous navigation, only the dataset construction and GP training steps need to be performed online and a small local portion of the TSDF values may be predicted for collision checking. Prediction at a single test-point took 7.7 μs. If complete TSDF reconstruction and mesh extraction are considered, our method took 66.4 sec to process 100 images.

2.8.4 Multi-Robot 3-D Evaluation

Finally, we evaluate our distributed GP regression on the Cow and Lady and SceneNN datasets. We split the RGB-D image sequences into equal parts and consider each as data obtained by a different robot. As in the 2-D simulation, we use three robots with communication weight

Table 2.2. Profiling our python implementation of Alg. 1 with average times obtained from 5 ShapeNet [2] with 100 RGB-D images each.

Online TSDF mapping without mesh reconstruction	
Data construction and training (42214 pseudo-points, 944970 updates)	16.02 s
Single pseudo-point update	17.13 μs
Single test-point prediction	7.68 μs
Image processing and training rate	**6.24 Hz**
Online TSDF mapping with mesh reconstruction	
Data construction and training (42214 pseudo-points, 944970 updates)	16.02 s
TSDF prediction (4138478 test points)	31.78 s
Marching cubes mesh reconstruction ($944 \times 969 \times 1066$ grid)	11.00 s
Other operations	7.58 s
Total reconstruction time (100 RGB-D images)	66.38 s
Image processing, training, and reconstruction rate	**1.51 Hz**

matrix W in (2.29). Each robot uses the distributed update rule in (2.27) and communication continues for 2 rounds after the last RGB-D image from the individual robot sequences is received. The individual robot parameters are the same as in the single-robot experiments in Sec. 2.8.1. The choice of additional communication rounds is due to Prop. 4, where we showed theoretically that $T+n-1$ rounds are needed for the local GP distributions to agree with the centralized GP estimator. As in the 2-D simulations, to verify Prop. 4 empirically, we compare the mean absolute error (MAE) in (2.30) between the GP mean and variance of an individual robot and the centralized estimator.

The results from the Cow and Lady dataset are reported in Fig. 2.13 and Fig. 2.14, while those from the SceneNN dataset are in Fig. 2.15 and Fig. 2.16. The local and centralized reconstruction results are identical in both data sets, which confirms the expected theoretical consistency. The mean and variance MAE curves also behave similarly in both data sets because the errors in the local GP regression are due to unobserved information, that has not yet been received by the robot, rather than measurement noise. As in the 2-D simulation, the peaks in the MAE curves are due to another robot in the network observing a new region that has not yet

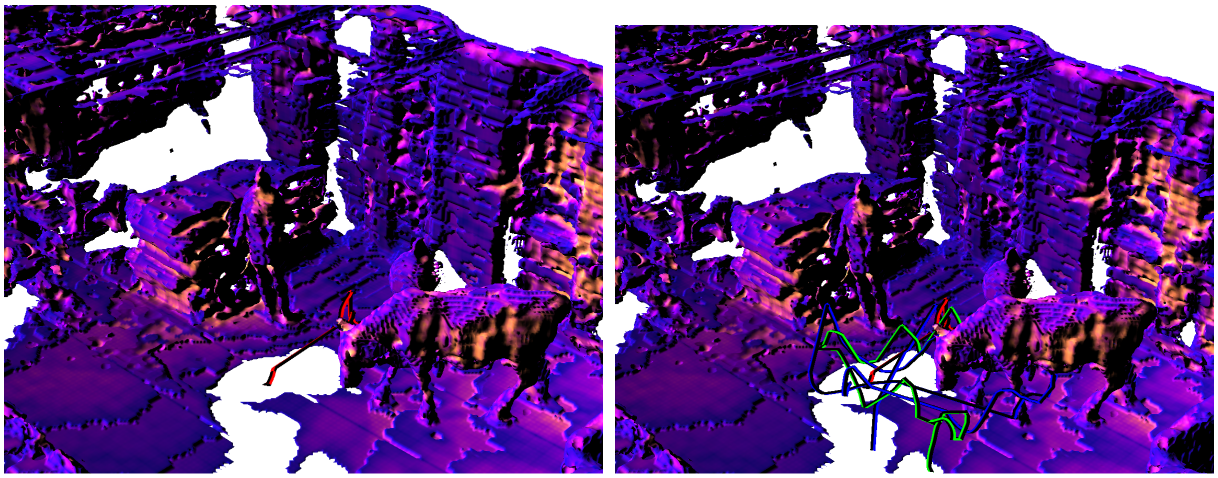


Figure 2.13. The Cow and Lady dataset [47] is divided into three equal sequences of about 275 depth images, and each is considered data obtained by one robot. The three camera trajectories are shown in red, green, and blue on the right. The left plot shows the final reconstruction obtained by the first robot. The right plot shows the final reconstruction obtained by centralized GP regression using the observations of all three robots. The orange hues indicate smaller variance. As expected, due to Prop. 4, the reconstruction of robot one is identical with that of the centralized estimator. The initial GP parameters for each robot and object class were $\mu^i_{0,l}(\mathbf{x}) = 0.15$ and $k^i_{0,l}(\mathbf{x},\mathbf{x}) = 5$.

been observed by this robot. These peaks quickly decrease, which indicates the fast empirical convergence of the distributed sparse GP algorithm.

2.9 Discussion

This paper developed an online Gaussian Process regression method that enables a robot team to build metric-semantic maps collaboratively. A theorem to compress repeated observations before GP training, combined with hierarchical data decomposition, allows scaling to large domains. The presence of distance information allows GP regression instead of computationally challenging GP classification to recover the semantic class distribution. Our approach achieves comparable accuracy to deep neural network techniques for scene reconstruction but offers better robustness to noise, incremental training, and uncertainty quantification. It also enables collaborative inference through distributed GP regression with guaranteed finite-time convergence to the distribution of a centralized estimator. Our probabilistic metric-semantic mapping results

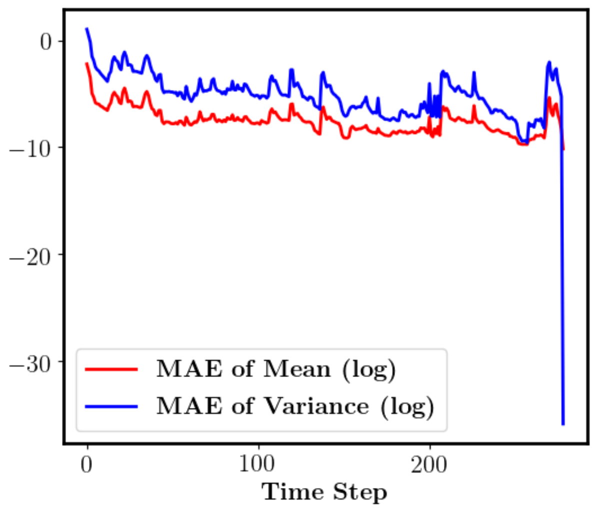


Figure 2.14. Log-space plot of the mean absolute error (MAE) between the mean (red) and variance (blue) predictions of robot 1 and centralized GP regression for the sequence in Fig. 2.13. When the data streaming stops at the end, the MAE approaches zero ($-\infty$ in log space).

offer a promising direction for future research in semantic task specifications and uncertainty-aware task planning.

Chapter 2, in full, is a reprint of the material as it appears in ”Ehsan Zobeidi, Alec Koppel and Nikolay Atanasov. Dense Incremental Metric-Semantic Mapping for Multiagent Systems via Sparse Gaussian Process Regression. IEEE Transactions on Robotics, vol. 38, no. 5, pp. 3133-3153, Oct. 2022”. The dissertation author was the primary investigator and author of this paper.

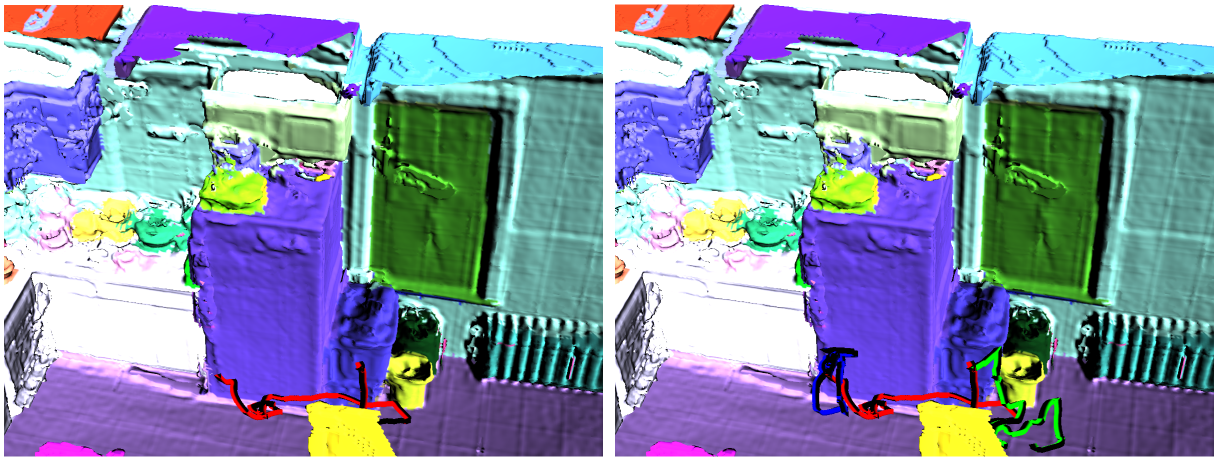


Figure 2.15. Sequence 255 of the SceneNN dataset [92] is divided into three equal sequences of about 800 RGB-D images, and each is considered data obtained by a different robot. The three camera trajectories are shown in red, green, and blue on the right. The left plot shows the final metric-semantic reconstruction obtained by the first robot. The right plot shows the final reconstruction obtained by centralized GP regression using the observations of all three robots. As expected, due to Prop. 4, the reconstruction of robot one is identical with that of the centralized estimator. The initial GP parameters for each robot and object class were $\mu^i_{0,l}(\mathbf{x}) = 0.09$ and $k^i_{0,l}(\mathbf{x},\mathbf{x}) = 5$.

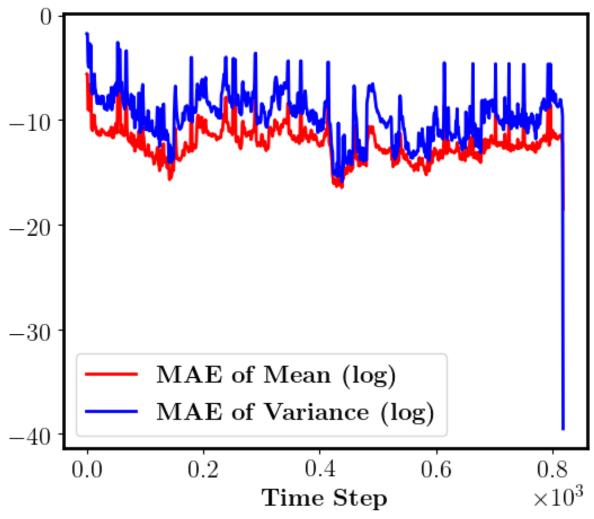


Figure 2.16. Log-space plot of the mean absolute error (MAE) between the mean (red) and variance (blue) predictions of robot 1 and centralized GP regression for the sequence in Fig. 2.15. When the data streaming stops at the end, the MAE approaches zero ($-\infty$ in log space).

Chapter 3

Signed Directional Distance Function Estimation using Deep Learning

Various representations for object shape and geometric surfaces have been proposed by the computer vision, computer graphics, and robotics communities. A unique challenge in robotics applications that involve operation in a priori unknown environments is that shapes and surfaces need to be reconstructed in real-time and immediately and efficiently used for collision or occlusion checking to enable autonomous navigation or manipulation. Shape estimation should be performed online using depth-camera or LiDAR observations, and shape models should ideally be trained with such partial-view measurements rather than complete 3D object models. Distance prediction for collision or occlusion checking should be performed highly efficiently and independently of the distance or viewing direction to the observed surface. As discussed below, many existing shape reconstruction methods have limitations when used for online shape reconstruction from partial views or efficient distance prediction from arbitrary views. In part, this is due to a lack of directional information in the geometry representations used by these methods.

To address this challenge, we propose a new 3D shape representation, called *signed directional distance function (SDDF)*, and a *DeepSDDF neural network model* for learning SDDF representations of objects from distance measurements. An SDDF $h(\mathbf{p}, \eta)$ returns the signed distance from a 3D position $\mathbf{p}$ along a unit-vector viewing direction η. We show that the

SDDF h of an object can be estimated online from sensor data and subsequently allows highly efficient prediction of the distance to the object surface from arbitrary position $\mathbf{p}$ and view η queries in a single look-up evaluation. We discuss the relationship of SDDF to existing shape models next.

3D object shape representations can be classified broadly into *explicit* and *implicit*. Explicit models parameterize an object surface directly using point cloud [95], polygonal mesh [96], or geometric primitive [97] representations. Mesh models can capture geometry, texture, and lighting properties and are widely used in computer vision and graphics. Novel mesh synthesis from partial views [98], however, is challenging, especially if, in addition to the vertex locations, prediction of the mesh topology and number of vertices is required. Implicit models parameterize an object surface indirectly, as a level set of a spatial function defined everywhere in 3D. Common implicit surface representations include volumetric occupancy [99], density [100] and signed distance function (SDF) [101] models. Recently, impressive results have been achieved using deep neural networks to approximate occupancy [94] and SDF [102]. Deep implicit surface models offer advantages with their ability to model arbitrarily complex shapes and topologies but face challenges related to efficient rendering and view synthesis [103]. For example, SDF returns the signed distance to the nearest object surface from a query position and cannot accommodate a desired viewing direction. As a result, distance rendering from SDF requires sphere tracing [104], an iterative process which evaluates the SDF distance multiple times along the desired view ray until it arrives at the observed surface. In contrast, our SDDF formulation returns the distance to the surface in a desired viewing direction directly, as a single function evaluation. This allows significant advantages in rendering speed. Moreover, most deep SDF models provide truncated distance estimates, guaranteed to be accurate only close to the surface. As we discuss below, our DeepSDDF formulation provide analytical confidence in the estimated distance accuracy, *regardless* of the distance to the observed surface. [105, 104, 106] enabled occupancy and density representation to be used for differential rendering. However, due to lack of direction in the geometry of their shape representation, still they struggle with computational load. In contrast, DeepSDDF, is directly differentiable with respect to arbitrary locations and directions, providing an efficient base for differential rendering applications. The direction in density based deep models mostly provide the information about light rather than geometry. However, our method provides a differentiable rendering framework by construction, while keeping the most efficient rendering time.

Understanding the key point that the methods based on previous shape representations struggle with efficiency due to lack of directional information in the geometry, we propose DeepSDDF, the deep model we developed based on the SDDF shape representation has the ability to predict accurate distance at any location and viewing direction, *arbitrarily far from the surface*, and the ability to *render novel distance views as a look-up operation*, obviating the need for mesh reconstruction or sphere tracing [107]. Capturing viewing direction allows training an SDDF model directly from depth camera or LiDAR data. In contrast, SDF training requires pre-processing the data to approximate the shortest distance to the observed surface. Producing such samples is challenging, adds noise to the training process, and is one reasons that SDF methods are frequently trained with 3D mesh supervision. Approximating an SDDF $h(\mathbf{p}, \eta)$ with a neural network appears more challenging due to the additional two-degree-of-freedom view-direction input η. Gropp et al. [1] observe that any valid SDF must satisfy a unit-norm constraint on its gradient (Eikonal equation), and introduce an Implicit Geometric Regularization (IGR) term to encourage a trained SDF model to satisfy this constraint. We show that, similarly, the value of any valid SDDF must decrease at constant unit rate along the viewing direction. Our key contribution is a neural network architecture for learning SDDFs that ensures that the gradient property is satisfied *by construction* and is used to decrease the input dimensionality of an SDDF. Encoding the gradient property in the SDDF model ensures that only valid SDDF functions may be generated. In contrast, the IGR regularization term encourages convergence to a valid SDF function but does not remove possible parameter local minima. This makes our DeepSDDF model significantly more sample efficient than IGR, allowing parameter convergences with little training data despite having both position and direction as inputs. More importantly, the gradient property also provides analytical confidence about the SDDF prediction accuracy *regardless of the distance to the object surface*. In contrast, SDF models are typically truncated with a small threshold around the surface and require a wavefront algorithm [108] to approximate the signed distance far away from the surface. A unique feature of our SDDF model is that it allows differentiable and arbitrary distance view synthesis at the speed of a single forward neural-network pass. For example, at any desired location $\mathbf{p}$, a distance image may be synthesized by simply querying an SDDF model $h(\mathbf{p},\eta)$ for a set of rays η corresponding to the image pixels.

We show that, DeepSDDF, is able to target both object *instance-level* and object *category-level* shape reconstruction as well as the ability to *render novel distance views* accurately and efficiently based on SDDF representation. Our deep model is inspired by DeepSDF [102], which trains a decoder-only network $h(\mathbf{p};\mathbf{z})$ that from a latent shape code $\mathbf{z}$ can model the SDF shape representation of a whole category of objects, such as cars, or airplanes, instead of just individual instances. The latent code $\mathbf{z}$ captures the shape of a specific instance, and the SDF decoder $h(\mathbf{p};\mathbf{z})$ generates the distance to the instance surface at $\mathbf{p}$. Training the model across multiple instances from the same category allows generalization to unseen instances and shape interpolation. Optimizing the latent code at test time also allows shape completion of a novel instance from a partial view. We demonstrate that an SDDF decoder $h(\mathbf{p},\eta;\mathbf{z})$ (DeepSDDF) augmented with a latent shape code $\mathbf{z}$, is similarly capable of representing a category of shapes, allowing shape completion and shape interpolation.

In summary, we make the following contributions.

- We propose a new signed directional distance function (SDDF) for shape representation. A key advantage of SDDF is that it returns distance in any desired viewing direction through a single regression, allowing synthesis of novel views without sphere tracing and direct training from distance data without 3D mesh supervision. We derive a gradient property, that distance along a ray decreases linearly, satisfied by any SDDF. This guarantees the accuracy of SDDF predictions analytically, regardless of the distance to the object surface.

- We propose, DeepSDDF, a neural network model to learn SDDF shape representation. Similar to a deep method for SDF, a DeepSDDF, augmented with a latent shape code, is capable of category-level shape completion and shape interpolation. Due to specifications of SDDF, DeepSDDF satisfies the gradient property, that distance linearly decreases along a ray, by design. SDDF allows DeepSDDF to do ray-tracing as fast as a single pass. To avoid the need for a large training set with data from many views, we develop a data augmentation technique to ensure that an SDDF model trained from a small dataset is multi-view consistent.

Our model is demonstrated in qualitative and quantitative experiments using the synthetic ShapeNet dataset [2] and the real YCB dataset [4].

3.1 Background

This section reviews popular 3D shape representations, including mesh, point cloud, occupancy, SDF, and density (radiance) models, as well as neural network techniques for their reconstruction from sensor data.

Mesh and chart representations

Explicit shape representations model the surface of an object directly, e.g., as a collection of flat polygons [109, 110, 111, 112, 113, 114, 115]. Viewing the surface as a collection of connected charts allows parameterization of object surface by neural network [116, 117, 3]. AtlasNet [118] parametrizes each chart with a multi-layer perceptron that maps a flat square to the real chart. The deep geometric prior (DGP) [3] model improves the AtlasNet results using Wasserstein distance and enforcing a consistency condition to fit the charts. Finding the proper number of patches for different topologies is challenging, and ray tracing is not possible through a single pass.

Geometric primitive representations

Other explicit shape representations use geometric primitives, such as points, cuboids, or quadrics. PointNet [119] proposes a new architecture for point cloud feature extraction that respects the permutation invariance of point clouds. Shape completion from partial point cloud data is investigated by [5, 118, 120, 121, 122, 123] using an encoder-decoder structure to estimate the point cloud of unseen shapes. Point cloud models are simple and efficient but do not provide a continuous shape representation. In cases where a coarse model is sufficient, 3D volumetric primitives can be used to represent the shape [96], including cuboids [124], quadrics [125], or superquadrics [97].

Occupancy representations

A widely used implicit shape representation is based on binary spatial occupancy (occupied vs free) or probability of occupancy. An occupancy model may be stored in a regular grid or adaptive-resolution octree [99, 126, 127] after discretization. OctNet [128, 129] is a deep model that defines convolutions directly over octrees, exploiting the sparsity and hierarchical partitioning of 3D space. As occupancy models do not provide distance information directly, efficient ray-tracing may be be a challenge. Niemeyer et al. [105] enable differentiable rendering of occupancy shape and texture representations by deriving the gradients of the predicted depth map with respect to the network parameters.

Signed distance function representations

The surface of an object may also be represented implicitly as the zero level-set of a signed distance function. An SDF associates with any point in space the signed distance to the closed object surface. SDF values may be estimated in a discrete octree data structure [6, 130, 131, 94] or continuously using Gaussian process [132, 133] or neural network [102, 1] regression. DeepSDF [102] developed an auto-decoder model for approximating continuous SDF values, which enables encoding category-level shape using a latent shape code in the neural network. This work demonstrated that various object topologies can be captured as a single differentiable implicit function, inspiring interest in learned SDF representations [134, 135, 136, 137, 138, 139]. IGR [1] improves DeepSDF by incorporating a unit-norm gradient constraint on the SDF values in the training loss function. Efficient rendering an SDF model is provided by the implicit differentiable renderer (IDR) [104] using sphere tracing. Although the IDR implementation is very efficient, due to the intrinsic limitation of the SDF shape representation, evaluation of each ray requires multiple neural network forward passes.

Density and radiance representations

In a density-based shape representation, an object is viewed as fog with different densities at different locations [100]. In this model, the geometry is not direction-based but, if the density represents radiance, different brightness can be observed from different views. This shape representation is similar in spirit to probabilistic occupancy. NeRF [106, 140, 141], that it achieves impressively realistic scene reconstruction, learns to predict the volume density and radiance of a scene at arbitrary positions and viewing directions using RGB images as input. NeRF allows direction-based rendering but since it represents radiance, instead of directional geometry, it requires sampling to approximate a radiance integral over the view ray, which cannot be done with a single neural network pass.

3.2 Problem Statement

We focus on learning a common shape representation for multiple object instances from the same category, such as cars, airplanes, or benches. We rely on supervision from distance measurements, e.g., from a depth camera or a LiDAR scanner. A distance measurement is modeled as a collection of rays (e.g, corresponding to depth camera pixels or LiDAR beams) along which the distance from the sensor position (e.g., depth camera optical center or LiDAR sensor frame origin) is measured to the observed surface. Let η_i be a vector on the unit sphere $S^{n-1} := \{\eta \in \mathbb{R}^n \mid \|\eta\|_2 = 1\}$ in n dimensions specifying the direction of the i-th sensor ray. Let the distance measurement obtained from sensor position $\mathbf{p}_i \in \mathbb{R}^n$ along direction η_i be

$d_i \in \mathbb{R} \cup \{\infty\}$. In practice, the dimension n is 2 or 3 and distance measurements of rays that do not hit a surface are set to ∞.

Problem 1. *Let $\mathscr{D}_l := \left\{(\mathbf{p}_{i,l}, \eta_{i,l}, d_{i,l})\right\}_i$ be distance measurements obtained from different object instances l of the same category. Learn a latent shape encoding $\mathbf{z}_l \in \mathbb{R}^m$ for each instance l and a function $h(\mathbf{p}, \eta, \mathbf{z})$ that can predict the distance from any point $\mathbf{p}$ along any direction η to the surface of an object instance with shape $\mathbf{z}$ from the same category.*

3.3 SDDF

This section proposes a new signed directional distance function for implicit shape representation. Sec. 3.3.1 defines SDDF and mathematically formulates the condition that the distance to a surface decreases linearly along the viewing direction. Sec. 3.3.2 and Sec. 3.3.3 derive the class of functions that satisfy the SDDF condition and study their properties.

3.3.1 Signed Directional Distance Function

We define a signed directional distance function $h(\mathbf{p}, \eta)$ to model the distance measured by a distance sensor from position $\mathbf{p}$ to the surface of an object $\mathscr{O}$ along viewing direction η.

Definition 3. *A* signed directional distance function *(SDDF) $h : \mathbb{R}^n \times S^{n-1} \mapsto \mathbb{R}$ of set $\mathscr{O} \subset \mathbb{R}^n$ measures the signed distance from a point $\mathbf{p} \in \mathbb{R}^n$ to the set boundary $\partial\mathscr{O}$ in direction $\eta \in S^{n-1}$:*

$$h(\mathbf{p}, \eta) := \min\left\{d \in \mathbb{R} \;\middle|\; \mathbf{p} + d\eta \in \partial\mathscr{O}\right\}. \tag{3.1}$$

Unlike an SDF [102], which measures the distance from $\mathbf{p}$ to the surface $\partial\mathscr{O}$ along the direction that minimizes the distance, an SDDF measures the distance to $\partial\mathscr{O}$ in a *specific* direction η. Also, unlike an SDF, which is negative inside the surface that it models, an SDDF is negative behind the observer's point of view. A key property is that, if the SDDF of a set is known, we can generate arbitrary distance views to the set boundary. In other words, we can image what a distance sensor would see from any point $\mathbf{p}$ in any viewing direction η. The SDDF

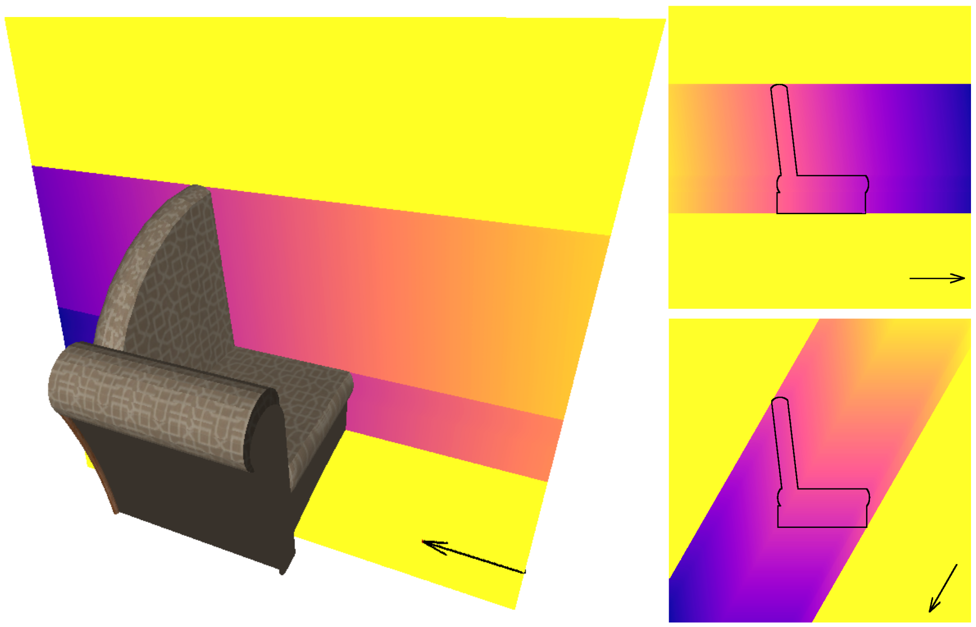


Figure 3.1. Visualization of the SDDF of a sofa viewed from three different directions (indicated by arrows). The color at each point shows the SDDF at that point with fixed direction. The colormap indicates the signed distance from small (purple) to large (yellow). When the view rays do not intersect the object the distance is infinitely large (yellow regions).

definition is illustrated in Fig. 3.1 and Fig. 3.19 (in Appendix 3.5.8). Our definition provides the smallest signed distance to the object surface along a viewing direction. This makes the distance continuous along a viewing direction with positive values before the first surface and negative values afterwards.

An SDDF possesses a property that the distance to the surface $\partial\mathscr{O}$ decreases linearly as the observer approaches $\partial\mathscr{O}$ along the viewing direction η. To formulate this property mathematically, consider an observer located at position $\mathbf{p}_1$. Let the SDDF to the surface $\partial\mathscr{O}$ in direction η be $h(\mathbf{p}_1,\eta)$. Suppose that the observer moves a distance of δ units along the viewing direction η towards $\partial\mathscr{O}$. Now, the observer is at position $\mathbf{p}_2 = \mathbf{p}_1 + \delta\eta$ and the SDDF $h(\mathbf{p}_2,\eta)$ in direction η is decreased by δ units, i.e., $\delta = h(\mathbf{p}_1,\eta) - h(\mathbf{p}_2,\eta)$. This property is stated below, where $\nabla_{\mathbf{p}} h(\mathbf{p},\eta)^\top \eta = \lim_{\delta\to 0} \frac{h(\mathbf{p}+\delta\eta,\eta)-h(\mathbf{p},\eta)}{\delta}$ is the gradient of h with respect to $\mathbf{p}$ projected along the direction η.

Lemma 2. *The gradient of an SDDF $h(\mathbf{p},\eta)$ with respect to the position $\mathbf{p}$ projected to the* *viewing direction η satisfies:*

$$\nabla_{\mathbf{p}} h(\mathbf{p},\eta)^\top \eta = -1. \tag{3.2}$$

3.3.2 SDDF Structure

In this section, we study the class of functions that satisfy the SDDF gradient property noted in Lemma 2. We show that, while the domain $\mathbb{R}^n \times S^{n-1}$ of an SDDF has $2n-1$ degrees of freedom, the constraint in (3.2) removes one additional degree of freedom. To see this, we first simplify the condition in (3.2) by defining a function $g(\mathbf{p},\eta) := h(\mathbf{p},\eta) + \mathbf{p}^\top \eta$. Since the viewing direction η is a unit vector, we have $\eta^\top \eta = 1$ and (3.2) is equivalent to the requirement that:

$$\nabla_{\mathbf{p}} g(\mathbf{p},\eta)^\top \eta = 0. \tag{3.3}$$

The zero-gradient condition in (3.3) suggests that $g(\mathbf{p},\eta)$ is constant along some direction and, hence, has only $2n-2$ degrees of freedom. To show this rigorously, we rotate the coordinate system so that the viewing direction η lies along the last coordinate axis. For example, in $n=3$ dimensions we can align η with the z-axis, showing that the third element of the gradient of $g(\mathbf{p},\eta)$ should be zero or equivalently that $g(\mathbf{p},\eta)$ is constant along the third dimension in the rotated reference frame.

The rotation matrix $\mathbf{R} \in SO(n)$ that rotates a unit vector $\mathbf{x} \in S^{n-1}$ to another unit vector $\mathbf{y} \in S^{n-1}$ with $\mathbf{y} \neq -\mathbf{x}$ along the sphere geodesic [142] (shortest path) is:

$$\mathbf{R} = \mathbf{I} + \mathbf{y}\mathbf{x}^\top - \mathbf{x}\mathbf{y}^\top + \frac{1}{1+\mathbf{x}^\top \mathbf{y}}(\mathbf{y}\mathbf{x}^\top - \mathbf{x}\mathbf{y}^\top)^2. \tag{3.4}$$

Using (3.4), we can obtain an explicit expression for the rotation $\mathbf{R}_\eta$ that aligns a viewing direction η with the unit vector $\mathbf{e}_n = [0,\ldots,0,1]^\top$. Lemma 3 and 4 provide the rotation matrices for the 2D and 3D cases.

Lemma 3. *A vector $\eta = [a,b]^\top \in S^1$ can be rotated to $\mathbf{e}_2 \in S^1$ via the rotation matrix $\mathbf{R}_\eta =$*

$$\begin{bmatrix} b & -a \\ a & b \end{bmatrix} \in SO(2).$$

Lemma 4. *A vector $\eta = [a,b,c]^\top \in S^2$ can be rotated to $\mathbf{e}_3 \in S^2$ via the rotation matrix $\mathbf{R}_\eta \in SO(3)$ below:*

$$\mathbf{R}_\eta = \begin{cases} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} & \text{if } \eta = -\mathbf{e}_3, \\ \begin{bmatrix} 1-\frac{a^2}{1+c} & -\frac{ab}{1+c} & -a \\ -\frac{ab}{1+c} & 1-\frac{b^2}{1+c} & -b \\ a & b & c \end{bmatrix} & \text{otherwise}. \end{cases} \tag{3.5}$$

To reveal the structure implied by the condition in (3.3), we express it in a rotated coordinate frame where $\mathbf{e}_n = \mathbf{R}_\eta \eta$ and $\mathbf{q} = \mathbf{R}_\eta \mathbf{p}$. By the chain rule:

$$0 = \frac{\partial g}{\partial \mathbf{p}} \eta = \frac{\partial g}{\partial \mathbf{q}} \frac{\partial \mathbf{q}}{\partial \mathbf{p}} \eta = \frac{\partial g}{\partial \mathbf{q}} \mathbf{R}_\eta \eta = \frac{\partial g}{\partial \mathbf{q}} \mathbf{e}_n = \frac{\partial g}{\partial q_n}. \tag{3.6}$$

The set of functions that satisfy (3.6) do not depend on the last element of $\mathbf{q}$. In other words, any function $g(\mathbf{p},\eta)$ that satisfies (3.6) can be expressed as $g(\mathbf{p},\eta) = f(\mathbf{P}\mathbf{R}_\eta \mathbf{p},\eta)$ for some function $f : \mathbb{R}^{n-1} \times S^{n-1} \mapsto \mathbb{R}$ and a projection matrix $\mathbf{P} := [\mathbf{I}\ \mathbf{0}] \in \mathbb{R}^{(n-1)\times n}$, which drops the last element of $\mathbf{q} = \mathbf{R}_\eta \mathbf{p}$. We summarize this structural property of SDDF below.

Proposition 5. *A function $h : \mathbb{R}^n \times S^{n-1} \mapsto \mathbb{R}$ is a valid SDDF, i.e., satisfies Def. 3 and the gradient condition* (3.2) *in Lemma 2, if and only if:*

$$h(\mathbf{p},\eta) = f(\mathbf{P}\mathbf{R}_\eta \mathbf{p},\eta) - \mathbf{p}^\top \eta \tag{3.7}$$

for some function $f : \mathbb{R}^{n-1} \times S^{n-1} \mapsto \mathbb{R}$ and $\mathbf{R}_\eta$ defined in (3.4) *with $\mathbf{x} = \eta$, $\mathbf{y} = \mathbf{e}_n$.*

Proposition 5 shows that space of SDDF functions is isomorphic to a space of real-valued functions with $2n-2$ degrees of freedom. For example, any SDDF in 2D with position $\mathbf{p} = [x,y]^\top$ and direction $\eta = [a,b]^\top$ such that $a^2+b^2 = 1$ can be expressed as $h(x,y,a,b) = f(bx - ay, a, b) - (ax+by)$. The gradient condition in (3.2) is satisfied by construction: $\nabla_{\mathbf{p}} h(\mathbf{p},\eta)^\top \eta = (f'[b,-a] - [a,b]) \cdot [a,b] = (f'b-a)a - (f'a+b)b = -a^2 - b^2 = -1$, where f' denotes the derivative of f with respect to its first argument evaluated at $(bx-ay,a,b)$.

3.3.3 Infinite SDDF Values

Proposition 5 allows learning SDDF representations of object shape from distance measurements without the need to enforce structure constraints explicitly. A remaining challenge is that the measured distance at some sensor position $\mathbf{p}$ and viewing direction η not directly looking towards an object surface may be infinite. Since a regression model cannot predict infinite values directly, we use an invertible function ϕ to squash the distance measurements to a finite range.

Lemma 5. *Let $\phi : \mathbb{R} \mapsto \mathbb{R}$ be a function with non-zero derivative, $\phi'(x) \neq 0$, for all $x \in \mathbb{R}$. Then, for any function $g : \mathbb{R}^n \times S^{n-1} \mapsto \mathbb{R}$ and vector $\eta \in S^{n-1}$, we have:*

$$\nabla_{\mathbf{p}} g(\mathbf{p},\eta)^\top \eta = 0 \quad \textit{if and only if} \quad \nabla_{\mathbf{p}} \phi(g(\mathbf{p},\eta))^\top \eta = 0. \tag{3.8}$$

Proof. The claim is concluded by the chain rule, $0 = \nabla_{\mathbf{p}} \phi(g(\mathbf{p},\eta))^\top \eta =$
$\phi'(g(\mathbf{p},\eta)) \nabla_{\mathbf{p}} g(\mathbf{p},\eta)^\top \eta$, and since $\phi'(g(\mathbf{p},\eta))$ is never zero. □

Since ϕ' in Lemma 5 is never zero, ϕ is either strictly increasing or strictly decreasing by the mean value theorem. In both cases, it has an inverse ϕ^{-1}. Such functions can be used to squash the distance values to a finite range. Examples include logistic sigmoid $\sigma(x) := (1+\exp(-x))^{-1}$, hyperbolic tangent $\tanh(x)$, and the Gaussian error function $\mathrm{erf}(x)$. For better performance in distance reconstruction, ϕ should be chosen with a linear region covering most possible distances and saturation regions covering the less likely (e.g., very large or very small) distances. For example, the logistic sigmoid, hyperbolic tangent and Gaussian error functions all have linear regions around zero, which is well-suited for our SDDF definition, which allows positive and negative distances. The functions $\tanh(x)$ and $\mathrm{erf}(x)$ are odd (symmetric about the origin), which makes them good choices for squashing large positive or negative distances without distortion. An additional scaling parameter λ can be added, e.g., $\tanh(\lambda x)$, to determine the slope of the linear region around the origin. Hereafter we assume ϕ is strictly increasing and define $q(\mathbf{p},\eta) := \phi(f(\mathbf{P}\mathbf{R}_\eta\mathbf{p},\eta))$ such that as in Proposition 5:

$$h(\mathbf{p},\eta) = \phi^{-1}(q(\mathbf{p},\eta)) - \mathbf{p}^\top\eta. \tag{3.9}$$

The formulation in (3.9) allows training a neural network parameterization of $q(\mathbf{p},\eta)$ with possibly infinite distance data. Due to Lemma 5, (3.9) is guaranteed to satisfy the SDDF property $\nabla_\mathbf{p} h(\mathbf{p},\eta)^\top\eta = -1$ by construction.

3.4 DeepSDDF

This section proposes a neural network architecture and a cost function for learning instance-level and category-level SDDF shape models.

3.4.1 DeepSDDF Neural Network

Based on the formulation in (3.9), we design *DeepSDDF*, a neural network architecture for SDDF learning, shown in Fig. 3.2. The network takes position-view pairs $(\mathbf{p},\eta)$ as input and requires corresponding distance measurements d for supervision. The architecture computes $\mathbf{P}\mathbf{R}_\eta\mathbf{p}$ analytically and then uses a fully connected autodecoder to map $\mathbf{P}\mathbf{R}_\eta\mathbf{p}$ and η to $y = \phi(f(\mathbf{P}\mathbf{R}_\eta\mathbf{p},\eta))$. We ensure that the autodecoder output does not exceed the maximum squashed distance, $m = \min\{y, \phi(\infty)\}$, and compare its value with the distance measurement, $\phi(d + \mathbf{p}^\top\eta)$, transformed according to (3.9). This neural network design ensures that the SDDF gradient property in Lemma 3.2 is satisfied by construction.

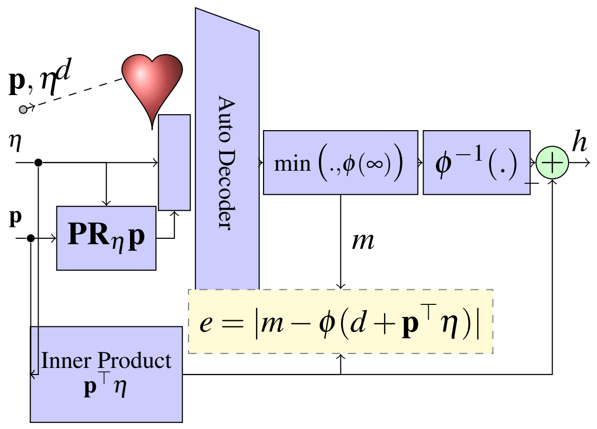


Figure 3.2. Neural network model for DeepSDDF. Given a position $\mathbf{p}$, viewing direction η and measured distance d, the model rotates $\mathbf{p}$ to new coordinates $\mathbf{R}_\eta \mathbf{p}$, whose last component does not effect the SDDF value and is removed via a projection matrix $\mathbf{P}$. The projected input is processed by an autodecoder to predict a squashed distance value m, which may be converted to an SDDF value h or compared to a modified distance $d + \mathbf{p}^\top \eta$ in the error function.

3.4.2 Instance-Level DeepSDDF Training

Given distance measurements $\mathscr{D}_l$ from a single object instance l, as in Problem 1, we can learn an SDDF shape representation $h(\mathbf{p}, \eta)$ by optimizing the autodecoder parameters of the DeepSDDF model in Fig. 3.2. We split the training data $\mathscr{D}_l$ into two sets, distinguishing whether the distance measurements are finite or infinite:

$$\begin{aligned} \mathscr{F}_l &:= \{(\mathbf{p}, \eta, d) \in \mathscr{D}_l \mid d < \infty\}, \\ \mathscr{I}_l &:= \{(\mathbf{p}, \eta, d) \in \mathscr{D}_l \mid d = \infty\}, \end{aligned} \tag{3.10}$$

and define an error function for training the parameters θ:

$$\begin{aligned} e(\theta; \mathscr{F}, \mathscr{I}) := &\frac{\alpha}{|\mathscr{F}|} \sum_{(\mathbf{p},\eta,d)\in\mathscr{F}} |\phi(d + \mathbf{p}^\top \eta) - q_\theta(\mathbf{p}, \eta)|^p \\ &+ \frac{\beta}{|\mathscr{I}|} \sum_{(\mathbf{p},\eta,d)\in\mathscr{I}} r(\phi(\infty) - q_\theta(\mathbf{p}, \eta))^p + \gamma \|\theta\|_p^p, \end{aligned} \tag{3.11}$$

where $\alpha, \beta, \gamma > 0$ are weights, $p \geq 1$, and r is a rectifier, such as ReLU $r(x) = \max\{0, x\}$, GELU $r(x) = x\Phi(x)$, or softplus $r(x) = \log(1 + \exp(x))$. In the experiments, we use $p = 1$ and

$r(x) = \max\{0, x\}$. The last term in (3.11) is used to regularizes the network parameters θ but in our experiments we set γ to zero. The first term encourages the autodecoder $q_\theta(\mathbf{p}, \eta)$ to predict the squashed distance values accurately. We introduced a rectifier r in the second term in (3.11) to allow the output of $q_\theta(\mathbf{p}, \eta)$ to exceed $\phi(\infty)$, which we observed empirically leads to faster convergence. To address that $q_\theta(\mathbf{p}, \eta)$ may exceed $\phi(\infty)$, we modify its conversion to an SDDF as: $h_\theta(\mathbf{p}, \eta) = \phi^{-1}(\min\{q_\theta(\mathbf{p}, \eta), \phi(\infty)\}) - \mathbf{p}^\top \eta$.

3.4.3 Category-Level DeepSDDF Training

Next, we consider learning an SDDF shape model for a complete object category with L instances. Inspired by DeepSDF [102], we introduce a latent code $\mathbf{z}_l \in \mathbb{R}^m$ to model the shape of each instance l and learn it as part of the neural network parameters with structure $q_\theta(\mathbf{p}, \eta, \mathbf{z}_l)$ described in Sec. 3.5. Given finite $\mathscr{F}_l$ and infinite $\mathscr{I}_l$ distance measurements, we optimize $\mathbf{z}_l$ independently, for each instance l, and θ jointly, across all instances using the same error as in (3.11):

$$\begin{aligned}
\min_{\theta, \{\mathbf{z}_l\}_l} & \frac{\alpha}{\sum_l |\mathscr{F}_l|} \sum_l \sum_{(\mathbf{p},\eta,d) \in \mathscr{F}_l} |\phi(d + \mathbf{p}^\top \eta) - q_\theta(\mathbf{p}, \eta, \mathbf{z}_l)|^p \\
& + \frac{1}{\sum_l |\mathscr{I}_l|} \sum_l \sum_{(\mathbf{p},\eta,d) \in \mathscr{I}_l} \beta r(\phi(\infty) - q_\theta(\mathbf{p}, \eta, \mathbf{z}_l))^p \\
& + \gamma \|\theta\|_p^p + \sigma \frac{1}{L} \sum_l \|\mathbf{z}_l\|_p^p,
\end{aligned} \tag{3.12}$$

where the last term regularizes the latent codes.

Training a category-level SDDF shape model allows predicting the shape of a previously unseen instance online from a partial observation. Assume that the category-level neural network parameters θ are already trained offline and we have an average category-level shape encoding $\bar{\mathbf{z}} \in \mathbb{R}^m$ (e.g., obtained by using a fixed $\mathbf{z}$ for all instances l during training or simply as the mean of $\{\mathbf{z}_l\}_l$). We initialize the shape code for a newly observed instance with $\bar{\mathbf{z}}$ and optimize it using distance measurements $\mathscr{F}$ and $\mathscr{I}$ and the same error function as before:

$$\begin{aligned}\min_{\mathbf{z}} \frac{\alpha}{|\mathscr{F}|} \sum_{(\mathbf{p},\eta,d)\in\mathscr{F}} |\phi(d+\mathbf{p}^\top\eta) - q_\theta(\mathbf{p},\eta,\mathbf{z})|^p \\ + \frac{\beta}{|\mathscr{I}|} \sum_{(\mathbf{p},\eta,d)\in\mathscr{I}} r(\phi(\infty) - q_\theta(\mathbf{p},\eta,\mathbf{z}))^p + \sigma\|\mathbf{z}\|_p^p.\end{aligned} \tag{3.13}$$

The optimized latent shape code $\mathbf{z}^*$ captures all geometric information about the instance and can be used to synthesize novel distances to its surface from any point $\mathbf{p}$ in any viewing direction η by a single forward pass through the SDDF model.

3.4.4 Multi-View Consistency

Compared to SDF whose input is a position $\mathbf{p}$ only, SDDF includes a viewing direction η and hence needs diverse training data with respect to η to guarantee *multi-view consistency*. We provide a data augmentation technique that augments a training set with additional synthetic data by judging which of the observed surface points in the dataset may be visible from other views. We emphasize that no additional training data is required. Rather for each surface point, new directions from which the point is visible are added to the training set. The new viewing directions are synthesized around the original direction from which the point was initially observed. This augmentation technique allows training an SDDF model from a small dataset with limited view directions in a multi-view consistent way.

Proposition 5 reduces the input dimension of an SDDF function $h(\mathbf{p},\eta)$ from $2n-1$ to $2n-2$. To model 3D shape, we need to represent a 4D SDDF function. In contrast, an SDF model [102] has a 3D input, which may even be reduced to a 2D surface using an Eikonal constraint [1]. Hence, training a multi-view consistent SDDF model might require a larger data set with distance measurements from many positions $\mathbf{p}$ and directions η. To reduce the necessary data, we develop an approach to synthesize additional data from the initial training set $\mathscr{D}_l := \{(\mathbf{p}_{i,l}, \eta_{i,l}, d_{i,l})\}_i$.

Remark. *The data augmentation process uses the same point cloud data as the original training* *set. For each point, it chooses additional view directions from which the point is visible.*

Given an arbitrary position $\hat{\mathbf{p}} \in \mathbb{R}^n$, we describe how to synthesize both finite and infinite (no surface hit) distance measurements $\hat{d}$ along different view rays $\hat{\eta}$ originating at $\hat{\mathbf{p}}$. Let $\mathscr{P}_l := \{\mathbf{p} + d\eta \mid (\mathbf{p}, \eta, d) \in \mathscr{D}_l, d < \infty\}$ be a point cloud representation of the training data.

Infinite Ray Synthesis

To synthesize infinite rays, we project the point cloud $\mathscr{P}_l$ to the desired image frame and select the directions $\hat{\eta}$ of all pixels that do not contain a projected point. These directions correspond to views with infinite distance. The points may be inflated with a finite radius to handle sparse point cloud data.

Finite Ray Synthesis

If a point $\mathbf{q} \in \mathscr{P}_l$ is observable from $\hat{\mathbf{p}}$, we can obtain a synthetic measurement with distance $\hat{d} = \|\mathbf{q} - \hat{\mathbf{p}}\|_2$ in direction $\hat{\eta} = \frac{1}{\hat{d}}(\mathbf{q} - \hat{\mathbf{p}})$. The challenge is to decide which points in $\mathscr{P}_l$ are visible from $\hat{\mathbf{p}}$. For $\mathbf{q} \in \mathscr{P}_l$, let $\mathbf{p}$ be the start point of the ray that observed $\mathbf{q}$ originally. We know that $\mathbf{q}$ is observable from $\mathbf{p}$. In contrast, for all $\mathbf{u} \in \mathscr{P}_l \setminus \{\mathbf{q}\}$ and all $\varepsilon > 0$, $\mathbf{q}$ is not observable from $\mathbf{u} - \varepsilon(\mathbf{q} - \mathbf{u})$, since $\mathbf{u}$ is in the way. Hence, $\mathbf{q}$ is observable when we look at it in the direction $\frac{\mathbf{q}-\mathbf{p}}{\|\mathbf{q}-\mathbf{p}\|_2}$ and unobservable in the direction $\frac{\mathbf{q}-\mathbf{u}}{\|\mathbf{q}-\mathbf{u}\|_2}$ for all $\mathbf{u} \in \mathscr{P}_l \setminus \{\mathbf{q}\}$. For convenience of representation, translate all points such that $\mathbf{q}$ is at the origin, project all points on a unit sphere around the origin, and rotate all of the points such that $\mathbf{p}$ maps to $\mathbf{e}_3 = [0,0,1]^\top$. Formally, this can be achieved with the transformation:

$$T_{\mathbf{q}}(\mathbf{x}) := \begin{cases} [0,0,0]^\top, & \text{if } \mathbf{x} = \mathbf{q}, \\ \mathbf{R}_\eta \frac{\mathbf{x}-\mathbf{q}}{\|\mathbf{x}-\mathbf{q}\|_2}, & \text{otherwise}, \end{cases} \tag{3.14}$$

where $\eta = \frac{\mathbf{p}-\mathbf{q}}{\|\mathbf{p}-\mathbf{q}\|_2}$. To decide whether $\mathbf{q}$ is observable from $\hat{\mathbf{p}}$, equivalently we should decide whether the origin is observable from $T_{\mathbf{q}}(\hat{\mathbf{p}})$. Let $\mathscr{P}_{\mathbf{q}} := T_{\mathbf{q}}(\mathscr{P}_l \setminus \{\mathbf{q}\})$. The origin is observable from $\mathbf{e}_3$ and a region around it and unobservable from all $\mathbf{u} \in \mathscr{P}_{\mathbf{q}}$. See Fig. 3.3 for an illustration.

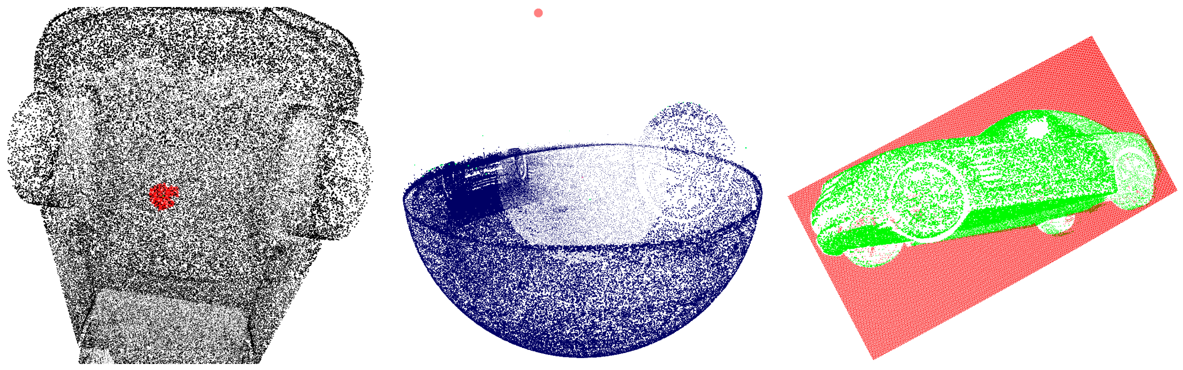


Figure 3.3. A new distance view (right) is synthesized from a point cloud $\mathscr{P}_l$ (left) by deciding whether each point $\mathbf{q}$ (left, red) is visible from the new view $\hat{\mathbf{p}}$. The point cloud is projected on a sphere around $\mathbf{q}$ (middle) to judge the visibility from $\hat{\mathbf{p}}$ (middle, red).

Spherical Convex Hull Data Augmentation

We approximate the region of points around $\mathbf{e}_3$ that can observe the origin. The vertices adjacent to $\mathbf{e}_3$ in the convex hull of $\mathscr{P}_\mathbf{q} \cup \{\mathbf{e}_3\}$ represent the boundary. We sort the boundary points based on their azimuth so that the geodesic among them represents the boundary. The origin is observable from the part of sphere that contains the $\mathbf{e}_3$. We provide a lemma that allows the convex hull computation to be performed in 2D.

Lemma 6. *Let $\mathscr{P}$ be a set of points on the unit sphere. The boundary points of $\mathscr{P}$ with respect to $\mathbf{e}_3$ are points that are adjacent vertices to $\mathbf{e}_3$ in the convex hull of $\mathscr{P} \cup \{\mathbf{e}_3\}$. Let m be a function that maps a point on the unit sphere to the plane $z = 0$ with center $\mathbf{e}_3$, i.e., $m([x,y,z]^\top) := [\frac{x}{1-z}, \frac{y}{1-z}]^\top$. A point $\mathbf{u} \in \mathscr{P}$ is a boundary point if and only if $m(\mathbf{u})$ is a vertex of the convex hull of $m(\mathscr{P})$.*

Proof. For $\mathbf{u} = [\mathbf{u}_x, \mathbf{u}_y, \mathbf{u}_z]^\top \in \mathscr{P}$, suppose that $m(\mathbf{u})$ is not a vertex of the convex hull of $m(\mathscr{P})$. Then, there exists a set of points $\{\mathbf{u}^i = [\mathbf{u}^i_x, \mathbf{u}^i_y, \mathbf{u}^i_z]^\top\}_{i=0}^n \subset \mathscr{P} \setminus \{\mathbf{u}\}$ and coefficients $\{\alpha_i\}_{i=0}^n$, $0 < \alpha_i < 1$, $\sum_{i=0}^n \alpha_i = 1$ such that $m(\mathbf{u}) = \sum_{i=0}^n \alpha_i m(\mathbf{u}^i)$. Let $\beta := \frac{1}{\sum_{i=0}^n \alpha_i \frac{1-\mathbf{u}_z}{1-\mathbf{u}^i_z}}$, and $\gamma_i = \beta \alpha_i \frac{1-\mathbf{u}_z}{1-\mathbf{u}^i_z}$, so that $\sum_{i=0}^{n} \gamma_i = 1$. Since all points are on the unit sphere, we have $\gamma_i > 0$, $\beta > 0$, and:

$$\begin{aligned}
\sum_{i=0}^{n} \gamma_i \mathbf{u}_x^i &= \beta(1-\mathbf{u}_z)\sum_{i=0}^{n} \alpha_i \frac{\mathbf{u}_x^i}{1-\mathbf{u}_z^i} \\
&= \beta(1-\mathbf{u}_z)\frac{\mathbf{u}_x}{1-\mathbf{u}_z} = \beta \mathbf{u}_x, \\
\sum_{i=0}^{n} \gamma_i \mathbf{u}_z^i &= \beta(1-\mathbf{u}_z)\sum_{i=0}^{n}(\alpha_i \frac{1-(1-\mathbf{u}_z^i)}{1-\mathbf{u}_z^i}) \\
&= \beta \sum_{i=0}^{n} \alpha_i \frac{1-\mathbf{u}_z}{1-\mathbf{u}_z^i} - \beta(1-\mathbf{u}_z)(\sum_{i=0}^{n} \alpha_i) \\
&= 1-\beta(1-\mathbf{u}_z).
\end{aligned}$$

Hence, $\sum_{i=0}^{n} \gamma_i \mathbf{u}^i = [\beta \mathbf{u}_x, \beta \mathbf{u}_y, 1-\beta(1-\mathbf{u}_z)]^\top = (1-\beta)\mathbf{e}_3 + \beta \mathbf{u}$. Note that $(1-\beta)\mathbf{e}_3 + \beta \mathbf{u}$ is on the segment from $\mathbf{e}_3$ to $\mathbf{u}$, so $0 < \beta < 1$; otherwise the convex combination of points on the unit sphere ($\sum_{i=0}^{n} \gamma_i \mathbf{u}^i$) will be out of unit sphere. This means that the segment between $\mathbf{e}_3$ and $\mathbf{u}$ intersects with the convex hull of $\mathscr{P}$ at another point $(1-\beta)\mathbf{e}_3 + \beta \mathbf{u}$, which implies that $\mathbf{u}$ is not a boundary point of $\mathscr{P}$. The converse statement can be proven similary by reversing the steps above. $\square$

Discretized Spherical Convex Hull Data Augmentation

To accelerate the convex hull computation further, we propose an approximation using discretization. We discretize the azimuth of the sphere into N segments. Let $E_i = \max\{el(\mathbf{u}) \mid \mathbf{u} \in \mathscr{P}_\mathbf{q}, az(\mathbf{u}) \in [\frac{2\pi i}{N}, \frac{2\pi(i+1)}{N})\}$, $0 \le i < N$ be the maximum elevation of points in $\mathscr{P}_\mathbf{q}$ with azimuth in $[\frac{2\pi i}{N}, \frac{2\pi(i+1)}{N})$. Then, let i determine the interval $[\frac{2\pi i}{N}, \frac{2\pi(i+1)}{N})$ that contains the azimuth of $T_\mathbf{q}(\hat{\mathbf{p}})$. We consider $\mathbf{q}$ observable from $\hat{\mathbf{p}}$ if the elevation of $T_\mathbf{q}(\hat{\mathbf{p}})$ is larger than E_i. In the experiments, we accelerate the computation further by sub-sampling $\mathscr{P}_\mathbf{q}$.

3.5 Evaluation

This section presents qualitative and quantitative evaluation of the DeepSDDF model for shape learning. Sec. 3.5.1 presents results for instance-level shape modeling in comparison to the deep geometric prior (DGP) [3] and the implicit geometric regularization (IGR) [1] models on the synthetic ShapeNet dataset [2]. Sec. 3.5.2 presents single-instance shape modeling using real data from the YCB dataset [4].

Sec. 3.5.3 and Sec. 3.5.4 apply DeepSDDF to category-level shape modeling, demonstrating shape completion from a single distance view and shape interpolation between different instances. The accuracy of DeepSDDF for shape completion is compared against IGR [1], which improves over DeepSDF [102] due to the Eikonal regularization but otherwise uses the same network. Both our model (by construction) and IGR (by loss function) capture structural constraints for SDDF and SDF, respectively, making a quantitative comparison interesting. We also compare the results against a group of category-level shape modeling methods that utilize point-cloud models without considering surface continuity or gradient properties, including AtlasNet [118], GRNet [5], PCN [120], FoldingNet [121], TopNet [122], and MSN [123]. Sec. 3.5.5 demonstrates scene-level reconstruction using real depth images from the SceneNN dataset [143] and discusses the utility of DeepSDDF for robotics applications, including accurate shape modeling for navigation or manipulation and high-speed depth and visibility prediction for active perception.

More results, exploring different aspects of our method are provided in the Appendices, while Sec. 3.6 discusses the limitations of our SDDF formulation. We emphasize that our DeepSDDF model can synthesize point clouds with arbitrary resolution from arbitrary views as efficiently as a single neural-network forward pass. The generated point clouds can be converted to a different representation, such as occupancy grid, depth images, or mesh if desired.

Additional ablation studies are available, in Sec. 3.5.6, providing training details and more shape completion and shape interpolation results, Sec. 3.5.7, investigating the effect of the decoder network size further, Sec. 3.5.8, reporting 2D results, and Sec. 3.5.9, showing category-level shape learning with few training set instances and varying levels of noise.

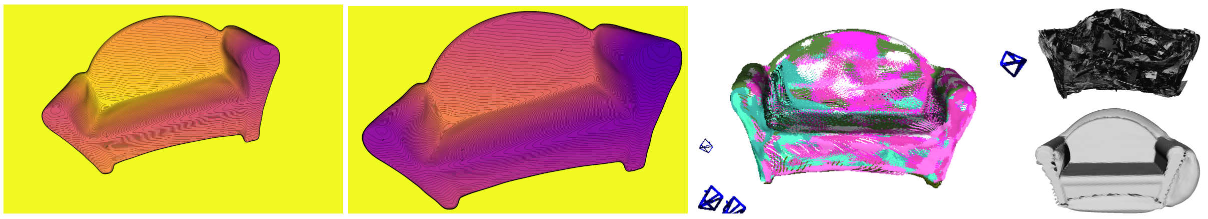


Figure 3.4. SDDF reconstruction of a sofa. The first two columns show distance images synthesized by DeepSDDF along the same view direction but different distance from the object. The third column shows point clouds synthesized from different arbitrary camera views, indicated in different colors to emphasize multi-view consistency. The fourth column shows DGP (top) and IGR (bottom) reconstructions of the same sofa.

Metrics

Our evaluation results use shape reconstruction metrics introduced by Occupancy Networks [94] based on point-cloud comparison, including Chamfer distance, Completeness, and Accuracy. For all metrics *lower* is *better* and all metrics are zero for two identical point clouds.

3.5.1 Instance-Level Shape Modeling (Synthetic Data)

We evaluate the performance of DeepSDDF for instance-level shape reconstruction using 5 object instances from different categories in the ShapeNet dataset [2].

Network Architecture

We use an autodecoder with 16 layers, 512 hidden units per layer, and a skip connection from the input to layers $4, 8, 12$ to represent the SDDF model $q_\theta(\mathbf{p}, \eta)$ introduced in Sec. 3.4.

Baseline Algorithms

We compare against DGP [3], an explicit shape model using local surface parameterizations fused in a manifold atlas, and IGR [1], an implicit shape model using decoder-only deep SDF esimation with a loss function to enforce unit-norm (Eikonal) SDF gradients. The DGP radius is set to guarantee at least 128 patches per object. DeepSDDF and IGR were trained with learning rate 0.005 that decreases every $1k$ epoch by factor of 0.5 for $10k$ epochs.

Training Details

We use truck, airplane, sofa, boat, and car instances from ShapeNet [2], normalized to a unit cube. Distance images with resolution 512×512 are generated as training data from 8 camera views facing the object from azimuth $\frac{k\pi}{4}$ and elevation $\frac{(-1)^k\pi}{4}$ for $k = 0, \ldots, 7$ on a sphere. Each distance image is subsampled to contain at most $100k$ finite and $100k$ infinite distance measurement rays. Both DGP and IGR were trained using the point-cloud obtained from all points with finite distance measurements and augmented with normals obtained using the method of [144]. Normals were not used to train DeepSDDF.

Timing

We compare the average ray-tracing speed of DeepSDDF and IGR [1] for distance prediction along a single viewing direction because efficient rendering time is important in many applications. Ray tracing for SDF models is implemented using sphere tracing. We used an open-source sphere-tracing algorithm from IDR [104], which is based on IGR. We used default algorithm parameters, except for $sphere\_tracing\_iters = 50$, which led to faster and accurate results in our experiments. Sphere-tracing time also depends on the radius of a bounding sphere enclosing the surface of interest. The test objects were normalized to a unit sphere. DeepSDDF does not require a bounding volume and, by definition of the model, ray tracing requires a *single* forward pass through the neural network model, which leads to an order of magnitude speed gain over sphere tracing. In real applications, where the bounding volume might be larger, we expect that the speed superiority of DeepSDDF will be more significant. The timing setup is equivalent in the following sections.

Results

Quantitative results are presented in Table 3.1. The shape reconstruction accuracy of DeepSDDF is comparable to DGP but, unlike DGP, our model can handle category-level shape modeling as shown in Sec. 3.5.3 and Sec. 3.5.4. DeepSDDF outperforms IGR significantly, showing that the model is generalizes well with limited training data. Visualizations of the

DeepSDF, DGP, and IGR reconstructions of a sofa instance are shown in Fig. 3.4. By inspecting the distance level sets in distance images synthesized by the DeepSDDF model, we can see that the model captures shape details accurately. The images get brighter as the view moves further away from the object because the measured distances at each pixel increase. On the other hand, the level sets in the distant view remain parallel to the close-up view. The two-view comparison shows that the directional condition of the SDDF model in (3.2) holds. The learned DeepSDDF model can generate distance predictions at arbitrary views. To visualize the learned model, we choose arbitrary camera locations facing the object and show point cloud predictions in Fig. 3.4 (third column). The different color point clouds in Fig. 3.4 are synthesized from different camera views and the fact that they align well on the object surface verifies the multi-view consistency of the learned shape model. Additionally, since the point clouds are generated from arbitrary views, DeepSDDF can synthesize novel views accurately. The results from DGP and IGR for the same instance are shown in Fig. 3.4.

3.5.2 Instance-Level Shape Modeling (Real Data)

In this section, we show that DeepSDDF can handle real data from the YCB dataset [4] directly.

Network Architecture

We use an autodecoder with 8 layers, 512 hidden units per layer, and a skip connection from the input to layer 4.

Training Details

We used depth images, segmentation masks, and camera poses for 20 object instances from YCB [4]. The masks are used to separate the (finite) rays that hit an object from the background (provided as infinite rays to our model). For each object in YCB, there are 600 views from which images are taken. The data was split randomly into a 95% training set and 5% test set. For each instance, we trained two different DeepSDDF models. The first one was trained

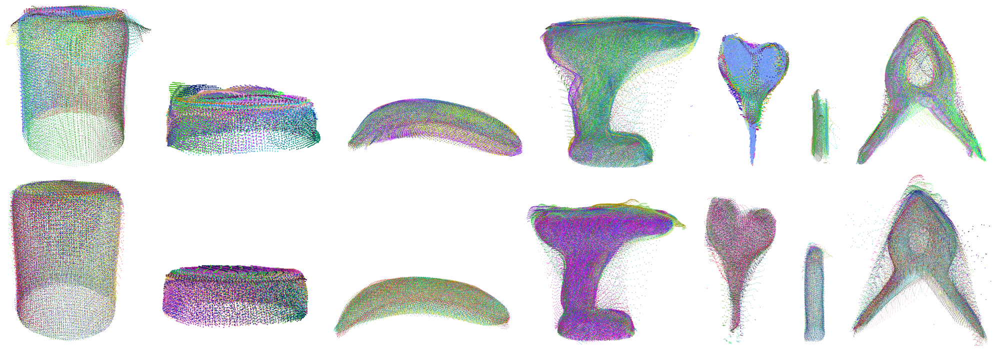


Figure 3.5. Novel-view point clouds synthesized by a DeepSDDF model trained using real data from YCB [4]. The first row shows the output of DeepSDDF trained with real data only. The second row shows the output of DeepSDDF trained with real data plus synthetic data augmentation.

only with the real data, while the second one was trained with our discretized convex hull data augmentation (see Sec. 3.4.4) with 1000 synthesized random views. Note that no new real data is used for training the second model. The data augmentation, synthesizes novel view directions based on the visibility of the real data points.

Results

Qualitative results are presented in Fig. 3.5. Quantitative results are provided in Table 3.2. This experiment shows that DeepSDDF predicts novel views (views from which the object has never been observed) accurately from real data.

3.5.3 Category-Level Shape Modeling

In this section, we explore the capability of our method to represent a whole category of object shapes.

Network Architecture

We use an autodecoder with 16 layers, 512 hidden units per layer, and a skip connection from the input to layers $4, 8, 12$ to represent the SDDF model $q_\theta(\mathbf{p}, \eta, \mathbf{z})$ introduced in Sec. 3.4 with dimension of the latent shape code $\mathbf{z}$ set to 256.

Baseline Algorithms

We compare against the SDF model IGR [1] and the point-cloud models GRNet [5], AtlasNet [118], PCN [120], FoldingNet [121], TopNet [122], and MSN [123].

Training Details

DeepSDDF and IGR were trained over 5 categories from ShapeNet [2]. For each instance, we collect data from 8 views similar to Sec. 3.5.1. DeepSDDF and IGR were trained for 1000 epochs, with 2000 random samples, and with learning rate 0.0005, 0.0001 for the network weights and latent code weights, respectively. The learning rate decreases by factor of 2 every 500 epochs for IGR (as suggested in [1]) and every 200 epochs for DeepSDDF. We did not retrain the point-cloud synthesis models, and used the results reported in [5]. To make the comparison to the point-cloud synthesis models as fair as possible, DeepSDDF was trained on the same ShapeNet categories with the same train/test splits and was evaluated at view rays that collide with the points used for testing of the baseline methods. Additional training details and results are provided in the Sec. 3.5.6.

Shape Completion
Results

The shape reconstruction accuracy and timing of the DeepSDDF and IGR models are presented in Table 3.3. The results show that our model is an order of magnitude more accurate than IGR with limited training data, despite training IGR with normals and encoding additional directional information in SDDF. This can be explained by the fact that IGR encourages the satisfaction of the SDF gradient property [1] through regularization, while DeepSDDF encodes the SDDF gradient property (Lemma 3.2) by construction, making DeepSDDF more data efficient. In a separate experiment in Appendix 3.5.9, we show that DeepSDDF, while capturing both position and orientation geometry, can learn shape space from few instances (see airplane in Fig. 3.7). Comparing the results of IGR(1) and IGR(2), in which point clouds are extracted after mesh reconstruction, with IGR(3), in which point clouds are obtained through sphere tracing,

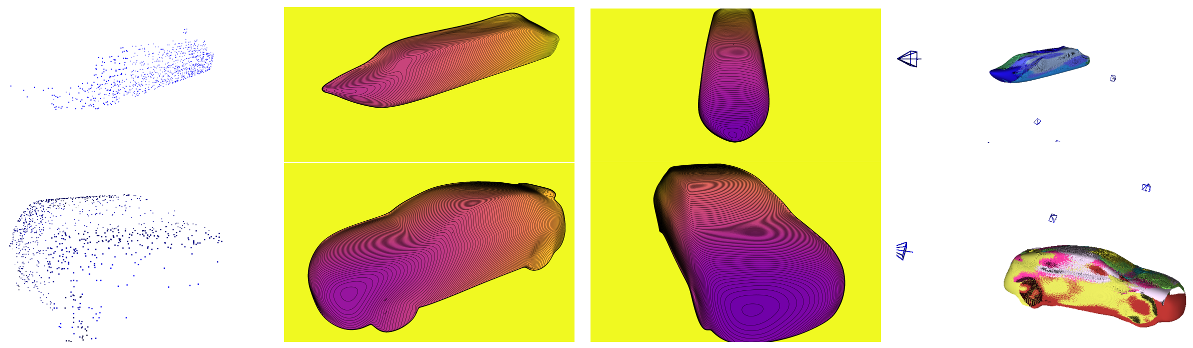


Figure 3.6. SDDF shape completion using distance measurements (first column) from unseen boat (first row) and car (second row) instances. The trained DeepSDDF model can synthesize novel distance views (second and third columns) or point clouds from arbitrary camera views which are multi-view consistent (fourth column). In the last column, the point cloud from each arbitrary view is shown with random color.

shows that the accuracy of deep SDF estimation may deteriorate with distance from the surface. The results also show that DeepSDDF is an order of magnitude faster than IGR for distance queries because DeepSDDF requires a single forward pass through the network, while IGR performs iterative sphere tracing.

We compare DeepSDDF versus the point-cloud reconstruction baselines in Table 3.4. The reconstructed objects are not normalized to a unit-length bounding box in these experiments since the baseline methods did not do this. These methods, despite DeepSDDF, do not challenge with modeling continuous representation or incorporating a geometric property by construction. Still, we see that DeepSDDF performs close to these methods (or even sometimes better).

Shape Interpolation

Finally, we demonstrate that DeepSDDF is able to capture the latent shape space of an object category continuously and meaningfully. Fig. 3.7 presents results for linear interpolation between the latent shape codes of two object instances from the same category. The intermediate shapes do not exist in the training set and are imagined by the DeepSDDF decoder based on the interpolated latent code $\mathbf{z}$.

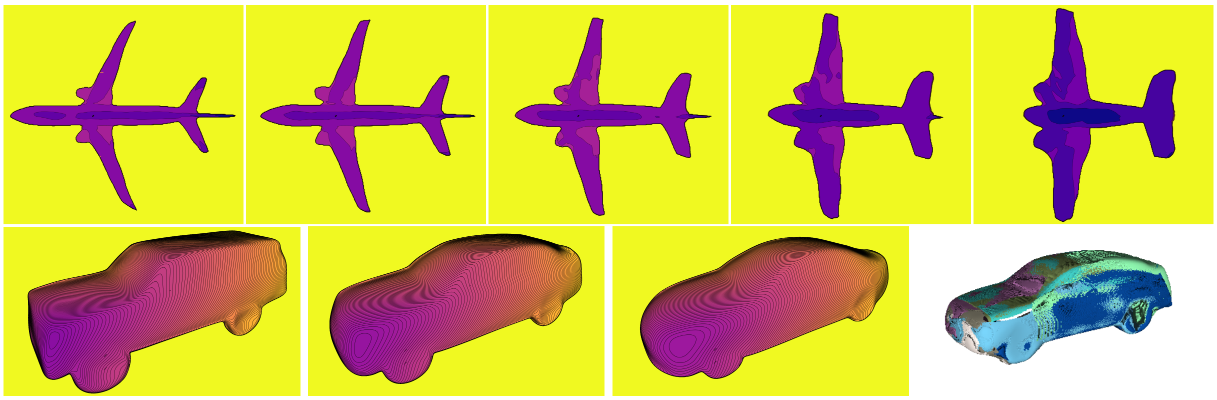


Figure 3.7. SDDF shape interpolation between two instances. The left-most and right-most column in the first row show the DeepSDDF output from the same view for two different airplane instances from the training set. The three columns in the middle are generated by using a weighted average of the latent codes of the left-most and right-most instances as an input to the DeepSDDF network. In each row, from left to right, the latent code weights with respect to the left-most instance are 1, 0.75, 0.5, 0.25, 0, respectively. Note how the shapes transform smoothly from the left-most to the right-most instance with intermediate shapes looking like valid airplanes. The second row shows interpolation (in column 2) between two learned car shapes (in columns 1, 3). The last column shows the point cloud reconstructions of the interpolated car instance from several different views (indicated in different colors to emphasize the multi-view consistency of the model).

3.5.4 Ablation Studies

In this section, we explore the effect of the DeepSDDF decoder depth (16 vs 8 layers) for modeling challenging ShapeNet [2] categories with delicate structure, such as Lamp and Bench.

Network Architecture

We use two autodecoders, one with 16 layers, 512 hidden units per layer, and a skip connection from the input to layers $4, 8, 12$, and another one with 8 layers, 512 hidden units, and a skip connection from input to layer 4 (similar to the IGR network structure) to represent the SDDF model $q_\theta(\mathbf{p}, \eta, \mathbf{z})$ introduced in Sec. 3.4 with latent shape code dimension set to 256.

Baseline Algorithms

We compare against IGR [1], which uses an autodecoder with 8 layers, 512 hidden units, and a skip connection from input to layer 4.

Training Details

The models were trained on the Lamp and Bench categories from ShapeNet [2]. For each category, we randomly picked 80% of the instances for training and the rest for testing. Each instance was normalized to a unit sphere and 500 camera views uniformly distributed on a sphere facing the object were used to collect training data. For each view, distance images with resolution 256×256 are generated and randomly subsampled such that are at most $2000k$ finite and $2000k$ infinite rays per instance. The training parameters, such as learning rates, were set the same way as in Sec. 3.5.3. No synthetic data augmentation was used for DeepSDDF.

Results

The results are provided in Table. 3.5. The shape reconstruction accuracy of DeepSDDF is still better than IGR, although the training data in this experiments is significantly more than the experiment in the previous section. The accuracy of the 16-layer DeepSDDF model is slightly better than the 8-layer model. However, the two results are very close, showing that the DeepSDDF model is able to accurately capture both position and orientation geometry with an 8-layer decoder.

3.5.5 Scene-Level Reconstruction and Robotics Applications

In this section, we apply DeepSDDF to learn part of a scene using real depth images from the SceneNN dataset [143]. This experiment shows that a single DeepSDDF network is capable of modeling multiple objects and that our data augmentation technique (Appendix 3.4.4) is effective for real data captured from a small set of views. Once the SDDF model is trained, it can be used to perform visibility and visible volume queries very efficiently (as a single SDDF network forward pass). Fast assessment of the visible volume of an environment from novel views is a key component in the next-best-view problem [145] and other active perception problems [146] in robotics.

Network Architecture

The DeepSDDF network structure is the same as the one in Sec. 3.5.1.

Training Details

We use sequence 255 from the SceneNN dataset [143]. We use 38 depth images chosen every 50 steps starting from image 0. For each image, we randomly choose 20k colliding rays and 20k non-colliding rays, effectively subsampling the depth image resolution. Since the real training data is limited, we use our discretized spherical convex hull data augmentation approach in Sec. 3.4.4 to synthesize additional training data. The same training procedure as in Sec. 3.5.1 is used.

Scene Reconstruction and Visibility Queries

An RGBD sample and a reconstruction of the 3D scene in SceneNN sequence 255 are shown in Fig. 3.8. Additionally, the SDDF function in a horizontal plane cutting the scene is shown. To illustrate the utility of an SDDF model in active perception applications, we consider two arbitrary poses (shown in Fig. 3.8) and evaluate the visible volume from their views. We approximate the visible volume by considering the volume corresponding to each pixel as a pyramid. If the dimensions of pixel at depth 1 are w, h, then the volume corresponding with a ray with depth d is $\frac{whd^3}{3}$. The sum of these volumes for all collided rays predicted by DeepSDDF provide the visible volume. As the resolution of the camera increases the accuracy of the visible volume prediction increases. For example for the yellow camera in Fig. 3.8 with resolution 480×640 the visible volume is 0.307, computed by the DeepSDDF model. With resolution 240×320, the computed visible volume is 0.305.

Similarly, the visibility of a query point $\mathbf{q}$ of interest may be evaluated as single DeepSDDF pass. A visibility query requires checking whether $\mathbf{q}$ is visible from a camera with position $\mathbf{p}$, i.e., checking $\|\mathbf{p}-\mathbf{q}\| > h(\mathbf{p}, \frac{\mathbf{q}-\mathbf{p}}{\|\mathbf{q}-\mathbf{p}\|})$, where h is SDDF. For example, the red point in Fig. 3.8 is visible from the yellow camera but it is not visible from the purple camera, which can be verified in less than a millisecond.

Other Robotics Applications

This paper introduced a new SDDF shape representation and an associated neural network model for learning SDDF in both instance- and category-level. We envision that this model may be beneficial for robotics applications, including manipulation, safe navigation, and unknown environment exploration. For example, the ability of DeepSDDF for accurate shape modeling of delicate objects may be useful in dexterous robot manipulation scenarios. The ability of DeepSDDF for shape completion from a single point cloud observations may be useful for geometry prediction in autonomous navigation and exploration of unknown environments. Our real data experiments show that DeepSDDF can be trained directly with depth or LiDAR data coming from the onboard sensors of a mobile robot. Finally, many robotics applications require real-time operation in partially known environments. Often, ray-tracing for collision or visibility prediction is a bottleneck, which may be alleviated by the ability of DeepSDDF to perform ray-tracing as a single forward pass with timing and accuracy independent of the distance to the observed surface.

3.5.6 Network Architecture and Training Details

We present additional details about the network architecture for $q_\theta(\mathbf{p}, \eta, \mathbf{z})$ and the training procedure. We use a soft-plus activation function $\frac{1}{\beta}\ln(1+\exp(\beta x))$ with $\beta = 100$. The inputs are positions $\mathbf{p} \in \mathbb{R}^3$, view direction $\eta \in S^2$, and latent code $\mathbf{z} \in \mathbb{R}^{256}$. The third component of η (c in Lemma 3) may be very close to -1. To avoid numerical problems, we let $\eta = [\sin(\theta)\cos(\phi), \sin(\theta)\sin(\phi), \cos(\theta)]^\top$. Using $s_\theta := \sin(\theta)$, $c_\theta := \cos(\theta)$, $s_\phi := \sin(\phi)$, $c_\phi := \cos(\phi)$, and applying the fact that $\frac{{s_\theta}^2}{1+c_\theta} = \frac{1-{c_\theta}^2}{1+c_\theta} = 1 - c_\theta$, the rotation matrix $\mathbf{R}_\eta$ in Lemma 4 used to map η to the standard basis vector $\mathbf{e}_3$ becomes:

$$\mathbf{R}_\eta = \begin{bmatrix} 1-(1-c_\theta)c_\phi^2 & -(1-c_\theta)s_\phi c_\phi & -s_\theta c_\phi \\ -(1-c_\theta)s_\phi c_\phi & 1-(1-c_\theta)s_\phi^2 & -s_\theta s_\phi \\ s_\theta c_\phi & s_\theta s_\phi & c_\theta \end{bmatrix}.$$

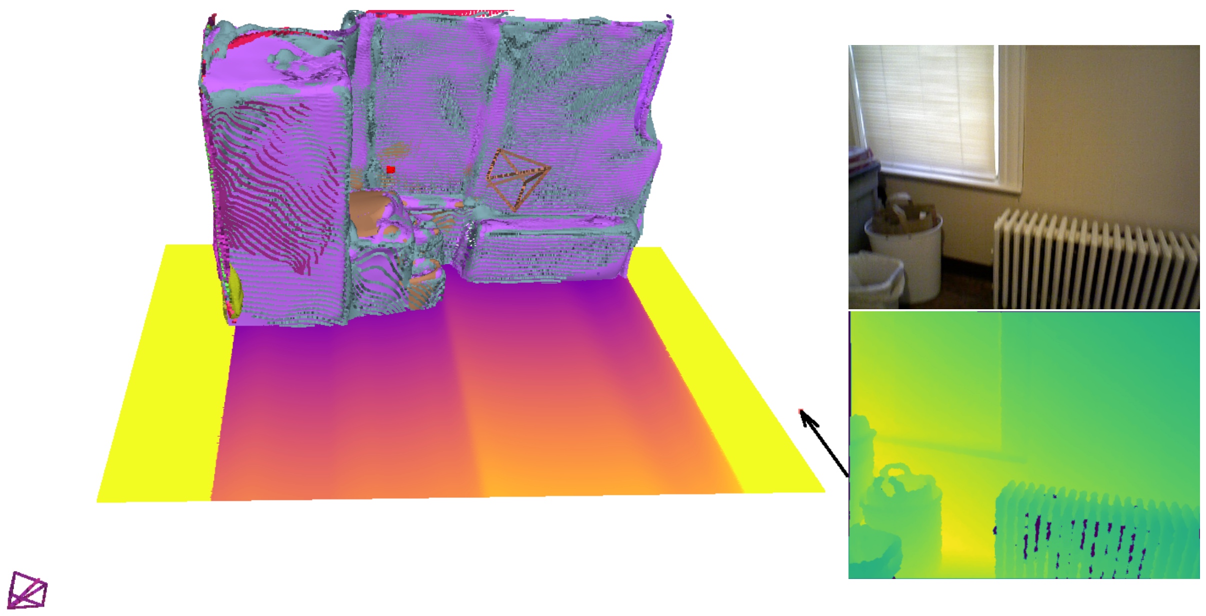


Figure 3.8. Scene-level SDDF reconstruction using 38 real depth images from the SceneNN dataset [143]. A sample RGBD image is shown on the right. A point cloud surface reconstruction from 8 test views, two of which are shown as a purple camera frame (left) and yellow camera frame (close to the surface). The ground plane shows the SDDF value at each point in the viewing direction indicated by the black arrow with yellow color indicating infinite values (truncated by some max value) and red-to-blue transition indicating large to small signed distance. The red point on the surface is a sample point which is visible from the yellow camera but not from the purple camera. The SDDF model can handle such visibility queries very efficiently (as a single forward pass through the neural network model).

Using the dimension reduction in Proposition 5, the final network inputs are $\mathbf{P}\mathbf{R}_\eta \mathbf{p} \in \mathbb{R}^2$, $\eta \in S^2$, and $\mathbf{z} \in \mathbb{R}^{256}$.

All experiments are done on a single GTX 1080 Ti GPU with the PyTorch deep learning framework [147] and the ADAM optimizer [148]. For single shape estimation, the network is trained with initial learning rate of 0.005, decreasing by a factor of 2 every 1000 steps for $10k$ iterations. In each iteration, we pick a batch of $100k$ samples randomly from the synthesized training data and use $\alpha = 1$, $\beta = 0.5$, $\gamma = 0$, $p = 1$, $r(x) = \max\{0,x\}$ in the error function in (3.11). For category-level shape estimation, the network is trained with initial learning rate of 0.0005 for the network parameters θ and 0.0001 for latent code $\mathbf{z}$, both decreasing by a factor of 2 every 200 steps for $1k$ iterations. In each iteration, for each object we pick a batch of $2k$

samples randomly from the union of the synthesized and original samples. We use $\alpha = 1$, $\beta = 1$, $p = 1$, $r(x) = \max\{0, x\}$, $\sigma = 0.001$ and Euclidean norm regularization $||\mathbf{z}||_2^2$ for the latent shape code in the error function (3.13).

To train the IGR network [1], for each object we picked a batch of $2k$ samples randomly from the original point cloud and augmented them with normals. Note that for our method we pick a total of $2k$ samples for each object, including the original data and synthesized data as well as finite rays and infinite rays. We use the default training parameters for IGR as provided in the open-source implementation [1]. The only parameter we adjusted was the initial learning rate because there was a discrepancy between the open source code and the IGR paper. We chose the setting that provided better results, namely initial learning rate of 0.0005 for network parameters θ and 0.0001 for latent code $\mathbf{z}$, both decreasing by a factor of 2 every 500 steps for $1k$ iterations.

We use the default training parameters for the DGP network [3], except increasing the upsamples-per-patch parameter from 8 to 20 and adjusting the radius parameter to guarantee at least 128 patches for each object.

To produce training data for the category-level experiments in Sec. 3.5.3, we normalize each instance to a unit box and generate 8 distance images with resolution 512×512 using PyRenderer [149]. Each distance image is subsampled to have at most 12500 infinite rays and 12500 finite rays. Hence, there are at most $100k$ finite and $100k$ infinite rays for each object. To accelerate the data augmentation procedure, we further subsample the point cloud produced by the finite rays in each view from 12500 to 1250 points, when computing the convex hull approximation in Sec. 3.4.4. This provides a subsampled point cloud across all views with at most $10k$ points, which was further subsampled to $2k$ points using the diversipy python package [150, 151]. To generate synthetic data, we choose $1k$ random azimuth and elevation views on a sphere around the point cloud. Infinite rays are obtained by projecting the original point cloud (with $100k$ points) to an image plane with resolution 128×128 for each imaginary camera and selecting the unoccupied pixel directions. Finite rays are generated using the procedure described in Sec. 3.4.4. In Sec. 3.5.4, to make the optimal situation for IDR sphere-tracing, we normalize each instance to a unit sphere and generate 500, 256×256 distance images with random views, then we randomly pick rays such that at most we have at most $2000k$ finite and $2000k$ infinite rays for each object. Then directly use this data for training the DeepSDDF (without applying the data augmentation technique). Additional qualitative results for shape completion are presented in Fig. 3.9 and for shape interpolation in Fig. 3.10, Fig. 3.11, Fig. 3.12, Fig. 3.13, Fig. 3.14, Fig. 3.15.

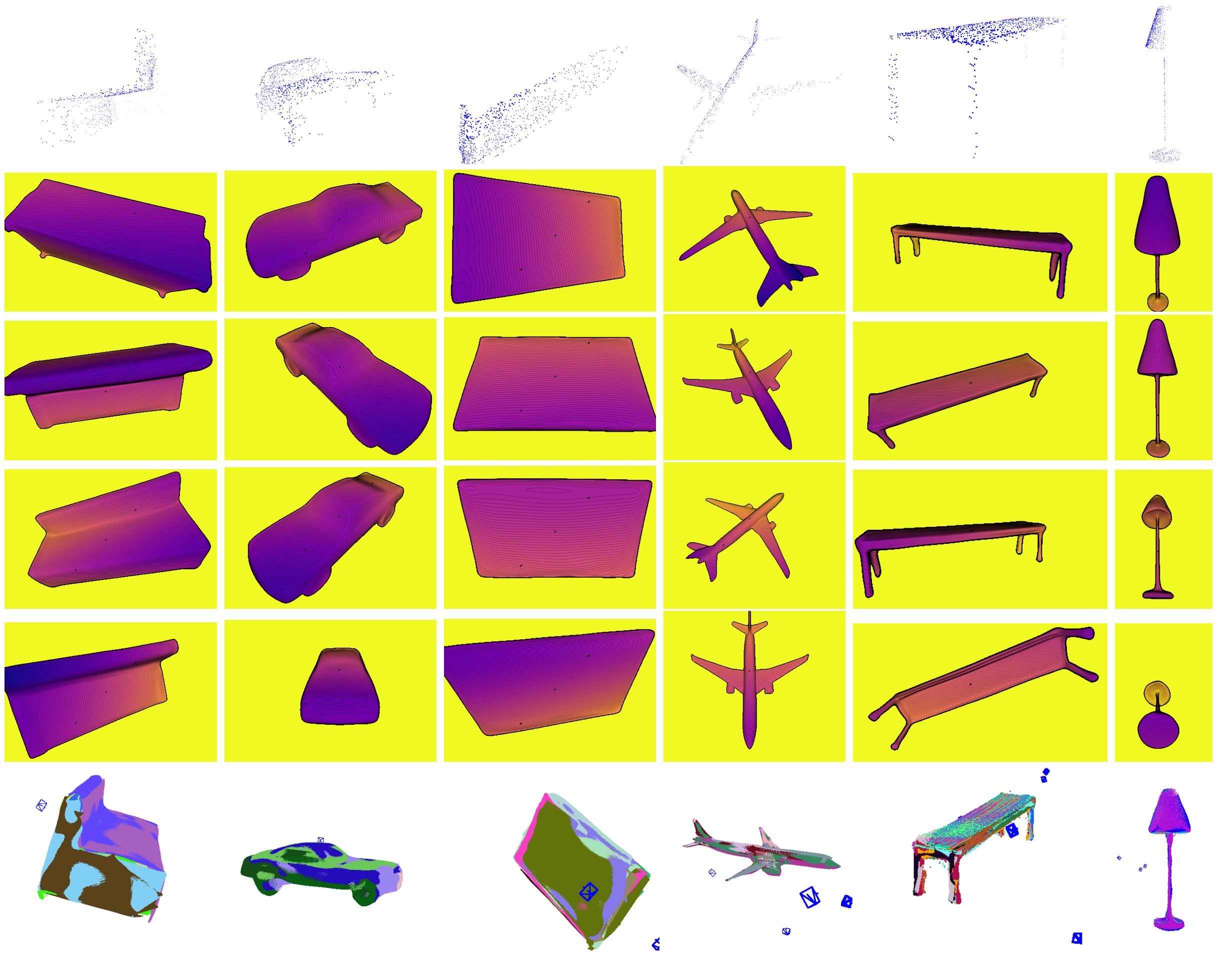


Figure 3.9. SDDF shape completion using $1k$ finite and $1k$ infinite rays from a single distance view and the corresponding point cloud (upper row) from an unseen object instance. After latent code optimization, the DeepSDDF model can synthesize novel distance views (the four middle rows), and novel point clouds from arbitrary views (last row). In the last row, the point cloud reconstructed from each arbitrary view is shown with different color.

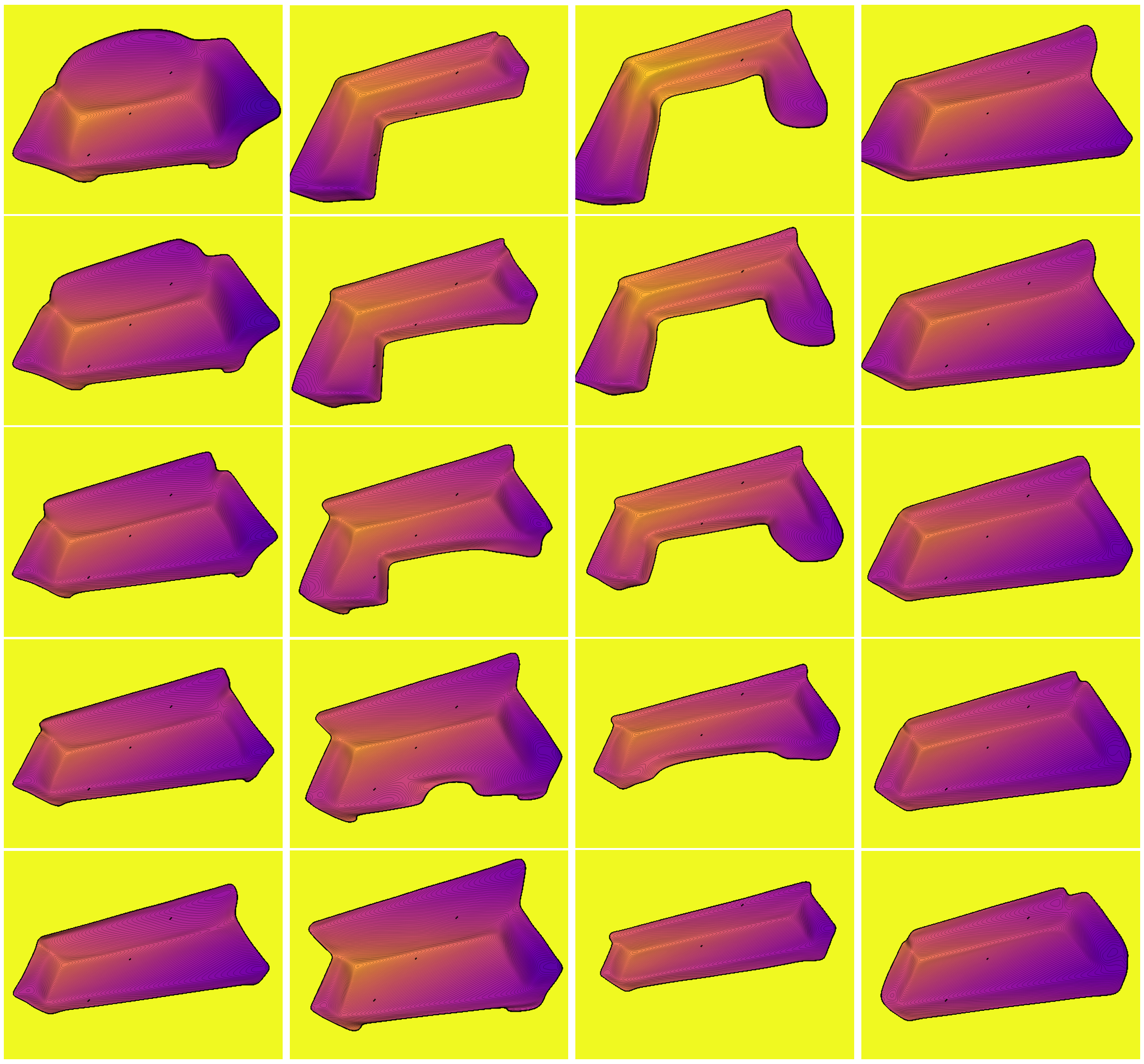


Figure 3.10. SDDF shape interpolation between two sofa instances. The first and last row show the SDDF output from the same view for two different instances from the training set. The rows in the middle are generated by using a weighted average of the latent codes of the upper-most and down-most instances as an input to the DeepSDDF network. In each column, from top to bottom, the latent code weights with respect to the upper-most instance are 1, 0.75, 0.5, 0.25, 0, respectively. Note how the shapes transform smoothly from top to bottom with intermediate shapes looking like valid sofas. This demonstrates that the DeepSDDF model represents the latent shape space continuously and meaningfully.

3.5.7 Effect of the Network Size on the Performance

This section evaluates the effect of the number of layers and number of neurons per layer in the DeepSDDF model on the performance qualitatively and quantitatively. The results are obtained using a single sofa instance, shown in Fig. 3.4 in main paper. We use the same settings as the single-object experiment in Sec. 3.5.1 with data augmentation from 10*k* random views. The default model in the paper has 16 layers with 512 neurons per layer with skip connections every 4 layers.

First, we varied the number of layers in the neural network, while keeping a skip connection every 4 layers. We evaluated the DeepSDDF model with 4 (with out skip connection), 8, 12, 16 layers. Quantitatively, as seen in Fig. 3.16, at first the error decreases significantly and then it increases slowly. Qualitatively, in Fig. 3.17, with more layers the model can capture finer details about the shape but even with 8 layers the shape is reconstructed very well.

Second, in the default setup with 16 layers, we kept an equal number of neurons per layer but varied the number as 8, 32, 128, 512. As we see in Fig. 3.16, at first the error decreases significantly and then continues to decrease slowly. In Fig. 3.18, we see that with fewer neurons per layer the model cannot capture the shape details very well. In comparison to the changing number of layers experiment, we see that the model is qualitatively more sensitive to the number of neurons.

3.5.8 2D Evaluation

This section shows that an DeepSDDF model can be used in 2D, e.g., with distance measurements obtained from a LiDAR scanner. We simulated a Hokuyo UTM-30LX Lidar scanner with 1081 rays per scan moving along a manually specified trajectory in an environment containing a 2D shape. See Fig. 3.19 for an example. The Lidar scans were used as training data for the DeepSDDF and IGR models. After training, the DeepSDDF network can generate distance values to the object contours at any location and in any viewing direction. Fig. 3.19 visualizes the models learned by IGR and DeepSDDF for a heart-shaped 2D object. The distance predictions of the SDDF model are shown at every 2D location for several fixed viewing directions. We see that the DeepSDDF model recognizes the boundary between free space and the object well. The parallel distance level sets indicate that the condition in Lemma 2 in the main paper indeed holds everywhere.

3.5.9 Small Training Set without Synthesized Data

The larger dataset, in number of objects, we have the better results we get. In this section, we study the performance of the DeepSDDF model when only a small training set of objects is available and the data augmentation technique, described in Appendix 3.4.4 is not used. We show that even in this case DeepSDDF can learn the latent space continuously and meaningfully. The results are generated using 100 car and 200 airplane instances from the ShapeNet dataset [2]. To generate training data, we use the functions in PyTorch3D [152] for ray casting. For each object, we choose 1000 random locations uniformly distributed on a sphere with orientations facing the object. Each distance image was down-sampled to have at most 5000 finite rays and 5000 infinite rays. An DeepSDDF model with 8 fully connected layers, 512 hidden units per layer, and a skip connection from the input to the middle layer is used. All experiments are done on a single GTX 1080 Ti GPU using PyTorch [147] and the ADAM optimizer [148] with initial learning rate of 0.005.

Single Instance Shape Representation

For single-instance shape estimation, we schedule the learning rate to decrease by a factor of 2 every 500 steps for 9k iterations. In each iteration, we pick a batch of 128^2 samples randomly from the input data. The result is provided in Fig. 3.20.

Shape Completion and Interpolation

For Car category-level shape estimation, the network is trained for 500 epochs with 100 instances. The learning rate is scheduled to decrease by a factor of 2 every 50 epochs. In each iteration, for each shape, 8000 random samples are picked uniformly out of the training data for that instance. For the Airplane category, the network is trained for 5000 epochs with 200 shapes. The learning rate is scheduled to decrease by a factor of 2 every 250 epochs. In each iteration, for each instance 10k random samples are picked uniformly. Shape completion results are provided in Fig. 3.21, while shape interpolation results are provided in Fig. 3.22. These show the latent space are learnt continuously and meaningfully.

Effect of Measurement Noise on the Performance

We present qualitative and quantitative results about the effects of noisy distance data and different number of layers and neurons per layer in the model on the performance of the DeepSDDF model. The results are obtained for a single Airplane instance, shown in Fig. 3.20. We obtained 500 finite rays and 500 infinite rays from 1000 random locations uniformly distributed on a sphere around the instance with orientations facing the object. The DeepSDDF model was trained for 9k iterations in several different settings. The default setting has 8 layers with 512 neurons per layer and noise-free distance data for training. Training this model takes about 688 seconds. A distance view synthesized by the trained DeepSDDF model is shown in Fig. 3.24 (the first image).

First, keeping the network structure fixed, we varied the standard deviation of zero-mean Gaussian noise added to the distance measurements. To have a sense about the noise magnitude, note that the radius of the sphere on which the camera locations were picked was 0.6. Qualitatively, as we see in Fig. 3.24, the more the noise increases, the fewer details the DeepSDDF model can capture. Second, we varied the number of layers (fixing the number of neurons to 512) and the number of neurons (fixing the number of layers to 8) in the neural network, keeping a skip connection to the middle layer, to measure the training time, as shown in Fig. 3.23.

3.6 Limitations

Compared to SDF, SDDF has a higher dimensional input, including both position $\mathbf{p}$ and viewing direction η. In Sec. 3.3, we observed that any valid SDDF satisfies a gradient condition (Lemma 2) that allowed us to reduce the required input dimension by one. Nonetheless, training an SDDF model requires more data than an SDF model because it needs to capture a variety of viewing directions. When an SDDF model is trained with real data, it may not be possible to obtain data with sufficient viewing direction variability. To address this limitation, we developed a data augmentation technique (Sec. 3.4.4) that generates more viewing directions from which the points in a given point cloud are observable. The method does not generate new data but rather provides additional valid viewing directions, which help ensure multi-view consistency of the SDDF model. The experiments in Sec. 3.5.1 and Sec. 3.5.3 show that an SDDF model can be trained with real subsampled depth images from only 8 views. Data augmentation alone may not completely solve the problem when there is limited training data. Since SDDF models geometry rather than appearance (e.g., in contrast with NeRF [106]), an SDDF model can be pre-trained in simulation with depth images or LiDAR scans from a variety of viewing directions, followed by fine-tuning with a smaller set of real data without significant accuracy loss. Another limitation of our formulation is that, although it can model instance-level and category-level shape, it is limited to individual objects rather than complete scenes. This means that the current formulation assumes that the observer is collecting distance views from outside of the observed surface and there are no other surfaces behind the observer. Our future work will focus on extending the SDDF formulation to scene-level modeling that allows arbitrary surfaces in front of and behind the observer as well as online training from streaming sensor data.

3.7 Discussion

This work proposed a signed directional distance function (SDDF) as an implicit shape representation of object shape. Any valid SDDF was shown to satisfy a gradient condition, which should be respected by neural network approximations. We designed DeepSDDF, a neural network model for SDDF estimation that that guarantees the gradient condition by construction. Our model enables direct supervision from depth camera or LiDAR sensors and efficient ray-tracing as a single network forward pass. We introduced a data augmentation technique to ensure multi-view consistency of DeepSDDF in limited training data regimes. DeepSDDF showed superior shape reconstruction accuracy and rendering time in various instance-level and category-level shape learning experiments. SDDF is a promising representation for robotics applications requiring accurate continuous shape and surface modeling and efficient visibility prediction.

Chapter 3, in full, is a reprint of the material as it appears in ”Ehsan Zobeidi and Nikolay Atanasov. A deep signed directional distance function for object shape representation. arXiv preprint arXiv:2107.11024 (2021)”. The dissertation author was the primary investigator and author of this paper.

Table 3.1. Comparison between DGP [3], IGR [1], and DeepSDDF with exact (e) and discretized (d) spherical convex hull data augmentation (Appendix 3.4.4) on 5 object instances from ShapeNet [2] using metrics from [94] with meshes normalized to unit-length bounding box. Three versions of IGR are evaluated: IGR(1), using the same test points as DeepSDDF, IGR(2), producing a uniform point cloud from the reconstructed mesh, and IGR(3) using IDR [104] sphere-tracing.

Class	Method	Chamfer-L_2	Chamfer-L_1	Completeness	Accuracy	Time (sec.)
Truck	DeepSDDF (e)	5.186e-05	4.776e-03	4.616e-03	4.935e-03	1.407e-06
	DeepSDDF (d)	5.727e-05	4.857e-03	4.705e-03	5.008e-03	1.495e-06
	IGR(1)	1.902e-02	8.947e-02	1.670e-02	1.622e-01	–
	IGR(2)	1.259e-02	5.961e-02	6.026e-03	1.132e-01	–
	IGR(3)	1.948e-02	9.345e-02	2.504e-02	1.618e-01	1.551e-05
	DGP	8.842e-05	7.388e-03	6.842e-03	7.935e-03	–
Airplane	DeepSDDF (e)	2.085e-05	3.468e-03	3.016e-03	3.919e-03	1.442e-06
	DeepSDDF (d)	3.217e-05	3.824e-03	3.055e-03	4.593e-03	1.474e-06
	IGR(1)	1.951e-02	9.124e-02	1.769e-02	1.648e-01	–
	IGR(2)	2.422e-02	8.450e-02	4.305e-03	1.646-01	–
	IGR(3)	1.965e-02	9.304e-02	2.130e-02	1.647e-01	1.552e-05
	DGP	4.900e-05	5.141e-03	4.512e-03	5.770e-03	–
Sofa	DeepSDDF (e)	2.692e-05	3.837e-03	3.354e-03	4.321e-03	1.494e-06
	DeepSDDF (d)	3.883e-05	3.941e-03	3.332e-03	4.549e-03	1.413e-06
	IGR(1)	1.054e-02	6.361e-02	2.905e-02	9.816e-02	–
	IGR(2)	9.831e-03	4.603e-02	4.488e-03	8.757e-02	–
	IGR(3)	1.142e-02	6.904e-02	4.030e-02	9.778e-02	1.500e-05
	DGP	1.882e-04	1.074e-02	9.546e-03	1.194e-02	–
Boat	DeepSDDF (e)	1.318e-05	3.304e-03	3.074e-03	3.533e-03	1.390e-06
	DeepSDDF (d)	1.421e-05	3.311e-03	3.083e-03	3.539e-03	1.425e-06
	IGR(1)	3.053e-02	1.140e-01	2.147e-02	2.066e-01	–
	IGR(2)	2.198e-02	8.078e-02	3.281e-03	1.582e-01	–
	IGR(3)	3.138e-02	1.190e-01	3.009e-02	2.079e-01	1.358e-05
	DGP	2.399e-05	4.015e-03	3.848e-03	4.182e-03	–
Car	DeepSDDF (e)	4.543e-05	4.371e-03	4.469e-03	4.273e-03	1.432e-06
	DeepSDDF (d)	4.315e-05	4.223e-03	4.276e-03	4.170e-03	1.442e-06
	IGR(1)	3.521e-02	1.083e-01	2.000e-02	1.966e-01	–
	IGR(2)	1.914e-01	2.679e-01	8.813e-03	5.271e-01	–
	IGR(3)	1.068e-01	2.044e-01	3.623e-02	3.727e-01	1.389e-05
	DGP	5.988e-05	5.927e-03	5.809e-03	6.046e-03	–
Avg.	DeepSDDF (e)	3.164e-05	3.951e-03	3.705e-03	4.196e-03	1.433e-06
	DeepSDDF (d)	3.712e-05	4.031e-03	3.690e-03	4.371e-03	1.449e-06
	IGR(1)	2.296e-02	9.332e-02	2.0982e-02	1.656e-01	–
	IGR(2)	5.200e-02	1.077e-01	5.382e-03	2.101 e-01	–
	IGR(3)	3.774e-02	1.157e-01	3.059e-02	2.009e-01	1.470e-05
	DGP	8.189e-05	6.642e-03	6.111e-03	7.174e-03	–

Table 3.2. Shape reconstruction accuracy of an 8-layer DeepSDDF model trained with real data from YCB [4] on 20 object instances. DeepSDDF(1) is trained with real data only, while DeepSDDF(2) uses discretized spherical convex hull data augmentation.

Class	Method	Chamfer-L_2	Chamfer-L_1	Completeness	Accuracy	Time (sec.)
002_master_chef_can	DeepSDDF(1)	3.191e-06	1.002e-03	6.775e-04	1.327e-03	5.902e-07
	DeepSDDF(2)	1.259e-06	8.834e-04	7.882e-04	9.786e-04	6.277e-07
003_cracker_box	DeepSDDF(1)	1.991e-05	1.982e-03	6.615e-04	3.302e-03	5.786e-07
	DeepSDDF(2)	6.264e-06	1.394e-03	7.997e-04	1.990e-03	5.861e-07
004_sugar_box	DeepSDDF(1)	2.501e-06	1.019e-03	7.455e-04	1.293e-03	5.779e-07
	DeepSDDF(2)	1.319e-06	8.673e-04	7.171e-04	1.017e-03	5.824e-07
005_tomato_soup_can	DeepSDDF(1)	2.260e-06	9.234e-04	6.396e-04	1.207e-03	5.813e-07
	DeepSDDF(2)	1.538e-06	9.605e-04	8.877e-04	1.033e-03	6.124e-07
006_mustard_bottle	DeepSDDF(1)	5.281e-06	1.489e-03	8.673e-04	2.111e-03	5.811e-07
	DeepSDDF(2)	4.072e-06	1.303e-03	8.522e-04	1.754e-03	5.989e-07
007_tuna_fish_can	DeepSDDF(1)	1.267e-06	7.619e-04	5.966e-04	9.271e-04	5.827e-07
	DeepSDDF(2)	1.402e-06	9.122e-04	1.027e-03	7.965e-04	5.960e-07
008_pudding_box	DeepSDDF(1)	1.725e-06	9.583e-04	7.523e-04	1.164e-03	5.842e-07
	DeepSDDF(2)	1.895e-06	1.000e-03	8.004e-04	1.199e-03	6.056e-07
009_gelatin_box	DeepSDDF(1)	1.495e-06	8.888e-04	6.974e-04	1.080e-03	5.831e-07
	DeepSDDF(2)	1.461e-06	9.553e-04	8.321e-04	1.078e-03	5.960e-07
010_potted_meat_can	DeepSDDF(1)	1.571e-06	8.316e-04	6.073e-04	1.055e-03	5.784e-07
	DeepSDDF(2)	1.271e-06	8.465e-04	6.478e-04	1.045e-03	5.960e-07
011_banana	DeepSDDF(1)	1.255e-06	8.172e-04	6.301e-04	1.004e-03	5.815e-07
	DeepSDDF(2)	1.273e-06	8.538e-04	7.924e-04	9.152e-04	5.976e-07
019_pitcher_base	DeepSDDF(1)	5.578e-06	1.254e-03	6.580e-04	1.850e-03	5.812e-07
	DeepSDDF(2)	1.194e-05	1.125e-03	7.268e-04	1.523e-03	5.900e-07
021_bleach_cleanser	DeepSDDF(1)	4.454e-06	1.378e-03	8.556e-04	1.901e-03	5.848e-07
	DeepSDDF(2)	3.053e-06	1.209e-03	9.064e-04	1.513e-03	5.937e-07
024_bowl	DeepSDDF(1)	2.281e-06	8.160e-04	6.790e-04	9.530e-04	5.839e-07
	DeepSDDF(2)	3.270e-06	1.056e-03	1.028e-03	1.085e-03	5.870e-07
035_power_drill	DeepSDDF(1)	4.755e-06	1.082e-03	6.103e-04	1.554e-03	5.825e-07
	DeepSDDF(2)	5.167e-06	1.193e-03	7.235e-04	1.663e-03	5.962e-07
036_wood_block	DeepSDDF(1)	1.255e-05	1.462e-03	6.209e-04	2.304e-03	5.823e-07
	DeepSDDF(2)	3.950e-06	9.273e-04	6.022e-04	1.252e-03	5.972e-07
037_scissors	DeepSDDF(1)	3.566e-05	1.785e-03	7.537e-04	2.817e-03	5.904e-07
	DeepSDDF(2)	9.896e-06	1.881e-03	8.948e-04	2.868e-03	6.110e-07
040_large_marker	DeepSDDF(1)	4.923e-05	3.413e-03	5.016e-04	6.325e-03	5.812e-07
	DeepSDDF(2)	3.365e-05	3.754e-03	5.945e-04	6.913e-03	6.036e-07
051_large_clamp	DeepSDDF(1)	6.143e-05	1.767e-03	7.469e-04	2.787e-03	5.862e-07
	DeepSDDF(2)	7.151e-06	1.719e-03	1.042e-03	2.395e-03	6.074e-07
052_extra_large_clamp	DeepSDDF(1)	6.485e-06	1.459e-03	7.115e-04	2.208e-03	5.838e-07
	DeepSDDF(2)	7.923e-05	2.315e-03	9.836e-04	3.646e-03	5.998e-07
061_foam_brick	DeepSDDF(1)	1.349e-06	8.015e-04	6.968e-04	9.062e-04	5.841e-07
	DeepSDDF(2)	1.649e-06	8.862e-04	8.561e-04	9.163e-04	5.940e-07
Avg.	DeepSDDF(1)	1.121e-05	1.294e-03	6.854 e-04	1.903e-03	5.829e-07
	DeepSDDF(2)	8.831e-06	1.302e-03	8.251e-04	1.779e-03	5.989e-07

Table 3.3. Comparison between DeepSDDF and IGR [1] over 5 categories from ShapeNet [2], using metrics from [94] with meshes normalized to unit-length bounding box. Three versions of IGR are evaluated: IGR(1), using the same test points as DeepSDDF, IGR(2), producing a uniform point cloud from the reconstructed mesh, and IGR(3) using IDR [104] sphere-tracing.

Class	Method	Chamfer-L_2	Chamfer-L_1	Completeness	Accuracy	Time (sec.)
Car	DeepSDDF	1.971e-04	7.982e-03	7.609e-03	8.355e-03	1.190e-06
	IGR(1)	3.599e-03	3.869e-02	1.646e-02	6.092e-02	–
	IGR(2)	2.771e-03	3.273e-02	1.444e-02	5.102e-02	–
	IGR(3)	9.091e-03	6.937e-02	3.279e-02	1.059e-01	1.772e-05
Airplane	DeepSDDF	2.332e-04	7.707e-03	6.407e-03	9.008e-03	1.175e-06
	IGR(1)	1.774e-02	8.798e-02	2.076e-02	1.551e-01	–
	IGR(2)	1.260e-02	7.100e-02	1.774e-02	1.242e-01	–
	IGR(3)	2.840e-02	1.172e-01	2.880e-02	2.057e-01	2.317e-05
Watercraft	DeepSDDF	5.209e-04	1.288e-02	1.147e-02	1.428e-02	1.192e-06
	IGR(1)	7.075e-03	5.480e-02	2.751e-02	8.210e-02	–
	IGR(2)	6.981e-03	5.374e-02	2.733e-02	8.014e-02	–
	IGR(3)	3.326e-02	1.231e-01	3.855e-02	2.077e-01	2.010e-05
Sofa	DeepSDDF	3.850e-04	1.221e-02	1.093e-02	1.348e-02	1.195e-06
	IGR(1)	1.151e-02	6.923e-02	4.083e-02	9.764e-02	–
	IGR(2)	1.132e-02	6.904e-02	4.202e-02	9.607e-02	–
	IGR(3)	5.414e-02	1.563e-01	5.594e-02	2.567e-01	2.239e-05
Display	DeepSDDF	9.200e-04	1.789e-02	1.551e-02	2.027e-02	1.184e-06
	IGR(1)	8.883e-03	5.063e-02	3.493e-02	6.633e-02	–
	IGR(2)	8.306e-03	5.069e-02	3.491e-02	6.647e-02	–
	IGR(3)	2.139e-02	9.600e-02	4.030e-02	1.516e-01	1.951e-05
Avg.	DeepSDDF	4.512e-04	1.173e-02	1.038e-02	1.307e-02	1.187e-06
	IGR(1)	9.761e-03	6.026e-02	2.809e-02	9.241e-02	–
	IGR(2)	8.395e-03	5.544e-02	2.728e-02	8.358e-02	–
	IGR(3)	2.925e-02	1.123e-01	3.927e-02	1.855e-01	2.057e-05

Table 3.4. Comparison between DeepSDDF and baseline methods reported in GRNet [5] over 4 categories from ShapeNet [2]. The errors are scaled by $\times 10^{-3}$.

Class	Metric	Car	Airplane	Watercraft	Sofa	Avg.
AtlasNet	Chamfer-L_2	0.3237	0.1753	0.4177	0.5990	0.3789
	Chamfer-L_1	10.105	6.366	10.607	12.990	10.017
PCN	Chamfer-L_2	0.2445	0.1400	0.4062	0.5129	0.3259
	Chamfer-L_1	8.696	5.502	9.665	11.676	8.8847
FoldingNet	Chamfer-L_2	0.4676	0.3151	0.7325	0.8895	0.6011
	Chamfer-L_1	12.611	9.491	14.987	15.969	13.2645
TopNet	Chamfer-L_2	0.3513	0.2152	0.4359	0.6949	0.4243
	Chamfer-L_1	10.898	7.614	11.124	14.779	11.1037
MSN	Chamfer-L_2	0.4711	0.1543	0.3853	0.5894	0.4000
	Chamfer-L_1	10.776	5.596	9.485	11.895	9.438
GRNet	Chamfer-L_2	0.2752	0.1531	0.2122	0.3613	0.2504
	Chamfer-L_1	9.447	6.450	8.039	10.512	8.612
DeepSDDF	Chamfer-L_2	0.17351	0.26172	0.39942	0.23767	0.26808
	Chamfer-L_1	8.31331	5.90701	9.80779	9.8593	8.47185

Table 3.5. Comparison between DeepSDDF and IGR [1] on the challenging Lamp and Bench categories from ShapeNet [2] using metrics from [94] with meshes normalized to unit sphere. Two versions of DeepSDDF are evaluated: DeepSDDF(1), with 16 layers, and DeepSDDF(2), with 8 layers. Three versions of IGR are evaluated: IGR(1), using the same test points as DeepSDDF, IGR(2), producing a uniform point cloud from the reconstructed mesh, and IGR(3) using IDR [104] ray-tracing.

Class	Method	Chamfer-L_2	Chamfer-L_1	Completeness	Accuracy	Time (sec.)
Lamp	DeepSDDF(1)	5.310e-03	3.531e-02	2.503e-02	4.558e-02	1.175e-06
	DeepSDDF(2)	5.704e-03	3.866e-02	2.376e-02	5.356e-02	5.992e-07
	IGR(1)	2.222e-02	9.344e-02	4.729e-02	1.395e-01	–
	IGR(2)	2.011e-02	8.962e-02	4.833e-02	1.309e-01	–
	IGR(3)	1.094e-01	2.355e-01	7.112e-02	3.999e-01	1.839e-05
Bench	DeepSDDF(1)	1.682e-03	2.214e-02	1.808e-02	2.619e-02	1.179e-06
	DeepSDDF(2)	1.853e-03	2.352e-02	1.800e-02	2.905e-02	5.975e-07
	IGR(1)	1.331e-02	7.277e-02	3.111e-02	1.144e-01	–
	IGR(2)	1.176e-02	6.725e-02	3.132e-02	1.031e-01	–
	IGR(3)	3.539e-02	1.380e-01	5.946e-02	2.166e-01	1.788e-05
Avg.	DeepSDDF(1)	3.496e-03	2.872e-02	2.155e-02	3.588e-02	1.177e-06
	DeepSDDF(2)	3.778e-03	3.109e-02	2.088e-02	4.130e-02	5.983e-07
	IGR(1)	1.776e-02	8.310e-02	3.920e-02	1.269e-01	–
	IGR(2)	1.593e-02	7.843e-02	3.982e-02	1.170e-01	–
	IGR(3)	7.239e-02	1.867e-01	6.529e-02	3.082e-01	1.813e-05

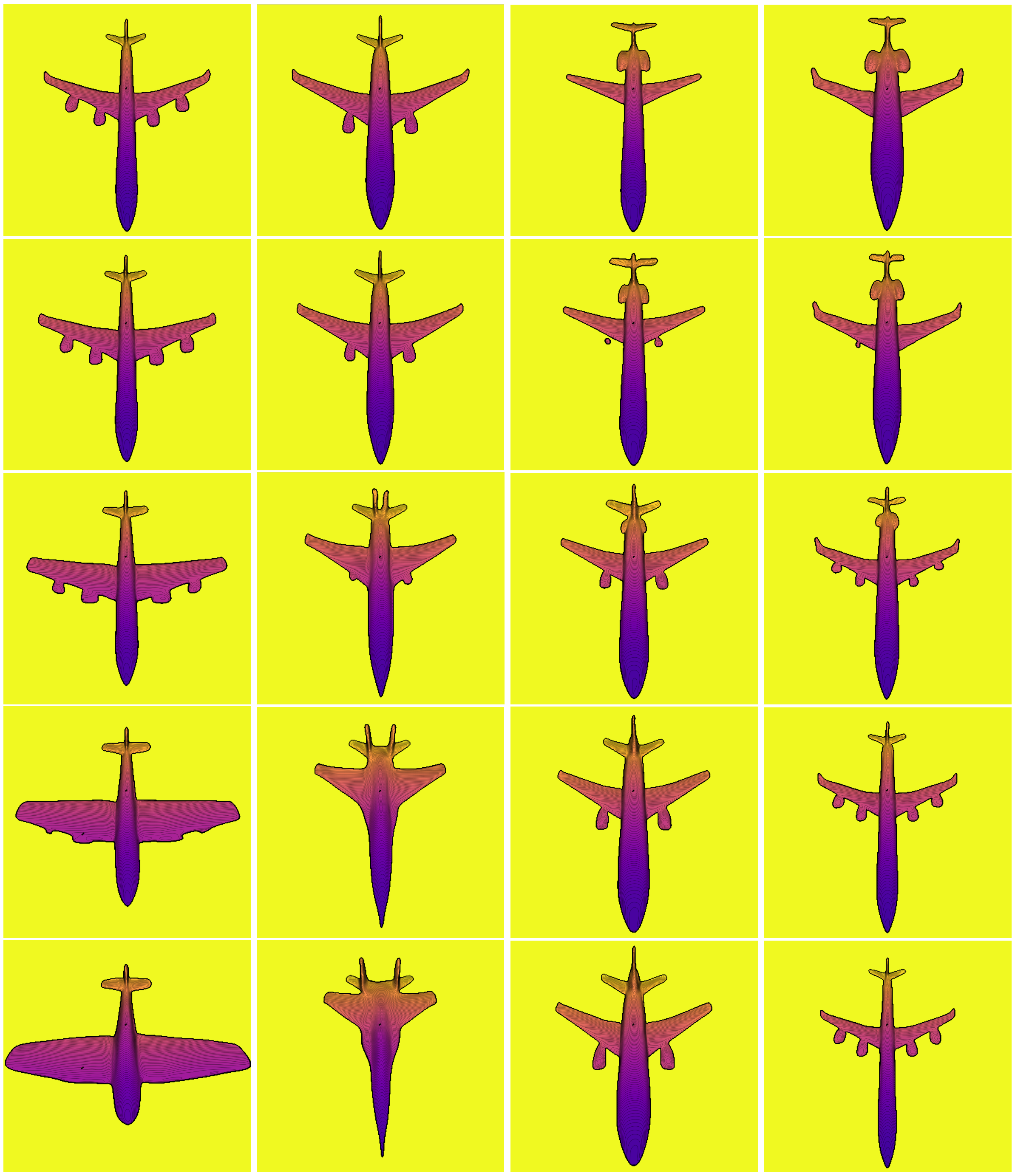


Figure 3.11. SDDF shape interpolation between two airplane instances. The first and last row show the SDDF output from the same view for two different instances from the training set. The rows in the middle are generated by using a weighted average of the latent codes of the upper-most and down-most instances as an input to the DeepSDDF network. In each column, from top to bottom, the latent code weights with respect to the upper-most instance are 1, 0.75, 0.5, 0.25, 0, respectively. Note how the shapes transform smoothly from top to bottom with intermediate shapes looking like valid airplanes. This demonstrates that the DeepSDDF model represents the latent shape space continuously and meaningfully.

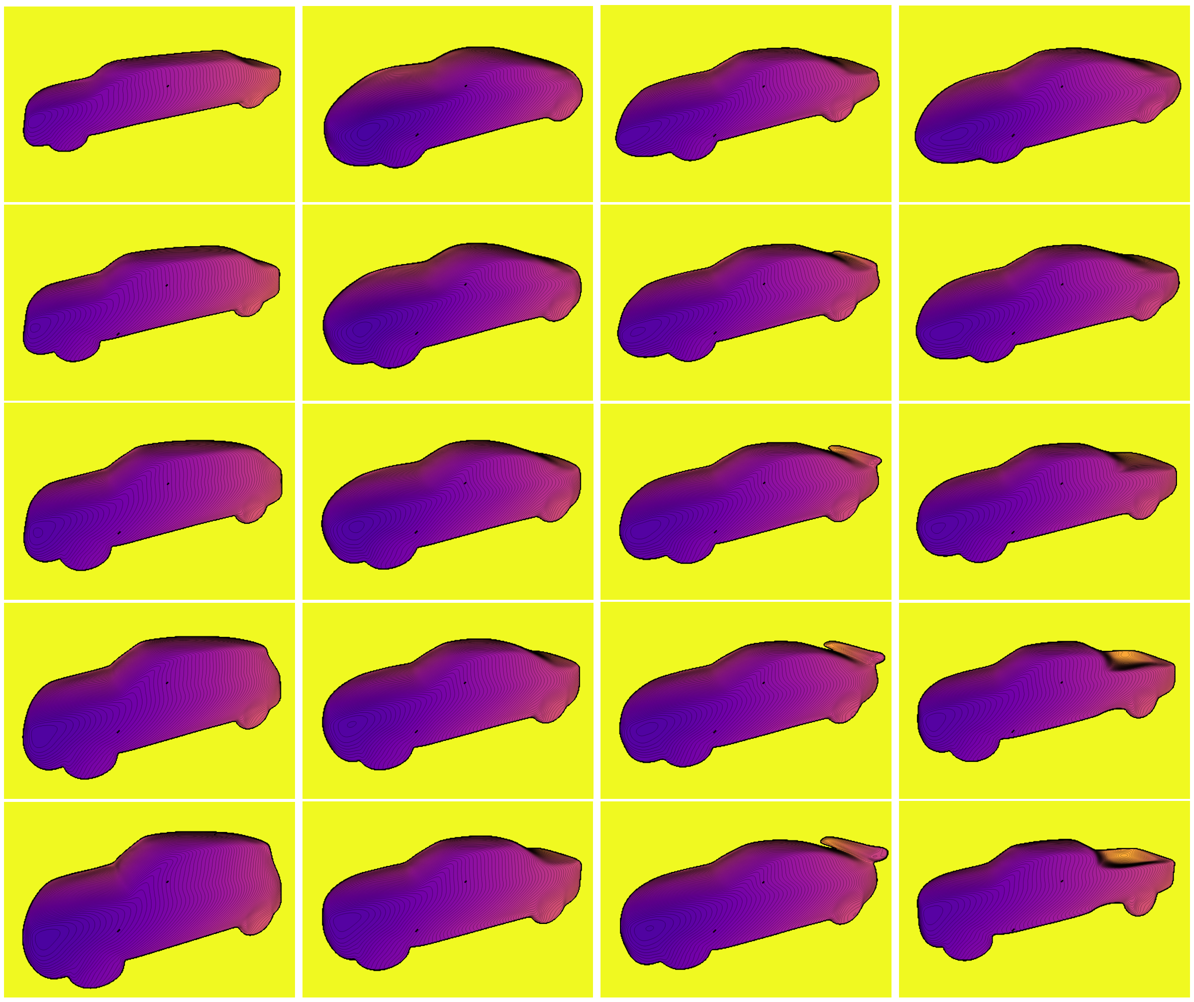


Figure 3.12. SDDF shape interpolation between two car instances. The first and last row show the SDDF output from the same view for two different instances from the training set. The rows in the middle are generated by using a weighted average of the latent codes of the upper-most and down-most instances as an input to the DeepSDDF network. In each column, from top to bottom, the latent code weights with respect to the upper-most instance are 1, 0.75, 0.5, 0.25, 0, respectively. Note how the shapes transform smoothly from top to bottom with intermediate shapes looking like valid cars. This demonstrates that the DeepSDDF model represents the latent shape space continuously and meaningfully.

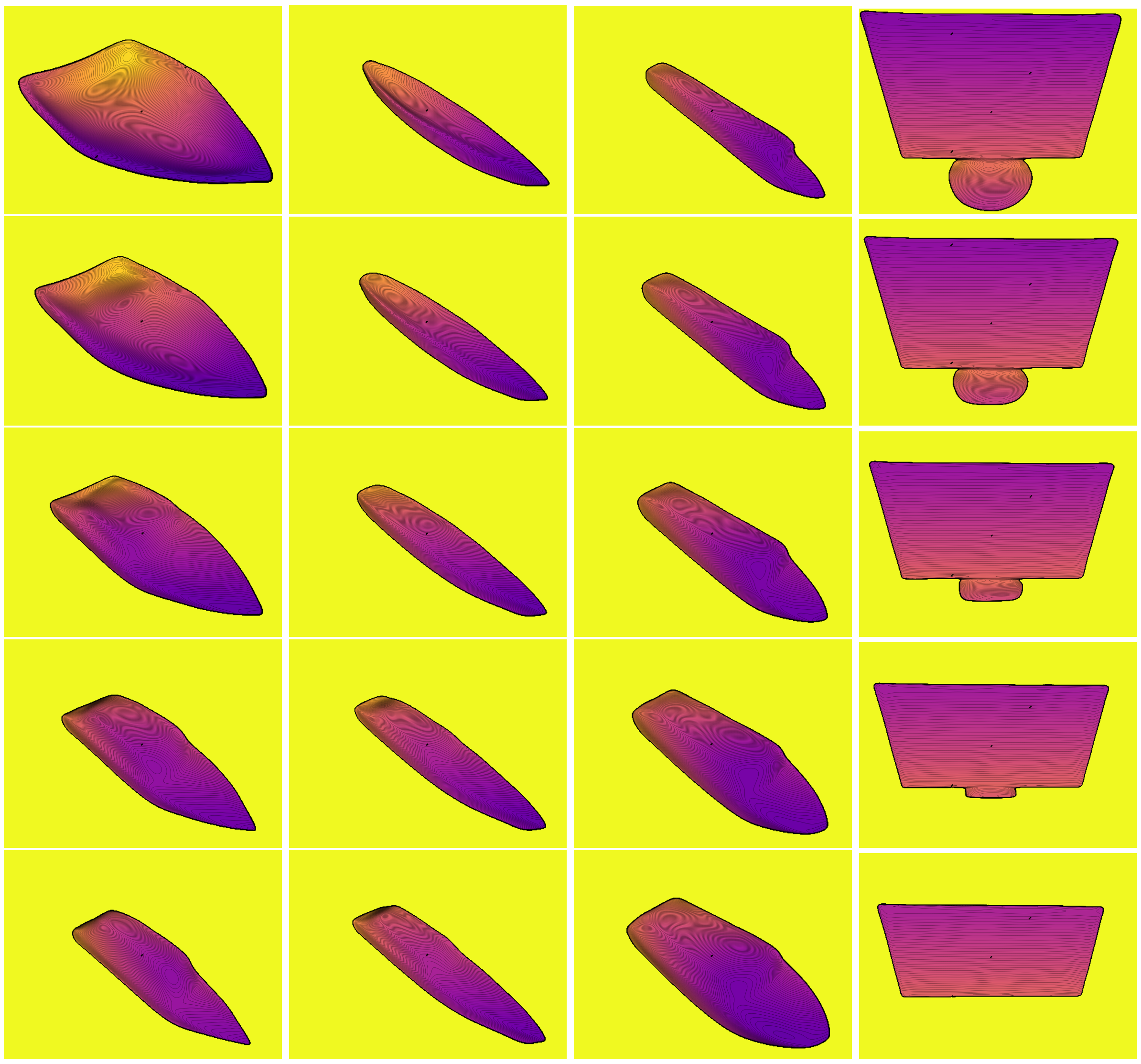


Figure 3.13. SDDF shape interpolation between two watercraft instances in the first three columns and two display instances in the last column. The first and last row show the SDDF output from the same view for two different instances from the training set. The rows in the middle are generated by using a weighted average of the latent codes of the upper-most and downer-most instances as an input to the DeepSDDF network. In each column, from top to bottom, the latent code weights with respect to the upper-most instance are 1, 0.75, 0.5, 0.25, 0, respectively. Note how the shapes transform smoothly from top to bottom with intermediate shapes looking like valid instances. This demonstrates that the DeepSDDF model represents the latent shape space continuously and meaningfully.

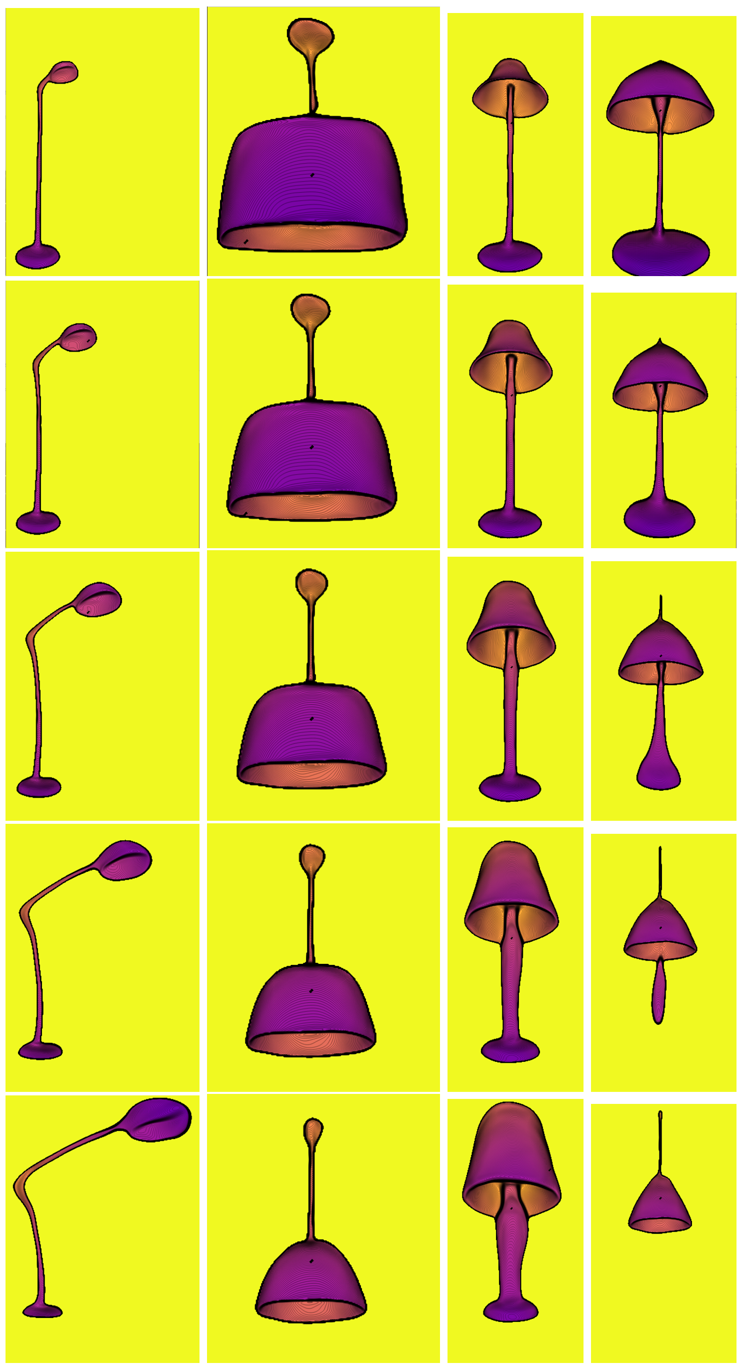


Figure 3.14. SDDF shape interpolation between two lamp instances. The first and last row show the SDDF output from the same view for two different instances from the training set. The rows in the middle are generated by using a weighted average of the latent codes of the upper-most and down-most instances as an input to the DeepSDDF network. In each column, from top to bottom, the latent code weights with respect to the upper-most instance are 1, 0.75, 0.5, 0.25, 0, respectively.

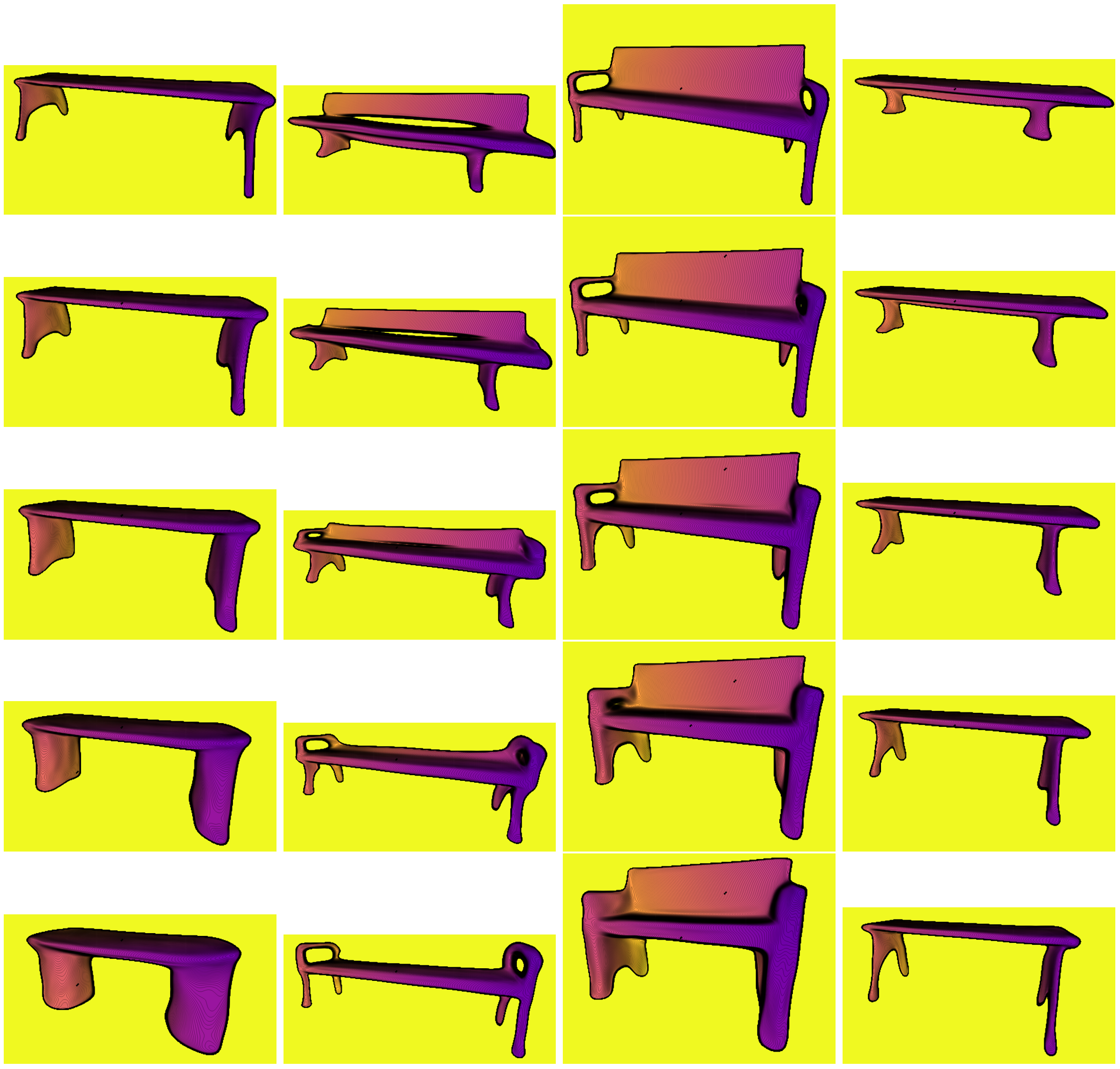


Figure 3.15. SDDF shape interpolation between two bench instances. The first and last row show the SDDF output from the same view for two different instances from the training set. The rows in the middle are generated by using a weighted average of the latent codes of the upper-most and down-most instances as an input to the DeepSDDF network. In each column, from top to bottom, the latent code weights with respect to the upper-most instance are 1, 0.75, 0.5, 0.25, 0, respectively. Note how the shapes transform smoothly from top to bottom with intermediate shapes looking like valid benches. This demonstrates that the DeepSDDF model represents the latent shape space continuously and meaningfully.

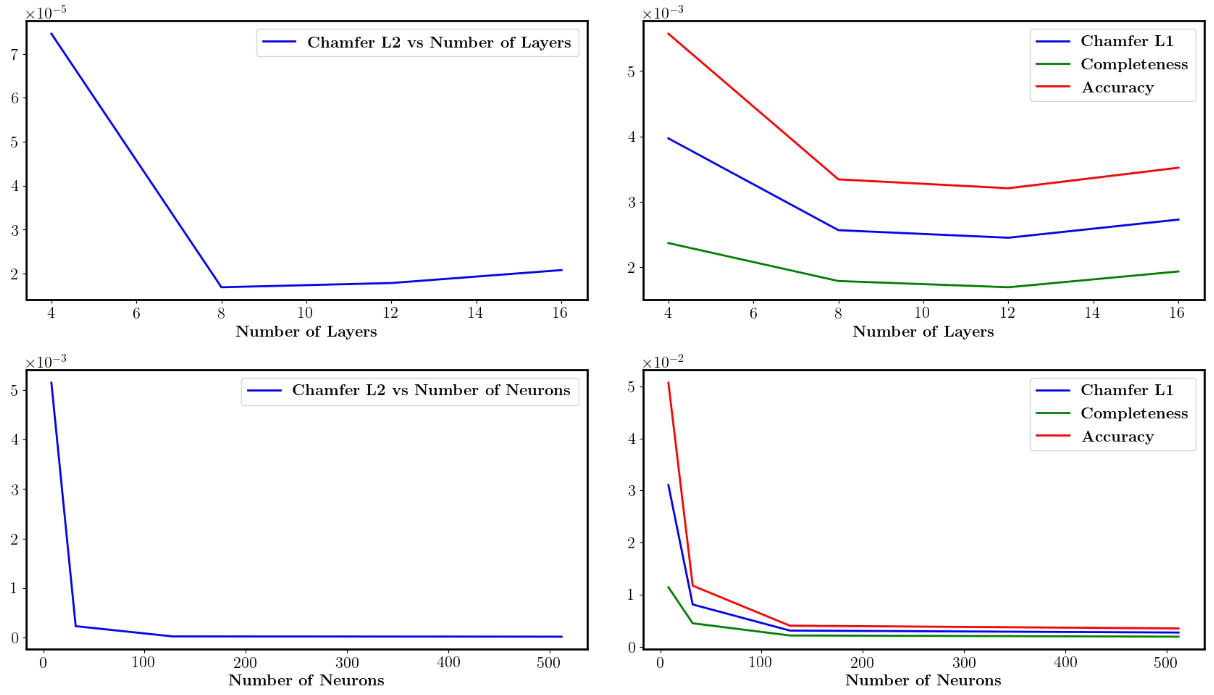


Figure 3.16. The first two figures show the error as the number of layers changes between $4, 8, 12, 16$ layers. The second two figures show the error as number of neurons changes between $8, 32, 128, 512$. In the first and third figures, Chamfer-L_2 distance is measured, and the second and forth figures contain Chamfer-L_1 distance, Accuracy and Completeness.

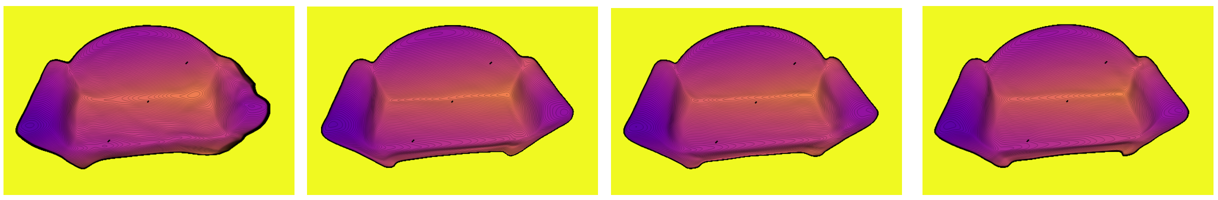


Figure 3.17. Distance views synthesized by our DeepSDDF model trained with the same data with 512 neurons per layer and different numbers of layers: 4 (first), 8 (second), 12 (third), 16 (forth).

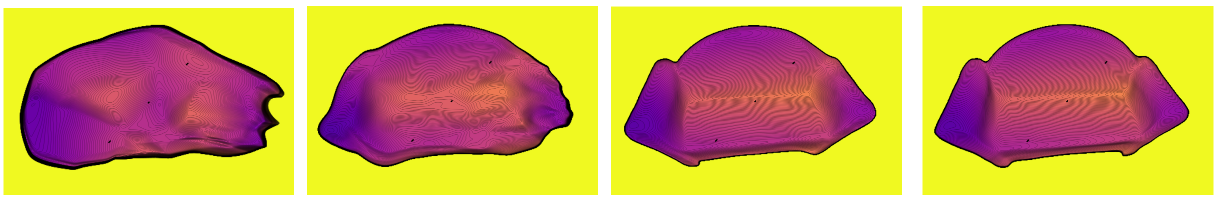


Figure 3.18. Distance views synthesized by our DeepSDDF model trained with the same data with 16 layers and different numbers of neurons per layer: 8 (first), 32 (second), 128 (third), 512 (forth).

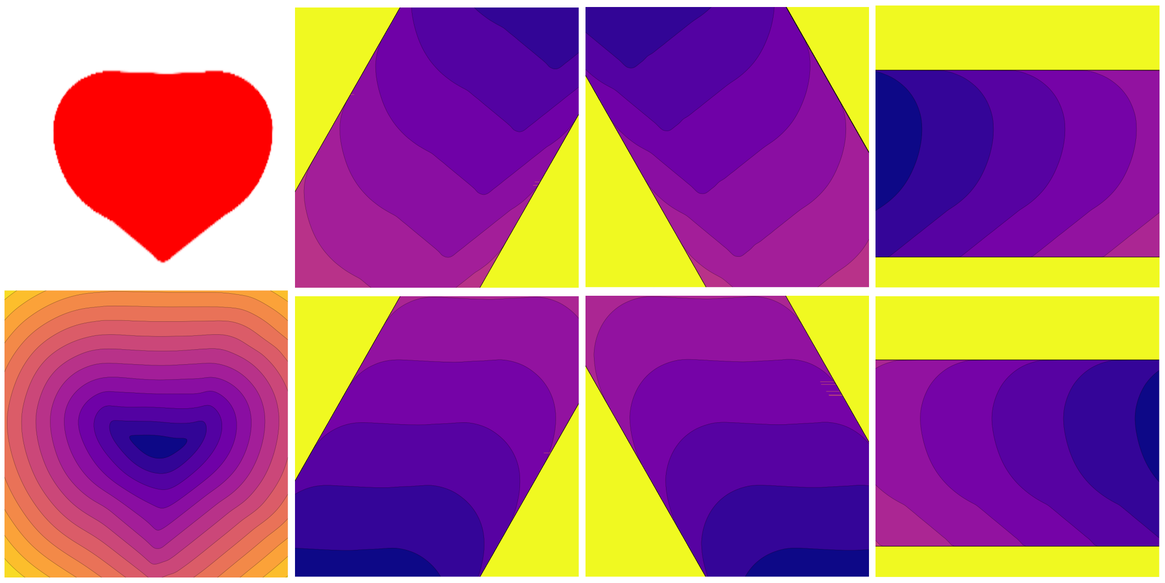


Figure 3.19. Ground-truth 2D instance (top left) and signed distance field (bottom left) learned by IGR [1]. The remaining plots show the output of our DeepSDDF model with a fixed direction at each 2D location in the image. The fixed viewing directions for the six plots from left to right and top to down are $\frac{\pi}{3}$, $\frac{2\pi}{3}$, π, $\frac{4\pi}{3}$, $\frac{5\pi}{3}$, 2π, respectively. To produce good color contrast, in all images infinite distance values (corresponding to rays in free space) are set to 1.

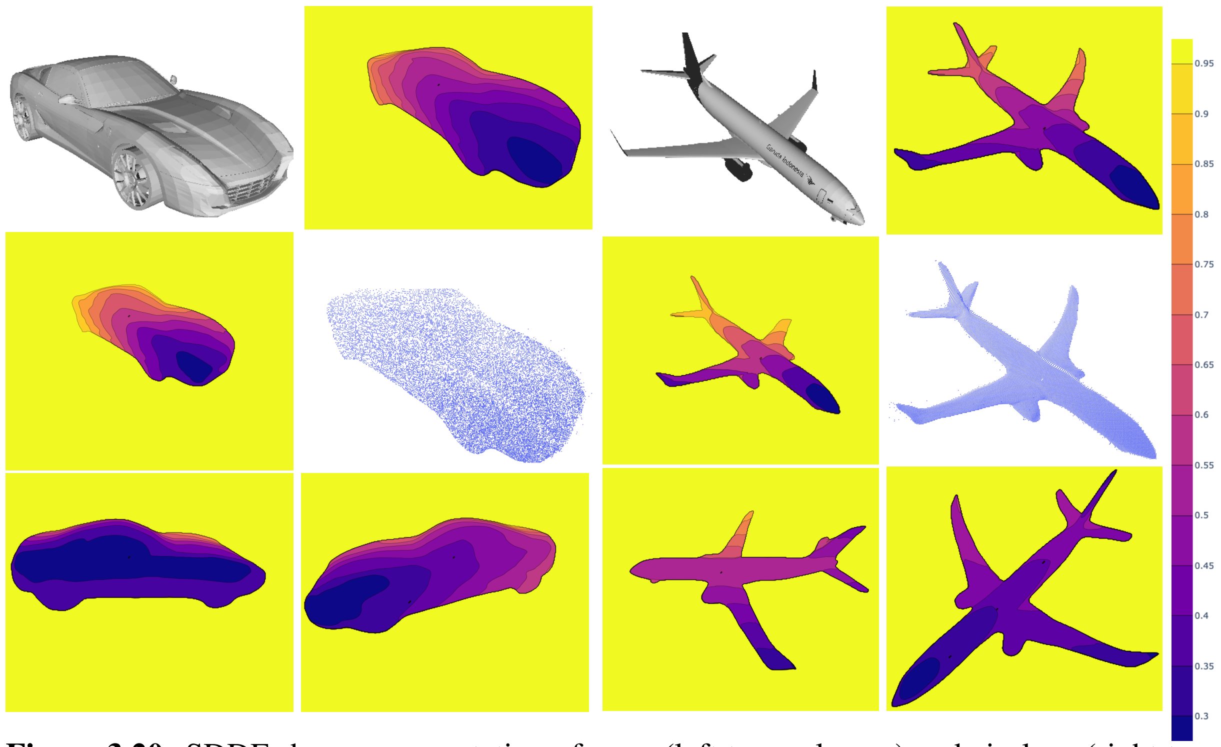


Figure 3.20. SDDF shape representation of a car (left two columns) and airplane (right two columns) instance. In each two columns, the ground-truth model is shown on the top left, a distance image synthesized by the DeepSDDF model is shown on the top right, and a point cloud generated from the distance image is shown in the middle right. The middle left shows a distance image synthesized from the same view but further distance from the object. Note that the level sets in the distant view remain parallel to the close-up view but more yellowish, indicating the distance increase. The third row includes more distance images synthesized by our DeepSDDF model from other views. To produce good color contrast, in all images we set infinite distance values (corresponding to rays in free space) to 1.

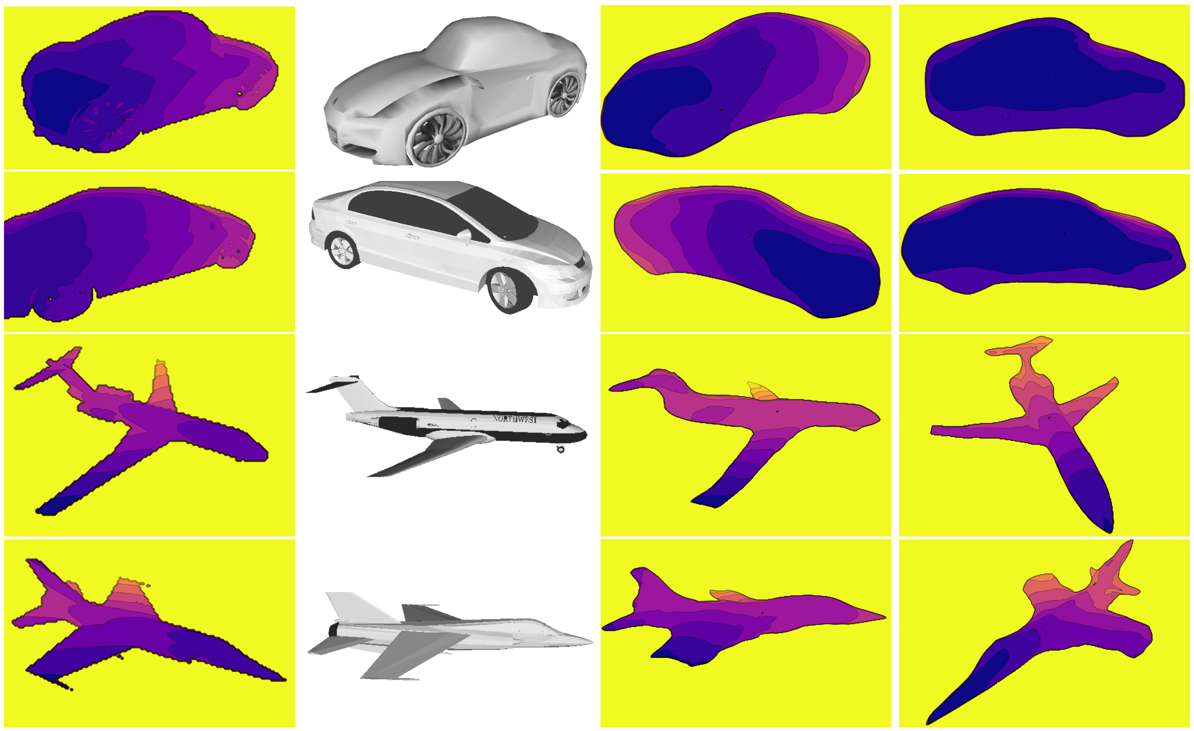


Figure 3.21. SDDF shape completion using a distance image (left column) from an unseen object instance (second column). After latent code optimization, the DeepSDDF model can synthesize novel distance views (third and fourth columns).

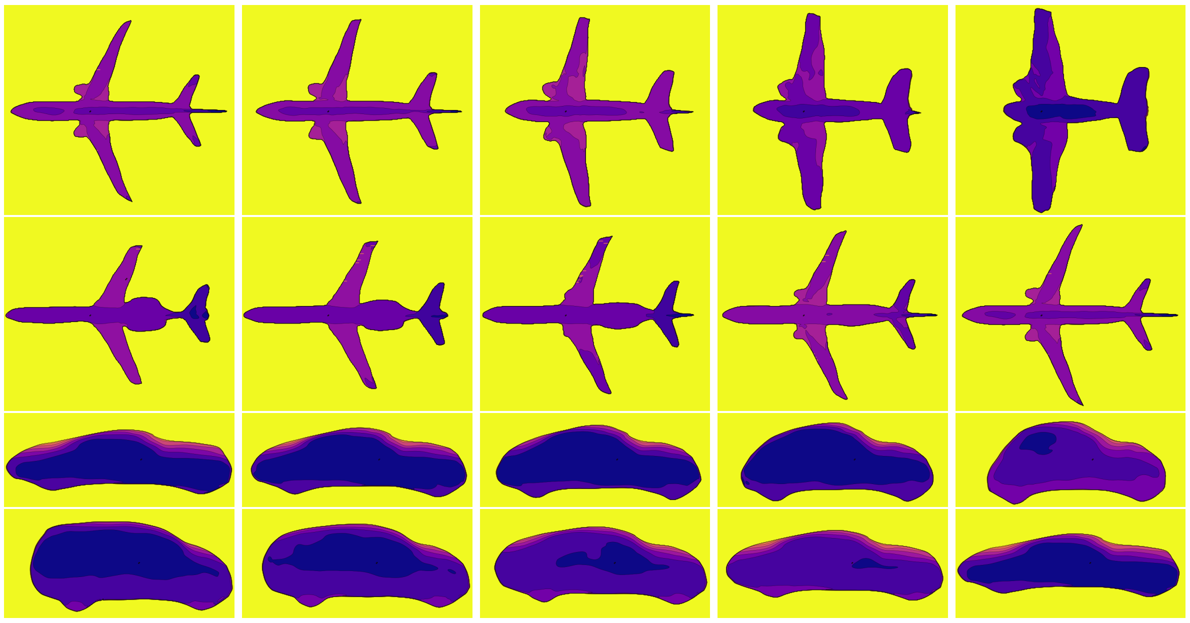


Figure 3.22. SDDF shape interpolation between two instances. The left-most and right-most columns show the SDDF output from the same view for two different instances from the training set. The three columns in the middle are generated by using a weighted average of the latent codes of the left-most and right-most instances as an input to the DeepSDDF network. In each row, from left to right, the latent code weights with respect to the left-most instance are 1, 0.75, 0.5, 0.25, 0, respectively. Note how the shapes transform smoothly from the left-most to the right-most instances with intermediate shapes looking like valid cars and airplanes. This demonstrates that the DeepSDDF model represents the latent shape space continuously and meaningfully.

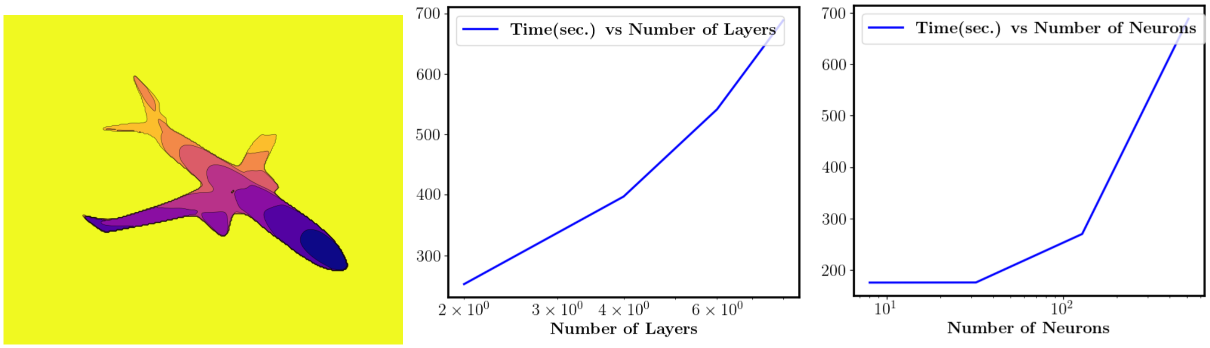


Figure 3.23. Distance image (left) produced by our DeepSDDF model with 8 layers, 512 neurons per layer, and noiseless training data. The model training time is shown as a function of the number of layers (middle) and number of neurons per layer (right). Note that in the right plot the *x*-axis is in log scale and the number of neurons is equal in all layers. The error of the point cloud, obtained from 100 distance images produced by our DeepSDDF model at 100 fixed poses, with respect to the ground-truth instance mesh in different settings: noisy distance data (left), changing number of network layers (middle), and changing numbers of neurons per layer (right). Note that in the right plot the *x*-axis is in log scale and the number of neurons is equal in all layers.

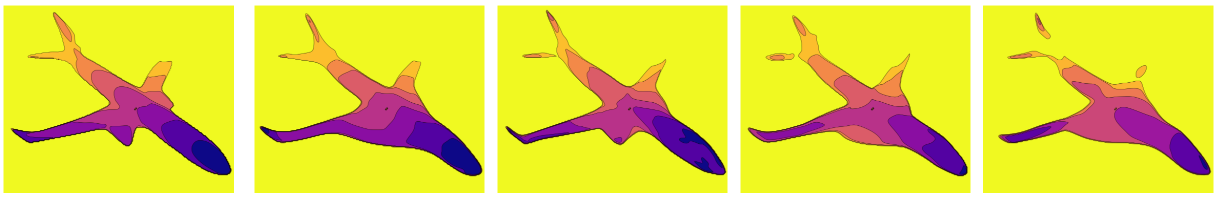


Figure 3.24. Distance views synthesized by our DeepSDDF model with default structure (8 layers and 512 neurons per layer) trained on noisy data. The standard deviation of the Gaussian noise added to the distance data from left to right is 0 (without noise), 0.1, 0.2, 0.3, 0.4, respectively.

Chapter 4

Conclusion

In this thesis, we have investigated the problem of mapping and shape representation for robotic applications. We considered the challenges in the mapping and addressed them within a single method. Based on existing shape representation signed distance function and Gaussian process regression, we proposed an online, continuous, probabilistic, 3D representation of geometric surfaces and semantic classes in the environment. Then we have extended our method to a distributed mapping algorithm that could be used by a network of robots which communicate with each other through a one-hop network. We have elaborated on the method to provide more efficient methods to address echo in the network and handle the large dimension of mapping problem. For all we have provided theoretical guarantee on convergence. However, there are a lot of room to improve this method like incorporating other information like temperature in the method. Another interesting direction is that we can investigate better optimization of the kernels or pseudo points or the way we transfer the data from depth images to SDF data.

In continue, observing the limitations that the popular shape representations have, we have proposed a novel shape representation signed directional distance function (SDDF). SDDF extends the SDF definition by measuring distance in a desired viewing direction rather than to the nearest point. As a result, SDDF removes post-processing steps for view synthesis required by SDF, such as surface extraction via marching cubes or rendering via sphere tracing, and allows ray-tracing through a single function call. SDDF also encodes by construction the property that distance decreases linearly along the viewing direction. We showed that this enables dimensionality reduction in the function representation and guarantees the prediction accuracy independent of the distance to the surface. Recent advances demonstrate impressive performance of deep neural networks for shape learning, including DeepSDF for SDF, Occupancy Networks for occupancy, AtlasNet for meshes, and NeRF for density. Our second contribution, DeepSDDF, is a deep neural network model for SDDF shape learning. Similar to DeepSDF, we show that DeepSDDF can model whole object categories and interpolate or complete shapes from partial views. However, there are a lot of room to build upon this shape representation. For example extending the method to scene level. Our current implementation includes only the geometry, we can add the color to it as well. Improving the frequency of representation is another challenge since the ideas developed for SDF do not necessarily work here. Another aspect is to improve the multi-view consistency aspect. More importantly, this a novel shape representation and it is very interesting to explore using SDDF and DeepSDDF for various applications that have been done before with other shape representation.

Another interesting aspect of this thesis is that we have investigated different learning methods like Gaussian processes (GP) and deep learning in this thesis. We grow a sense about their pros and cons. For example we observed that GP performs better in presence of noise, can capture the higher frequencies in comparison to neural networks. Additionally, it is much easier to provide an incremental and distributed method with GP, while it is very challenging with neural networks. On the other side, when we need to learn a lot of information like the shape representation over a class, deep learning works much better. In future, it is very interesting to explore other machine learning tools like graph neural networks and their performance for the mapping and shape representation as well.

Bibliography

[1] A. Gropp, L. Yariv, N. Haim, M. Atzmon, and Y. Lipman, "Implicit geometric regularization for learning shapes," *arXiv preprint arXiv:2002.10099*, 2020.

[2] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An Information-Rich 3D Model Repository," *arXiv:1512.03012*, 2015.

[3] F. Williams, T. Schneider, C. Silva, D. Zorin, J. Bruna, and D. Panozzo, "Deep geometric prior for surface reconstruction," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[4] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The ycb object and model set: Towards common benchmarks for manipulation research," in *2015 international conference on advanced robotics (ICAR)*, pp. 510–517, IEEE, 2015.

[5] H. Xie, H. Yao, S. Zhou, J. Mao, S. Zhang, and W. Sun, "Grnet: Gridding residual network for dense point cloud completion," in *European Conference on Computer Vision*, pp. 365–381, Springer, 2020.

[6] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Conference on Computer Graphics and Interactive Techniques*, pp. 303–312, 1996.

[7] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Eurographics Symposium on Geometry Processing*, 2006.

[8] X. Wang, M. R. Oswald, I. Cherabier, and M. Pollefeys, "Learning 3d semantic reconstruction on octrees," in *German Conference on Pattern Recognition*, pp. 581–594, Springer, 2019.

[9] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, 2013.

[10] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3d reconstruction at scale using voxel hashing," *ACM Transactions on Graphics (ToG)*, vol. 32, no. 6, pp. 1–11, 2013.

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

[12] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, “Semanticfusion: Dense 3d semantic mapping with convolutional neural networks,” in *2017 IEEE International Conference on Robotics and automation (ICRA)*, pp. 4628–4635, IEEE, 2017.

[13] A. Hermans, G. Floros, and B. Leibe, “Dense 3D Semantic Mapping of Indoor Scenes from RGB-D Images,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2631–2638, 2014.

[14] A. Kundu, Y. Li, F. Dellaert, F. Li, and J. M. Rehg, “Joint semantic segmentation and 3d reconstruction from monocular video,” in *European Conference on Computer Vision*, pp. 703–718, Springer, 2014.

[15] J. Zhang, C. Zhu, L. Zheng, and K. Xu, “Fusion-aware point convolution for online semantic 3d scene segmentation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4534–4543, 2020.

[16] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[17] C. E. Rasmussen, “Gaussian processes in machine learning,” in *Summer School on Machine Learning*, pp. 63–71, Springer, 2003.

[18] S. O’Callaghan and F. Ramos, “Gaussian process occupancy maps,” *The International Journal of Robotics Research (IJRR)*, vol. 31, no. 1, pp. 42–62, 2012.

[19] S. Kim and J. Kim, “Occupancy Mapping and Surface Reconstruction Using Local Gaussian Processes With Kinect Sensors,” *IEEE Trans. on Cybernetics*, vol. 43, no. 5, pp. 1335–1346, 2013.

[20] M. G. Jadidi, J. V. Miró, R. Valencia, and J. Andrade-Cetto, “Exploration on Continuous Gaussian Process Frontier Maps,” in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 6077–6082, 2014.

[21] R. Senanayake and F. Ramos, “Building continuous occupancy maps with moving robots,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[22] M. Ghaffari Jadidi, L. Gan, S. Parkison, J. Li, and R. Eustice, “Gaussian Processes Semantic Map Representation,” *arXiv:1707.01532*, 2017.

[23] J. Hensman, A. Matthews, and Z. Ghahramani, “Scalable Variational Gaussian Process Classification,” in *International Conference on Artificial Intelligence and Statistics*, pp. 351–360, 2015.

[24] T. Galy-Fajou, F. Wenzel, C. Donner, and M. Opper, "Multi-class gaussian process classification made conjugate: Efficient inference via data augmentation," in *Uncertainty in Artificial Intelligence Conference*, pp. 755–765, 2020.

[25] D. Hernández-Lobato, J. Hernández-lobato, and P. Dupont, "Robust multi-class gaussian process classification," *Advances in neural information processing systems*, vol. 24, pp. 280–288, 2011.

[26] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs," in *Advances in neural information processing systems*, pp. 1257–1264, 2006.

[27] J. Hensman, N. Durrande, and A. Solin, "Variational Fourier features for Gaussian processes," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 5537–5588, 2017.

[28] A. Koppel, "Consistent online Gaussian Process regression without the sample complexity bottleneck," in *American Control Conference (ACC)*, pp. 3512–3518, 2019.

[29] A. Koppel, A. S. Bedi, K. Rajawat, and B. M. Sadler, "Optimally compressed nonparametric online learning," *IEEE Signal Processing Magazine*, 2020.

[30] V. Tresp, "A bayesian committee machine," *Neural computation*, vol. 12, no. 11, pp. 2719–2741, 2000.

[31] S. Kim and J. Kim, "Recursive Bayesian Updates for Occupancy Mapping and Surface Reconstruction," in *Australasian Conference on Robotics and Automation (ACRA)*, 2014.

[32] M. Bauer, M. van der Wilk, and C. E. Rasmussen, "Understanding probabilistic sparse gaussian process approximations," in *Advances in neural information processing systems*, pp. 1533–1541, 2016.

[33] C. E. Rasmussen and Z. Ghahramani, "Infinite mixtures of gaussian process experts," in *Advances in neural information processing systems*, pp. 881–888, 2002.

[34] A. Nedić, A. Olshevsky, and C. A. Uribe, "Distributed learning for cooperative inference," *arXiv preprint:1704.02718*, 2017.

[35] A. Jadbabaie, P. Molavi, A. Sandroni, and A. Tahbaz-Salehi, "Non-bayesian social learning," *Games and Economic Behavior*, vol. 76, no. 1, pp. 210–225, 2012.

[36] E. Zobeidi, A. Koppel, and N. Atanasov, "Dense incremental metric-semantic mapping via sparse gaussian process regression," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[37] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[38] E. Vespa, N. Nikolov, M. Grimm, L. Nardi, P. H. J. Kelly, and S. Leutenegger, "Efficient Octree-Based Volumetric SLAM Supporting Signed-Distance and Occupancy Mapping," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1144–1151, 2018.

[39] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *European Conf. on Computer Vision*, 2014.

[40] R. Dubé, A. Cramariuc, D. Dugas, H. Sommer, M. Dymczyk, J. Nieto, R. Siegwart, and C. Cadena, "SegMap: Segment-based mapping and localization using data-driven descriptors," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 339–355, 2020.

[41] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.

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

[43] L. Teixeira and M. Chli, "Real-time mesh-based scene estimation for aerial inspection," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4863–4869, 2016.

[44] E. Piazza, A. Romanoni, and M. Matteucci, "Real-time cpu-based large-scale three-dimensional mesh reconstruction," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1584–1591, 2018.

[45] R. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-Time Dense Surface Mapping and Tracking," in *IEEE Int. Symposium on Mixed and Augmented Reality*, pp. 127–136, 2011.

[46] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "Elasticfusion: Real-time dense slam and light source estimation," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016.

[47] 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 Int. Conf. on Intelligent Robots and Systems*, 2017.

[48] 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 Int. Conf. on Intelligent Robots and Systems*, 2019.

[49] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. McDonald, "Real-time large-scale dense RGB-D SLAM with volumetric fusion," *The International Journal of Robotics Research (IJRR)*, vol. 34, no. 4-5, pp. 598–626, 2015.

[50] M. Klingensmith, I. Dryanovski, S. S. Srinivasa, and J. Xiao, "Chisel: Real Time Large Scale 3D Reconstruction Onboard a Mobile Device using Spatially Hashed Signed Distance Fields," in *Robotics: science and systems*, vol. 4, p. 1, Citeseer, 2015.

[51] L. Han and L. Fang, "FlashFusion: Real-time Globally Consistent Dense 3D Reconstruction using CPU Computing," in *Robotics: Science and Systems (RSS)*, 2018.

[52] O. Kähler, V. A. Prisacariu, and D. W. Murray, "Real-time large-scale dense 3d reconstruction with loop closure," in *European Conference on Computer Vision (ECCV)*, pp. 500–516, 2016.

[53] V. Reijgwart, A. Millane, H. Oleynikova, R. Siegwart, C. Cadena, and J. Nieto, "Voxgraph: Globally Consistent, Volumetric Mapping Using Signed Distance Function Submaps," *IEEE Robotics and Automation Letters*, 2020.

[54] M. Grinvald, F. Furrer, T. Novkovic, J. J. Chung, C. Cadena, R. Siegwart, and J. Nieto, "Volumetric Instance-Aware Semantic Mapping and 3D Object Discovery," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 3037–3044, 2019.

[55] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers, "Real-time camera tracking and 3d reconstruction using signed distance functions.," in *Robotics: Science and Systems*, 2013.

[56] H. Oleynikova, M. Burri, Z. Taylor, J. I. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online uav replanning," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2016.

[57] K. Saulnier, N. Atanasov, G. Pappas, and V. Kumar, "Information theoretic active exploration in signed distance fields," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2020.

[58] S. Kim and J. Kim, "GPmap: A Unified Framework for Robotic Mapping Based on Sparse Gaussian Processes," in *International Conference on Field and Service Robotics*, 2015.

[59] J. Wang and B. Englot, "Fast, accurate gaussian process occupancy maps via test-data octrees and nested bayesian fusion," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 1003–1010, 2016.

[60] F. Ramos and L. Ott, "Hilbert maps: Scalable continuous occupancy mapping with stochastic gradient descent," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1717–1730, 2016.

[61] W. Zhi, L. Ott, R. Senanayake, and F. Ramos, "Continuous occupancy map fusion with fast bayesian hilbert maps," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4111–4117, 2019.

[62] R. Senanayake and F. Ramos, "Bayesian Hilbert Maps for Continuous Occupancy Mapping in Dynamic Environments," in *Conference on Robot Learning (CoRL)*, vol. 78 of *Proceedings of Machine Learning Research*, pp. 458–471, 2017.

[63] R. Senanayake, S. O'Callaghan, and F. Ramos, "Learning highly dynamic environments with stochastic variational inference," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2532–2539, 2017.

[64] V. Guizilini and F. Ramos, "Learning to Reconstruct 3D Structures for Occupancy Mapping," in *Robotics: Science and Systems*, 2017.

[65] S. Guo and N. A. Atanasov, "Information filter occupancy mapping using decomposable radial kernels," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7887–7894, 2019.

[66] V. Vineet, O. Miksik, M. Lidegaard, M. Nießner, S. Golodetz, V. A. Prisacariu, O. Kähler, D. W. Murray, S. Izadi, P. Pérez, and P. H. S. Torr, "Incremental dense semantic stereo fusion for large-scale semantic scene reconstruction," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 75–82, 2015.

[67] S. Sengupta and P. Sturgess, "Semantic octree: Unifying recognition, reconstruction and representation via an octree constrained higher order mrf," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1874–1879, IEEE, 2015.

[68] S. Yang, Y. Huang, and S. Scherer, "Semantic 3D occupancy mapping through efficient high-order CRFs," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 590–597, 2017.

[69] Z. Zhao and X. Chen, "Building 3D semantic maps for mobile robots using RGB-D camera," *Intelligent Service Robotics*, vol. 9, no. 4, pp. 297–309, 2016.

[70] L. Gan, R. Zhang, J. W. Grizzle, R. M. Eustice, and M. Ghaffari Jadidi, "Bayesian spatial kernel smoothing for scalable dense semantic mapping," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 790–797, 2020.

[71] K. Zheng and A. Pronobis, "From pixels to buildings: End-to-end probabilistic deep networks for large-scale semantic mapping," *arXiv preprint arXiv:1812.11866*, 2018.

[72] Y. Siddiqui, J. Thies, F. Ma, Q. Shan, M. Nießner, and A. Dai, "RetrievalFuse: Neural 3D Scene Reconstruction with a Database," *IEEE International Conference on Computer Vision (ICCV)*, 2021.

[73] R. Chabra, J. E. Lenssen, E. Ilg, T. Schmidt, J. Straub, S. Lovegrove, and R. Newcombe, "Deep local shapes: Learning local sdf priors for detailed 3d reconstruction," in *European Conference on Computer Vision (ECCV)*, pp. 608–625, Springer, 2020.

[74] C. Jiang, A. Sud, A. Makadia, J. Huang, M. Nießner, T. Funkhouser, *et al.*, "Local implicit grid representations for 3d scenes," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6001–6010, 2020.

[75] S. Peng, M. Niemeyer, L. Mescheder, M. Pollefeys, and A. Geiger, "Convolutional occupancy networks," in *European Conference on Computer Vision (ECCV)*, pp. 523–540, Springer, 2020.

[76] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[77] K. Rahnama Rad and A. Tahbaz-Salehi, "Distributed parameter estimation in networks," in *IEEE Conference on Decision and Control (CDC)*, pp. 5050–5055, 2010.

[78] N. Atanasov, R. Tron, V. M. Preciado, and G. J. Pappas, "Joint estimation and localization in sensor networks," in *IEEE Conference on Decision and Control (CDC)*, pp. 6875–6882, 2014.

[79] A. Nedić, A. Olshevsky, and C. A. Uribe, "Distributed Gaussian learning over time-varying directed graphs," in *Asilomar Conference on Signals, Systems and Computers*, pp. 1710–1714, 2016.

[80] S. Choudhary, L. Carlone, C. Nieto, J. Rogers, H. I. Christensen, and F. Dellaert, "Distributed mapping with privacy and communication constraints: Lightweight algorithms and object-based models," *The International Journal of Robotics Research*, vol. 36, no. 12, pp. 1286–1311, 2017.

[81] P. Koch, S. May, M. Schmidpeter, M. Kühn, C. Pfitzner, C. Merkl, R. Koch, M. Fees, J. Martin, D. Ammon, and A. Nüchter, "Multi-robot localization and mapping based on signed distance functions," *Journal of Intelligent & Robotic Systems*, vol. 83, no. 3-4, pp. 409–428, 2016.

[82] P. Lajoie, B. Ramtoula, Y. Chang, L. Carlone, and G. Beltrame, "DOOR-SLAM: Distributed, Online, and Outlier Resilient SLAM for Robotic Teams," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1656–1663, 2020.

[83] A. Milioto and C. Stachniss, "Bonnet: An Open-Source Training and Deployment Framework for Semantic Segmentation in Robotics using CNNs," in *IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[84] T. D. Bui, J. Yan, and R. E. Turner, "A unifying framework for sparse gaussian process approximation using power expectation propagation," *stat*, vol. 23, no. 1050, 2016.

[85] J. Oliva, B. Póczos, and J. Schneider, "Distribution to distribution regression," in *International Conference on Machine Learning*, pp. 1049–1057, PMLR, 2013.

[86] K. B. Petersen and M. S. Pedersen, "The matrix cookbook," 2008.

[87] A. Tahbaz-Salehi and A. Jadbabaie, "A Necessary and Sufficient Condition for Consensus Over Random Networks," *IEEE Transactions on Automatic Control*, vol. 53, no. 3, pp. 791–795, 2008.

[88] A. Nedic and A. Ozdaglar, "Distributed Subgradient Methods for Multi-Agent Optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[89] L. Moreau, "Stability of multiagent systems with time-dependent communication links," *IEEE Transactions on Automatic Control*, vol. 50, no. 2, pp. 169–182, 2005.

[90] F. Saadatniaki, R. Xin, and U. A. Khan, "Decentralized optimization over time-varying directed graphs with row and column-stochastic matrices," *IEEE Transactions on Automatic Control*, vol. 65, no. 11, pp. 4769–4780, 2020.

[91] T. S. Newman and H. Yi, "A survey of the marching cubes algorithm," *Computers & Graphics*, vol. 30, no. 5, pp. 854–879, 2006.

[92] B.-S. Hua, Q.-H. Pham, D. T. Nguyen, M.-K. Tran, L.-F. Yu, and S.-K. Yeung, "Scenenn: A scene meshes dataset with annotations," in *International Conference on 3D Vision (3DV)*, 2016.

[93] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *Computer Graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[94] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3D reconstruction in function space," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4460–4470, 2019.

[95] W. Yifan, F. Serena, S. Wu, C. Öztireli, and O. Sorkine-Hornung, "Differentiable Surface Splatting for Point-based Geometry Processing," *ACM Transactions on Graphics, SIGGRAPH ASIA*, vol. 38, no. 6, 2019.

[96] S. Tulsiani, H. Su, L. J. Guibas, A. A. Efros, and J. Malik, "Learning shape abstractions by assembling volumetric primitives," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2635–2643, 2017.

[97] D. Paschalidou, A. O. Ulusoy, and A. Geiger, "Superquadrics revisited: Learning 3d shape parsing beyond cuboids," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[98] A. Kanazawa, S. Tulsiani, A. A. Efros, and J. Malik, "Learning category-specific mesh reconstruction from image collections," in *European Conference on Computer Vision (ECCV)*, 2018.

[99] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: an efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[100] J. T. Kajiya and B. P. Von Herzen, "Ray tracing volume densities," *ACM SIGGRAPH computer graphics*, vol. 18, no. 3, pp. 165–174, 1984.

[101] 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*, pp. 127–136, 2011.

[102] 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 Visioan and Pattern Recognition (CVPR)*, pp. 165–174, 2019.

[103] S. Liu, Y. Zhang, S. Peng, B. Shi, M. Pollefeys, and Z. Cui, "DIST: Rendering deep implicit signed distance function with differentiable sphere tracing," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2019–2028, 2020.

[104] L. Yariv, Y. Kasten, D. Moran, M. Galun, M. Atzmon, B. Ronen, and Y. Lipman, "Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 2492–2502, 2020.

[105] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger, "Differentiable volumetric rendering: Learning implicit 3D representations without 3D supervision," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3504–3515, 2020.

[106] 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*, pp. 405–421, 2020.

[107] J. C. Hart, "Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces," *The Visual Computer*, vol. 12, no. 10, pp. 527–545, 1996.

[108] 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 Int. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[109] L. Gao, Y.-K. Lai, J. Yang, Z. Ling-Xiao, S. Xia, and L. Kobbelt, "Sparse data driven mesh deformation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 3, pp. 2085–2100, 2021.

[110] Q. Tan, L. Gao, Y.-K. Lai, and S. Xia, "Variational autoencoders for deforming 3d mesh models," in *IEEE conference on computer vision and pattern recognition*, pp. 5841–5850, 2018.

[111] K. Zhou, J. M. Snyder, X. Liu, B. Guo, and H.-y. Shum, "Large mesh deformation using the volumetric graph laplacian," Oct. 23 2007. US Patent 7,286,127.

[112] O.-C. Au, C.-L. Tai, L. Liu, and H. Fu, "Dual Laplacian Editing for Meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 3, pp. 386–395, 2006.

[113] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel, “Laplacian surface editing,” in *ACM SIGGRAPH Symposium on Geometry Processing*, pp. 175–184, 2004.

[114] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum, “Mesh editing with Poisson-based gradient field manipulation,” in *ACM SIGGRAPH*, pp. 644–651, 2004.

[115] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel, “Interactive multi-resolution modeling on arbitrary meshes,” in *Conference on Computer Graphics and Interactive Techniques*, pp. 105–114, 1998.

[116] H. Maron, M. Galun, N. Aigerman, M. Trope, N. Dym, E. Yumer, V. G. Kim, and Y. Lipman, “Convolutional neural networks on surfaces via seamless toric covers.,” *ACM Trans. Graph.*, vol. 36, no. 4, pp. 71–1, 2017.

[117] A. Sinha, A. Unmesh, Q. Huang, and K. Ramani, “Surfnet: Generating 3d shape surfaces using deep residual networks,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6040–6049, 2017.

[118] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry, “A papier-mâché approach to learning 3d surface generation,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 216–224, 2018.

[119] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 652–660, 2017.

[120] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert, “Pcn: Point completion network,” in *2018 International Conference on 3D Vision (3DV)*, pp. 728–737, IEEE, 2018.

[121] Y. Yang, C. Feng, Y. Shen, and D. Tian, “Foldingnet: Point cloud auto-encoder via deep grid deformation,” in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 206–215, 2018.

[122] L. P. Tchapmi, V. Kosaraju, H. Rezatofighi, I. Reid, and S. Savarese, “Topnet: Structural point cloud decoder,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 383–392, 2019.

[123] M. Liu, L. Sheng, S. Yang, J. Shao, and S.-M. Hu, “Morphing and sampling network for dense point cloud completion,” in *AAAI conference on artificial intelligence*, vol. 34, pp. 11596–11603, 2020.

[124] S. Yang and S. Scherer, “Cubeslam: Monocular 3-d object slam,” *IEEE Transactions on Robotics*, vol. 35, no. 4, pp. 925–938, 2019.

[125] L. Nicholson, M. Milford, and N. Sünderhauf, “QuadricSLAM: Dual quadrics from object detections as landmarks in object-oriented SLAM,” *IEEE Robotics and Automation Letters*, vol. 4, no. 1, pp. 1–8, 2018.

[126] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, “3d-r2n2: A unified approach for single and multi-view 3d object reconstruction,” in *European Conference on Computer Vision (ECCV)*, pp. 628–644, Springer, 2016.

[127] M. Tatarchenko, A. Dosovitskiy, and T. Brox, “Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs,” in *IEEE International Conference on Computer Vision (ICCV)*, pp. 2088–2096, 2017.

[128] G. Riegler, A. Osman Ulusoy, and A. Geiger, “Octnet: Learning deep 3d representations at high resolutions,” in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3577–3586, 2017.

[129] G. Riegler, A. O. Ulusoy, H. Bischof, and A. Geiger, “Octnetfusion: Learning depth fusion from data,” in *IEEE International Conference on 3D Vision (3DV)*, pp. 57–66, 2017.

[130] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser, “3dmatch: Learning local geometric descriptors from rgb-d reconstructions,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1802–1811, 2017.

[131] Z. Chen and H. Zhang, “Learning implicit fields for generative shape modeling,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5939–5948, 2019.

[132] 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*, vol. 6, no. 2, pp. 2461–2468, 2021.

[133] E. Zobeidi, A. Koppel, and N. Atanasov, “Dense Incremental Metric-Semantic Mapping for Multi-Agent Systems via Sparse Gaussian Process Regression,” *IEEE Transactions on Robotics (TRO)*, vol. 38, no. 5, pp. 3133–3153, 2022.

[134] K. Genova, F. Cole, A. Sud, A. Sarna, and T. Funkhouser, “Local deep implicit functions for 3d shape,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[135] C. M. Jiang, A. Sud, A. Makadia, J. Huang, M. Niessner, and T. Funkhouser, “Local implicit grid representations for 3d scenes,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[136] C.-H. Lin, C. Wang, and S. Lucey, “SDF-SRN: Learning Signed Distance 3D Object Reconstruction from Static Images,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[137] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein, “Implicit neural representations with periodic activation functions,” in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 7462–7473, 2020.

[138] V. Sitzmann, M. Zollhöfer, and G. Wetzstein, "Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations," in *Advances in Neural Information Processing Systems*, 2019.

[139] Q. Xu, W. Wang, D. Ceylan, R. Mech, and U. Neumann, "DISN: Deep Implicit Surface Network for High-quality Single-view 3D Reconstruction," in *Advances in Neural Information Processing Systems*, pp. 492–502, 2019.

[140] R. Martin-Brualla, N. Radwan, M. S. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth, "NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections," *arXiv preprint arXiv:2008.02268*, 2020.

[141] K. Schwarz, Y. Liao, M. Niemeyer, and A. Geiger, "GRAF: Generative Radiance Fields for 3D-Aware Image Synthesis," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 20154–20166, 2020.

[142] J. Ángel Cid and F. A. F. Tojo, "A Lipschitz condition along a transversal foliation implies local uniqueness for ODEs," *Electronic Journal of Qualitative Theory of Differential Equations, arXiv:1801.01724*, vol. 36, no. 4, pp. 1–13, 2018.

[143] B.-S. Hua, Q.-H. Pham, D. T. Nguyen, M.-K. Tran, L.-F. Yu, and S.-K. Yeung, "SceneNN: A scene meshes dataset with annotations," in *IEEE International Conference on 3D Vision (3DV)*, pp. 92–101, 2016.

[144] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[145] N. Atanasov, B. Sankaran, J. L. Ny, G. Pappas, and K. Daniilidis, "Nonmyopic View Planning for Active Object Classification and Pose Estimation," *IEEE Transactions on Robotics (T-RO)*, vol. 30, no. 5, pp. 1078–1090, 2014.

[146] J. A. Placed, J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. A. Castellanos, "A Survey on Active Simultaneous Localization and Mapping: State of the Art and New Frontiers," *IEEE Transactions on Robotics*, pp. 1–20, 2023.

[147] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *Advances in Neural Information Processing Systems (NeurIPS) Autodiff Workshop*, 2017.

[148] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[149] M. Matl, "Pyrender." https://github.com/mmatl/pyrender, 2019.

[150] S. Salomon, G. Avigad, A. Goldvard, and O. Schütze, "Psa–a new scalable space partition based selection algorithm for moeas," in *EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II*, pp. 137–151, Springer, 2013.

[151] D. P. Hardin and E. B. Saff, “Discretizing manifolds via minimum energy points,” *Notices of the AMS*, vol. 51, no. 10, pp. 1186–1194, 2004.

[152] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari, “Accelerating 3d deep learning with pytorch3d,” *arXiv:2007.08501*, 2020.