UNIVERSITY OF CALIFORNIA SAN DIEGO

Scene-Level Neural Implicit Surface Representations Using Signed Directional Distance Functions

A thesis submitted in partial satisfaction of the
requirements for the degree Master of Science

in

Engineering Sciences (Mechanical Engineering)

by

Hojoon Shin

Committee in charge:

Professor Nikolay Atanasov, Chair
Professor Henrik Christensen
Professor Sylvia Herbert

2023

The thesis of Hojoon Shin is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

DEDICATION

I dedicate this work to my parents and family. For all their love and support that has pushed me to this achievement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Nikolay Atanasov for his support as my advisor and chair of the committee. All of this would have been impossible without his invaluable advice and boundless patience. I would also like to acknowledge Alex Paskal, Leo Liu, and Zhirui Dai, for working alongside me throughout this project. Finally, I would like to thank the members of the Existential Robotics Laboratory, who have all provided me with so much support throughout my time at the lab.

Chapter 3 is, in part, coauthored with Paskal, Alex; Liu, Peiran; Dai, Zhirui; Atanasov, Nikolay. The thesis author was the primary author of this chapter.

ABSTRACT OF THE THESIS

Scene-Level Neural Implicit Surface Representations Using Signed Directional Distance Functions

by

Hojoon Shin

Master of Science in Engineering Sciences (Mechanical Engineering)

University of California San Diego, 2023

Professor Nikolay Atanasov, Chair

Recent advances in the field of neural implicit surface representation has led to the use of signed distance functions as a continuous representation of complex surfaces. Though signed distance functions allow for a very simple and concise mapping of the environment, rendering these surfaces from signed distance functions are non-trivial and require iterative methods such as ray marching, leading to high render times. The inclusion of direction in these functions have been proposed and applied to various object-level applications, but have yet to be proposed for scene-level applications. This thesis explores the idea of scene-level directional distance functions and some of the problems that it faces. It then proposes three different methods through

which the directional distance function can be implemented and potential solutions to some of the drawbacks of the directional distance function.

Chapter 1

Introduction

1.1 Background

Recent developments in robotics have allowed robots to be placed into operation in a larger scale than ever before. Autonomous mobile robots (AMRs), in particular, have risen to popularity for both commercial (i.e. delivery robots) and private (i.e. robot vacuum) applications. One of the fundamental problems in the study of the AMRs is the reconstruction of expressive maps of an explored environment from robot observations. Mapping allows a robot to not only conduct key functions such as localization, but also generate representations that can be then further analyzed for future use. Numerous mapping algorithms and surface representation techniques have been developed for purposes ranging from scene reconstruction to large scale hierarchical semantic mapping.

There are largely two primary methods of surface representation. Explicit surface representation is one of the most commonly used type of map representation. Explicit surface representation refers to techniques that define a surface as the range of a parameterized function [8]. It is most commonly represented as a set of vertices and faces, making them very intuitive and easy to manipulate. However, such representations require the surface to be discretized in the form of voxels or polygons, leading to a high memory requirement and discontinuities between patches. An alternative to the explicit surface representation is the implicit surface representation. The implicit representation, such as the signed distance functions (SDF), defines

a volumetric function $g : \mathbb{R}^3 \rightarrow \mathbb{R}$. This function is called an implicit function, with the surface S defined as its zero level set, $S = \{p \in \mathbb{R}^3 \rightarrow g(p) = 0\}$. The SDF is a special formulation of an implicit function where in which the function returns the distance to the closest surface. This formulation theoretically allows for a continuous representation of any arbitrary object, but the objects in the real world cannot be represented analytically due to their complexity. Hence, the SDF was typically discretized and voxelized, which led to similar problems faced in explicit representations.

Fortunately, the development of deep learning in recent years has unlocked the field of neural implicit surface representations that are capable of modeling detailed surfaces without discretization. By utilizing multi-layered perceptrons (MLPs) to represent the implicit function, the complex implicit representations of real life objects could be captured. These methods, however, also faced another problem: rendering. Due to the nature of implicit functions, the surface of the object or scene cannot be extracted directly and must use iterative techniques, such as ray marching, to find the zero level set. In a neural representation, this requires multiple forward passes through the network, which can be computationally expensive. In addition, iterative algorithms are only able to compute the distance to a specified error tolerance, which meant that the estimation itself contains error and may even actually terminate rays that pass near a surface but does not actually hit the surface. Another major problem is that these neural implicit models does not allow for incremental or local updates to the map. Because all input data affects all the weights in the neural network, every observation will modify the map estimation regardless of how distant the map and the data sample may be from each other.

In order to resolve these limitations, we propose two solutions. First, to resolve the problem rendering, we suggest the incorporation of direction in scene-level SDFs. These SDFs with direction will be henceforth collectively referred to as directional distance functions (DDF). Incorporating direction in the SDF allows for depth estimation in a single forward pass, without the need for iterative methods of rendering. Second, to solve the problem of local updates, we propose the use of neural feature grids to encode the local area using a combination

of the neighboring features. This means that each data sample only alters the neighboring features, allowing for updates in just a local region. We combine both of these methods to formulate feature-based signed directional distance functions (FSDDF). There are two different implementations of the DDF presented in this paper. We first present the first scene-level DDF formulation without the use of feature grids. We then extend this method to one that utilizes a 2D feature grid by decoupling direction from position, and another that combines both the direction and position to utilize a 3D feature grid. This is the first implementation of DDFs at the scene-level and opens the possibility of fast implicit mapping for continuous surface representations.

The remainder of this paper will be structured as follows. First we discuss existing prior works regarding SDFs at both the object and scene-level along with those regarding DDFs. We then formulate the specified problem mathematically and discuss the technical approaches taken. Finally, we present the results of our implementation along with quantitative and qualitative analysis.

1.2 Related Works

1.2.1 Implicit Object Representations

The first paper to utilize SDF using neural networks was DeepSDF in [13], which utilized an autoencoder/autodecoder structure encoded a library of shape priors into a latent vector. The latent vector was then concatenated with the queried pose to estimate the SDF. This formulation allowed the network to learn the SDF without any discretization by optimizing the latent vector with a simple $L1$ loss function. This sprouted many extensions to the method, such as [22], which combined multiple-view stereo networks and SDF networks to allow deep image features to improve the SDF estimation, and [11], which utilizes disentangled latent spaces to allow for reconstruction of articulated objects with high levels of deformation.

Implicit geometric regularization (IGR) [5] proposed an addition to the DeepSDF method

and introduced the eikonal constraint in the loss function. The method stated that point normals could be utilized to encourage the gradient of the SDF at the point to be similar to that of the point normal. The eikonal loss was also contained in the loss function in the form of the expectation of the eikonal loss. IGR demonstrated superb accuracy when used for estimating SDFs at the object level and presented the first use of the eikonal constraint in the loss function for learning SDFs.

Neural radiance fields (NeRF) [10] and its extension paper [2] presented another idea of neural implicit object representation by utilizing volume rendering to estimate volumetric radiance fields. By utilizing volume rendering techniques, the model was capable of dealing with discontinuity along the queried ray, as the method involves sampling multiple points along a ray to estimate the RGB color and volume density. However, since these methods were designed for novel view synthesis, its estimated surfaces contain significant amounts of noise. NeuS [18] adapted this volume rendering methodology to estimate SDF rather than radiance fields.

1.2.2 Implicit Scene Representation

A field that was developing concurrently with the object-level SDF was the representation of scenes using these implicit formulations. iMAP, developed in [16], was one of the first methods that utilized implicit surface representations for scene representation. iMAP utilized a single MLP to learn the scene geometry in real time from a handheld RGB-D camera to simultaneous estimate the camera pose and build a dense 3D model of the scene occupancy with color. This was achieved with no prior data and demonstrated a 10 Hz tracking frequency and 2 Hz map updates. Building further upon the idea of real-time operation, iSDF [12] utilized a very similar pipeline to estimate SDFs instead of occupancy using self-supervision. Though the iSDF demonstrated high accuracy at extremely fast speeds, the model struggles to scale to multi-room datasets, as each input to the network updated the network parameters for the entire map and solely relied on keyframe windows to maintain accuracy on past data. In addition, most of the datasets used by these papers consists of single rooms with a distinct lack of narrow corridors and thin walls.

NICE-SLAM [24] proposed the idea of using a neural implicit feature grid to estimate the occupancy and color from RGB-D inputs. NICE-SLAM utilized feature grids that stored feature vectors at each grid index. For every queried point, the neighboring grid point features were trilinearly interpolated to output the final feature vector, which was then decoded by a decoder MLP. This was then extended by [23] and [21] to work with RGB images and to estimate SDF rather than occupancy respectively. Though these methods are able to operate in real time, they suffer from the limitations of SDFs in that rendering of views requires multiple forward passes through the network, making it highly inefficient.

1.2.3 Directional Distance Prediction

An intuitive step up from regular SDF is to incorporate direction. The key idea here is that the the viewing direction is added to the distance function, so that the function returns the distance to the surface along the specified viewing direction. For simplicity, we will henceforth refer to these functions as directional distance functions (DDFs). As this field has only been proposed very recently, there are only a very limited number of works that fit our application.

The first implementation of DDF was the signed directional distance function (SDDF) [25]. Developed as an extension of DeepSDF, SDDF is designed to learn shape representation for categorical objects and uses an MLP to learn the signed distance for a given object latent code, pose, and viewing direction. The paper also proposes a novel dimension reduction algorithm that helps to simultaneously reduce the number of input dimensions and embed the eikonal constraint into the model structure, hence removing the need for a separate eikonal loss. Though effective, the same method cannot be applied to scene level data as the dimension reduction technique does not account for occlusion.

Another very similar idea was the probabilistic directed distance fields (PDDF) [1]. PDDF is a formulation of DDFs where the measured distance is always looking forward along the viewing direction. Due to this definition, the paper takes special steps to deal with discontinuities arising from occlusion through applying a probability distribution along the ray that is a mixture

of delta functions. This allows the network to output a depth prediction based on the most probable continuous piece of the function along the ray. However, the network was again only applied for object level and hence does not take into consideration the discontinuity along the direction dimension.

Hybrid directional and signed distance function (HDSDF) [20] proposed utilizing both SDF and DDF in its formulation for depth estimation. Since SDF and DDF are related in that $SDF(p) = \min_{v} DDF(p,v)$ for a given pose p amd viewing direction v, HDSDF learns to simultaneously predict DDF and SDF values. For the DDF prediction, the network learns to output DDF estimations along with ray-hitting probability from points and directions on a unit sphere that fully encloses the object. Fast inverse rendering (FIRe) [19], which was written by the same authors, extends the HDSDF to include two sets of 2D feature grids and can be considered to be the most analogous to the method implemented in this paper. FIRe decomposes the 3D input (x,y,z) in SDFs into three 2D feature grids, while for DDFs, the 6D input (x,y,z,dx,dy,dz) is decomposed into 15 2D feature grids. Then a higher dimensional feature is interpolated for each feature grid under the assumption that the learned function is linear in a higher dimensional space. Though this method demonstrated exceptional accuracy and runtimes, it cannot be extended to the scene level, as the queried DDF is bounded to a unit sphere that surrounds the entire object.

1.2.4 Discontinuity Approximation Using Neural Networks

A key factor to training neural networks to learn DDFs is the ability of the network to approximate discontinuity. There are two major types of discontinuity in DDFs that are worth discussing. First is the discontinuity along a fixed viewing direction resulting from occlusion. This problem was also present at the object level and hence was tackled by previous work such as [1] using probability distributions of mixtures of delta functions. The second is the discontinuity in depth measurement due to the viewing direction from a fixed position. This is a major problem that become particularly important in scene-level applications because each ray from the LiDAR may be hitting entirely different objects, resulting in a piecewise discontinuous output.

There has been extensive work conducted regarding the approximation of piecewise continuous functions using neural networks. Ismailov [7] proved that a three-layered neural network is capable of representing any multivariate function, including discontinuous functions. However, the paper explicitly states that the argument is strictly an existence result and that there is no guarantee that existing learning algorithms are capable of converging to the desired function. The primary method of dealing with such piecewise continuities have been to utilize classification identify the piece at which the queried point belongs. Bernholdt et al. [3] suggested clustering and classifying the input data prior to conducting regression on the data. The clustering and classifying front end could either be parametric or non-parametric, while a neural network was used for the regression. Cho et al. [4] partitioned the input data under the assumption that the input data only affects a single piece of the piecewise continuous function. Using this assumption, a $k \times k$ weak Jacobian block matrix can be computed, for a piecewise continuous with k partitions. Each block of the represents the Jacobian of each corresponding piece, which can be used to optimize the network parameters corresponding to the specific continuous piece. However, both of these methods cannot be applied to our circumstance, as it is impossible to cluster the surface scan of an unknown environment a-priori.

Another approach that was to utilize jump functions to model the discontinuity. Selmic et al. [14] utilized sigmoid jump blocks, which appends a regular MLP with nodes that pass through sigmoid jump basis functions as an activation. These sigmoid jump blocks takes the position of the discontinuity c as a parameter in order to network to apply the jump at the appropriate position. Llanas et al. [9] also utilized step functions to model the discontinuities while combining the piecewise continuous functions together to for a single continuous function. By decoupling the approximation of the discontinuity and the continuous function itself, the network was able to effectively estimate the piecewise continuous function accurately. Both of these methods, however, requires prior knowledge regarding the position of the discontinuities, which is impossible in an unknown environment.

Chapter 2

Problem Formulation

2.1 Definitions

We first illustrate the definition of DDFs that will be used in this paper. The DDF is defined with a point $p \in \mathbb{R}^n$ and the direction $v \in \mathscr{S}^{n-1}$ input. The output of the DDF is then the distance to the surface along the queried direction. Please note that though the definition in Eq.2.1 uses a unit vector to represent the viewing direction, the rotation can be represented using any other forms, such as roll-pitch-yaw.

We also adopt a similar direction convention that is used in [25], where the network estimates the direction to the closest surface along the viewing ray with negative depth representing a distance behind the field-of-view (FOV). We utilize a modified version of the definition, as shown in Eq.2.1, where the set S, O refers to the set of points on the surface and within the obstacle respectively. Note that with the definition, the queries that are within the obstacle are now measuring the distance to the surface behind the FOV. This allows the depth measurement along the ray pointing into the object surface to be continuous, allowing the network to have an easier time modeling the surface. Though this still causes rays originating from the insider of the surface pointing out to have a discontinuous depth measurement, this is an extremely unlikely circumstance in our problem formulation, as an operating robot would always be positioned outside an obstacle. An illustration of both the SDDF and PDDF interior point definition is shown in Fig. 2.1.

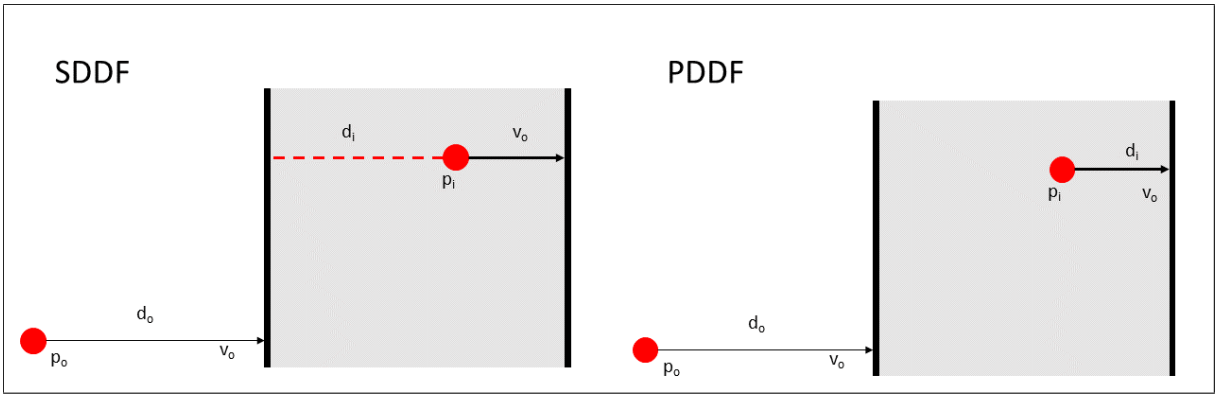


Figure 2.1. Difference in the modified SDDF and PDDF definition for interior sampling. p_o, p_i represents the (x,y) coordinate of the sample point, v_o represents the viewing direction, and d_o, d_i the depth label of the query.

$$SDDF(p,v) = \begin{cases} 0, & \text{if } p \in S \\ \min\{d > 0 \mid p + dv \in S\}, & \text{if } p \notin O \\ \max\{d < 0 \mid p + dv \in S\}, & \text{if } p \in O \end{cases} \tag{2.1}$$
$$p \in \mathbb{R}^n,\ v \in \mathscr{S}^{n-1},\ ||v||_2 = 1$$

In additional, though not directly enforced, the estimated SDDF would ideally adhere to the directional eikonal constraint, where the gradient of the SDDF along a viewing direction is fixed to be -1. This can be mathematically defined as:

$$\nabla_p SDDF(p,v)^T v = -1 \tag{2.2}$$

2.2 Problem Statement

For the application of DDFs, we consider a randomly positioned robot in a 2D room environment. The pose of the robot is known and provided in the form of (p,v), where $p = (x,y)$ and $v = (dx,dy)$ is the orientation of the robot represented in unit vectors. At each step, we

receive depth information from a 2D LiDAR in the form of d, where the number of depth measurements per scan j depends on the FOV and resolution of the sensor. In order to mimic the FOV of a depth camera, we limit the LiDAR FOV to 60° with an angular resolution of 0.5°. We refer to the input dataset as D_i, where i is the index of the input data (p_i, v_i, d_i). Hence, a dataset with k number of poses and j scans per pose will contain a total of $n = j \times k$ samples. It should be noted that the p_i and v_i samples in the dataset are not unique, as there may be two entries at the sample pose with different viewing directions or vice versa. The aim is then to learn a function that represents the map of the surrounding environment. Specifically, we seek the parameters θ such that:

$$\min_{\theta}\{\sum_{i}|SDDF(p_i, v_i\ ;\ \theta) - d_i|\} \tag{2.3}$$

The surrounding environment is assumed to be an enclosed indoor home environment with multiple rooms. This multiple-room setup creates a significant amount of occlusion and thin walls, which both SDFs and SDDFs tend to struggle with. A sample input data is shown in Fig.2.2. For evaluation of the results, we utilize a simple L1 error between the estimated depth and the ground truth input from the scans. In the following section we discuss three different methods to tackle this problem.

2.3 Datasets

There will be three different room environments that the methods will be evaluated upon. First, we utilize a vary simple initial testing dataset that consists of a simple square room with dimensions 5×5 meters. As a simple geometric shape of a square does not contain any occlusion while also containing discontinuities such as the sharp corners, it creates a decent baseline from which we can evaluate the performance.

The second room environment that was used to test the methodology was a custom made room dataset with a couple distinct points. First, the dataset contains three rooms with open

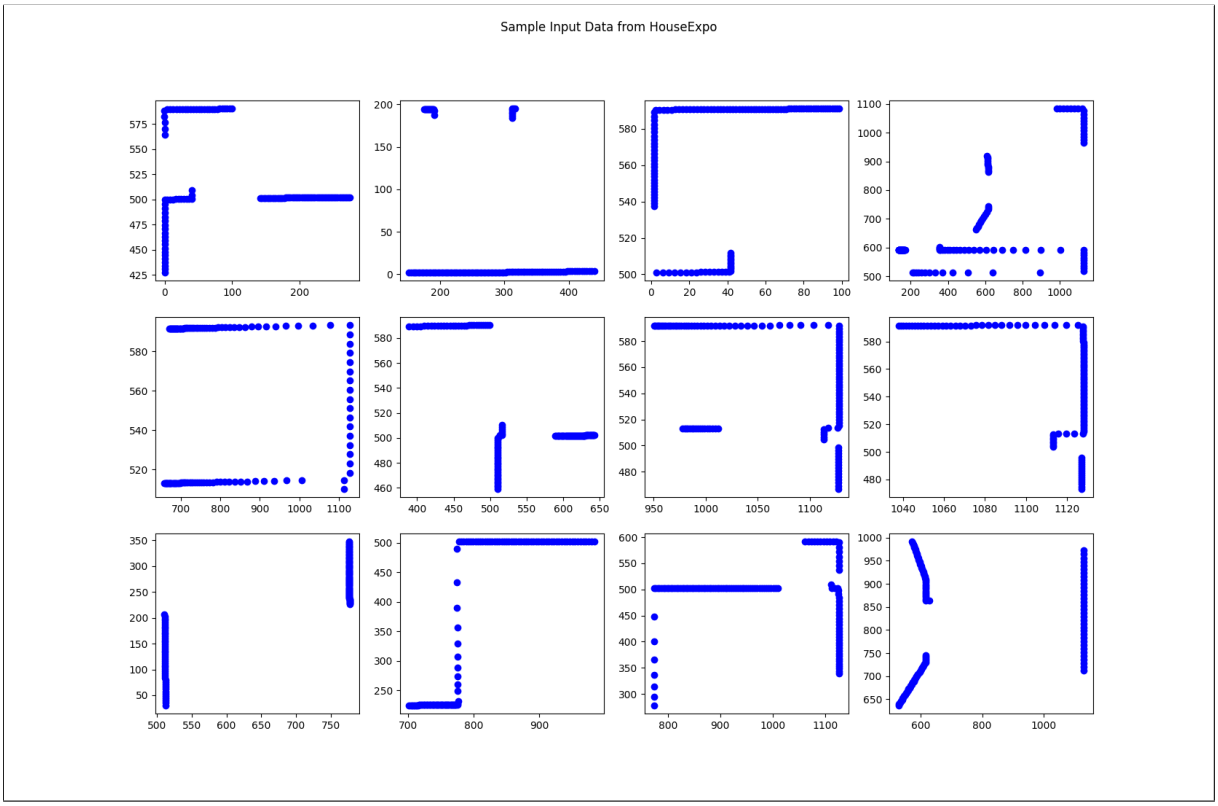
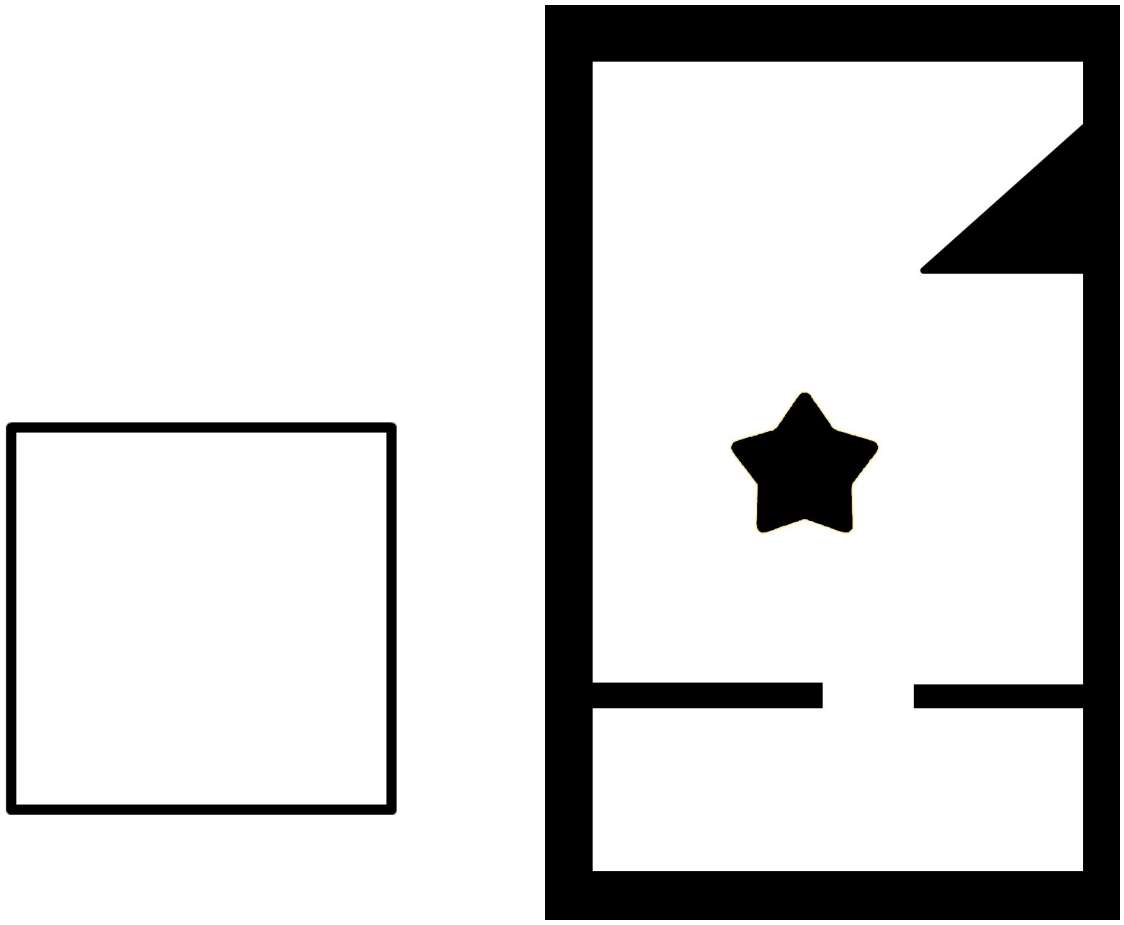


Figure 2.2. Example input from the HouseExpo dataset taken with a 60° FOV and angular resolution of 0.5°

doors, meaning that the rooms contains thin walls that existing works have not tested extensively on. The room also contained a large star in the center of the room, that causes occlusion and creates great amount of discontinuity in the LiDAR scans. Both these factors make the shape room a very difficult dataset to learn. The room was also relatively larger, with a map dimension of 17×11 meters.

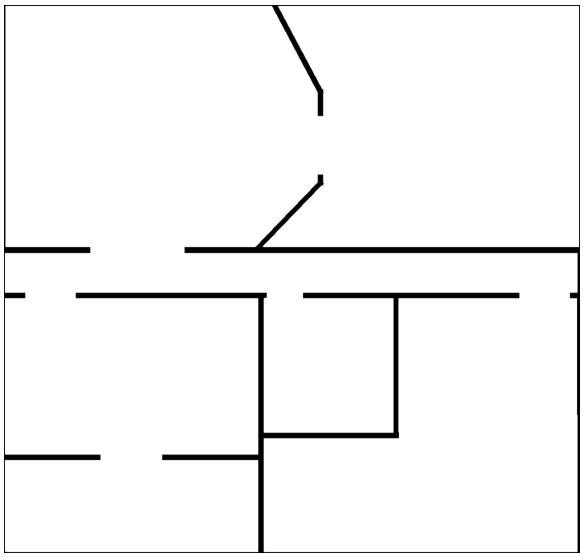
As a final test, we utilize the room layouts that are part of the HouseExpo [17].The complete dataset contains 35,126 2D floor plans of multiple rooms in each floor plan. We chose a room from the dataset with many discontinuities and corridors to truly test the accuracy of the estimation. The layouts for all the rooms discussed are shown in Fig.2.3.

We randomly sample the free space for 700 different pose samples for the box dataset and 1000 poses for the shape room and HouseExpo dataset. For each pose, we generate a 2D LiDAR scan with the sensor parameters stated previous. Since each of the scans contain 120 sample rays, the box dataset contains 84,000 input data points, while the shape room and HouseExpo contains 120,000. These points can then be further expanded using data augmentation techniques that will be described later in this paper. For the evaluation of the results, 100 additional random poses that were not used for the training dataset were sampled from each of the environments and a similiar laser scan was generated for each of the test poses.



(a) Box Dataset

(b) Shape Room Dataset



(c) HouseExpo Dataset

Figure 2.3. Full layout of the three different datasets.

Chapter 3

SDDF without Feature Grid

3.1 Network Architecture

We first present a method of estimation SDDF without the use of feature grids, the model of which is shown in Fig.3.1. We will refer to this method as the SDDF for the rest of the paper. As we can see, there are two major portions in the model. First is the encoder network, which is a simple five-layer MLP with leaky ReLU activation. Each hidden linear layer in the encoder contains 256 nodes. The encoder takes in the position and direction (p, v) and encodes the information into a feature vector f. Here, the position p is taken in Cartesian coordinates $p = (x, y)$ while the direction is assumed to be in the form of a unit vector $v = (dx, dy)$. Once encoded, the feature vector is concatenated with the original input (p, v) and then passed to the decoder MLP, which is identical to the encoder MLP, except that the output is now the SDDF.

3.2 Network Training and Inference

For training the network, stochastic gradient descent was used with an Adam optimizer. We only require a very simple loss function using the depth loss $\mathscr{L}_{depth}$ with a simple L1 loss as follows:

$$\mathscr{L}_{depth} = \frac{1}{n_{input}} \sum_{i=1}^{n_{input}} |SDDF(x_i, y_i, v_i) - d_i| \tag{3.1}$$

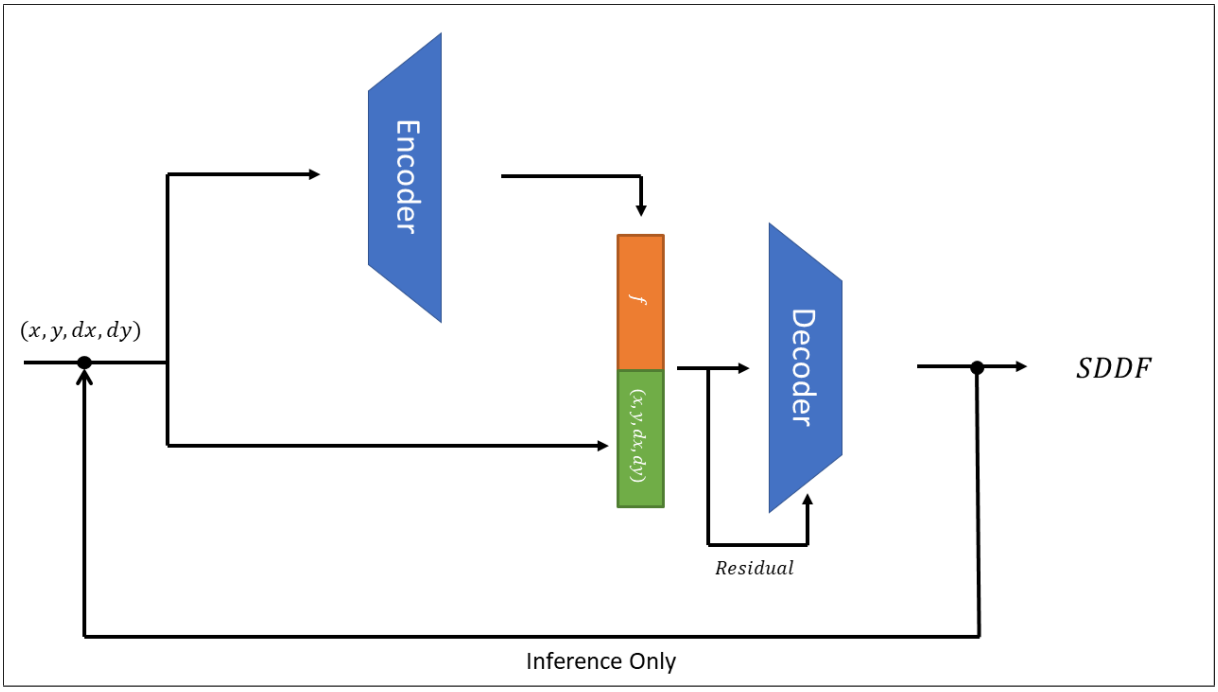


Figure 3.1. Architecture of the SDDF with no feature grid. Given coordinates (x, y) and directional unit vector (dx, dy), the encoder computes a feature vector f which is then concatenated with the original 4D input to decode the final SDDF estimate.

where the n_{input} is the total number of samples in the training data. The results of the training are shown in Sec.6.4.

When evaluating the network, we utilize double-forward pass technique to leverage the fact that the SDDF follows the eikonal constraint along a viewing direction. To this end, we first run the input (p, v) through the network to obtain the first estimate of the depth d_{coarse}. Using this estimated depth, we move along the ray a point closer to the surface and query the network again for to return d_{fine}. We then combine these two estimations together to compute the final depth estimate. Specifically,

$$d_{final} = \gamma_{coarse} d_{coarse} + (1 - \gamma_{coarse}) d_{fine} \tag{3.2}$$

The hyperparameter γ_{coarse} is the weighting factor $0 \leq \gamma_{coarse} \leq 1$ of the coarse estimate with respect to the fine estimate.

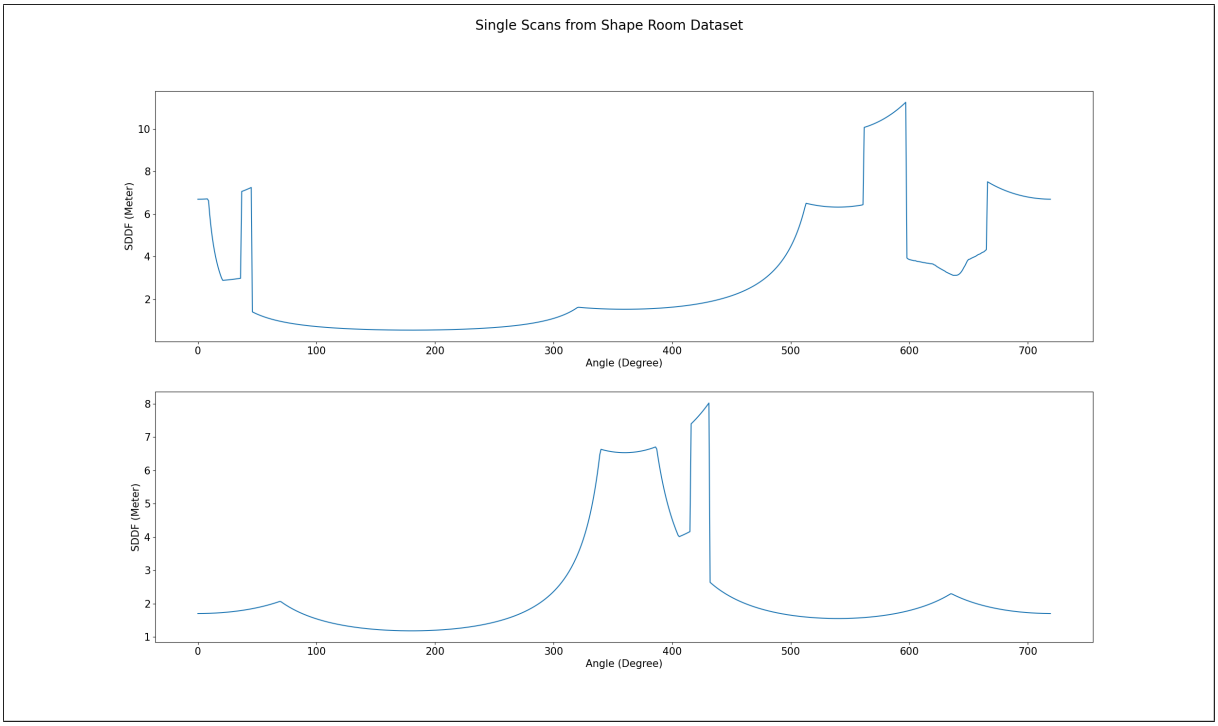


Figure 3.2. Two sample input scans from the shape room dataset. The horizontal axis represents the ray index of a 360° LiDAR scan with a 0.5° resolution, resulting in a total of 720 points. The piecewise discontinuity can be clearly observed from both scans.

3.3 Discontinuity Treatment

As mentioned previously, there are two types of discontinuity that network must be able to handle for scene-level SDDF. In Sec.1.2.4, we discussed some prior works regarding approximation of piecewise discontinuous functions and Sec.1.2.3 contained prior attempts to tackle discontinuity with regards to the depth discontinuity along a single ray. Another source of discontinuity that has not been addressed by previous work at the object level is the discontinuity due to the changes in viewing direction. This becomes immediately evident when simply plotting the depth values from a single LiDAR scan. Based on the surface that each individual ray hits, the depth measurement may change drastically, as shown in Fig.3.2.

To evaluate the different methods of approximating discontinuity, we implemented neural network approximations of a piecewise discontinuous quadratic function. As mentioned in section 1.2.4, many of the existing works in this field is ill-suited for our problem situation.

However, the sigmoid jump block in particular can be adapted by assuming that the jump position is a zero. In other words, the sigmoid jump block *SigJump* will have the effective formulation as follows,

$$SigmoidalJump(x) = \sum_{k=0}^{L} \alpha_k \sigma(m_k x + n_k) + \sum_{k=0}^{N} \alpha_k \varphi_k(x) \tag{3.3}$$

where α_k is the weight of the k^{th} node and the σ is a sigmoid activation. The φ_k is the sigmoidal jump approximation function defined as follows

$$\varphi_k(x) = \begin{cases} 0, & \text{for } x < 0 \\ (1 - e^{-x})^k, & \text{for } x \geq 0 \end{cases} \tag{3.4}$$

For analysis we utilize a sigmoidal jump block with five different jump nodes, or $k = 5$. We also include a parameterized ReLU activation, where in which the slope of the negative portion of the leaky ReLU was a parameter that was optimized by the network. Instead of have just a single parameter, each node in the layer would receive its own parameter, meaning that each node would have a differently sloped leaky ReLU activation. We present the results of these tests in Sec.6.1.

3.4 Data Sub-Sampling

From our formulation, for each set of input data (p_i, v_i), we can acquire j number of depth measurements, the directions of which are determined by the sensor type. For example, if using a 2D LiDAR with a 60° FOV and 0.5° resolution, we acquire 120 data points per p_i, v_i pair. While we could directly utilize these measurements, there is still additional information that can be exacted to allow for more information in the space. Since each depth measurement means that the ray passes through free space, we can further sample points along the ray to augment our dataset while also indirectly implying the eikonal constraint in our input data.

To this end, we adapt the sub-sampling method that was used in [12], for fast sub-sampling of our data. We use both the stratified and Gaussian sampling along the ray. The stratified sampling takes in two parameters: d_{min},which indicates the minimum distance along the ray the stratified sampling takes place, and d_{delta}, which indicates the distance past the depth measurement the stratified sampling continues to. This helps to create additional samples with the same viewing direction from different poses. The Gaussian sampling is conducted by drawing from a normal distribution to create offset distances from the original depth measurement. The standard deviation of the Gaussian distribution σ is a hyper-parameter that can be tuned. In summary, given n_{strat} desired stratified samples, n_{gauss} desired Gaussian samples, the sub-sampling can be expressed in Alg.1,

Algorithm 1. Data Sub-Sampling

Given ray (p,v) with depth d and hyper-parameters $n_{strat}, n_{gauss}, d_{min}, d_{delta}, \sigma$
$\{d_{strat}\} \leftarrow range(d_{min}, d+d_{delta}, n_{strat})$
$\{d_{gauss}\} \leftarrow d \sim \mathcal{N}(0,\sigma)$
for d_{strat} in D_{strat} **do**
    $p_{strat} \leftarrow p + d_{strat}(cos(v), sin(v))$
    $D_{strat} \leftarrow (p_{strat}, v, d_{strat})$
end for
for d_{gauss} in D_{gauss} **do**
    $p_{gauss} \leftarrow p + (d_{gauss}+d)(cos(v), sin(v))$
    $D_{gauss} \leftarrow (p_{gauss}, v, d-(d_{gauss}+d))$
    $D_{supp} \leftarrow (p_{gauss}, -v, d-(d_{gauss}+d))$ ▷ For directional supplement only
end for

We can also utilize the same method to generate interior samples. Gaussian depths that are greater than zero in the above sampling methods indicates that the sampled point will be inside the detected obstacle. These are our first negative or interior samples that we can generate. Due to our definition of the SDDF, the viewing direction for these additional interior samples are the same as the exterior samples, except that the distances will now be negative. This is also reflected in the last line of the algorithm, as $d-(d_{guass}+d) \leq 0$ for $d_{guass} \geq 0$. Though generating more interior points using a larger σ would generate a wider range of data samples, it can potentially cause incorrect data samples, particular if the obstacle the robot is looking at is

a thin wall, hence causing the point sampled to be in free space rather than in occupied space. Hence, it is beneficial to keep the σ parameter sufficient small to limit such erroneous labels.

3.5 Directional Augmentation

Though the existing subsampling method is great for providing additional positional samples, additional directional variety is critical in training an SDDF network, especially if the dataset is along a trajectory where the variety in directions in the input data set is insufficient. In other words, for an input dataset consisting of a poses along with n number viewing directions and depths $\{p, v_j, d_j\}_{j=1,..,n}$, we want to augment the dataset m additional sets $\{p_l, v_l, d_l\}_{l=1,..,m}$, such that $v_i \neq v_l$.

Because these samples represent the distance to a surface in the given direction, the samples must generated must simultaneously pass through only free space and must hit a guaranteed surface point. This limits the potential samples that can be taken, as the only points that are certain to be on a surface are the surface points from the input data, given by $s_i = p_i + d_i v_i$. Another constraint is that the pose p_l must also lie in free space. There are a couple ways we can guarantee this. First, we could sample along existing rays, as any point along the input ray is in free space. However, doing so limits the range of augmented viewing directions and can potentially bias the input data. Therefore, to maximize the variety of data the augmentation provides, we utilize the sampled surface points to draw polygons to define areas of free space, henceforth referred to as safe polygons.

The safe polygon first chooses a closest surface points within a given scan to consider as vertices of the polygon. Let us refer to these vertices as $\{s_k\}_{k=1,...,a}$ Using these closest points along with the pose p, we define a polygon where in which points can be considered to be in free space. We can then sample m number of candidate points within this safe polygon and draw rays v_k to each of the k vertices. We must then verify that each of these candidate rays are not occluded, in other words, that the drawn polygon is convex. To do so, we choose one of the v_j as

an anchor and compute the angle θ_k between the anchor and the other directions. The polygon is guaranteed to be convex if the difference in angles are of the same sign. The summary of the algorithm is shown in Alg.2.

Algorithm 2. Data Augmentation

Given $S = \{s_i\}_{i=1,\ldots,n}$ surface points in a scan with pose (p_0, v_0)
Choose $C = \{s_k\} \in S$ where C are the a closest surface points
for point $p = (x, y)$ inside polygon drawn from C and p_0 **do**
    $v_k \leftarrow s_k - p$
    Compute θ_k for v_k w.r.t a chosen $v_1 \in V$
    $\delta\theta_k \leftarrow \theta_k - \theta_{k-1}$
    if $|\sum sign(\{\delta\theta_k\})| = a$ **then**
        $D_{aug} \leftarrow (p, v_k, d_k)$
    end if
end for

The final dataset D_{input} that is used to train the network can then be represented as

$$D_{input} = D_i \cup D_{strat} \cup D_{gauss} \cup D_{supp} \cup D_{aug} \tag{3.5}$$

resulting in a total of $n_{input} = n + (n_{strat} + 2 \times n_{gauss}) \times j \times k + n_{aug}$ datasamples in the dataset.

Chapter 3 is, in part, coauthored with Paskal, Alex; Liu, Peiran; Dai, Zhirui; Atanasov, Nikolay. The thesis author was the primary author of this chapter.

Chapter 4

SDDF with 2D Feature Grid

4.1 Network Architecture

We first present a scene level SDDF architecture with a 2D feature grid. For convenience, we will refer to SDDF with the feature grid as FSDDFs, where 2D FSDDF indicates a FSDDF with 2D feature grid. The overall network structure is shown in Fig.4.1. The first component of the network is the interpolator that holds the feature grid used to interpolate the feature vector f. We utilize a similar rectilinear feature grid used in [24, 23], except that we use a single feature grid rather than a hierarchical setup. While [19] included direction in the feature grid by pairing the dimensions in sets of two, we decouple the direction from the position so that the feature grid only takes in the position $p = (x, y)$ while the directional component is concatenated to the feature vector prior to the decoding. This means that the feature grid can be interpolated using bi-linear interpolation. Given a queried point (x, y) and the four neighboring feature grid coordinates and features $\{(x_1, y_1, f_{11}), (x_2, y_1, f_{21}), (x_1, y_2, f_{12}), (x_2, y_2, f_{22})\}$, the output feature f can be interpolated as follows:

$$\begin{aligned} f = \frac{1}{(x_2 - x_1)(y_2 - y_1)} \{ & f_{11}(x_2 - x)(y_2 - y) + f_{21}(x - x_1)(y_2 - y) \\ & + f_{12}(x_2 - x)(y - y_1) + f_{22}(x - x_1)(y - y_1) \} \end{aligned} \tag{4.1}$$

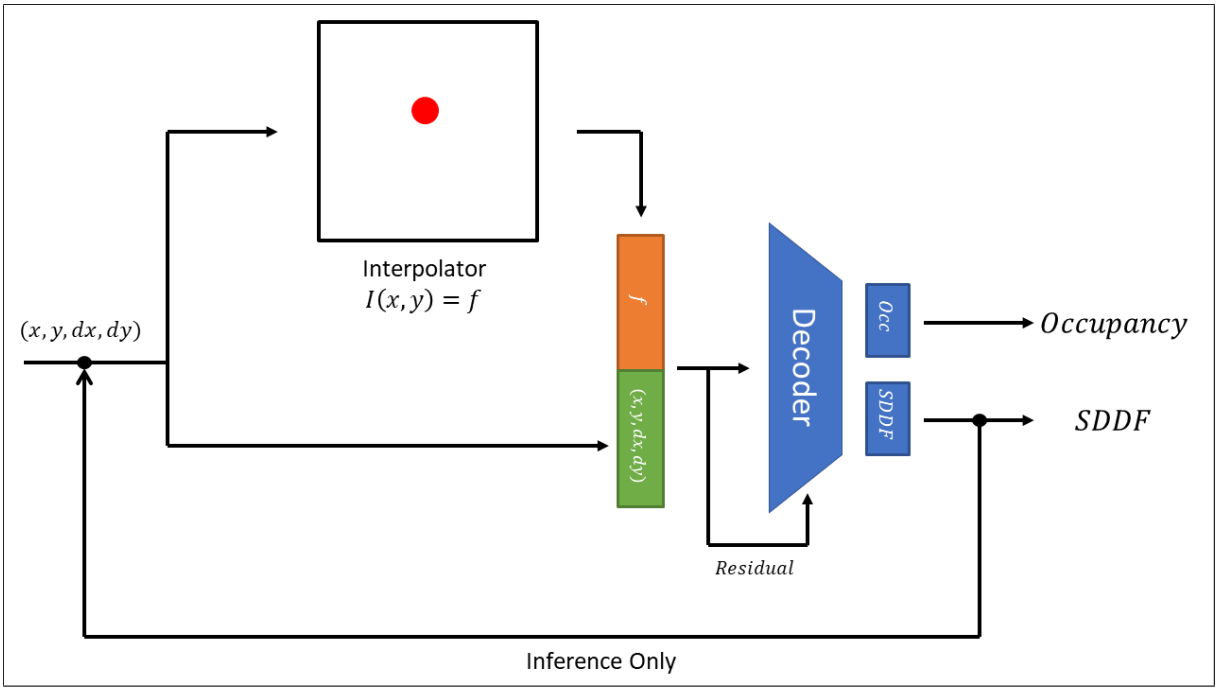


Figure 4.1. Architecture of the SDDF with a 2D feature grid. Given coordinates (x, y) and directional unit vector (dx, dy), the feature grid interpolates a feature vector f from just the position (x, y)

The resolution of the feature grid is a hyper-parameter that can be tuned depending on the environment and the variance of the input data. Though having a finer resolution can increase the accuracy of the SDDF estimation, there is a risk of the grid not being trained at all, which could lead to failure. Utilizing a feature grid enables two things: first, the network is now capable of making local updates to the map, where the observations only affect the grid coordinates relevant to the specific area, rather than the entire map. Second, the map is now subdivided to allow the feature to encode information in a very local region, a characteristic that we will take advantage of during the inference stage of the network.

The output feature is then passed to the decoder MLP to output the final SDDF estimate. The feature vector is concatenated with the position p_i and the direction $v_i = (dx, dy)$ such that $||v_i||_2 = 1$. Hence, the input dimension to the MLP becomes $c + 4$. The MLP consists of five hidden linear layers, with 256 nodes in each layer, and a leaky ReLU activation in between each layer. From various testing, we have observed that the leaky ReLU activation allowed for the

most accurate estimation of piecewise discontinuous functions in neural networks. This will be discussed in further detail in Sec.3.3. There is also a residual layer at the third linear layer, where the original inputs are concatenated to the input. This was a design based on the ResNet [6] design. There are two separate output layers to the decoder MLP. The first is a standard linear output layer that outputs the estimate of the SDDF value. The second, however, consists of a linear output layer followed by a sigmoid activation. This is because this output layer estimates the occupancy of the queried point. Estimating output layer complements the SDDF estimation as the occupancy determines the sign of the SDDF, allowing for a more accurate estimation of depth.

4.2 Occupancy Labels

Occupancy labels for the input data points can also similarly concurrently with the subsampling methods described in Sec.3.4. If the computed depth label is greater than the depth measurement to the surface, the occupancy would be 1, as the point would be within the obstacle. In our implementation, the stratified samples were used to generate free space data while the Gaussian samples were used to generate occupied data. One of the major problems with this method of interior sampling is that the parameter d_{delta} is not a completely accurate assumption. If, for example, d_{delta} was unnecessarily large, it is entirely possible for the sampled point to have exited through the surface on the other side of the object. If we were utilizing these methods at the object level, this would not be much of a problem. However, because we are working with scenes with narrow corridors and thin walls, this problem is a definite possibility. Hence, we must maintain the d_{delta} to be sufficiently small, which naturally greatly limits the depth into the surface at which we can sample.

There is also one additional method we can employ to generate additional occupancy information. Though the Gaussian samples generate samples in the surface, the sampled directions are all point from free space into the surface, meaning that there are no samples in the reverse

direction. Hence, in order to generate some data from within the surface, we utilize the same interior Gaussian samples, but with a flipped direction. However, since we cannot generate a depth label for this point, we only utilize the occupancy portion of the network. Doing so will allow the network to have knowledge regarding the occupancy boundary when the estimate is from within the surface. These supplemental interior samples will be henceforth referred to as directional supplement samples and can be seen in the last line of Alg.1.

4.3 Network Training and Inference

In order to train the feature grid and the decoder networks, take a two-step approach with initial pre-training followed by feature training. During the pre-training stage, all the parameters in both the feature grid and the decoder are optimized. Once the pre-training is complete, the weights of the decoder are frozen, while the grid features continue to be optimized. This freezing of the weights in the decoder allow the network to learn the environment purely through its features while the decoder remains unchanged.

We utilize two different loss functions to optimize the network. First, we utilize the same depth loss defined in Eq.3.1. For the occupancy, we utilize a binary cross entropy loss $\mathscr{L}_{BCE}$, as the occupancy estimation problem can be described as a simple binary classification problem. Given the output of the occupancy estimation as o_{pred} and occupancy label o_{gt}, the loss can be specifically calculated as follows:

$$\mathscr{L}_{BCE} = \frac{1}{n_{input}} \sum_{i=1}^{n_{input}} -w_i[o_{gt,i} log(o_{pred,i}) + (1 - o_{gt,i}) log(1 - o_{pred,i})] \tag{4.2}$$

Combining the two losses together, the total loss can be computed as follows:

$$\mathscr{L}_{total} = \lambda_{depth}\mathscr{L}_{depth} + \lambda_{occ}\mathscr{L}_{occ} \tag{4.3}$$

where $\lambda_{depth}, \lambda_{occ}$ are hyper-parameters representing the weights of the two components.

When running an inference on the network, we utilize the same double-forwad pass technique used in the SDDF formulation with no feature grids. However, due to the ability of the feature grid to represent highly accurate local information, we see an even greater increase in the accuracy of the estimation.

Chapter 5

SDDF with 3D Feature Grid

5.1 Network Architecture

The overall network structure is shown in Fig.5.1. Similar to the 2D FSDDF, we will refer to this method as the 3D FSDDF. Though very similar to the network model used in Chapter 4, there are a couple of key changes that differentiates the two. Instead of taking unit vectors to represent direction, we now utilize the yaw angle to represent v. By doing so, we decrease the number of input dimensions by one. In addition, the feature grid is now represented using a cube, as the feature now takes in the full (x, y, v) as an input. This results in a slight modification in the interpolation method utilized in the feature grid that will be discussed further below.

The feature f computed by the interpolation is also now no longer concatenated with the original input, but rather directly passed on to the decoder. As the feature grid now incorporates direction in its formulation, there is no longer a need to concatenate the initial inputs for the directional information. While the decoder architecture is largely the same, with linear layers and leaky ReLU activation, it no longer contains a separate output layers for occupancy and depth, but instead simply estimates the SDDF value. In addition, we utilize eight linear layers instead of five, with the residual layer at the fifth linear layer instead of the third.

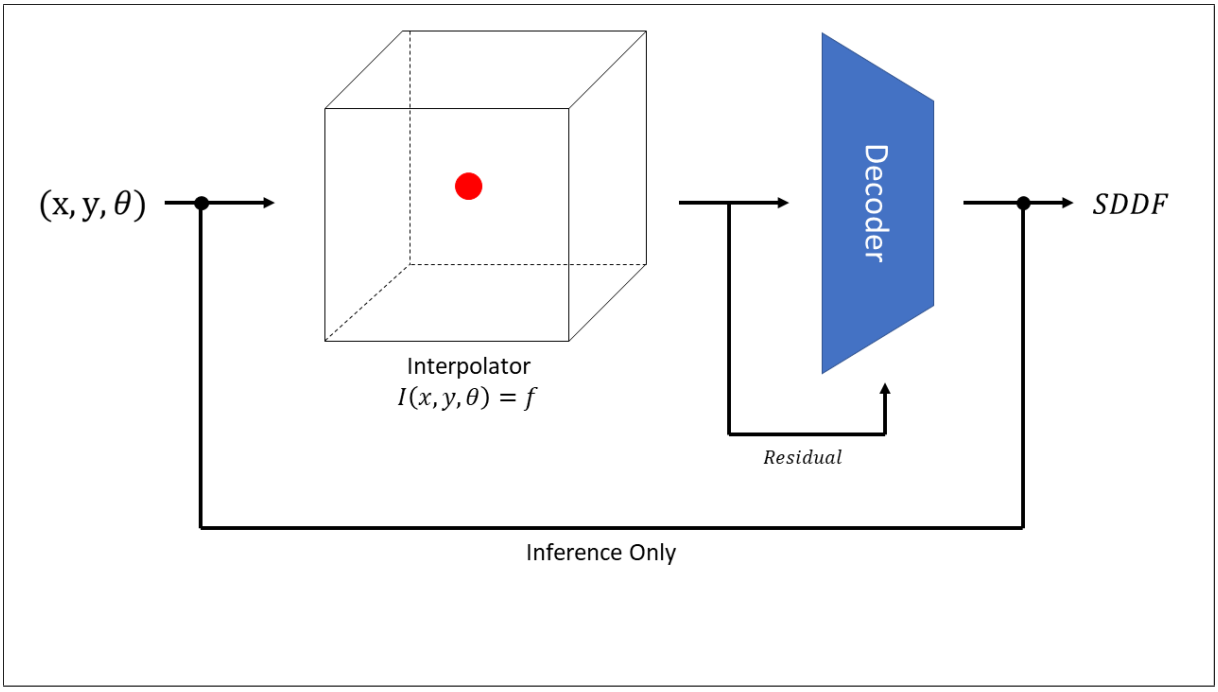


Figure 5.1. Architecture of the SDDF model with a 3D feature grid. Though the architecture is largely the same, the inclusion of rotation in the feature grid requires a modified interpolation method using Slerp

5.2 Rectilinear Feature Grid with Rotation

The feature grid design is very similar to the one used in the 2D feature grid, except that we add an additional dimension of rotation. The feature grid is therefore defined by (x, y, v), where v is the yaw angle. In all previous works, the interpolation in the feature grid has been done through bi-linear or tri-linear interpolation. Now, if the network was purely interpolation rotation angles in radians, the bilinear interpolation would be justified, as the angles in radians for a unit circle represents the great circle distance, meaning that the bilinear interpolation would be interpolating based on the great circle distance rather than Euclidean distance. However, the problem arises when utilizing these weights for interpolating the feature vectors. The simple bi-linear interpolation appears to be incapable of accurately interpolating between features in the latent space. Hence, we utilize the weighing factor used in spherical linear interpolation (Slerp) [15]. The Slerp interpolation given $p_1, p_2 \in S^2$ and interpolation factor t is given as follows,

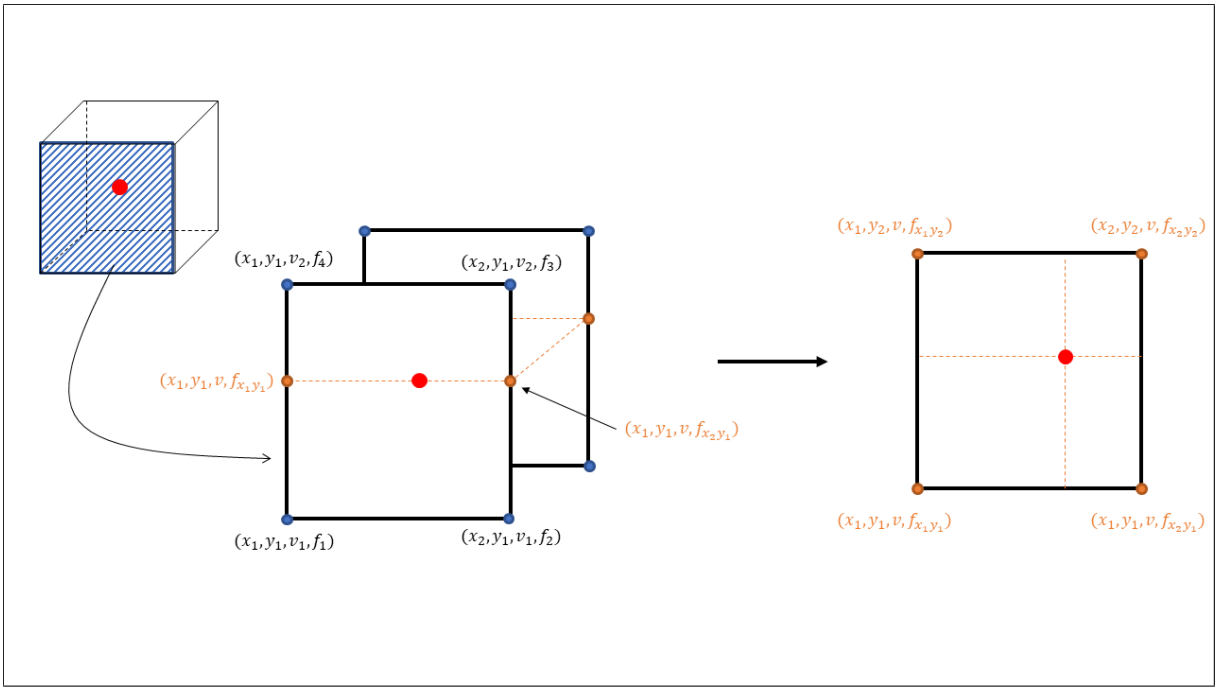


Figure 5.2. Slerp interpolation process within the 3D feature grid. The rotation dimension is interpolated first before using bilinear interpolation for the remaining dimensions

$$Slerp(p_1, p_2, t) = \frac{sin((1-t)\Omega)}{sin(\Omega)} p_1 + \frac{sin(t\Omega)}{sin(\Omega)} p_2 \tag{5.1}$$

where Ω is the angle between p_1 and p_2 in radians. The Slerp interpolation provides interpolation based on the great circle geodesic between two Using this equation, we can compute a similar interpolation factor based on the sine ratios between the yaw angles of the grid coordinates. The other two dimensions of p can be interpolated bilinearly, as they line in the Euclidean space. Given a queried point (p, v) and the neighbor eight sets of points as shown in Fig.5.2, we can then compute the feature grid interpolation as follows.First, we interpolate using the Slerp weighing factors for the theta dimension. Using the notation shown in the figure, this can be computed as below:

$$
\begin{aligned}
f_{x1y1} &= \frac{sin(v_2 - v)}{sin(v_2 - v_1)} f_1 + \frac{sin(v - v_1)}{sin(v_2 - v_1)} f_4 \\
f_{x2y1} &= \frac{sin(v_2 - v)}{sin(v_2 - v_1)} f_2 + \frac{sin(v - v_1)}{sin(v_2 - v_1)} f_3 \\
f_{x1y2} &= \frac{sin(v_2 - v)}{sin(v_2 - v_1)} f_5 + \frac{sin(v - v_1)}{sin(v_2 - v_1)} f_8 \\
f_{x2y2} &= \frac{sin(v_2 - v)}{sin(v_2 - v_1)} f_6 + \frac{sin(v - v_1)}{sin(v_2 - v_1)} f_7
\end{aligned}
\tag{5.2}
$$

Once the v dimension gone, the feature grid becomes the standard 2D feature grid, where in which we can conduct bilinear interpolation as described in Eq.4.1 to compute the final feature.

Chapter 6

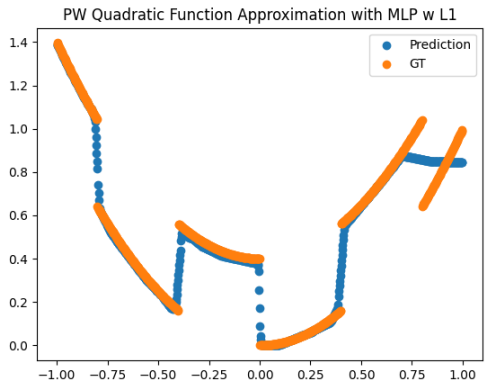
Results

6.1 Discontinuity Analysis

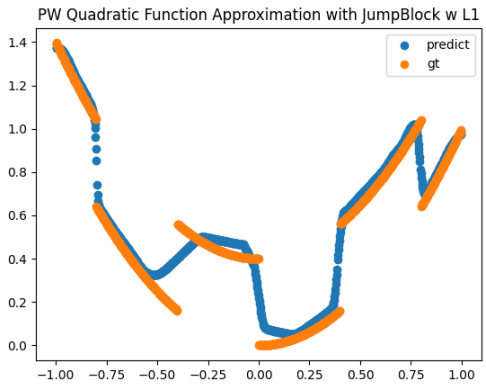
We first present the results of the discontinuity analysis discussed in Sec.3.3. The first MLP we test is the baseline MLP that utilizes the standard ReLU activation in between linear layers. The second is the discontinuity approximation using sigmoid jump blocks, as discussed in Sec.1.2.4. We then present the MLP utilizing the parametrized leaky ReLU activation and long with the standard leaky ReLU activation, which was adopted for the methods in this paper. All of the networks were optimized using the L1 loss between the ground truth and the estimate.

For estimating the piecewise quadratic function, we first generate sample data along the quadratic function for a total of . Then each of the above networks were trained on the generated dataset . Fig.6.1 shows the results.

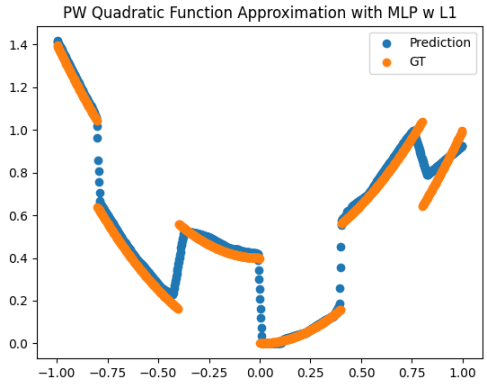
We immediately see that the sigmoidal jump block approach with significantly over-smooths the function and estimates the discontinuity with curves. This is detrimental when estimating surfaces as the curvature means that any points that are sampled to be point in that direction will estimate a direction that is between the discontinuities rather than jumping cleanly from once piece to another. ReLU, PReLU, and Leaky ReLU actually visually performs quite similarly, though the ReLU activation fails to model the final discontinuity at the end of the curve. Furthermore, though the PReLU activation does have a cleaner estimation of the function, it also contains more straggling points between discontinuities. Leaky ReLU on the other hand,



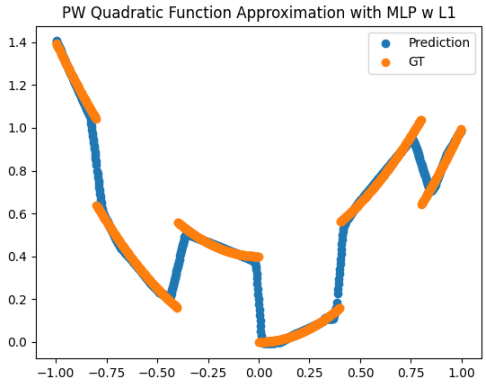
(a) ReLU



(b) Sigmoidal Jump Block



(c) Leaky ReLU



(d) PReLU

Figure 6.1. Estimation of a piecewise quadratic function using various activation functions

Table 6.1. L1 Loss from Various Methods for Piecewise Quadratic

	ReLU	Leaky ReLU	Sigmoid Jump Block	Parametric Leaky ReLU
Piecewise Quadratic	0.4081	0.4077	0.4237	0.4038

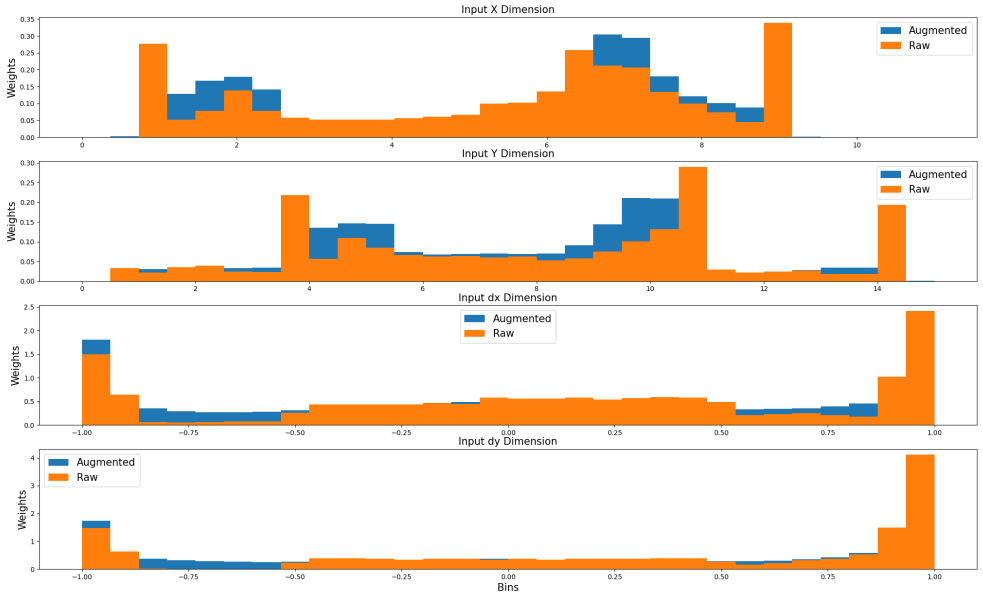
has less of this problem while having some noise in the piecewise estimation.

For quantitative evaluation, we look at the L1 loss computed each estimation method, as summarized in Table.6.1. For this evaluation, 500 test points were sampled along the piecewise quadratic function and evaluated by each network. The mean L1 loss was then computed for each of the results. Interestingly, the leaky ReLU and ReLU performed quantitatively similarly, with the leaky ReLU being slightly more accurate. The PReLU, however, actually outperformed all other methods, despite the trailing points. Hence, in terms of accuracy, the PReLU appears to be the best choice out of the other methods. However, due to the fact that t he PReLU requires additional parameters for each activation layer, but still produces a similar performance to that of the leaky ReLU, we decided to utilize the leaky ReLU over the PReLU in our implementations.

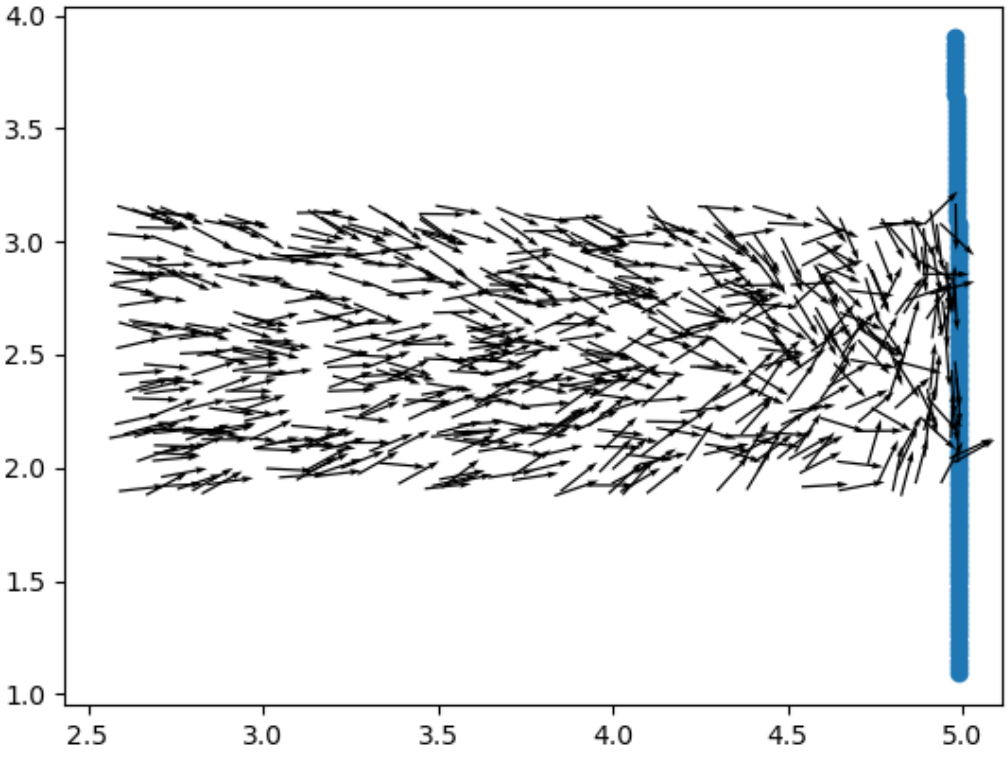
6.2 Directional Augmentation

As discussed previously, we also augment our dataset, the trajectory dataset in particular, with directional augmentation. To analyze the effects of the augmentation on the input data, we take a look at the augmented data and compare it with the original. If the augmentation was successful, we would see an increase in the variance of the chosen dataset. We utilize the shape room dataset for this purpose and Fig.6.2 shows a histogram of the results for each dimension.

We see quite clearly from the histogram that the augmentation technique does in fact significantly increase the variance of data that we see. The variance of the rotation dimension in particular increased by a significant margin, from 0.414 to 0.445 for dx and 0.476 to 0.532 for dy, which is invaluable for training SDDFs as mentioned above. The one drawback we do observe from this directional augmentation is the increase in computation time. Though the algorithm



(a) Directional Augmentation Histogram



(b) Augmentation Output of a Scan

Figure 6.2. Results of the directional augmentation on the shape room dataset. The histogram depicts the distribution of the input data for each dimension.

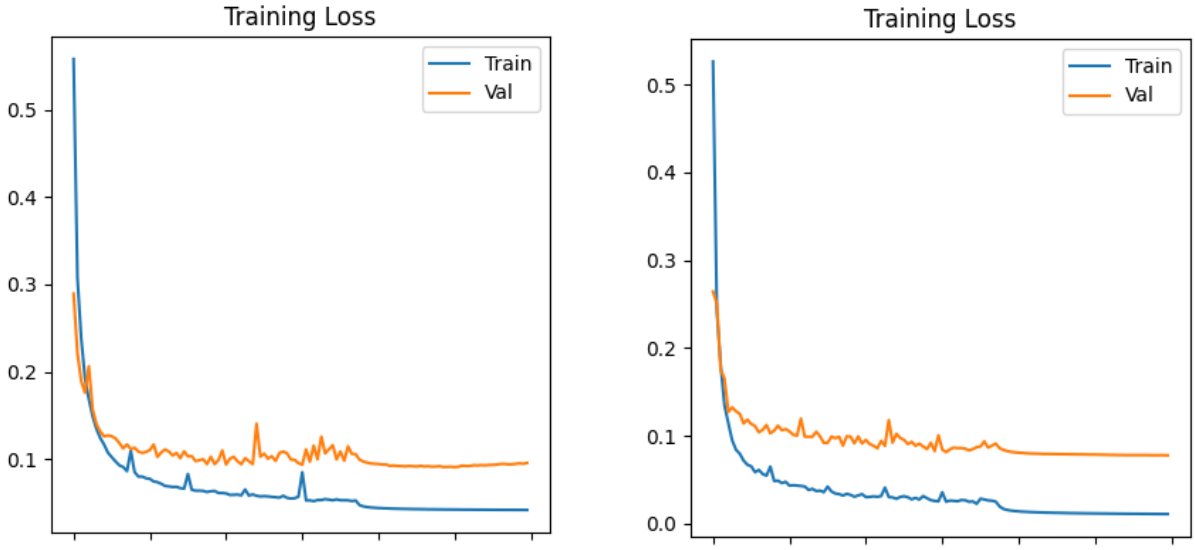
itself is pretty efficient, with a computational complexity of $\mathcal{O}(m \times n)$, where m is the number of close point samples and n is the number of desired augmented samples, the increase in the amount of data used for training does actually cause a significant performance impact. Of course, this computation impact is heavily dependent on how densely the dataset is augmented and can be adjusted depending on the situation. Furthermore, it would also be possible to replace the majoring of the subsampling methods with the augmentation, such that most of the subsampled points are from the augmentation with a greater variation in direction rather than the subsampling method that only provides samples in the same viewing drection.

6.3 Feature Grid Pretraining and Training

As mentioned previously, the training process for the 2D and 3D FSDDF methods are split into two steps. First we have the pretraining step, where we optimize the parameters for every portion of the network, and the training step, where the parameters of the decoder are frozen and only the grid features are optimized. This was done to allow the grid features handle the representation of the environment while the decoder was simply interpreting the features to compute the SDDF. Though simplistic, we can see the effects of this method in the training and validation losses during network training process. Fig. 6.3 depicts the training and validation losses on the HouseExpo dataset for both the 2D and 3D FSDDF methods.

The first observation that we make is the drop in training loss caused by freezing the feature grid. We can observe that freezing the feature grid causes a significant drop in both the training and validation loss, allowing the losses to fall below the plateau for both the 2D and 3D FSDDFs. However, we actually do observe that the validation loss actually increases for the 2D FSDDF, indicating that the network is overfitting. Hence, it is clear that choosing the correct number of pretraining epochs and training epochs is critical to the accuracy of the network.

Though pretraining/training split that we utilized worked relatively well, it is evident that there are some severe limitations to the efficacy of the method. The performance of



(a) Training and validation loss from the 2D FSDDF

(b) Training and validation loss from the 3D FSDDF

Figure 6.3. Sample training and validation losses from the 2D and 3D FSDDF methods on the HouseExpo Dataset

the network varies greatly based on the level of overfitting, which depends greatly on the pretraining/traing epoch splits. In addition, though the drops in the losses when freezing the decoder are significant, the rest of the training does not seem to improve the losses by very much. A potential improvement of this method would be to actually used pretrained neural features. Instead of training the neural features from the input datasets, it might be better to actually pretrain the neural features using other geometric datasets containing a significantly wider range of room. This would be akin to the latent code autoencoder adopted by DeepSDF and would create generalized neural features rather than ones specific to a particular dataset.

6.4 SDDF without Feature Grid

We now present the results of the scene-level SDDF model without feature grid trained on the three room environments. For each of these rooms, The normal distribution that was used to sample the Gaussian samples were given by $\mathcal{N}(0, 0.001)$. We did not utilize directional augmentation for the random poses. The model was trained with a learning rate of $1e^{-3}$ and

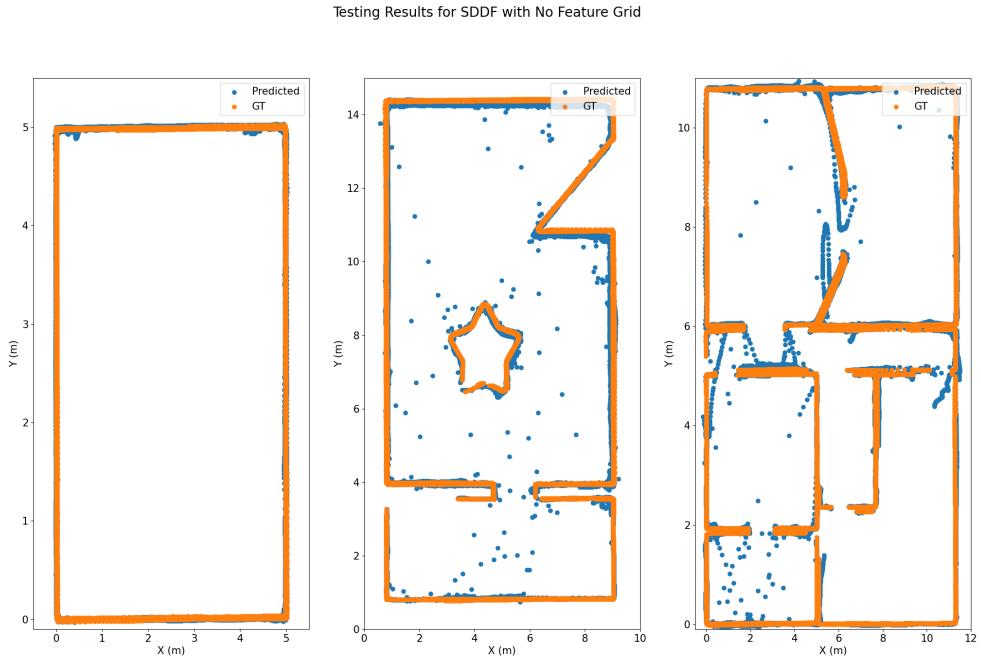
batch size of 5000. After training, the network was tested on a testing dataset that consisted of 100 novel poses that the network had not seen before. The results of the testing on the three rooms are shown in Fig.6.4.

As we can see, the results of the training for the square room is extremely clean, which is as expected as the dataset is the simplest out of the three we utilize. The key observations that should be made are the network is fully capable of modeling the sharp discontinuous corners, while the flat walls are still correctly estimated. There is still some noise around the corners, though they are not significant enough to warrant concern.

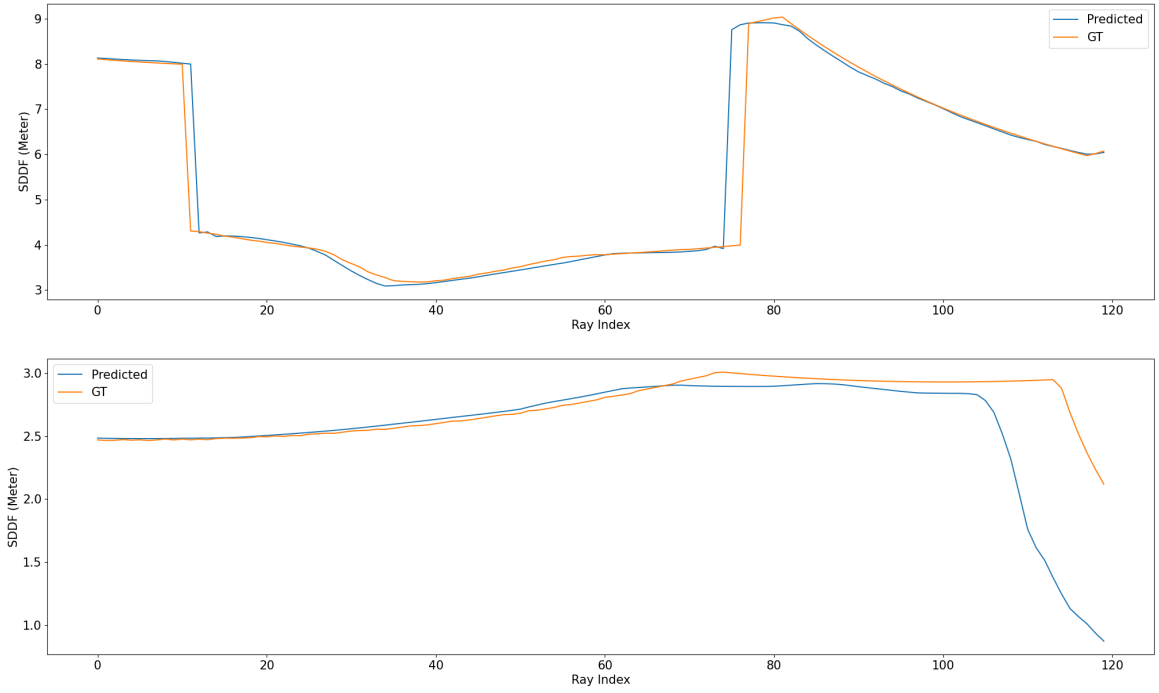
The interesting observations appear starting from the shape room dataset, where we now observe significantly more noise in the output. There are couple observations that immediately stick out, however. First, we see that the start at the center which should be completely free is largely intact, indicating that the network is capable of modeling occlusion in the dataset. However, we do observe limitation of the leaky ReLU activation in capturing discontinuity, as there are many noisy points at positions where such discontinuity occurs. Hence, we can conclude that though the leaky ReLU activation works wonderfully compared to the ReLU and sigmoidal activation, it still incapable of allowing the network to cleanly model the discontinuous jump.

These limitations become even clearer when we sample a single scan and visualize the estimation from the results, as shown in Fig.6.4b. There are clear areas where in which the network struggles to estimate the function. When there are abrupt changes to the function, which are caused by discontinuities due to rotation, the network struggles to settle onto the correct estimate, also known as the Gibb's phenomenon.

Finally, in the HouseExpo dataset, the network clearly struggles with the narrow corridors and numerous doorways, as there are erroneous estimations in the free space that are closing off what should be openings into another room. This could be due to the fact that from points far away, the doorsways are not visible, meaning that the network assumes there are walls rather than doorways along the corridor. Regardless, the network still does a remarkable job in capturing the



(a) SDDF without feature grid tested on the three different room environments



(b) Single 60° scan estimated using the SDDF without feature grid from the shape room dataset

Figure 6.4. Testing results of the SDDF without feature grid on the three datasets

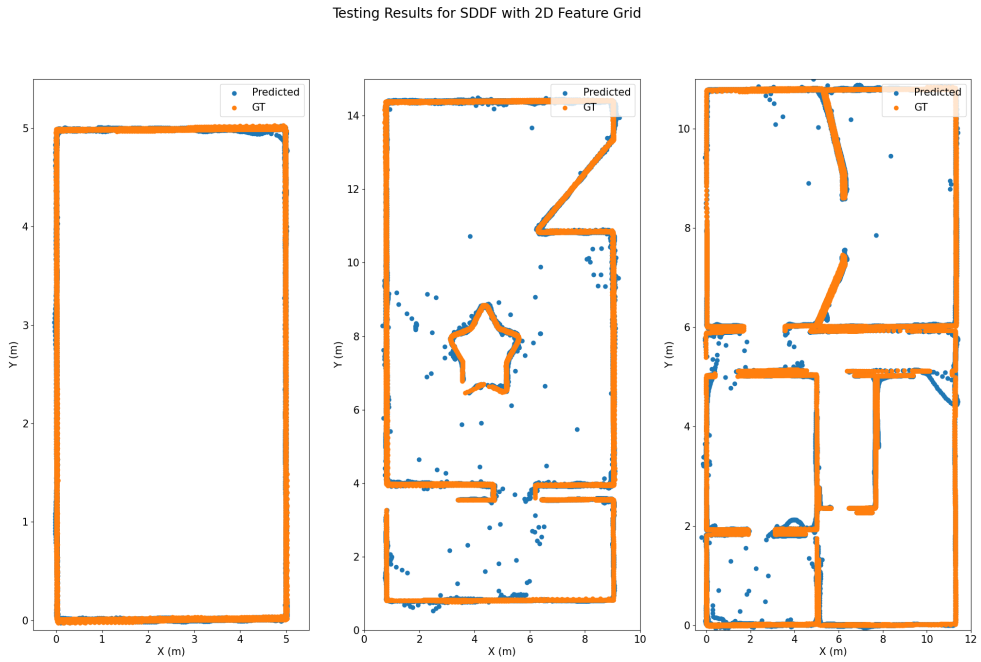
multiple layered walls and occlusion in the rooms.

6.5 SDDF with 2D Feature Grid

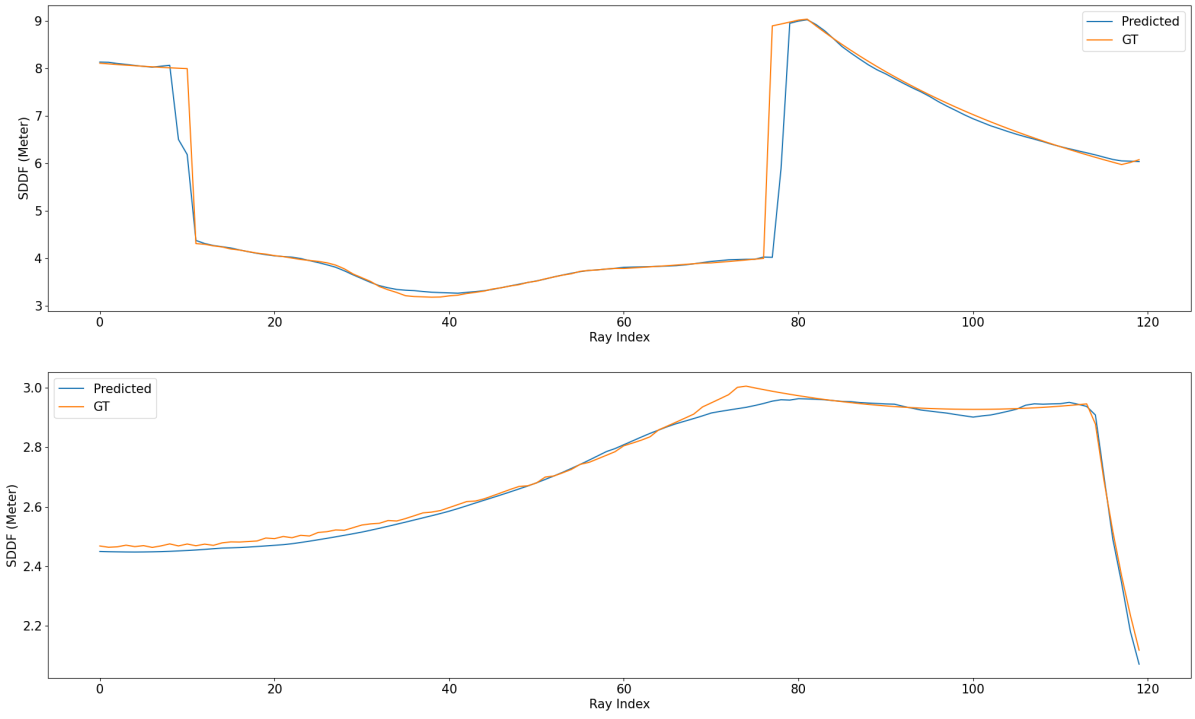
Now we evaluate the implementation of the feature into the SDDF estimation. For this model, we utilize the same parameters as the SDDF without a feature grid for sampling. The feature grid was designed to be of resolution (10×10) for the box dataset, (15×15) for the shape room dataset, and (12×12) for the HouseExpo dataset. For the training, we pretrained the network for 60 epochs then froze the decoder and continued training for another 60 epochs. We utilized different learning rates for the pretraining and training phases, with $plr = 1e^{-4}$ being used for pretraining and $lr = 1e^{-5}$ being used for the final training. The results of the estimations are shown in Fig.6.5.

We can clearly observe that the estimated depths are significantly cleaner than the previous SDDF method without a feature grid. This may be due to the fact that the feature grid allows the network to overfit onto small regions of the map, instead of having to learn the map in its entirety. The double-forward pass really shines here, as it moves the query along the ray to a grid cell that models the depth more accurately.

The box dataset still exhibits a very clean result, with even less noise than compared to the no feature grid method. We also see a significant reduction in the noise for the shape room and HouseExpo datasets, though there are still some narrow doorways that are predicted to be closed off in some poses. It is clear that the inclusion of the feature grid has definitely help the network learn the environment significantly better. In the single scan predictions, we see that the Gibbs phenomenon is minimized, with the network almost being able to perfectly predict the depth scan of the second sample.

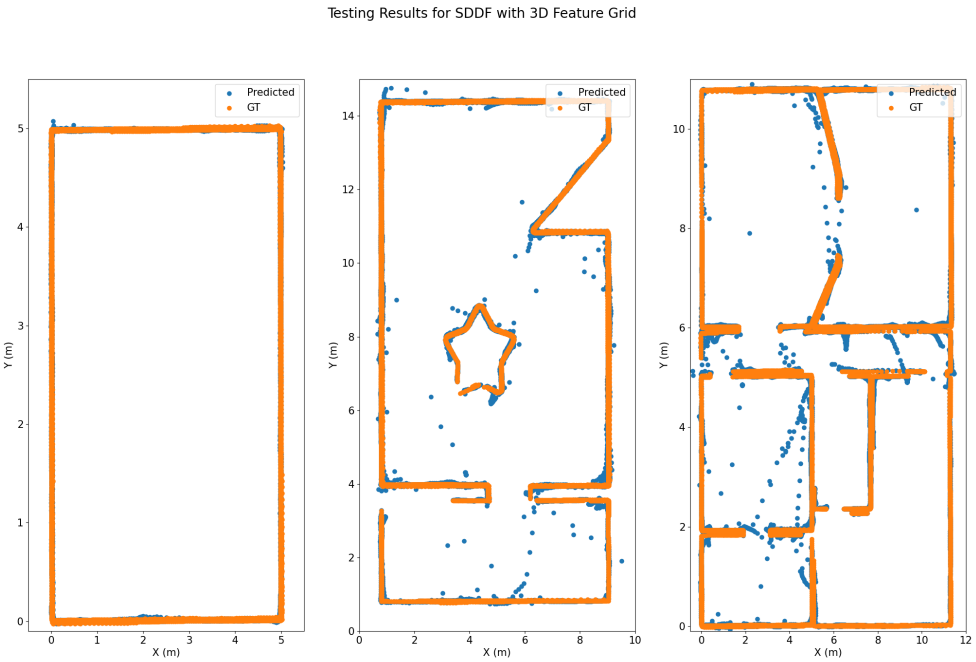


(a) 2D FSDDF tested on the three different room environments

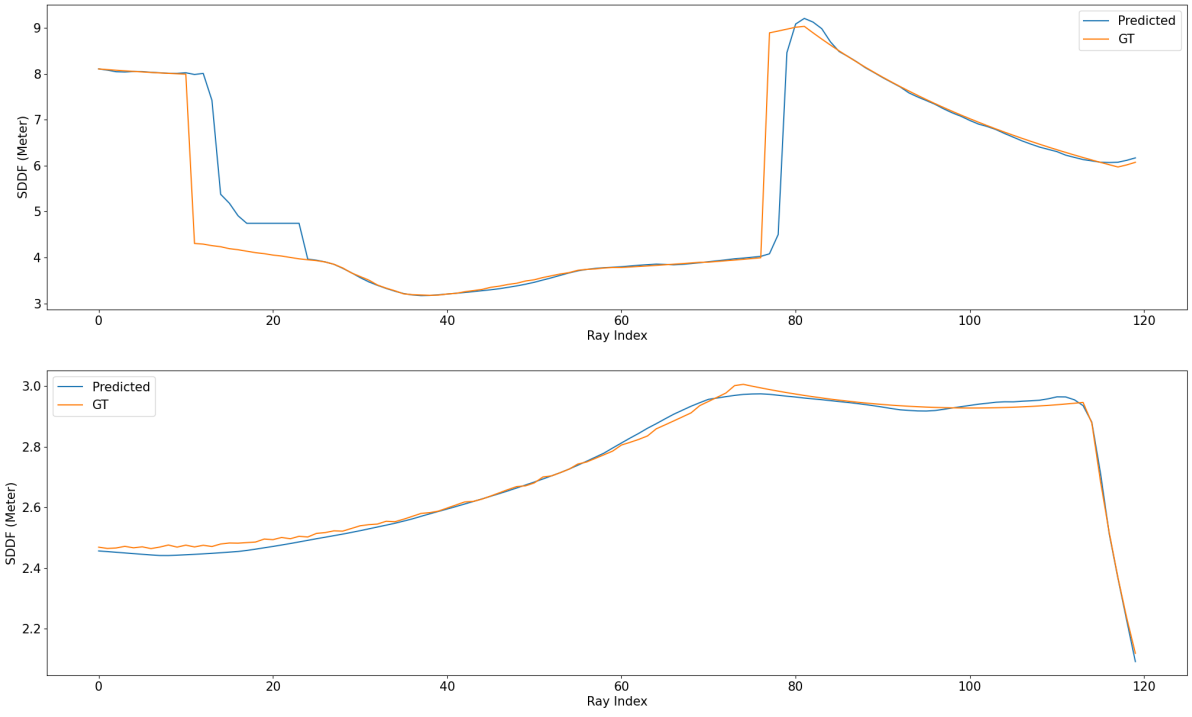


(b) Single 60° scan estimated using the 2D FSDDF

Figure 6.5. Testing results of the 2D FSDDF on the three datasets



(a) 3D FSDDF tested on the three different room environments



(b) Single 60° scan estimated using the 3D FSDDF

Figure 6.6. Testing results of the 3D FSDDF on the three datasets

6.6 SDDF with 3D Feature Grids

Finally, we demonstrate the results of the SDDF model with 3D feature grids. Similar to the previous feature grid model, the same parameters for the subsampling and training were used, but the additional of the v dimension in the feature grid adds an addition to the resolution of the feature grid. The resolution of the feature grid was set to be $(10 \times 10 \times 10)$ for the box dataset, $(15 \times 15 \times 20)$ for the shape room dataset, and $(12 \times 12 \times 20)$ for the HouseExpo dataset. The results of the estimations are shown in Fig.6.6.

With the inclusion of the rotation in our feature grid, we see an unbelievable drop in the training loss, where there are almost no noise in the free space of the map on the training data. This is most evident in the Shape Room dataset, where the start at the center is extremely clearly defined and the training loss is almost a third of what the other methods were at. However, unfortunately this accuracy is not reflected in the testing results. The validation loss does not fall as much as the training loss does, and remains at a level that is on par with the previous 2D feature grid, indicating that the network is heavily overfitting to the training data.

Qualitatively, we observe a similar performance as the 2D feature grid for all three datasets, though the network predicts the box dataset arguably cleaner. An interesting observation is that with the 3D formulation the narrow doorway in the HouseExpo dataset is intact indicating that the network was able to predict these gaps more accurately than before. However, the single scane figures indicate a problem in the results. We see that there is a significant offset in the drop of the depth value for the first scan. There could be a multitude of reasons behind this, but the primary cause may be due to the fact that because now rotation is included in the feature grid, the interpolation in the rotation space sometimes does not make sense. For example, if two consecutive rays are hitting separate surfaces with no connection between them, interpolating the two depth values will yield an erroneous measure that is halfway between the two depth values. Of course, it is not a direct translation, as we're interpolating feature vectors rather than depth values, but the analogy still holds.

Table 6.2. L1 losses of the different methods on the testing datasets for each room. FSDDF refers to the SDDF method with feature grids

	SDDF	2D FSDDF	3D FSDDF
Box	0.0196	0.0116	0.0125
Shape Room	0.116	0.0923	0.112
HouseExpo	0.0802	0.0442	0.122

A summary of the average L1 error for all the various methods in each of the room datasets are shown in Table 6.2. As we can see, the 2D FSDDF outperformed all the other methods by a significant margin, especially in the complex shape room and HouseExpo datasets. It is clear that the inclusion of the feature grid significantly improves the accuracy of the SDDF in general. Unfortunately, as stated previously, the 3D FSDDF does not perform as well as expected, largely due to the heavy overfitting we observed. Still, it is able to beat out the no-feature-grid method for the box and shape room datasets, indicating that despite overfitting, it still performs admirably. It may be possible that the occupancy prediction in the 2D FSDDF helped its ability to generalize, and could potentially aid the 3D FSDDF in the feature. Until then, the 2D FSDDF appears to be the best way to implemented directional distance functions for scene level datasets.

Chapter 7

Closing Remarks

7.1 Conclusion

In this thesis, we have explored the idea of utilizing directional information in SDFs to create SDDFs for the purpose of implicit surface representation. We have demonstrated that the leaky ReLU function out-performs the ReLU and sigmoidal jump block activation layers when it comes to approximating piecewise continuous functions with neural networks, while also proposing a novel directional augmentation technique that improves the variance of direction in the input data, especially along a trajectory.

We then proposed three different methods of implementing the SDDF in a 2D environment. The first method utilized a simple encoder MLP and decoder MLP architecture with a concatenation with the original input data in the middle, mimicking residual networks in computer vision. The second utilized a 2D neural feature grid that replaced the encoder MLP and decoupled the rotation from the position to allow the feature grid to use bilinear interpolation to compute neural features. The final method was an extended the second method to include the rotation dimension in the feature grid and utilize Slerp to interpolate in the rotation space while maintaining the bilinear interpolation for the position. All three methods were capable of modeling the surrounding environment, though there are clear differences in performance between the methods. The feature grid appears to significantly improve the performance of the estimations, as can be observed from the results of the 2D and 3D FSDDF. This can be attributed

to the subdivision of the mapping area using a feature grid, allowing the features to represent a smaller area around the map more accurately. However, there appears to be a trade-off, in that if the feature grid is too fine, the amount of sampling may be insufficient to train some of the feature grids, causing erroneous interpolation.

Another factor that must be discussed is the inclusion of the rotation in the feature grid. As we can see from the 3D FSDDF, the training results of the 3D feature grid is superb. However, in its current state, it is incapable of generalizing to the testing dataset and overfits heavily onto the training data. This could be due to a couple things. First, the Slerp interpolation itself could be the problem, meaning that the interpolation method is not accurate in the higher-dimensional space, hence leading to the feature overfitting to the training data. Another is the possibility that the network contains too many parameters and that the geometry can be represented using a smaller network. Nevertheless, this demonstrates that including rotation in the feature grid can significantly improve the estimation of the results.

All three methods presented are viable options when attempting to model SDDF in 2D for the scene-level environment. Leveraging the fast rendering speeds of SDDFs, one could easily see applications in fields such as active mapping, where in which poses can be queried for scene prediction to evaluate the novelty of a particular pose, and novel view reconstruction.

7.2 Future Work

Though the progress we have made is further than any prior work regarding scene-level SDDFs, there is still some obvious fine tuning that must be done before the results can be considered finalized. First, as mentioned above, though the formulation of the FSDDF with the 3D feature grid has the greatest potential for accuracy, it is still struggling to generalize onto the rest of the dataset. This could be caused by a multitude of factors such as the accuracy of the Slerp weighting and still requires investigating in the future. In addition, the current methods could also be applied to a windowed environment, where instead of learning the entire dataset as

a batch, we learn the environment incrementally. This would demonstrate the performance of the feature grid when data is provided sequentially and verify the real-time performance of the implemented algorithm. Finally, different methods of pretraining and training can be researched. For example, instead of utilizing the first couple training epochs of the data as pretraining, it may be possible to generalize the process and have the grid features trained on prior set of data, then be fine tuned onto the targeted environment. This could potentially help alleviate some of the problems regarding overfitting.

In addition to the potential improvements to the existing model, there are a couple extensions to the work we have shown here as well. First and most intuitively, is to apply these methodologies in 3D. We have already demonstrated progress in this regard and while not included in this thesis, have been able to approximate room geometry using the SDDF representation without feature grids. Despite being in the initial stages of the development, the method achieves near equal training loss as the Slerp formulation of the FSDDF, though still currently faces problems regarding over-fitting and generalization. Another is the extension of the directional augmentation algorithm into 3D. Though the algorithm described in this thesis is efficient and can greatly improve the diversity of data, scaling the algorithm to 3D is actually non-trivial. To this end, we have researched utilizing convex hull decomposition to create safe convex polygons in 3D with a computational complexity of $O(n \log n)$. Applying this to the 3D trajectory datasets could potentially lead to improvements in generalization. As SDDF and FSDDF are still in the early stages of its development, the possibilities for their application are endless and further research could lead to some significant breakthroughs.

Bibliography

[1] Tristan Aumentado-Armstrong, Stavros Tsogkas, Sven Dickinson, and Allan D. Jepson. Representing 3d shapes with probabilistic directed distance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19343–19354, June 2022.

[2] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields, 2021.

[3] David E. Bernholdt, Mark R. Cianciosa, Clement Etienam, David L. Green, Kody J. H. Law, and Jin Myung Park. Cluster, classify, regress: A general method for learning discountinous functions. *CoRR*, abs/1905.06220, 2019.

[4] Minsu Cho, Ameya Joshi, Xian Yeow Lee, Aditya Balu, Adarsh Krishnamurthy, Baskar Ganapathysubramanian, Soumik Sarkar, and Chinmay Hegde. Differentiable programming for piecewise polynomial functions. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020.

[5] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. In *Proceedings of Machine Learning and Systems 2020*, pages 3569–3579. 2020.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[7] Vugar Ismailov. A three layer neural network can represent any multivariate function, 2022.

[8] Yiyi Liao, Simon Donné, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[9] Bernardo Llanas, Sagrario Lantarón, and Francisco J. Sáinz. Constructive approximation of discontinuous functions by neural networks. *Neural Process. Lett.*, 27(3):209–226, 2008.

[10] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[11] Jiteng Mu, Weichao Qiu, Adam Kortylewski, Alan Yuille, Nuno Vasconcelos, and Xiaolong Wang. A-sdf: Learning disentangled signed distance functions for articulated shape representation, 2021.

[12] Joseph Ortiz, Alexander Clegg, Jing Dong, Edgar Sucar, David Novotny, Michael Zollhoefer, and Mustafa Mukadam. isdf: Real-time neural signed distance fields for robot perception. *arXiv preprint arXiv:2204.02296*, 2022.

[13] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[14] R.R. Selmic and F.L. Lewis. Neural-network approximation of piecewise continuous functions: application to friction compensation. *IEEE Transactions on Neural Networks*, 13(3):745–751, 2002.

[15] Ken Shoemake. Animating rotation with quaternion curves. *SIGGRAPH Comput. Graph.*, 19(3):245–254, jul 1985.

[16] Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew Davison. iMAP: Implicit mapping and positioning in real-time. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[17] Li Tingguang, Ho Danny, Li Chenming, Zhu Delong, Wang Chaoqun, and Max Q.-H. Meng. Houseexpo: A large-scale 2d indoor layout dataset for learning-based algorithms on mobile robots. *arXiv preprint arXiv:1903.09845*, 2019.

[18] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction, 2023.

[19] Tarun Yenamandra, Ayush Tewari, Nan Yang, Florian Bernard, Christian Theobalt, and Daniel Cremers. Fire: Fast inverse rendering using directional and signed distance functions, 2022.

[20] Tarun Yenamandra, Ayush Tewari, Nan Yang, Florian Bernard, Christian Theobalt, and Daniel Cremers. Hdsdf: Hybrid directional and signed distance functions for fast inverse rendering. 03 2022.

[21] Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[22] Jingyang Zhang, Yao Yao, and Long Quan. Learning signed distance field for multi-view surface reconstruction, 2021.

[23] Peng Zhu, Cui Larsson, Geiger Oswald, and Pollefeys. Nicer-slam: Neural implicit scene encoding for rgb slam. June 2022.

[24] Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R. Oswald, and Marc Pollefeys. Nice-slam: Neural implicit scalable encoding for slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.

[25] Ehsan Zobeidi and Nikolay Atanasov. A deep signed directional distance function for object shape representation. *CoRR*, abs/2107.11024, 2021.