# Online Learning of Piecewise Polynomial Signed Distance Fields for Manipulation Tasks

Ante Marić, Yiming Li, and Sylvain Calinon

***Abstract*— Reasoning about distance is indispensable for establishing or avoiding contact in manipulation tasks. To this end, we present an online method for learning implicit representations of signed distance using piecewise polynomial basis functions. Starting from an arbitrary prior shape, our approach incrementally constructs a continuous representation from incoming point cloud data. It offers fast access to distance and analytical gradients without the need to store training data. We assess the accuracy of our model on a diverse set of household objects and compare it to neural network and Gaussian process counterparts. Distance reconstruction and real-time updates are further evaluated in a physical experiment by simultaneously collecting sparse point cloud data and using the evolving model to control a manipulator.**

***Index Terms*— Signed Distance Fields; Representation Learning; Incremental Learning; Machine Learning for Robot Control**

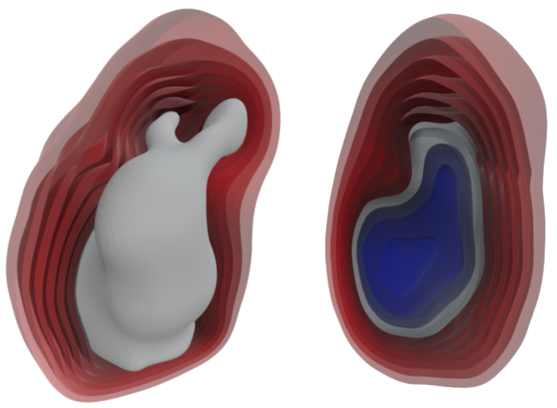


Fig. 1: Mesh and distance level-sets of the *Stanford Bunny* reconstructed from piecewise polynomial basis functions. The model is learned incrementally from 1283 randomly sampled surface points and corresponding normal vectors.

## I. Introduction

Scene representation is a naturally emerging topic in robotics as a basis for physical interaction. In recent years, implicit modeling methods have been used as compact representations of environment properties such as distance, occupancy, and color. Signed distance functions (SDFs) model distances to closest occupied points by assigning zero values to surfaces, negative values to surface interiors, and positive values elsewhere. Previously used in environment mapping and collision avoidance settings [1], they have recently seen use in robotic manipulation as the field moves towards exploring contact-rich behaviors [2]. In such scenarios, distance representations can be exploited to quickly and robustly retrieve gradients for a variety of tasks. Furthermore, modeling the full range of distances, as opposed to only the zero level-set, can be beneficial for reasoning about making or breaking contact, deformation, or penetration, with possible extensions for representing active agents such as users or robots. Commonly used implicit SDF models in robotics rely mainly on neural architectures or Gaussian process models, while alternative formulations remain less explored. Earlier computer graphics work implicitly encodes SDFs using basis functions, with recent extensions showing promising results aimed at simulation environments using piecewise polynomial representations [3]. In robotics, basis functions have been used to encode movements as superpositions of primitives [4]. Building on this viewpoint, basis function representations of implicit distance can be seen as a step toward combining movement with shape primitives of robot environments or objects of interest.

In many robotics settings, quickly building an environment model from sparse incoming data takes priority over achieving highly accurate representations. Doing so in an online manner additionally enables the integration of feedback for adaptive behavior in previously unseen environments. We formulate an online method for learning implicit signed distance fields represented as piecewise polynomial basis functions. Our method uses a simple incremental least squares approach and regularization scheme in order to approximate distance fields from incoming surface points and normals. Accuracy and performance of the piecewise polynomial representation can be balanced through interpretable hyperparameters like polynomial degree and number of segments. Model behavior can also easily be influenced by imposing priors. Our representation does not require storage of training data and can be run in real-time on both GPU and CPU. We evaluate its accuracy on a set of diverse household objects, with comparisons to Gaussian process and neural network counterparts. The smoothness and continuity of our representation are highlighted by using an evolving model to guide the movement of a physical manipulator.

This work was supported by the State Secretariat for Education, Research and Innovation in Switzerland and the European Commission's Horizon Europe Program through the INTELLIMAN project (https://intelliman-project.eu/, HORIZON-CL4-Digital-Emerging Grant 101070136) and the SESTOSENSO project (http://sestosenso.eu/, HORIZON-CL4-Digital-Emerging Grant 101070310).

The authors are with the Idiap Research Institute, CH-1920 Martigny, Switzerland and also with the EPFL, 1015 Lausanne, Switzerland. ante.maric@idiap.ch; yiming.li@idiap.ch; sylvain.calinon@idiap.ch

## II. Implicit environment representations

In robotics, widely adopted scene representations use mapping approaches that rely on discretized occupancy grids and account for uncertainty [5]. Subsequent methods explicitly store distance information for real-time usage in dynamic settings by introducing distance fields [1], [6]. Following advancements in computer vision, recent work has been exploring implicit representations as scalable and compact alternatives for describing scenes without storing data in dense grids. Implicit representations of distance, volume, and color information have found application in robotics, with initial uses in navigation [7] and mapping [8]. Recent work uses implicit representations as visual encodings for grasping [9], whole-body manipulation [2], human-robot interaction [10], planning [11], and control [12].

### A. Implicit signed distance functions

Implicit signed distance functions model geometry through continuous functions, thus decoupling memory from spatial resolution. Seminal work utilizes neural networks for shape representation, showing higher performance than point cloud, mesh, or grid-based counterparts [13]. Additional efforts have been put into investigating regularization and modeling methods that allow learning such representations in online scenarios using raw point cloud data [14], [15]. Recent methods introduce neural architectures that jointly represent SDF and color to track and reconstruct unknown objects [16]. Similarly, Neural radiance fields (NeRFs) [17] jointly encode volume density and color. They have garnered much attention as environment representations, with recent extensions targeting real-time rendering [18] and dynamic scenes [19]. However, NeRFs do not offer direct access to distance or derivative information, which can be beneficial for interpreting task execution. Furthermore, neural representations often require large amounts of data and do not translate readily to lower data regimes found in modalities such as tactile or proximity sensing.

### B. Gaussian process implicit surfaces

Probabilistic models like Gaussian process implicit surfaces (GPIS) have been used to represent distances with account for uncertainty [20]. To extend the approach for mapping purposes, scaling issues of the Gaussian process have been addressed through the use of clustering and hierarchical models [21]. Subsequent combinations with implicit regularization methods enable accurate modeling of unsigned distances [22]. These methods were later combined to give a unified mapping, odometry, and planning framework [23].

### C. Basis function representations

Basis functions can describe complex representations as weighted superpositions of simple signals. In robotics, they are well-known as the underlying representation used to encode movement primitives [24], [25]. A detailed review can be found in [4]. Their role in computer graphics extends to higher-dimensional input space to implicitly represent shapes using SDFs [26]. Learning such representations from point clouds and normals can be achieved simply by solving a linear system of equations using a least squares approach or iterative optimization procedures [27]. The resulting implicit SDF models are continuous and smooth combinations of shape primitives with analytical access to distance and gradients. A recent example utilizes piecewise polynomial basis functions to represent detailed SDFs for high-definition rendering in simulation settings [3]. The following section describes an online formulation of piecewise polynomial SDF based on incremental learning, with the aim of guiding movement in manipulation tasks.

## III. Piecewise polynomial SDF

### A. Bernstein polynomial basis functions

The value of a univariate function $f(x)$ at input $x$ can be represented as a weighted sum of $K$ basis functions with

$$f(x) = \sum_{k=1}^{K} \boldsymbol{\phi}_k(x)\, w_k = \boldsymbol{\phi}(x)\, \boldsymbol{w}, \tag{1}$$

where $\boldsymbol{\phi}$ can come from any family of basis functions. For our SDF representation, we use Bernstein polynomial basis functions, which can be computed as

$$\boldsymbol{\phi}_k(x) = \frac{(K-1)!}{(k-1)!(K-k)!}\,(1-x)^{K-k}\,x^{k-1}, \tag{2}$$

$\forall k \in \{1, \ldots, K\}$. Instead of considering a global encoding which might require the use of high-order polynomials, we split the problem into a set of local fitting problems that can consider lower-order polynomials. We retain $C^1$ continuity between the concatenated segments by adding constraints on the weights of the form

$$w_K^a = w_0^b \tag{3}$$
$$w_1^b = -w_{K-1}^a + 2w_0^b, \tag{4}$$

where $a$ and $b$ are concatenated polynomials of order $K$, and $w_k^a$ is used to denote the $k$-th weight of polynomial $a$. Polynomial bases and their derivatives can then be expressed in matrix form as

$$\boldsymbol{\phi}(x) = \boldsymbol{T}(x)\boldsymbol{B}\boldsymbol{C}, \tag{5}$$
$$\frac{\partial \boldsymbol{\phi}(x)}{\partial x} = \frac{\partial \boldsymbol{T}(x)}{\partial x}\boldsymbol{B}\boldsymbol{C}, \tag{6}$$

with $\boldsymbol{T}(x) = \begin{bmatrix} 1 & x & x^2 & \cdots & x^K \end{bmatrix}$ being a polynomial feature map of input $x$, $\boldsymbol{B}$ the corresponding Bernstein coefficient matrix, and $\boldsymbol{C}$ a constraint matrix of the form

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots \\ 0 & 1 & \cdots & 0 & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \\ 0 & 0 & \cdots & 1 & 0 & \cdots \\ 0 & 0 & \cdots & 0 & 1 & \cdots \\ 0 & 0 & \cdots & 0 & 1 & \cdots \\ 0 & 0 & \cdots & -1 & 2 & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \end{bmatrix}, \tag{7}$$

enforcing (3) and (4).

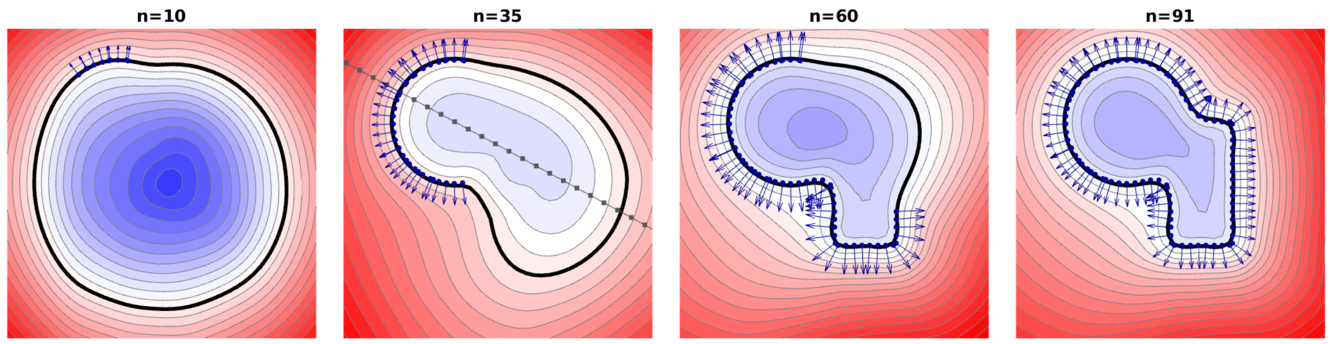


Fig. 2: Incremental model updates used to model a 2D shape, starting from a circular prior. Sampled points and normals are shown in dark blue, and the reconstructed zero-level contour in black. The reconstructed SDF is visualized as a color map with distance contours. The normal ray and regularization points of a single sample are displayed in the second image.

Successive Kronecker products can be used to extend the described representation to any number of input and output dimensions. For clarity and visualization purposes, we will continue the method description for a two-dimensional case. Namely, an extension to two-dimensional input space (Cartesian coordinates) and one-dimensional output (signed distance) can be calculated as

$$\boldsymbol{\Psi}(x,y) = \boldsymbol{\phi}(x) \otimes \boldsymbol{\phi}(y), \tag{8}$$

with partial derivatives and gradient computed analytically as

$$\frac{\partial \boldsymbol{\Psi}(x,y)}{\partial x} = \frac{\partial \boldsymbol{\phi}(x)}{\partial x} \otimes \boldsymbol{\phi}(y), \tag{9}$$
$$\frac{\partial \boldsymbol{\Psi}(x,y)}{\partial y} = \boldsymbol{\phi}(x) \otimes \frac{\partial \boldsymbol{\phi}(y)}{\partial y}, \tag{10}$$
$$\boldsymbol{\nabla\Psi}(x,y) = \frac{\partial \boldsymbol{\Psi}(x,y)}{\partial x} \otimes \frac{\partial \boldsymbol{\Psi}(x,y)}{\partial y}, \tag{11}$$

This can then be used to compute the distance and gradient values of the SDF at coordinate $(x, y)$ with

$$f(x,y) = \boldsymbol{\Psi}(x,y)\,\boldsymbol{w}, \tag{12}$$
$$\boldsymbol{\nabla} f(x,y) = \boldsymbol{\nabla\Psi}(x,y)\,\boldsymbol{w}. \tag{13}$$

The same procedure can be applied to calculate the Laplacian $\boldsymbol{\Delta} f(x,y) = \boldsymbol{\Delta\Psi}(x,y)\,\boldsymbol{w}$ and higher-order derivatives. The above representation extends analogously to accommodate three-dimensional Cartesian coordinates as input by applying an additional Kronecker product.

### B. *Computation of weights*

To approximate the SDF using polynomial basis functions, any method capable of solving a system of linear equations of the form $\boldsymbol{Aw} = \boldsymbol{s}$ can be employed. The simplest case can utilize a batch least squares estimate of the form

$$\boldsymbol{w} = (\boldsymbol{A}^\top \boldsymbol{A})^{-1} \boldsymbol{A}^\top \boldsymbol{s}, \tag{14}$$

or ridge regression as the regularized variant [28].

We use quadratic error terms in order to evaluate the fitting of distance and normal data for $N$ incoming samples

$$c_d(\boldsymbol{x}_n) = ||\boldsymbol{\Psi}(\boldsymbol{x}_n)\boldsymbol{w}||^2\,, \tag{15}$$
$$c_g(\boldsymbol{x}_n) = ||\boldsymbol{\nabla\Psi}(\boldsymbol{x}_n)\boldsymbol{w} - \boldsymbol{g}_n||^2\,, \tag{16}$$

with $\boldsymbol{x}_n = (x_n, y_n)$ denoting the $n$-th input sample, and $\boldsymbol{g}_n$ the corresponding ground truth normal. An additional *tension* term is used to constrain the curvature of the resulting distance field on $R$ control points

$$c_t(\boldsymbol{x}_r) = ||\boldsymbol{\Delta\Psi}(\boldsymbol{x}_r)\boldsymbol{w}||^2\,, \tag{17}$$

$\forall r \in \{1, \ldots, R\}$.

The model weights are learned by minimizing a combined cost

$$c = \sum_{n=1}^{N} \lambda_d^2 c_d(\boldsymbol{x}_n) + \lambda_g^2 c_g(\boldsymbol{x}_n) + \sum_{r=1}^{R} \lambda_t^2 c_t(\boldsymbol{x}_r), \tag{18}$$

with cost-tuning coefficients $\lambda_d$, $\lambda_g$, and $\lambda_t$. We construct our input and output vectors as concatenations of distance, normal, and tension features

$$\boldsymbol{\Psi}_d = [\boldsymbol{\Psi}(\boldsymbol{x}_1) \cdots \boldsymbol{\Psi}(\boldsymbol{x}_N)]^\top\,, \tag{19}$$
$$\boldsymbol{\nabla\Psi}_g = [\boldsymbol{\nabla\Psi}(\boldsymbol{x}_1) \cdots \boldsymbol{\nabla\Psi}(\boldsymbol{x}_N)]^\top\,, \tag{20}$$
$$\boldsymbol{\Delta\Psi}_t = [\boldsymbol{\Delta\Psi}(\boldsymbol{x}_1) \cdots \boldsymbol{\Delta\Psi}(\boldsymbol{x}_R)]^\top\,. \tag{21}$$

In the above, multidimensional features are flattened by stacking components to keep compatible dimensions. We finally minimize (18) by calculating (14) with

$$\boldsymbol{A} = \begin{bmatrix} \lambda_d \boldsymbol{\Psi}_d \\ \lambda_g \boldsymbol{\nabla\Psi}_g \\ \lambda_t \boldsymbol{\Delta\Psi}_t \end{bmatrix}, \quad \boldsymbol{s} = \begin{bmatrix} \boldsymbol{0}_d \\ \lambda_g \boldsymbol{g} \\ \boldsymbol{0}_t \end{bmatrix}, \tag{22}$$

where $\boldsymbol{g}$ is a vector of sampled normal components, and $\boldsymbol{0}_d$ and $\boldsymbol{0}_t$ are zero vectors with lengths compatible to $\boldsymbol{\Psi}_d$ and $\boldsymbol{\Delta\Psi}_g$, respectively.

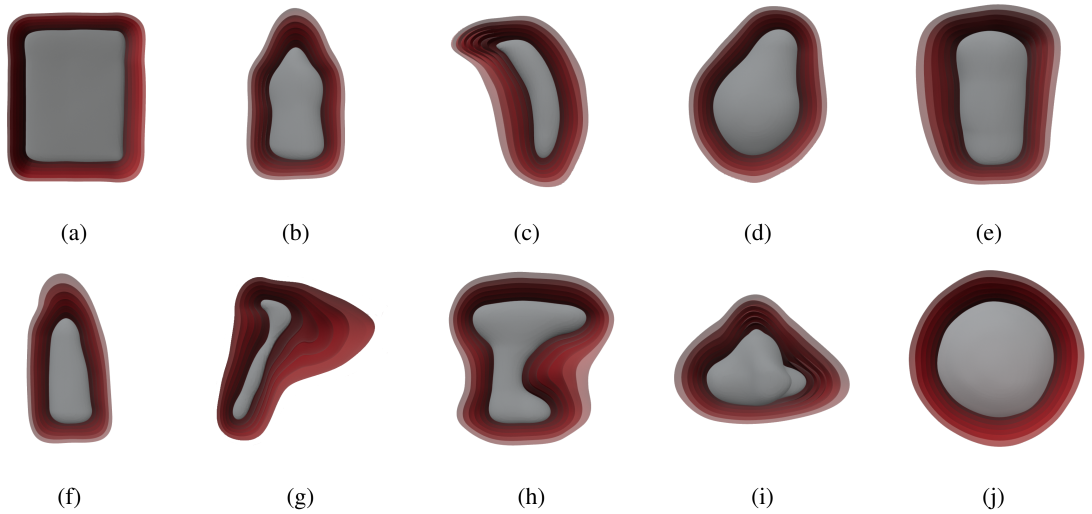


Fig. 3: Mesh and SDF reconstruction results for objects from the YCB test set: (a) `003_cracker_box`, (b) `006_mustard_bottle`, (c) `011_banana`, (d) `016_pear`, (e) `019_pitcher_base`, (f) `021_bleach_cleanser`, (g) `048_hammer`, (h) `035_power_drill`, (i) `063-a_marbles`, (j) `053_mini_soccer_ball`. All models were learned from 800 non-uniformly sampled points and normals.

### C. *Incremental formulation*

For online approximation, we employ an incremental variant of the least squares algorithm which allows us to gradually refine an initial estimate by providing samples one-by-one or in batches. Similar approaches have previously been used in the context of control [29]. The algorithm exploits the Sherman-Morrison-Woodbury relations [30] which connect subsequent inverses of a matrix after small-rank perturbations. After initializing the weight precision matrix $\boldsymbol{P} = \mathrm{cov}(\boldsymbol{w})^{-1}$ to $\boldsymbol{P}_0$, it can be incrementally updated as

$$\boldsymbol{P}_{\text{new}} = \boldsymbol{P} - \underbrace{\boldsymbol{P}\boldsymbol{A}_n^\top \left(\sigma^2 \boldsymbol{I} + \boldsymbol{A}_n \boldsymbol{P} \boldsymbol{A}_n^\top\right)^{-1}}_{\boldsymbol{K}} \boldsymbol{A}_n \boldsymbol{P}, \tag{23}$$

where $\sigma^2$ is the measurement noise variance, $\boldsymbol{A}_n$ the input matrix, and $\boldsymbol{I}$ an identity matrix of compatible dimensions. Starting from a prior $\boldsymbol{w} = \boldsymbol{w}_0$, weight updates can then be calculated by using the Kalman gain $\boldsymbol{K}$

$$\boldsymbol{w}_{\text{new}} = \boldsymbol{w} + \boldsymbol{K}\Big(\boldsymbol{s}_n - \boldsymbol{A}_n \boldsymbol{w}\Big). \tag{24}$$

The above iterative computation has no requirement of storing the training points and enables us to impose priors on our model through $\boldsymbol{P}_0$ and $\boldsymbol{w}_0$. We initialize the precision matrix as $\boldsymbol{P}_0 = (\boldsymbol{M}^\top \boldsymbol{M})^{-1}$, with $\boldsymbol{M} = \boldsymbol{B}\boldsymbol{C}$ following (5). The evolving precision matrix can be used to incrementally track weight covariance as the model is learned.

In order for our model to accurately approximate a distance function, the tension term needs to be enforced throughout the input space. We achieve this in an online setting by uniformly sampling a number of control points on the normal rays of incoming surface samples, as displayed in Figure 2. The full computation steps are summarized in Algorithm 1. We apply the same approach for three-dimensional inputs, with example reconstructions shown in Figures 1 and 3.

**Algorithm 1** Incremental computation of weights.

```
$\boldsymbol{P} = \boldsymbol{P}_0$ // Initialize precision matrix
$\boldsymbol{w} = \boldsymbol{w}_0$ // Initialize weights
for $n \leftarrow 1$ to $N$ do
    $\boldsymbol{A}_n = \boldsymbol{A}(\boldsymbol{x}_n)$ // Construct input matrix
    $\boldsymbol{K} = \boldsymbol{P}\boldsymbol{A}_n^\top \left(\sigma^2 \boldsymbol{I} + \boldsymbol{A}_n \boldsymbol{P} \boldsymbol{A}_n^\top\right)^{-1}$ // Compute gain
    $\boldsymbol{P} \leftarrow \boldsymbol{P} - \boldsymbol{K}\boldsymbol{A}_n\boldsymbol{P}$ // Update precision matrix
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{K}\Big(\boldsymbol{s}_n - \boldsymbol{A}_n \boldsymbol{w}\Big)$ // Update weights
end
```

## IV. Experiments

### A. *Reconstruction accuracy*

To evaluate the reconstruction accuracy of our method we use error metrics similar to [15]. Reconstructed distances are evaluated using the mean absolute error (MAE)

$$\mathrm{MAE}(\boldsymbol{x}) = \big|\hat{s}(\boldsymbol{x}) - s(\boldsymbol{x})\big|, \tag{25}$$

with $\hat{s}(\boldsymbol{x})$ denoting the estimated signed distance at point $\boldsymbol{x}$, and $s(\boldsymbol{x})$ the ground truth value. The accuracy of our model is first tested on different grid sizes by using cubic polynomials with a variable number of segments.

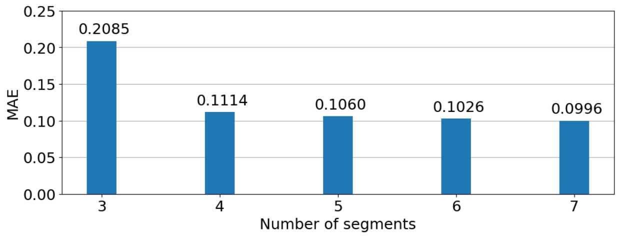
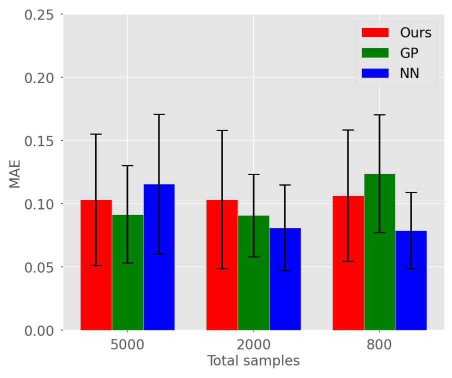


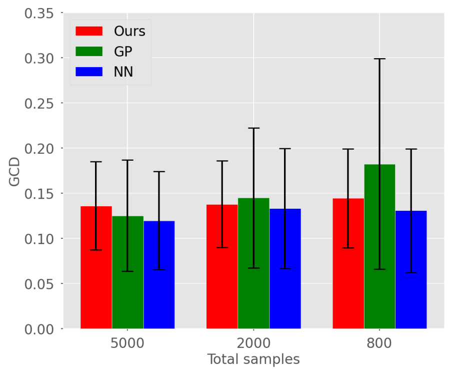
Fig. 4: Mean absolute error (MAE) using cubic polynomials with a varying number of segments.

Figure 4 displays the MAE results on our test data, which comprises of real point clouds from the Yale-CMU-Berkeley (YCB) dataset [31]. The test set consists of 10 household objects of diverse shapes, depicted in Figure 3. Ground truth SDFs are reconstructed from high-definition meshes with 512k polygons.

We compare the accuracy of our approach against two methods used in online settings: a *LogGP* model using the Matérn 3/2 kernel as in [22], and a 4-layer neural network with 256 neurons in each layer, using the loss and positional embedding described in [15]. Our comparison model uses cubic Bernstein polynomials with 6 segments per input dimension. All methods are evaluated on the same point cloud and normal data for each object from the test set. Accuracy is compared across different data volumes by varying the number of sampled points. Since *LogGP* models unsigned distance, we evaluate it against the absolute value of the ground truth. Figure 5a shows the resulting MAE comparisons. Distance accuracy near and far from object surfaces for different amounts of training samples is displayed in Table I.



(a) Mean absolute error (MAE)



(b) Gradient cosine distance (GCD)

Fig. 5: Distance and gradient reconstruction accuracy compared on varying amounts of data.

Additional comparisons are made with respect to reconstructed distance gradients by calculating the gradient cosine distance (GCD)

$$GCD(\boldsymbol{x}) = 1 - \frac{\nabla_{\boldsymbol{x}}\hat{s}(\boldsymbol{x}) \cdot \nabla_{\boldsymbol{x}}s(\boldsymbol{x})}{\|\nabla_{\boldsymbol{x}}\hat{s}(\boldsymbol{x})\|\|\nabla_{\boldsymbol{x}}s(\boldsymbol{x})\|}, \quad (26)$$

$|s| < 0.05$

| Samples | Ours | GP | NN |
|---|---|---|---|
| 5000 | $0.0574 \pm 0.0389$ | $0.0577 \pm 0.0379$ | $0.0245 \pm 0.0066$ |
| 2000 | $0.0562 \pm 0.0366$ | $0.0585 \pm 0.0425$ | $0.0385 \pm 0.0226$ |
| 800 | $0.0558 \pm 0.0368$ | $0.0539 \pm 0.0395$ | $0.0386 \pm 0.0249$ |

$|s| > 0.05$

| Samples | Ours | GP | NN |
|---|---|---|---|
| 5000 | $0.1043 \pm 0.0525$ | $0.0926 \pm 0.0392$ | $0.1174 \pm 0.0561$ |
| 2000 | $0.1044 \pm 0.0551$ | $0.0916 \pm 0.0325$ | $0.0818 \pm 0.0339$ |
| 800 | $0.1079 \pm 0.0522$ | $0.1252 \pm 0.0461$ | $0.0798 \pm 0.0299$ |

TABLE I: Comparison of the mean absolute error (MAE) near and far from object surfaces for varying numbers of training samples.

with $\nabla_{\boldsymbol{x}}\hat{s}(\boldsymbol{x})$ denoting the estimated distance gradient, and $\nabla_{\boldsymbol{x}}s(\boldsymbol{x})$ the corresponding ground truth. Gradients are recovered analytically for our method and numerically for the NN and GP models. Ground truth gradients are calculated numerically from corresponding ground truth SDFs. All numerical calculations are done on a dense grid of $128^3$ points. Figure 5b displays the GCD comparisons. For qualitative evaluation, Figure 3 shows the mesh and distance fields of the test objects, reconstructed from learned piecewise polynomial representations.

## B. *Computation time & memory requirements*

The computation time of updates and queries to our model increases quadratically based on the number of weights, which is determined by the desired number of segments and basis functions (polynomial degree). Computation time and memory requirements do not increase with the total number of training points, and we only store the combination weights of our basis function representation. For a set input dimension, the time complexity of a single incremental update is $\mathcal{O}(B^3 + S^2K^2B + SKB^2)$, with $S$ denoting the number of segments, $K$ the number of basis functions, and $B$ the sample batch size. Figure 6 shows mean update times for different numbers of segments and varying batch sizes using a *PyTorch* implementation and *NVIDIA GeForce MX550* GPU.

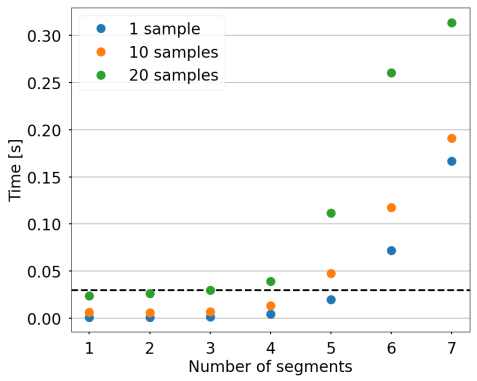


Fig. 6: Update time for cubic polynomials with varying numbers of segments and different batch sizes. The desired real-time cutoff of 30 $ms$ is denoted by a horizontal line.

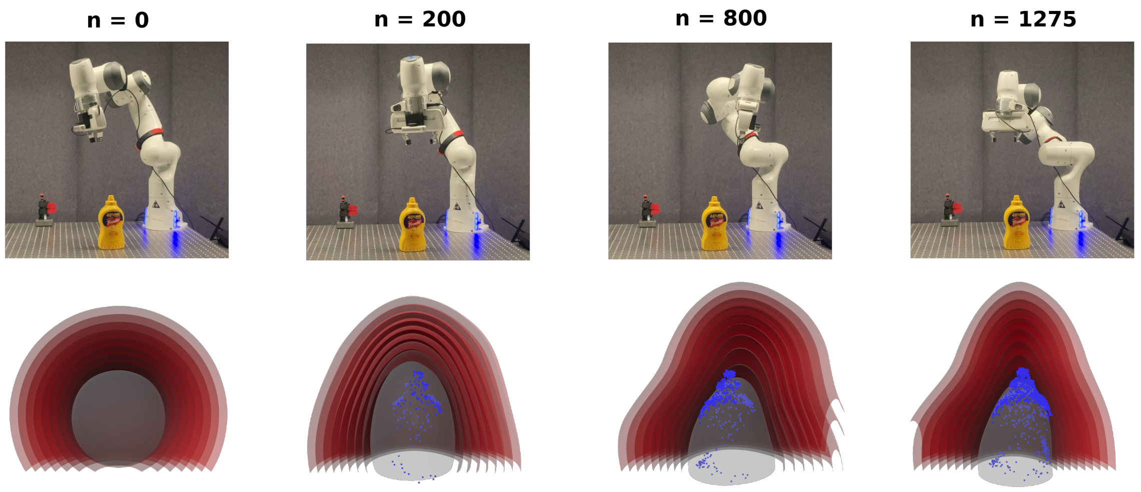


Fig. 7: A visualization of our physical experiment and results, showing the evolution of a piecewise polynomial SDF model starting from a spherical prior. As $n$ samples are collected, the updated model is queried in real-time to control the manipulator by following a tangential trajectory on the SDF level-sets while keeping normal orientation.

During accuracy evaluation, the mean time of a single incremental update for $S = 6$, $K = 4$, and $B = 10$ was $0.1176$ $s$ on the described setup. Update time can be further reduced by reducing the density of sampled regularization points. For queries and reconstruction, our method scales linearly with the number of weights, as we calculate only a weighted sum of basis functions. The time complexity of querying our model for a batch of $B$ points is $\mathcal{O}(SKB)$ with mean time $0.17$ $ms$ for a single query point in our tests. The mean time of full SDF reconstruction on a dense $128^3$ grid, required only for visualization of the results, was $28.79$ $s$ using the marching cubes algorithm [32].

## C. *Physical experiment*

In order to test the viability of our method in a real-world setting, we perform a physical experiment in an online learning scenario using a Franka Emika 7-DoF manipulator. Starting from a spherical prior, we incrementally update our model with sparse surface points and estimated normals collected by an in-hand Intel RealSense D415 sensor. At each timestep, we query normals from our evolving SDF model to keep the camera oriented toward the object while following a tangential trajectory on the level sets. Model updates are done in real-time and used as feedback for our controller, with a mean timestep of $31.4$ $ms$ on a $3.6$ $GHz$ Intel Core i9-9900K CPU. To enable real-time operation on CPU, our SDF model uses cubic polynomials and 4 segments per input dimension with point-by-point updates. Figure 7 shows our setup and the evolution of an SDF approximated from sparse partial point cloud data.

## V. Discussion

The results in Figure 4 demonstrate a reduction in the reconstruction error when increasing the number of segments. For cubic polynomials, this decline becomes less pronounced beyond 5 segments, partly due to the size and complexity of our test set. Qualitatively, employing a higher number of segments results in more detailed reconstructions, a characteristic that might not be fully captured by the MAE metric. For instance, comparing Figure 3b, reconstructed from a model using 6 segments, with the final result in Figure 7, reconstructed from a model using 4 segments, shows this effect. The number of segments and polynomial degree can be adjusted to balance accuracy and performance for the task at hand. Evaluations in Figure 5 and Table I demonstrate consistent performance across various data volumes. Employing 6 segments per input dimension yields MAE and GCD values similar to those of the Gaussian process and neural network comparison models. These results are maintained both near object surfaces and at greater distances, affirming the effectiveness of our regularization approach. Qualitative reconstructions in Figure 3 display visually accurate distance and mesh reconstruction results across our test set. It can be noted that the current regularization approach, paired with the tested hyperparameters, tends to over-smooth and compromise detail. This is especially visible when modeling thin or sharp objects, as depicted by Figure 3g.

The computation time and memory requirements of our formulation are independent of the total number of training points. As illustrated in Figure 6, the update time scales quadratically with the number of segments and cubically with batch size. Consequently, this constrains the applicability of our current formulation to smaller environments and data regimes typically found in tabletop manipulation settings. Our approach achieves fast query times as inference amounts to calculating a weighted superposition of basis functions. The physical experiment showcases real-time updates and intermediary reconstruction results, starting from a spherical prior. It demonstrates that continuity, smoothness, and fast access to distance and analytical gradients enable direct usage of our model for feedback-loop control, while validating it in a scenario with higher noise and sparse samples.

Future work will investigate the adaptation of the described method to clustering or hierarchical models, such as octrees. These approaches have been shown to improve SDF reconstructions and enable the handling of larger and more complex environments [3]. Varying other hyperparameters like polynomial degree or family of basis functions might yield additional improvements to the accuracy and detail of the reconstructed SDFs. Further efforts in scaling and increasing performance might focus on using local weight updates and optimized computation pipelines. On the method level, an interesting direction would involve further analysis of the evolving precision matrix to evaluate the quality of intermediary results during the learning process, or as an uncertainty measure in planning scenarios. The use of incrementally updated priors might also be of particular interest for extensions to dynamic scenes. Finally, combinations of multiple sensing modalities (e.g., tactile and proximity) need to be further explored and validated on a physical setup. Further experiments could also make use of the analytic representation by exploiting Boolean operators to combine the SDFs of multiple bodies and evaluate overlap (e.g., for uses in whole-body manipulation).

## VI. Conclusion

We presented an online formulation of piecewise polynomial signed distance functions. Starting from an arbitrary prior shape, our method incrementally builds a continuous and smooth distance representation from incoming surface point clouds and normal data. The results display reconstruction accuracy similar to Gaussian process and neural network comparison models on a test set consisting of various household objects. We have shown that memory and computation-time requirements of our underlying model do not increase with the total number of training points, and highlight the possibility of real-time updates. Finally, we validated our approach on a physical manipulator with noisy observations by using the evolving distance model for feedback-loop control.

## References

[1] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning," in *Proc. IEEE/RSJ IROS*, pp. 1366–1373, 2017.

[2] Y. Li, Y. Zhang, A. Razmjoo, and S. Calinon, "Learning robot geometry as distance fields: Applications to whole-body manipulation," *arXiv:2307.00533*, 2023.

[3] E. Pujol and A. Chica, "Adaptive approximation of signed distance fields through piecewise continuous interpolation," *Computers & Graphics*, vol. 114, pp. 337–346, 2023.

[4] S. Calinon, "Mixture models for the analysis, edition, and synthesis of continuous time series," in *Mixture Models and Applications* (N. Bouguila and W. Fan, eds.), pp. 39–57, Springer, Cham, 2019.

[5] 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.

[6] L. Han, F. Gao, B. Zhou, and S. Shen, "Fiesta: Fast incremental euclidean distance fields for online motion planning of aerial robots," *Proc. IEEE/RSJ IROS*, pp. 4423–4430, 2019.

[7] M. Adamkiewicz, T. Chen, A. Caccavale, R. Gardner, P. Culbertson, J. Bohg, and M. Schwager, "Vision-only robot navigation in a neural radiance world," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 2, pp. 4606–4613, 2022.

[8] E. Sucar, S. Liu, J. Ortiz, and A. Davison, "iMAP: Implicit mapping and positioning in real-time," *Proc. IEEE/CVF ICCV*, pp. 6209–6218, 2021.

[9] M. Breyer, J. J. Chung, L. Ott, S. Roland, and N. Juan, "Volumetric grasping network: Real-time 6 dof grasp detection in clutter," in *Proc. CoRL*, pp. 1602–1611, 2020.

[10] P. Liu, K. Zhang, D. Tateo, S. Jauhri, J. Peters, and G. Chalvatzaki, "Regularized deep signed distance fields for reactive motion generation," in *Proc. IEEE/RSJ IROS*, pp. 6673–6680, 2022.

[11] D. Driess, J.-S. Ha, M. Toussaint, and R. Tedrake, "Learning models as functionals of signed-distance fields for manipulation planning," in *Proc. CoRL*, 2021.

[12] Y. Li, S. Li, V. Sitzmann, P. Agrawal, and A. Torralba, "3d neural scene representations for visuomotor control," in *Proc. CoRL*, pp. 112–123, 2021.

[13] J. J. Park, P. R. Florence, J. Straub, R. A. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," *Proc. IEEE CVPR*, pp. 165–174, 2019.

[14] A. Gropp, L. Yariv, N. Haim, M. Atzmon, and Y. Lipman, "Implicit geometric regularization for learning shapes," in *Proc. ICML*, vol. 119, pp. 3789–3799, 2020.

[15] J. Ortiz, A. Clegg, J. Dong, E. Sucar, D. Novotny, M. Zollhoefer, and M. Mukadam, "isdf: Real-time neural signed distance fields for robot perception," in *Proc. RSS*, 2022.

[16] B. Wen, J. Tremblay, V. Blukis, S. Tyree, T. Muller, A. Evans, D. Fox, J. Kautz, and S. Birchfield, "Bundlesdf: Neural 6-dof tracking and 3d reconstruction of unknown objects," *Proc. IEEE CVPR*, pp. 606–617, 2023.

[17] 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," *Comm. ACM*, vol. 65, p. 99–106, Dec 2021.

[18] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, July 2023.

[19] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer, "D-NeRF: Neural Radiance Fields for Dynamic Scenes," in *Proc. IEEE CVPR*, 2020.

[20] S. Dragiev, M. Toussaint, and M. Gienger, "Gaussian process implicit surfaces for shape estimation and grasping," in *Proc. IEEE ICRA*, pp. 2845–2850, 2011.

[21] Z. H. Bhoram Lee, Clark Zhang and D. D. Lee, "Online continuous mapping using gaussian process implicit surfaces," *Proc. IEEE ICRA*, pp. 6884–6890, 2019.

[22] 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 (RA-L)*, vol. 6, no. 2, pp. 2461–2468, 2021.

[23] L. Wu, K. M. B. Lee, C. Le Gentil, and T. Vidal-Calleja, "Log-gpis-mop: A unified representation for mapping, odometry, and planning," *IEEE Transactions on Robotics (T-RO)*, vol. 39, no. 5, pp. 4078–4094, 2023.

[24] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors," *Neural Computation*, vol. 25, pp. 328–373, 02 2013.

[25] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic movement primitives," in *Proc. NIPS*, pp. 2616–2624, 2013.

[26] B. Jüttler and A. Felis, "Least-squares fitting of algebraic spline surfaces," *Advances in Computational Mathematics*, vol. 17, pp. 135–152, Jul 2002.

[27] G. Taubin, "Smooth signed distance surface reconstruction and applications," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pp. 38–45, 2012.

[28] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[29] J.-A. Ting, S. Vijayakumar, and S. Schaal, "Locally weighted regression for control," in *Encyclopedia of Machine Learning*, pp. 613–624, Springer, 2010.

[30] W. W. Hager, "Updating the inverse of a matrix," *SIAM Rev.*, vol. 31, pp. 221–239, 1989.

[31] B. Calli *et al.*, "Yale-cmu-berkeley dataset for robotic manipulation research," *Intl Journal of Robotics Research*, vol. 36, no. 3, pp. 261–268, 2017.

[32] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *Seminal graphics: pioneering efforts that shaped the field*, pp. 347–353, 1998.