Learning Models as Functionals of Signed-Distance Fields for Manipulation Planning

Danny Driess[1] **Jung-Su Ha**[1] **Marc Toussaint**[1] **Russ Tedrake**[2]
[1]TU Berlin, Germany
[2]Massachusetts Institute of Technology, USA

Abstract: This work proposes an optimization-based manipulation planning framework where the objectives are learned functionals of signed-distance fields that represent objects in the scene. Most manipulation planning approaches rely on analytical models and carefully chosen abstractions/state-spaces to be effective. A central question is how models can be obtained from data that are not primarily accurate in their predictions, but, more importantly, enable efficient reasoning within a planning framework, while at the same time being closely coupled to perception spaces. We show that representing objects as signed-distance fields not only enables to learn and represent a variety of models with higher accuracy compared to point-cloud and occupancy measure representations, but also that SDF-based models are suitable for optimization-based planning. To demonstrate the versatility of our approach, we learn both kinematic and dynamic models to solve tasks that involve hanging mugs on hooks and pushing objects on a table. We can unify these quite different tasks within one framework, since SDFs are the common object representation. Video: `https://youtu.be/ga8Wlkss7co`

Keywords: Manipulation Planning, Signed Distance Fields, Model Learning

1 Introduction

Manipulation planning is challenging for multiple reasons. On the one hand, planning robot motions to solve a task can be formulated as a decision problem over a high-dimensional, non-convex space, including discrete and continuous aspects. Especially long-horizon tasks that consist of multiple manipulation steps have the property that motions have to be coordinated globally with the future goal. This coupling of potentially all variables requires joint reasoning and makes the problem particularly challenging [1]. On the other hand, the problem solving capabilities of a planning framework is inherently dependent on its underlying models. The field of Task and Motion Planning (TAMP) has made significant progress in solving challenging multi-step, long-horizon tasks [2], ranging from ones that involve mainly kinematic models [3, 4, 5, 6, 7] to dynamic tasks that require reasoning about forces, friction etc. based on more general dynamic equations [8, 9, 10, 11, 12, 13, 14]. However, most TAMP approaches rely on carefully chosen abstractions and analytically defined models in order to be successful and efficient. In particular, TAMP often makes simplifying assumptions on the possible geometries of objects it can deal with to define manipulation constraints in the first place. It is unclear how these models can be grounded from sensor information.

To overcome these issues, a natural idea is to replace the analytic models in TAMP frameworks with learned ones. Recent advances in deep learning have enabled to learn predictive forward models even in high-dimensional observation spaces like images. The typical objective for learning a forward model is its predictive accuracy. However, having an accurate model does not necessarily imply that a planning framework can utilize it efficiently. While having an accurate forward prediction model might be sufficient for short-horizon tasks, especially for long-horizon tasks, learned models can exhibit too high combinatorics for sampling or non-informative gradients for achieving future goals.

This paper aims to address these challenges by learning models that can be used effectively by a planning framework while at the same time using a general object representation more closely related to senors spaces. To realize this, we present an optimization-based TAMP framework where the objectives are learned *functionals* of signed-distance fields (SDFs). The SDFs represent each object in the scene separately, while the functionals defined on top of them induce constraints on possible, physically plausible interactions between the objects within a trajectory optimization problem. The task planning aspect is realized by (discrete) decisions that determine which of those functionals are active at which phase of the planning horizon.

5th Conference on Robot Learning (CoRL 2021), London, UK.

We argue that representing objects as SDFs has multiple advantages. First, an SDF can be seen as an intermediate representation between raw perception like point-clouds or images and full state information. While not the focus of this work, many methods have been developed to learn and obtain SDFs from, e.g., image observations of the scene. Further, SDFs can represent arbitrary, non-convex geometries, which is beneficial, since manipulation problems and physical phenomena often depend on the geometry of the interacting objects. Finally, we show that SDFs are particularly suited for learning and representing models that can later be used within a planning framework effectively. Since our models are functionals of the SDFs, the constraints can take the information about whole objects into account to reason about their geometry and therefore especially the *interaction* between objects. Compared to a representation that only describes the surface of an object like point clouds or occupancy measures, a signed-distance field also provides information about the object at distance. As we experimentally show, this not only leads to models that perform better in their prediction accuracy compared to models learned on top of point-cloud or occupancy object representations, but SDFs also enable the functionals/learned models to have more useful gradients for planning.

In the experiments, we demonstrate the versatility of our approach by tackling two completely different tasks within one framework: On the one hand, a kinematic task where the goal is to hang mugs of different shapes on hooks of different shapes. On the other hand, a pushing scenario where boxes and L-shaped objects should be pushed to different goal regions by pushers of different sizes. In the first case, the model predicts whether the static interaction between SDFs leads to manipulation success, whereas in the latter case, the model predicts the forward dynamics in SDF space based on a history of SDF interactions of two objects. We show that our framework can be used to plan motions that involve multiple push phases. To summarize our main contributions, we propose

- To learn a novel class of kinematic and dynamic models as functionals of SDFs,
- A manipulation planning framework that utilizes these learned functionals as constraints,
- Comparison to other object representations showing the advantages of the SDFs.

2 Related Work

2.1 Signed Distance Fields as Object Representation

Representing objects or scenes as implicit surfaces [15, 16, 17] or SDFs [18, 19, 20, 21, 22, 23] is an active research topic, due to aforementioned advantages like learning shape completion, non-convex shapes etc. Our focus is not to obtain SDFs from observations in the first place. Conversely, we are interested in what can be done with SDFs in the context of model learning and manipulation planning. There are some works that utilize SDFs within trajectory optimization [24, 25, 26], but without learning or integration into a TAMP framework. While some recent approaches [27, 28, 29] have suggested that grasping of diverse objects can be addressed using implicit functions, we present a manipulation framework that utilizes SDFs for learning and formulating more general models.

2.2 Perceptual Models

There is great interest in learning predictive models in perception spaces, especially applied to the problem of pushing. So-called visual foresight approaches [30, 31, 32] aim to predict the evolution of the scene in image space. Our SDF dynamics model is also closely related to perception spaces, but, in comparison, is naturally differentiable. Xu et al. [33] use a voxelized SDF-based representation of the whole scene to predict the motion of an object when an action is applied. Our approach is more structured in the sense that we do not predict the scene flow for actions applied on a single object, but the dynamics of interacting of objects. In [34], the pushing dynamics in keypoints extracted from visual object observations is learned. However, their focus is to utilize the learned model to stabilize a trajectory with MPC. We focus on planning a complex pushing trajectory and not stabilization during execution. SE3 networks [35] learn a forward model that predicts a rigid transformation of an observed point cloud given actions. However, they need ground-truth transformations at training time (we only need SDF observations). Where most of these approaches differ from our approach is that they assume the model to be a function of the observation of a single object or the scene and an action as input. Therefore, these approaches are mostly limited to the same pusher geometry and make the assumption that actions can readily be applied to the object. Our model handles the interaction between objects of different shapes and can plan the contact establishment phase as well. Transporter networks [36] or deep visual reasoning [7] predict manipulation sequences from image spaces. However, no dynamic models are considered in these approaches.

2.3 Manipulation Planning (with learned models)

In [37, 38], a manipulation framework based on point cloud observations and manipulation primitives is proposed. Our method plans the complete motions based on learned dynamic models. Sutanto et al. [39] is related to our formulation in the sense that they learn manifolds that are used as constraints in sequential manipulation problems. However, there are no dynamic models or dependencies on the geometry of the involved objects in the learned constraints. You et al. [40] address a hanging task similar to our mug hanging experiment on a more diverse set of object categories. They use a point-cloud-based input representation to predict a hanging pose. Therefore, they need a special neural network for collision avoidance (similar to [41]), while our SDF based representation can handle collisions directly. Further, we learn a manifold of solutions instead of predicting a single hanging configuration. To summarize, what makes our approach unique is that we propose to use SDFs as a common object representation that is closely connected to perception to learn a variety of models that are able to take the interaction of objects into account and can be integrated in an optimization-based motion planning framework due to their differentiability.

3 Background on Signed-Distance Fields (SDFs)

Let $\Omega \subset \mathbb{R}^3$ be an object in the 3D Euclidean space. A function $\phi : \mathbb{R}^3 \to \mathbb{R}$, $\phi \in \Phi$ with $\phi(x) = -d(x, \partial\Omega)$ for $x \in \Omega$ and $\phi(x) = d(x, \partial\Omega)$ for $x \in \mathbb{R}^3 \backslash \Omega$ is called a signed-distance field of Ω in $\mathbb{R}^3$. Here, $d(x, \partial\Omega) = \inf_{x' \in \partial\Omega} \|x - x'\|_2$ and $\partial\Omega$ the boundary of Ω. We assume ϕ to be differentiable almost everywhere in $\mathbb{R}^3$. The way ϕ is defined ensures that inside the object, ϕ attains negative values, on the boundary zeros, and outside positive ones. We denote with the set Φ the space of all functions ϕ that are SDFs for some object.

Rigid Transformations of SDFs A central concept in this work is to rigidly transform SDFs in space. This can be realized by transforming the input where the SDF is queried. To simplify the notation, we define a *rigid transformation*, parameterized by $q \in \mathbb{R}^7$ (translation + quaternion),

$$T(q)[\phi](\cdot) := \phi\left(R(q)^T\left(\ \cdot\ - r(q)\right)\right) \tag{1}$$

of an SDF ϕ, where $R(q) \in \mathbb{R}^{3\times 3}$ is a rotation matrix and $r(q) \in \mathbb{R}^3$ the translation vector.

4 Manipulation Planning with Signed-Distance Functionals

The core idea of this work is to represent each object i in the scene as a signed-distance field ϕ^i in order to learn predictive models as functionals H of these SDFs. Based on the learned functionals, we formulate a trajectory optimization problem where the decision variable is a trajectory of rigid transformations applied on the initial SDFs as they have been observed in the initial scene.

More specifically, through interaction with the environment, we aim to learn functionals of the form $H : \Phi \times \cdots \times \Phi \to \mathbb{R}$ that map multiple SDFs of multiple, possible different objects at possibly different consecutive times to a real number. These are trained in a way that a value of zero implies that the SDFs as input are compatible with what has been learned through interaction with the environment. Otherwise, they should attain a positive value, hence functionals H discriminate correct from incorrect dynamics or desired from undesired manipulations.

The learned functionals then define constraints for the (hybrid) trajectory optimization problem

$$\min_{\substack{q_{0:KT}, q_t \in \mathbb{R}^{7\cdot n_O} \\ K \in \mathbb{N},\ s_{1:K}}} \sum_{t=1}^{KT} c\big(q_{t-l:t}, s_{k(t)}\big) \tag{2a}$$

$$\text{s.t.} \quad \forall_{H \in \mathbb{H}(s_{k(t)})} \ : \ H\left(\left(T(q_t^i)[\phi^i]\right)_{(t,i)\in\mathcal{I}_H(s_{k(t)})}\right) = 0 \tag{2b}$$

$$s_{1:K} \in \mathbb{S}(S), \quad q_0 = 0. \tag{2c}$$

The discrete variable s_k determines which functionals H from the set $\mathbb{H}(s_{k(t)})$ are active at which of the $K \in \mathbb{N}$ phases of the motion ($k(t) = \lfloor t/T \rfloor$). This number of phases is part of the decision problem. The trajectory $q_{0:KT}$ of rigid transformations is discretized in time into $T \in \mathbb{N}$ steps per phase. If n_O is the number of objects in the scene S, then $q_t \in \mathbb{R}^{7\cdot n_O}$, leading to $7 \cdot KTn_O$ continuous variables. Further, s_k selects through the set $\mathcal{I}_H(s_{k(t)})$ the time slice and object index tuples (t, i) that determine the SDFs ϕ^i, which have been transformed through q_t^i, at the times t of the trajectory on which the functional constraints H depends on. This problem formulation is inspired by LGP [10], but the constraints are replaced by learned functionals of SDFs. The set $\mathbb{S}(S)$ contains all valid sequences $s_{1:K}$ of such discrete variables for the scene S. The goal of the manipulation problem is specified through $\mathbb{S}(S)$ by s_K selecting a desired goal functional constraint that has to be fulfilled at the end q_{KT} of the trajectory. Solving (2) therefore involves a tree search over nodes $s_{1:K}$ such that the continuous optimization problem implied by the choice of $s_{1:K}$ at a node of the tree is feasible. The role of q in the optimization problem is not absolute object poses, but rather rigid transformations applied to the SDFs ϕ^i that represent the configurations of the objects as observed in the scene initially. With the term c, we can include regularizing motion costs. As will be described in sec. 5.1, the forward dynamic model we learn for pushing implies a constraint on the evolution of one object based on the motion of another object. Therefore, we only add motion costs to those degrees of freedom that can be interpreted as being controlled, meaning the motion of the other object. From the perspective of (2), there is no explicit notion of controlled actions.

5 Deep Signed-Distance Functionals

This section presents two main types of models we propose. First, a way of learning forward dynamic models that predict the dynamics in SDF space based on the interaction between objects. Second, a kinematic success model that determines whether a static configuration of interacting SDFs leads to manipulation success. All functionals we consider are of the form $H : \Phi \times \cdots \times \Phi \to \mathbb{R}$, i.e. they only take the SDFs of interacting objects as input, there is no explicit notion of position, orientation, action etc. Therefore, the functionals can be used at arbitrary locations in space.

Bounding-Box To define most of the following functionals and those in sec. 6, we utilize a set $\mathcal{X}$ with the property $\Omega \subset \mathcal{X} \subset \mathbb{R}^3$ for all objects Ω that are involved. This set should be large enough to cover the relevant workspace of the manipulation problem where the interaction between the objects should occur. A more detailed discussion about the role of $\mathcal{X}$ can be found in sec. 5.3.

5.1 Forward Dynamic Models

Generally, a forward model predicts future states/observations of a system given the current or additionally a history of states/observations. In the context of objects being represented solely as SDFs, we propose to learn a forward model $F : \Phi \times \cdots \times \Phi \to \Phi$ that predicts the SDF of an object ϕ_t^1 at time step t based on a history of SDF observations of the object $\phi^1_{t-l:t-1}$ until time $t-1$ and the motion of another object $\phi^2_{t-l:t}$ until time t. This means F as

$$\phi_t^1(\cdot) = F\big[\phi^1_{t-l:t-1}, \phi^2_{t-l:t}\big](\cdot) \tag{3}$$

is an SDF itself that can be queried in $\mathbb{R}^3$. Interactions between more than two objects are possible, but we focus on pair-interactions in the present work. If $l = 1$, F is a quasi-static model. Internally, F can be defined to either directly predict the SDF ϕ_t^1 as in (3) or the flow

$$\phi_t^1(\cdot) = \phi^1_{t-1}(\cdot) + F_{\text{flow}}\big[\phi^1_{t-l:t-1}, \phi^2_{t-l:t}\big](\cdot) \tag{4}$$

from ϕ^1_{t-1} to ϕ^1_t. In both cases, the functional H for planning is then naturally defined as

$$H_F\big(\phi^1_{t-l:t}, \phi^2_{t-l:t}\big) = \int_{\mathcal{X}} \Big(\phi_t^1(x) - F\big[\phi^1_{t-l:t-1}, \phi^2_{t-l:t}\big](x)\Big)^2 \,\mathrm{d}x. \tag{5}$$

For a perfect model F, this functional H_F attains a zero value if and only if the evolution of $\phi^1_{t-l:t}$ and $\phi^2_{t-l:t}$ is compatible with the underlying physical process in the space $\mathcal{X}$. Therefore, the loss function to train F is also (5) for a dataset $D = \big\{\big(\phi^1_{0:l}, \phi^2_{0:l}\big)_i\big\}_{i=1}^n$ of such consecutive SDF motions of the two objects. Since F takes as input the complete SDFs of the objects and not just values like the distance between objects and their contact point locations, it can learn to reason not only about these quantities, but also the contact geometry, relative object movements, center of mass and inertial parameters (assuming an equal density of the objects), all of which are necessary quantities to represent the dynamics. This way, F inherently takes the geometry of the objects into account. Note that usually, forward models are understood in terms of a function that maps the current state (history) and an (abstract) action a to the next state. For SDFs, this would mean a model of the form $\phi_t^1 = F[\phi^1_{t-l:t-1}, a_{t-1}]$. In our case, however, there is no notion of an abstract action, instead, our formulation learns a generic model of the interaction between two objects, where the motion of one object (ϕ^2) influences the other (ϕ^1). Therefore, while the transformation applied to ϕ^2 can be interpreted as an action, the model has no action as input and hence can deal with different geometries of ϕ^2, which is not possible in case of an abstract action without ϕ^2 also being an input.

5.2 Kinematic Success Models

Many tasks in manipulation planning can be specified in terms of static success models instead of a full forward dynamics model. We call a model that predicts whether a configuration of potentially multiple SDFs at the same time slice leads to manipulation success a kinematic success model. Assume through interaction with the environment, a dataset $D = \left\{(\phi^j)_{j\in\mathcal{I}_i}, y^i\right\}_{i=1}^n$ of SDFs representing $|\mathcal{I}_i|$ many objects has been obtained with $y^i = 1$ indicating that the configuration of SDFs leads to manipulation success, $y^i = 0$ to failure. Then learning H is similar to a classification problem, where $H\left((\phi^i)_{i\in\mathcal{I}}\right) = 0$ implies success prediction. This way, H can model a manifold of feasible configurations and not only a single solution. See sec. F for details (loss function etc.).

5.3 Learning Functionals with Neural Networks

So far, we have not discussed how functionals of the form $H : \Phi \times \cdots \times \Phi \to \mathbb{R}$ can be learned or even queried in the first place with usual function approximators like neural networks, since, in general, the neural network would have to take functions as infinite dimensional objects as input. To approximate this, we choose in this work the straight-forward approach by evaluating $\phi \in \Phi$ on a discretized version of the set $\mathcal{X}$, denoted by $\mathcal{X}_h$. As discussed previously, the set $\mathcal{X}$ should cover the relevant region of the workspace where the interaction between the objects takes place. We specifically do not assume $\mathcal{X}$ to be aligned or perfectly centered with the objects that are involved. This way, the dynamics model from sec. 5.1 can be realized by

$$F\left[\phi^1_{t-l:t-1}, \phi^2_{t-l:t}\right](x) \approx F_\theta(\phi^1_{t-l:t-1}(\mathcal{X}_h), \phi^2_{t-l:t}(\mathcal{X}_h), x) \tag{6}$$

with F_θ being usual neural network architectures. During training, the integral in (5) is approximated over the same discretized $\mathcal{X}_h$ for simplicity. Hence, the dataset to train F can contain the SDF observations at the grid points of $\mathcal{X}_h$ only. However, F_θ still approximates an SDF which can be queried at arbitrary $x \in \mathbb{R}^3$ and does not only predict the values on the grid points. For general functionals H, the evaluation is analogous, i.e. $H\Big(\left(\phi^i\right)_{i\in\mathcal{I}}\Big) \approx H_\theta\left(\left(\phi^i(\mathcal{X}_h)\right)_{i\in\mathcal{I}}\right)$. Technically, $\mathcal{X}_h \subset \mathbb{R}^{d\times h\times w}$ is a regular grid which allows us to encode $\phi(\mathcal{X}_h)$ using 2D or 3D convolutions. In contrast to an occupancy grid, the evaluation of $\phi(\mathcal{X}_h)$ contains more information about the object than whether there is an object at the grid point or not. Note that the differentiabilty of $H\Big(\left(T(q^i)[\phi^i(\mathcal{X}_h)]\right)_{i\in\mathcal{I}}\Big)$ with respect to q^i is maintained, which is another advantage of representing such models as functionals of SDF functions evaluated on a grid instead of static values on a grid. During training, it is sufficient to only have the SDF values evaluated on a gird, no other information like actions or velocity/pose estimations are needed.

6 Task Constraint Functionals

Here we present analytical functionals of SDFs that are useful to specify goals of a manipulation problem or other task aspects. These functionals are general as a direct consequence of our object representations being SDFs. Therefore, there is no advantage or need to learn these given the SDFs.

6.1 Pair-Collision between Objects

Collision avoidance is an inherent part of many task specifications. Given two SDFs ϕ_1, ϕ_2, we can measure whether they are in collision via their overlap integral

$$H_{\text{coll}}(\phi_1, \phi_2) = \int_{\mathcal{X}} [\phi_1(x) < 0]\,[\phi_2(x) < 0]\,\mathrm{d}x. \tag{7}$$

The indicator bracket $[\cdot]$ means $[P] = 1$ if P is true, otherwise $[P] = 0$. The integral in (7) integrates over the space where both SDFs are negative at the same time, which is only the case if the two objects overlap, hence are in collision. The gradients of (7) are smoothed using the sigmoid function $\sigma(z) = \frac{1}{1+\exp(-z)}$, i.e. $[\phi_{1,2}(x) < 0] = \sigma\left(-a\phi_{1,2}(x)\right)$ with a parameter $a > 0$.

6.2 Goal Region

If part of the task specification is that an object ϕ_1 is fully contained inside the boundary of another object ϕ_g, called the goal region, then a similar integral as for the pair-collision can be utilized

$$H_{\text{g}}(\phi_1, \phi_g) = \int_{\mathcal{X}} [\phi_1(x) < 0]\,[\phi_g(x) > 0]\,\mathrm{d}x \approx \int_{\mathcal{X}} \sigma\left(-a\phi_1(x)\right)\sigma\left(a\phi_g(x)\right)\mathrm{d}x. \tag{8}$$

Here, points outside of the goal region that are inside the object count towards the integral.

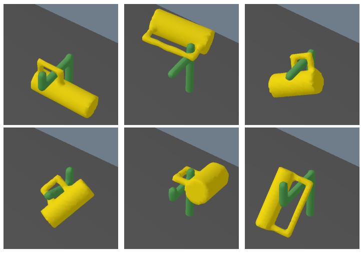

Figure 1: Found solution configurations by the optimizer using the learned model.

	solution found	success rate	**total scenes solved**
SDF opt. + sampling	98.7%	88.5%	**87.3%**
SDF opt. only	51.3%	93.5%	47.3%
SDF sampling only κ_1	83.8%	82.3%	68.9%
occupancy ψ, sampling κ_3	100.0%	34.0%	34.0%
pointnet++, sampling κ_1	100.0%	82.0%	52.0%

Table 1: Success rates of mug hanging experiment. Total scenes solved means the percentage of scenes in the evaluation dataset for which a solution was found that is not in collision and is stable when dropped. Only best results for each object representation shown. Full results see Tab. 3 and sec. A.5.

6.3 Establishing Contact between Objects

Establishing and maintaining contact between objects is central for many manipulation tasks. One way to model that the distance between two objects ϕ_1 and ϕ_2 should be zero is via the functional

$$H_{\text{PoC}}(\phi_1, \phi_2) = \min_{p \in \mathcal{X}} |\phi_1(p)| + |\phi_2(p)|. \tag{9}$$

7 Experiments

7.1 Mug-Hanging: Kinematic Success Model

In this experiment, we want to find rigid transformations applied on observed mugs of different shapes in a scene to hang them stably on hooks of different types. The functional H_{hang} is therefore a kinematic success model that takes the SDFs of the mug and the hook as input. To generate data to learn H_{hang}, we randomly sample scenes of different mug and hook shapes (1600 scenes for training, 400 for testing and 150 for evaluation). See Fig. 15 for examples of mugs and hooks in the evaluation data. Then we sample for each scene in the training and test data the position and orientation of the mug uniformly in the bounding box $\mathcal{X}$ until at least one successful configuration has been obtained where the mug does not fall on the ground when being dropped from the sampled configuration while at the same time not being in collision with the hook. We use Bullet [42] to simulate the dropping. In total, 20 configurations per scene are generated. Since sampling a successful configuration is a rare event, for the majority of the scenes, only one successful and 19 failure configurations are contained in the training and test data, making learning challenging. Another challenge of this task is that the model has to reason about both the hook and mug geometry jointly. Formulating an analytical model, e.g. on a mesh-based object representation, to model this constraint is non-trivial.

7.1.1 Performance with Optimization

Fig. 1 shows solution configurations found by our model H_{hang} as an optimization objective. Interestingly, the solutions not always contain the intuitive solution, but also ones where other parts of the hook are being utilized (middle column in Fig. 1). The optimization problem (2) to solve this mug hanging problem has two objectives, the learned kinematic success functional H_{hang} and the pair-collision H_{coll} from sec. 6.1. While in principle H_{hang} also learns to avoid collisions, we found that the robustness in avoiding collisions increases when including H_{coll}. The learned functional H_{hang} is, in general, non-convex in the rigid transformation q of the mug. Therefore, we observed that gradient-based optimization is not sufficient for the optimizer to find a feasible solution, i.e. where H_{hang} predicts zero, in every instance [43]. To overcome this issue, we restart the optimization procedure up to 20 times with a randomly sampled initial guess of the mug in $\mathcal{X}$. Fig. 16 shows an example of a sampled initial configuration from which the optimizer is started (left), then the optimized configuration (middle) and finally, the configuration after simulation. Tab. 1 shows the success rates on the evaluation scenes. As one can see, for the proposed approach where objects are represented as SDFs using optimization and sampling, in 98.7% of the evaluation scenes, a solution is found where H_{hang} predicts success and no collision is violated (first column). Out of these, 88.5% are stable configurations (checked by simulation) and the optimized configuration of the mug is collision free with the hook, leading to 87.3% total solved scenes (last column). When the optimization is run only once (second row), then only in 51.3% of the cases it converges to a feasible solution.

7.1.2 Comparison to Sampling, Point-Cloud and Occupancy Measure Representations

This section shows that learning a kinematic success model based on the proposed SDF object representation outperforms other representations (point-clouds and occupancy measures) and further highlights the advantage of the models learned with SDFs providing useful gradients by comparing it to sampling without optimization. For full results, refer to sec. A and Tab. 3. The sampling

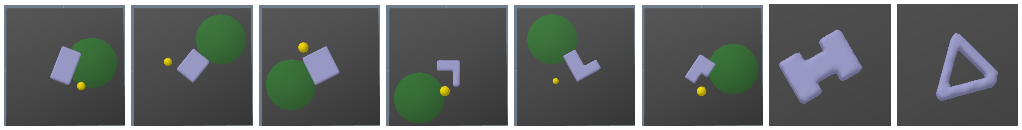

Figure 2: Different pushing scenarios in evaluation dataset. Yellow is the pusher, green the goal region and light blue the object. The two images from the right show out-of-distribution generalization shapes.

approach draws relative transformations of the mugs uniformly in $\mathcal{X}$ until the evaluation with the learned H_{hang} and the collision functional H_{coll} predicts a successful and collision free configuration with a threshold κ. As one can see in Tab. 1 and Tab. 3, our proposed approach has a significantly higher performance (87.3%) compared to the best threshold for pure sampling with SDF (68.9%) and the best of the other object representations (34% with occupancy measure, 52% with point-cloud).

7.2 Pushing Objects on a Table: Dynamic Model

In this experiment, we consider the task of pushing boxes and L-shaped objects of different dimensions with a spherical pusher of different radii into a goal region ϕ_g on a table. Fig. 2 visualizes typical objects, pushers and goal regions. The goal region has to be large enough that all possible objects fit. We again use Bullet as a simulator to generate data to train a dynamics model of the from described in sec. 5.1 with $l = 1$, i.e. H_F is a function of four SDFs ϕ_t^1, ϕ_{t-1}^1 (object) and ϕ_t^2, ϕ_{t-1}^2 (pusher). In total, 14975 different scenes (including shapes and initial configuration) are sampled where random push actions biased roughly towards the object center are applied until the object leaves the table. Since the dynamics and interaction of the objects in this scenario can be described in the 2D plane, the 3D signed distance functions of the objects are evaluated in the 2D set $\mathcal{X}_h \in \mathbb{R}^{140\times 140}$ only. Therefore, the model F predicts the dynamics of ϕ^1 in this 2D projection.

7.2.1 Forward Prediction Error

Tab. 2 shows the one-step prediction error on the evaluation dataset for the flow model F_{flow} (4) and the direct SDF prediction F (3). The way we utilize the model within the trajectory optimization problem never asks for predictions more than one step into the future. We train one single dynamics model for both box and L-shaped objects and different pushers. The prediction error is the RMSE of predicting the correct SDF values in $\mathcal{X}_h$. The last row shows the error if the model would simply predict the next state as the last state of the object. As one can see, F_{flow} achieves a lower error than F. This is due to F having to predict the complete SDF, while F_{flow} only the flow. In phases of the motion where there is no contact between the object and the pusher, both models F_{flow} and F have to learn that the object should not move (and F has to predict the complete SDF in this case as well), which is also non-trivial, but they accomplish this with low error.

	contact phase	no contact phase
F_{flow}	3.4 ± 1.6	1.4 ± 1.8
F	5.8 ± 1.7	5.2 ± 1.6
$\phi_t^1 = \phi_{t-1}^1$	10.8 ± 3.4	0

Table 2: RMSE [mm] on evaluation dataset.

7.2.2 Comparison to other Object Representations (Point-Cloud and Occupancy Measure)

In sec. A.6.1 and sec. A.6.2 we present and explain a comparison of the forward prediction error between models learned with object representations being SDFs, occupancy measures and point-clouds. As shown in Tab. 4 and Tab. 5, models learned with the SDF representation outperform models based on point-clouds and occupancy measures in their predictive performance. Further, in Tab. 5 and sec. A.4, we also show that one can learn image conditioned SDFs and dynamic models on top of the learned SDF simultaneously with no noticeable performance degradation.

7.2.3 Planning with the Learned Model and Execution Performance

Having learned the pushing dynamics prediction model, we now utilize it within (2) to solve the task of pushing the object into the goal region. There are four constraints. First, the dynamics model H_F and, second, the goal region H_{g}. While this seems to be enough to specify the problem fully, we add two additional constraints, H_{coll} and H_{PoC}. The discrete variable s_k of (2) decides whether there are one or two push phases. Only in a push phase, H_{PoC} is active. H_{coll} is always active. Similarly to the mug hanging experiment, local minima are a core issue as well. Therefore, we initialize the pusher position at phase 1 or 2 on a set of 4 different points around the object. These 4 points around the object are always the same in all scenarios, no matter of the size, shape or orientation of the object. Compared to other approaches where the action space has to be chosen much more carefully, we believe that this is a rather weak prior. The initialization also does not start from contact with the object or similar, because our problem contains the challenge of contact establishment and possible breakage to push from a different side to achieve the goal. Therefore, due to this initialization, there

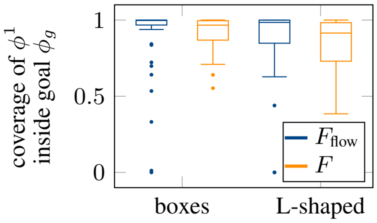

Figure 3: Pushing performance on evaluation scenes in terms of the amount of ϕ^1 that is inside of the goal region at the end of the execution. A value of 1 means that the object is fully contained in the goal region.

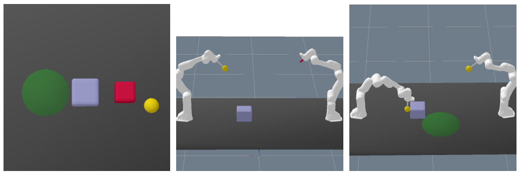

Figure 4: Generalization experiments for pushing scenario. Left: interaction between three objects to solve the task. Middle: the goal is that the right robot arm touches the object. Right: the two robots have to collaborate to push the object into the goal region, which would not be possible with one arm alone.

are 20 different optimization problems we solve for each scene (4 for one pushing phase, 4^2 for two pushing phases). To evaluate the performance, we execute the planning result with the least constraint violation and cost *open-loop* in the simulator. Despite the fact that pushing is unstable over long-horizons, our proposed approach achieves a high performance. As shown in Fig. 3, which plots the amount of the object that is inside of the goal region at the end of the execution, using the learned F_{flow}, the approach achieves 99.7% (median) coverage of the object inside the goal region on box pushing and 98.4% (median) on the L-shaped objects (50 evaluation scenes each). Please note that, although the goal region for small objects seems large, the optimizer usually moves the object until it is just barely inside the goal region and not any further. Therefore, even very small deviations during the open-loop execution already lead to some parts sticking out. For the larger objects in the evaluation scenes, the goal region is barely large enough. With the direct F, the performance is a bit worse, but still high (median 96.6% for boxes, 91.5% for L-shaped objects).

Sec. B demonstrates that our proposed SDF framework outperforms an approach where the optimization problem is formulated for objects represented as meshes and the analytic dynamic model from [13] for the pushing dynamics. Refer to Fig. 6 for quantitative results. Finally, in sec. A.6.3 and Fig. 5 we investigate the importance of models learned on top of SDF representations providing useful gradients for planning in comparison to models with point-clouds and occupancy measures.

7.2.4 Ablation Study

Sec. D presents an ablation study regarding the importance of the additional objectives H_{coll} and H_{PoC} in the optimization problem. Results in Fig. 11 show that while it is possible to solve the scenes without them, the performance greatly increases if they are part of the problem formulation.

7.2.5 Generalization to Out-of-Distribution Shapes, Multiple Objects, and Robots

In sec. C.1, we demonstrate that the framework and the learned model generalizes to shapes beyond the training distribution. See Fig. 2 and Fig. 7 for those shapes. Quantitative results presented in Tab. 6 and Fig. 8 indicate that the model achieves both high prediction accuracy and high performance when used for planning. Furthermore, as seen in Tab. 6, a model learned with SDFs generalizes significantly better than with point-clouds, also relative to the results obtained on-distribution. We further show in sec. C.3 and sec. C.2 that the framework is capable of generalizing to scenes that contain obstacles (Fig. 10) and a scenario where three objects interact in order to solve the task (Fig. 9 and Fig. 4). Finally, in sec. E we demonstrate multiple scenarios where the learned pushing dynamics model is embedded into a scene that contains robots. All these generalization experiments require no change in the methodology or to learn a new model, showing the generality and versatility of our proposed framework. For more details, refer to the respective sections in the appendix.

8 Conclusion

In this work, we have shown that the constraints of a trajectory optimization problem for solving manipulation problems can be formulated in terms of learned functionals of SDFs only. SDFs can serve as a *common* object representation across completely different tasks. The functionals can naturally model the interaction between objects of arbitrary shapes and can be learned directly from SDF observations, which is closely connected to perception. We have shown that learning models on top of SDFs outperform other object representations like point-clouds and occupancy measures both in terms of prediction accuracy and the ability to plan. The greatest challenge of our framework are local minima of the resulting trajectory optimization problem. While sampling strategies for initial guesses can mitigate this to some extend, it is an issue, which is not unique to our approach, but many nonlinear trajectory optimization formulations. While we have considered rigid objects in this work only, we believe that the proposed approach can be extended to deformables as well.

Acknowledgments

Danny Driess thanks the International Max-Planck Research School for Intelligent Systems (IMPRS-IS) for the support. This research has been supported by the German Research Foundation (DFG) under Germany's Excellence Strategy – EXC 2002/1–390523135 "Science of Intelligence". The authors thank the anonymous reviewers for their comments.

References

[1] D. Driess, J.-S. Ha, R. Tedrake, and M. Toussaint. Learning geometric reasoning and control for long-horizon tasks from visual input. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2021.

[2] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 2021.

[3] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical planning in the now. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[4] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. J. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *Proc. of the Int. Conf. on Robotics and Automation (ICRA)*, 2014.

[5] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki. An incremental constraint-based framework for task and motion planning. *International Journal on Robotics Research*, 2018.

[6] V. N. Hartmann, O. S. Oguz, D. Driess, M. Toussaint, and A. Menges. Robust task and motion planning for long-horizon architectural construction planning. arXiv:2003.07754, 2020.

[7] D. Driess, J.-S. Ha, and M. Toussaint. Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image. *arXiv:2006.05398*, 2020.

[8] I. Mordatch, E. Todorov, and Z. Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 31(4):1–8, 2012.

[9] M. Posa, C. Cantu, and R. Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.

[10] M. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Robotics: Science and Systems*, 2018.

[11] F. R. Hogan, E. R. Grau, and A. Rodriguez. Reactive planar manipulation with convex hybrid MPC. In *Int. Conf. on Robotics and Automation (ICRA)*, 2018.

[12] N. Doshi, F. R. Hogan, and A. Rodriguez. Hybrid differential dynamic programming for planar manipulation primitive. In *Int. Conf. on Robotics and Automation (ICRA)*, 2020.

[13] M. Toussaint, J.-S. Ha, and D. Driess. Describing physics for physical reasoning: Force-based sequential manipulation planning. *IEEE Robotics and Automation Letters*, 2020.

[14] D. Driess, J.-S. Ha, and M. Toussaint. Learning to solve sequential physical reasoning problems from a scene image. *The International Journal of Robotics Research*, to appear.

[15] Z. Chen and H. Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[16] S. Liu, S. Saito, W. Chen, and H. Li. Learning to infer implicit surfaces without 3d supervision. *arXiv preprint arXiv:1911.00767*, 2019.

[17] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[18] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[19] V. Sitzmann, E. R. Chan, R. Tucker, N. Snavely, and G. Wetzstein. Metasdf: Meta-learning signed distance functions. In *arXiv*, 2020.

[20] M. Atzmon and Y. Lipman. Sal: Sign agnostic learning of shapes from raw data. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[21] C. Jiang, A. Sud, A. Makadia, J. Huang, M. Nießner, T. Funkhouser, et al. Local implicit grid representations for 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6001–6010, 2020.

[22] M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and Z. Corse. Local optimization for robust signed distance field collision. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 2020.

[23] A. Fuhrmann and G. Sobottka. Distance fields for rapid collision detection in physically based modeling. 2003.

[24] K. Hauser. Semi-infinite programming for trajectory optimization with non-convex obstacles. *The International Journal of Robotics Research*, 2018.

[25] M. Zhang and K. Hauser. Semi-infinite programming with complementarity constraints for pose optimization with pervasive contact. In *IEEE International Conference on Robotics and Automation*, 2021.

[26] S. Pfrommer, M. Halm, and M. Posa. Contactnets: Learning of discontinuous contact dynamics with smooth, implicit representations. *Conference on Robot Learning*, 2020.

[27] M. Breyer, J. J. Chung, L. Ott, S. Roland, and N. Juan. Volumetric grasping network: Real-time 6 dof grasp detection in clutter. In *Conference on Robot Learning*, 2020.

[28] Z. Jiang, Y. Zhu, M. Svetlik, K. Fang, and Y. Zhu. Synergies between affordance and geometry: 6-dof grasp detection via implicit representations. *arXiv preprint arXiv:2104.01542*, 2021.

[29] M. Van der Merwe, Q. Lu, B. Sundaralingam, M. Matak, and T. Hermans. Learning continuous 3d reconstructions for geometrically aware grasping. In *Int. Conf. on Robotics and Automation (ICRA)*, 2020.

[30] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv:1812.00568*, 2018.

[31] R. Kandukuri, J. Achterhold, M. Moeller, and J. Stueckler. Learning to identify physical parameters from video using differentiable physics. In *Pattern Recognition: 42nd DAGM German Conference*, 2021.

[32] Z. Xu, J. Wu, A. Zeng, J. B. Tenenbaum, and S. Song. Densephysnet: Learning dense physical object representations via multi-step dynamic interactions. *arXiv preprint arXiv:1906.03853*, 2019.

[33] Z. Xu, Z. He, J. Wu, and S. Song. Learning 3d dynamic scene representations for robot manipulation. In *Conference on Robotic Learning (CoRL)*, 2020.

[34] L. Manuelli, Y. Li, P. Florence, and R. Tedrake. Keypoints into the future: Self-supervised correspondence in model-based reinforcement learning. In *Conference on Robotic Learning (CoRL)*, 2020.

[35] A. Byravan and D. Fox. Se3-nets: Learning rigid body motion using deep neural networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 173–180, 2017.

[36] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani, and J. Lee. Transporter networks: Rearranging the visual world for robotic manipulation. *Conference on Robot Learning (CoRL)*, 2020.

[37] A. Simeonov, Y. Du, B. Kim, F. R. Hogan, P. Agrawal, and A. Rodriguez. Learning to plan with pointcloud affordances for general-purpose dexterous manipulation. *Conference on Robot Learning*, 2020.

[38] S. Mukherjee, C. Paxton, A. Mousavian, A. Fishman, M. Likhachev, and D. Fox. Sim-to-real task planning and execution from perception via reactivity and recovery. *arXiv preprint arXiv:2011.08694*, 2020.

[39] G. Sutanto, I. M. R. Fernández, P. Englert, R. K. Ramachandran, and G. S. Sukhatme. Learning equality constraints for motion planning on manifolds. *arXiv preprint arXiv:2009.11852*, 2020.

[40] Y. You, L. Shao, T. Migimatsu, and J. Bohg. Omnihang: Learning to hang arbitrary objects using contact point correspondences and neural collision estimation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[41] M. Danielczuk, A. Mousavian, C. Eppner, and D. Fox. Object rearrangement using learned implicit collision functions. *arXiv:2011.10726*, 2020.

[42] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016–2021.

[43] J. Ortiz-Haro, J.-S. Ha, D. Driess, and M. Toussaint. Structured deep generative models for sampling on constraint manifolds in sequential manipulation. In *Proc. of the Annual Conf. on Robot Learning (CORL)*, 2021.

[44] X. Yan. Pointnet/pointnet++ pytorch. https://github.com/yanx27/Pointnet_Pointnet2_pytorch, 2019.

[45] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

[46] D. Driess, S. Schmitt, and M. Toussaint. Active inverse model learning with error and reachable set estimates. In *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, 2019.

A Comparisons to Point-Cloud and Occupancy Measure Object Representations as well as SDFs Learned from Images

The purpose of this section is to investigate the importance of the object representations being signed-distance functions for both the mug-hanging and the pushing scenario. In order to do so, we consider two other object representations, namely occupancy measures, which are functions that indicate whether there is an object at a certain location or not, and point-clouds which represent the surface of an object as a set of points in 3D space. Moreover, we also show that one can learn an image-conditioned SDF and the dynamics model simultaneously for the pushing scenario while maintaining high performance.

A.1 Occupancy Measure Object Representation

An occpuancy measure $\psi : \mathbb{R}^3 \to \{-1, 1\}$ is a function that is defined such that $\psi(x)$ is -1 if $x \in \mathbb{R}^3$ is inside the object ψ represents and $+1$ if x is outside the object (for the boundary of the object there are multiple conventions). The function ψ can therefore be interpreted as an indicator whether there is an object at x or not.

Formally, we can utilize a signed-distance function ϕ to define the corresponding occupancy measure ψ via

$$\psi(x) = \text{sign}(\phi(x)). \tag{10}$$

To make ψ differentiable, one can use

$$\psi_\text{s}(x) = \tanh(b\,\phi(x)) \tag{11}$$

for $b > 0$. The smaller b, the smoother the boundary and hence the gradients become more well-behaved. On the other hand, if b is too small, then the objects do not have well-defined boundaries anymore, which is an issue, since we want to model both collision avoidance and contact establishment at the same time. Further, if b is very small, then ψ locally around the surface of the object maintains the information of the SDF ϕ it was constructed with. We therefore test values of $b = 100$ and $b = 1000$.

Compared to occupancy values on a static grid, ψ contains more information, since it is a function of $x \in \mathbb{R}^3$, which means that it can also be transformed rigidly in space using (1) as for ϕ. For static occupancy grids, such transformations are also possible, but they require a post-processing step to recreate a valid occupancy grid after the transformation.

A.2 Point-Cloud Object Representation

As another object representation, we consider point-clouds, which are a set of 3D points on the surface of the object. Formally, we can define a point-cloud $\mathcal{P}$ for an object via the corresponding signed-distance function ϕ of the object as

$$\mathcal{P} = \left\{x_1, \ldots, x_N \in \mathbb{R}^3 : \forall_{i=1,\ldots,N}\ \phi(x_i) = 0\right\}. \tag{12}$$

In order to make the comparison fair, the number N of points is chosen such that the resolution of the SDF and the point-cloud is similar. More precisely, we first run the marching cube algorithm on the SDF evaluated on the same $\mathcal{X}_h$ as in the other experiments and then extract $\mathcal{P}$ as the vertex points of the obtained mesh.

As described in sec. 5.3, H.1 and I.1, we can encode the SDF as input to the functionals H via grid evaluation of the SDF function, followed by a convolutional neural network encoder architecture. For point-clouds, this is not directly possible. Therefore, we utilize a so-called pointnet architecture to encode the input point-clouds into a feature vector from which the dynamics model and the kinematic success model is defined. We use the pointnet and pointnet++ implementation from [44] for the point-cloud encoding.

A.3 Dynamics Model via Predicting Rigid Transformations

Our proposed forward dynamics model (3) and (4), described in sec. 5.1, predicts the SDF *function* ϕ_t^1 of object 1 at time t as a function of the history of SDF observations of the object $\phi_{t-l:t-1}^1$ until time $t-1$ and the motion of another object $\phi^2_{t-l:t}$ until time t. The model does not predict the values on a static grid, but as a function that can be queried in $\mathbb{R}^3$.

For point-clouds, there is no notion of a function that could be predicted. Instead, a forward dynamics model in point-cloud space means to predict the positions of the points at the next timestep. To achieve this, we learn a model

$$\Delta q^1 = F_q\big(\mathcal{P}^1_{t-1}, \mathcal{P}^2_{t-1}, \mathcal{P}^2_t\big) \tag{13}$$

that predicts the rigid transformation Δq^1 (in case of the pushing scenario $\Delta q^1 \in \mathbb{R}^3$, x, y-translation and α rotation) of the point-cloud $\mathcal{P}^1_{t-1}$ of object 1 at time $t-1$ as a function of the point-clouds $\mathcal{P}^2_{t-1}$ and $\mathcal{P}^2_t$ of object 2 such that

$$\mathcal{P}^1_t = R\big(\Delta q^1\big)\mathcal{P}^1_{t-1} + r\big(\Delta q^1\big). \tag{14}$$

This means whereas for training the model based on SDFs and occupancy measures it was sufficient to have a dataset of just SDF/occupancy observations, the training process for the point-cloud model requires those relative transformations of the point-cloud either as ground-truth data or via a (non-trivial) point-cloud registration step. We assume to have access to ground-truth rigid transformations Δq^1 to train the point-cloud model.

In order to compare the performance of this point-cloud model with the SDF and occupancy measure approach, we also train models

$$\Delta q^1 = F_q\big(\phi^1_{t-1}, \phi^2_{t-1}, \phi^2_t\big) \tag{15}$$

$$\Delta q^1 = F_q\big(\psi^1_{t-1}, \psi^2_{t-1}, \psi^2_t\big) \tag{16}$$

for both the SDF and occupancy measure that predict the rigid transformations of these object representations (as functions). In this case

$$\phi^1_t = T\big(\Delta q^1\big)\big[\phi^1_{t-1}\big] \tag{17}$$

$$\psi^1_t = T\big(\Delta q^1\big)\big[\psi^1_{t-1}\big] \tag{18}$$

with the transformation T as defined in (1).

As a side note, while we have focused on rigid objects in this work, our approach of predicting a function would directly be applicable to the deformable case for SDFs and occupancy measures, which would not directly be possible with a point-cloud model that predicts the rigid transformation of the points.

A.4 Learning Image Conditioned SDFs and Dynamics Model Simultaneously for Pushing Scenario

The experiments in this work assume to have access to the SDFs of the involved objects. Here, we show for the pushing scenario that it is possible to train a neural network that predicts the SDF from an image observation of the corresponding object and the dynamics model based on top of the learned SDF simultaneously with very little performance degradation compared to having the ground-truth SDF available. For this, we assume to have a segmented image $I \in \mathbb{R}^{h_I \times w_I}$ of each object. Then, we learn one function for all objects such that

$$\phi(\cdot, I) \in \Phi \tag{19}$$

is the SDF corresponding to the object masked in the image I. We assume an overhead camera perspective of the table such that the images I are masked depth images of the objects from above. We learn a single ϕ for both the pusher and the object that is being pushed. Also note that the masked image is not transformed into the origin or some other canonical frame, i.e. the image shows the configuration (position and rotation) of the object as it is in the scene. Therefore, no pose estimation step or similar is necessary, $\phi(\cdot, I)$ is the SDF of the object in the configuration as in the scene.

For quantitative results of this experiment, see Tab. 5 (second row) and sec. A.6.1 below.

A.5 Comparison Results for Mug-Hanging

Tab. 3 shows the success rates in terms of the percentage of successfully solved scenes for the mug hanging experiment for the object representations being SDFs (proposed approach) and occupancy measures/point-clouds as baselines. We investigate various different approaches, namely optimization + sampling, optimization only, sampling only with different feasibility thresholds κ, and two different pointnet architectures.

The results in Tab. 3 have to be interpreted in the following way. For each method (object representation, solution method, thresholds), we report in the first column the percentage of scenes for which the method (optimization + sampling, optimization only, sampling with different thresholds) found a solution within the maximum allowed functional evaluations. To allow for a fair comparison, both the optimization problem (including its up to 20 restarts) and the sampling procedure are allowed to use the exact same total maximum number of functional evaluations (20,000). Then the second column shows the percentage of the found solutions where the solution configuration leads to a stable hanging of the mug on the hook when dropped in the simulator. Since the simulator also simulates if the solution configuration of the mug is in collision with the hook, we report in the third column the percentage of the successfully simulated configurations that were initially not in collision with the hook, because they cannot be counted as success. Taking all these into account, the last column shows the total percentage of the scenes in the evaluation dataset for which a valid, i.e. collision free, and stable solution was found.

Note that due to the way it was generated, the test and training data distribution has a significantly different ratio of success/failure examples than when sampling the model until it predicts success. Therefore, when using the same feasibility threshold κ_2 as for evaluating the test data, the model can be overly optimistic. Hence, we investigate different feasibility thresholds $\kappa_1 = 0.15$, $\kappa_2 = 1.5$, $\kappa_3 = 0.015$, i.e. if $H_{\text{hang}}(\phi^1, \phi^2) < \kappa_i$, then the model predicts success.

		solution	simulation	collision	**total scenes**
		found	success	free	**solved**
signed-distance ϕ	**opt. + sampling**	98.7%	88.5%	100.0%	**87.3%**
	opt. only	51.3%	93.5%	98.6%	47.3%
	sampling only κ_1	83.8%	82.3%	100.0%	68.9%
	sampling only κ_2	100%	35.6%	100.0%	35.6%
	sampling only κ_3	26.0%	87.5%	100.0%	22.8%
occupancy measure ψ	ψ_s + opt. + sampling	34.0%	51.0%	100.0%	17.3%
	ψ + sampling κ_1	100%	17.3%	100%	17.3%
	ψ + sampling κ_2	100%	22.0%	100%	22.0%
	ψ + sampling κ_3	100%	34.0%	100%	34.0%
point-cloud pointnet	opt. + sampling	100.0%	79.3%	14.3%	11.3%
	opt. + sampling + SDF collision	99.3%	67.1%	100.0%	66.7%
	sampling κ_1, no collision	100.0%	48.0%	23.6%	11.3%
	sampling κ_2, no collision	100.0%	36.0%	53.7%	19.3%
	sampling κ_3, no collision	100.0%	78.0%	32.5%	25.3%
	sampling κ_1 + SDF collision	99.3%	35.6%	100.0%	35.3%
	sampling κ_2 + SDF collision	100.0%	21.3%	100.0%	21.3%
	sampling κ_3 + SDF collision	73.3%	70.9%	100.0%	52.0%
point-cloud pointnet++	opt. + sampling	0%	–	–	0%
	opt. + sampling + SDF collision	0%	–	–	0%
	sampling κ_1, no collision	100.0%	82.0%	63.4%	52.0%
	sampling κ_2, no collision	100.0%	36.0%	53.7%	19.3%
	sampling κ_3, no collision	0%	–	–	0%
	sampling κ_1 + SDF collision	0%	–	–	0%
	sampling κ_2 + SDF collision	100.0%	24.0%	100.0%	24.0%
	sampling κ_3 + SDF collision	0%	–	–	0%

Table 3: Comparison of success rates for mug hanging experiment between an object representation based on signed-distance functions ϕ (proposed approach), occupancy measures ψ (baseline) and point-clouds (baseline). Total scenes solved (last column) means the percentage of scenes in the evaluation dataset for which a solution was found where the learned model predicts success (first column) that is actually stable when dropped in the simulator from the optimized solution configuration (second column) and is not in collision before dropping (third column).

For the SDF model and sampling only, using the threshold κ_2, in 100% of the cases a solution is found, but only 35.6% of those are actually successful in simulation. For a 10-times smaller threshold κ_1, the success rate is high again (92.3%), but only in 83.8% of the scenes a feasible solution is sampled within the computational budget, leading to a total of 68.9% solved scenes with sampling compared to 87.3% with optimization and sampling, which shows the advantages of our models based on SDFs being differentiable with informative gradients.

As one can see in Tab. 3, the performance using the occupancy measures ψ or ψ_s is significantly worse than with ϕ. The best result with the occupancy measure was 34.0%, compared to 87.3% with our proposed approach. Especially with sampling, the model based on ψ very often mistakenly predicts that a sampled configuration will lead to success. With optimization (ψ_s + opt. + sampling), the success rate for a found solution is higher (51.0%). However, in only 34.0% of the scenes, the solver converges to a configuration within 20 restarts where the learned model with ψ_s predicts success. Therefore, the total percentage of successfully solved scenes in this case is only 17.3% with ψ_s compared to 87.3% with ϕ. One reason for the optimizer finding a solution in only 34.0% of the scenes with the model based on ψ_s is the fact that, although ψ_s is differentiable, its gradients are mostly zero. For ψ_s, we chose $b = 1000$.

For the point-cloud representation, there is no easy way to define the collision constraint H_{coll}, in contrast to both SDF and occupancy measure representations, where this can be realized by (7). In [40], where they also consider hanging objects with a point-cloud based object representation, they have to learn a separate collision predictor. We therefore additionally evaluate the point-cloud representation with the collision constraint (7) defined based on the same SDF of the object as for the evaluation of the SDF approach.

The best achieved result with pointnet is 66.7% of total solved scenes using optimization and the (differentiable) SDF collision constraint. However, without this additional knowledge of the SDF representation for differentiable collision checking when optimizing with the learned pointnet model, the best achieved result is 25.3%. For pointnet++, the best achieved percentage of totally solved scenes is 52.0% with sampling.

The results for the occupancy measure have been obtained by keeping everything the same as in sec. 7.1, i.e. same network architecture, same training, test and evaluation datasets, same thresholds, sampling strategies (with same random seeds) etc., except for ϕ being replaced by ψ and ψ_s when learning and evaluating H_{hang}. Furthermore, the collision constraint $\mathcal{H}_{\text{coll}}$ is also evaluated with ψ or ψ_s, respectively. For the point-cloud representation, we had to tune the hyperparameters of the pointnet++ architecture much more than we did tune the hyperparameters of our proposed approach (with the default parameters of pointnet++, we achieved a success rate of 0%). The gradients that a model learned with pointnet++ provides were not suitable for optimization at all (found solutions 0%). The training, test and evaluation datasets are also the same for the point-cloud architectures with the point-clouds obtained from the SDFs as described in sec. A.2. The thresholds and sampling strategies are also the same (with same random seeds).

A.6 Comparison Results for Pushing Objects on a Table

A.6.1 Forward Prediction Error in Observation Space

Tab. 4 shows the mean RMSE of the one-step predictions on the evaluation dataset (different random seed as for the training and test dataset) for both the proposed approach where the forward model is learned as a functional of SDFs ϕ and occupancy measures ψ as a baseline. More precisely, the error is calculated very similar to the dynamics model functional (5) as follows

$$\frac{1}{n}\sum_{i=1}^{n}\sqrt{\frac{1}{|\mathcal{X}_h|}\sum_{x\in\mathcal{X}_h}^{|\mathcal{X}_h|}\Big(\phi^1_{1,i}(x) - F\big[\phi^1_{0,i}(\mathcal{X}_h), \phi^2_{0:1,i}(\mathcal{X}_h)\big](x)\Big)^2} \tag{20}$$

with $\phi^k_{t,i}(x)$ meaning the SDF value of object k at time t for scene i at $x \in \mathcal{X}_h$ with $\mathcal{X}_h$ the discrete grid. The index t is $t = 0$ or $t = 1$ only here, since we are only interested in the one-step predictions, which is the data the dataset contains. We investigate this prediction error for ψ as defined in (10) and its smooth version ψ_s defined in (11) for different values of b. The absolute numbers of these errors in Tab. 4 are not directly comparable between ψ and ϕ, since the predictions/ground-truth values have different magnitudes. However, we show for each case the error if the model simply

		contact phase	no contact phase
signed-distance ϕ	F_{flow}	3.4 ± 1.6	1.4 ± 1.8
	F	5.8 ± 1.7	5.2 ± 1.6
	$\phi_t^1 = \phi_{t-1}^1$	10.8 ± 3.4	0
occupancy measure ψ	F_{flow}, ψ_s with $b = 100$	59.0 ± 16.6	10.0 ± 11.7
	$\psi_{s,t}^1 = \psi_{s,t-1}^1$ with $b = 100$	83.6 ± 25.1	0
	F_{flow}, ψ_s with $b = 1000$	100.2 ± 19.5	13.1 ± 16.7
	$\psi_{s,t}^1 = \psi_{s,t-1}^1$ with $b = 1000$	127.1 ± 27.0	0
	F_{flow}, ψ	109.7 ± 18.3	13.5 ± 19.5
	$\psi_t^1 = \psi_{t-1}^1$	134.9 ± 25.7	0

Table 4: Comparison of mean RMSE [mm] on evaluation dataset for pushing scenario between an object representation based on signed-distance functions ϕ (proposed approach) and occupancy measures ψ (baseline) with different parameters b for ψ_s as defined in (11).

		x [mm]	y [mm]	α [°]
SDF	ϕ	2.1 ± 2.2	2.1 ± 1.3	0.63 ± 0.69
	$\phi(\cdot; I)$ (with image encoder)	2.2 ± 2.3	2.1 ± 2.3	0.63 ± 0.63
occupancy measure	ψ_s with $b = 100$	2.7 ± 2.7	2.7 ± 2.7	0.76 ± 0.82
	ψ_s with $b = 1000$	2.8 ± 2.7	2.7 ± 2.7	0.80 ± 0.87
	ψ	2.8 ± 2.7	2.8 ± 2.7	0.81 ± 0.87
point-cloud	pointnet	3.3 ± 2.9	3.3 ± 3.0	0.88 ± 0.86
	pointnet++	2.7 ± 2.6	2.6 ± 2.5	0.79 ± 0.80

Table 5: Comparison of one-step Δq^1 prediction error between models learned on SDF, occupancy measure and point-cloud object representations.

predicts the last observed SDF/occupancy measure, i.e. either $\phi_t^1 = \phi_{t-1}^1$ or $\psi_t^1 = \psi_{t-1}^1$. Comparing a model based on SDFs vs occupancy measures, one can see that with SDFs, the prediction error relative to predicting the last observation as the future is significantly lower than with the occupancy measure. These results have been obtained by keeping everything the same as in sec. 7.2, except for the field values to train/test and evaluate the model being replaced by $\psi^k(\mathcal{X}_h)$ instead of $\phi^k(\mathcal{X}_h)$.

A.6.2 Forward Prediction Error in Rigid Transformation Space

As explained in sec. A.3, in order to compare models based on SDFs and occupancy measures with a model that has point-clouds as input, we show in Tab. 5 the forward prediction error of the models predicting the rigid transformation Δq^1 of the object that is being pushed.

As one can see, the model learned based on SDFs outperforms the other representations. The pointnet++ architecture achieves a similar error than the occupancy measure, pointnet has the highest error.

Furthermore, one can see in the second row of Tab. 5 that the case $\phi(\cdot, I)$, i.e. where the SDFs are learned functions conditioned on image observations of the objects, does not degrade the performance.

The training, test and evaluation datasets are the same as in sec. A.6.1, but contain the ground-truth rigid transformations additionally for training and error evaluation purposes.

A.6.3 Planning

Although being worse with respect to the one-step prediction error, the occupancy measure based model is qualitatively able to learn the dynamics of the pushing scenario. However, when trying to utilize the occupancy measure based model within the optimization problem, we could not solve a single scene of the evaluation scene dataset. This is caused by the fact that in the majority of the cases the optimization process got stuck in its initial condition directly, since the learned model as well as the other functionals based on the occupancy measure have zero or non-informative gradients, even in the smoothed version ψ_s. Therefore, for using the learned model with a gradient-based planning framework, the information encoded in the SDF is crucial. An SDF contains non-flat information

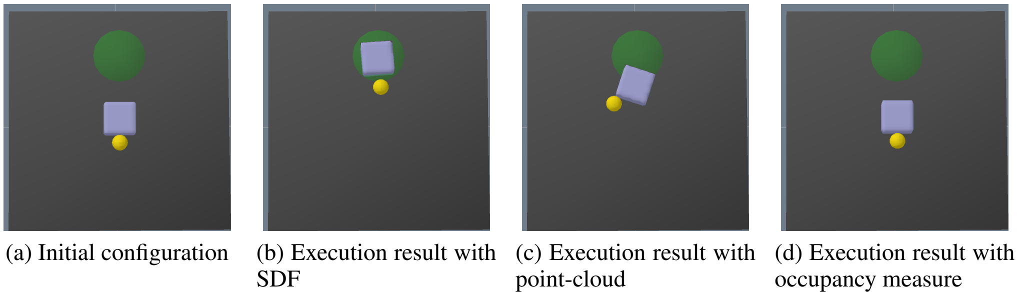
(a) Initial configuration
(b) Execution result with SDF
(c) Execution result with point-cloud
(d) Execution result with occupancy measure

Figure 5: Planning comparison for simple pushing scenario for models learned on top of SDF, occupancy measure and point-cloud object representations.

about the object at distance to it, whereas the occupancy measure is flat at distance to the object. Empirically, this property of the SDFs carries over to the models learned on top of them.

Similarly, planning with models learned on top of point-cloud encodings also could not solve a single scene of the evaluation scene dataset.

To investigate this further, we consider a very simple scenario (see Fig. 5) where the pusher already in the initial configuration has established contact with the object, as shown in Fig. 5a. The object should just be pushed straight ahead to the goal.

Fig. 5b-5d shows the execution result in the simulator of the trajectory the optimizer has converged to for this simple scenario. Here, the only constraint of the optimization problem apart from the goal specification is H_F (the dynamics model) and no other constraints like H_{coll} or H_{PoC}. As one can see, optimizing a trajectory with the occupancy measure based model does not move the pusher at all. The point-cloud based model leads to some movement, but it is clearly unable to solve the task. Using the SDF based model, planning is possible and hence solving this scene is no issue.

A.7 Discussion

For the mug-hanging scenario, the proposed approach of learning models based on SDF representations of the objects significantly outperforms the occupancy measure and point-cloud baselines. The reasons for this are not only a higher prediction accuracy of the learned functional H_{hang} with the SDF representation, but also that the model learned with SDFs provides more informative gradients for optimization. Furthermore, an SDF representation allows to define the collision constraint (in a differentiable way) naturally. For point-clouds, such collision constraints are not well-defined. Generally, the results also highlight the importance of optimization (and sampling for initial guess), compared to sampling alone.

Regarding the pushing scenario, the SDF object representation enables to learn models that outperform all other considered object representations in terms of the forward prediction error. Compared to a point-cloud representation, with SDFs (and occupancy measures) the learned dynamics model can directly predict in SDF (occupancy measure) space, requiring a dataset of such observations only, compared to ground-truth rigid object transformations as required for learning a model with point-clouds. When it comes to planning, the (learned and analytic) functionals defined in terms of SDFs provide informative gradients, which is crucial for planning success.

This shows that model learning and subsequent planning has to be considered together. Just from the fact that a model leads to acceptable prediction performance does not mean that it is useful at all for planning.

Further, we have also shown that one can learn an image conditioned SDF and the dynamics model based on the learned SDFs simultaneously with no noticeable performance degradation.

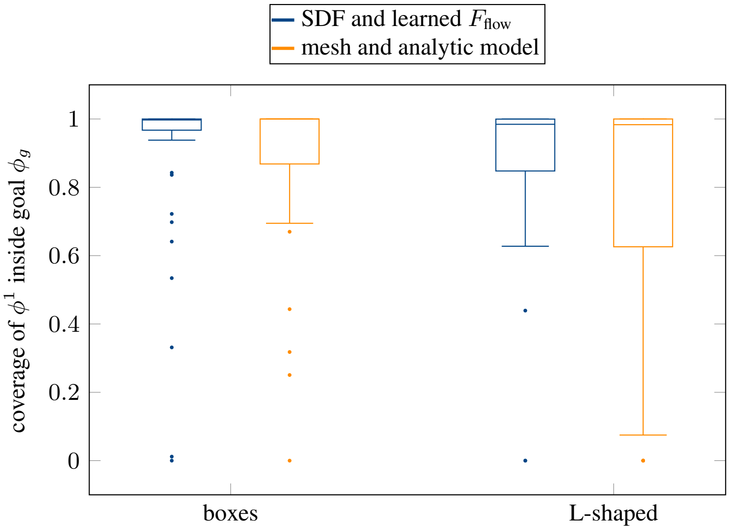

Figure 6: Performance comparison between proposed SDF approach with learned models and analytic mesh-based models for pushing scenario on evaluation scenes. A value of 1 means that the object is fully contained in the goal region.

B Comparison to Analytic Mesh-Based Model for Pushing Scenario

In this experiment, we utilize the analytic model from [13] that is based on mesh object representations to solve the same pushing scenarios as in Sec 7.2.3. The problem formulation remains the same, i.e. we have the same constraints (collision avoidance and contact establishment), same initializations, same discrete decisions etc., with the only difference that the functionals are replaced by mesh-based representations and the learned SDF dynamics model with an analytic one. Since it is unclear how to specify the same goal region constraint from sec. 6.2 with a mesh, we utilize the same goal-region SDF constraint (8) in this experiment, i.e. everything is based on meshes except the goal specification, which utilizes the same SDF as for the other experiments. These choices allow for a fair comparison.

Fig. 6 shows the performance in terms of the amount of object ϕ^1 that is inside the goal region at the end of the execution both for the proposed SDF approach (F_{flow}) and the analytic mesh-based approach. As one can see, especially for the L-shaped objects, our proposed method outperforms the mesh-based analytic variant significantly, but also for boxes the performance with the SDF models is better.

Possible reasons for this are mainly two fold. On the one hand, the mismatch between the analytic model and the simulator leads to the object rotating more in the optimized trajectory than when executed in the simulator. System identification (including potentially a more complex friction model, which makes planning much harder) for each object shape would be required to improve on this. On the other hand, the optimized trajectories obtained with the mesh models very often push on a corner of the objects. Such corner pushes are unstable when executed. We have seen less such behavior of pushing at corners when using SDFs and the learned models on top of them. A possible explanation for this is that the data generation for the learned model is performed via random pushes, which leads to only little data where there are pushes on corners.

As mentioned in sec. 7.2.3 and sec. I.2, directly solving (2) on the pushing scenario often leads the optimizer to converge to an infeasible local minimum if the pusher has to go around the object to achieve the goal. This is not a fact that is caused by using learned models or SDF object representations. The same holds true for the analytic model and mesh-based object representations. Therefore, we also require the initializations described in sec. I.2 for this experiment with the analytic, mesh-based models. Similar to the other experiments, these 4 rough initializations address the issue successfully.

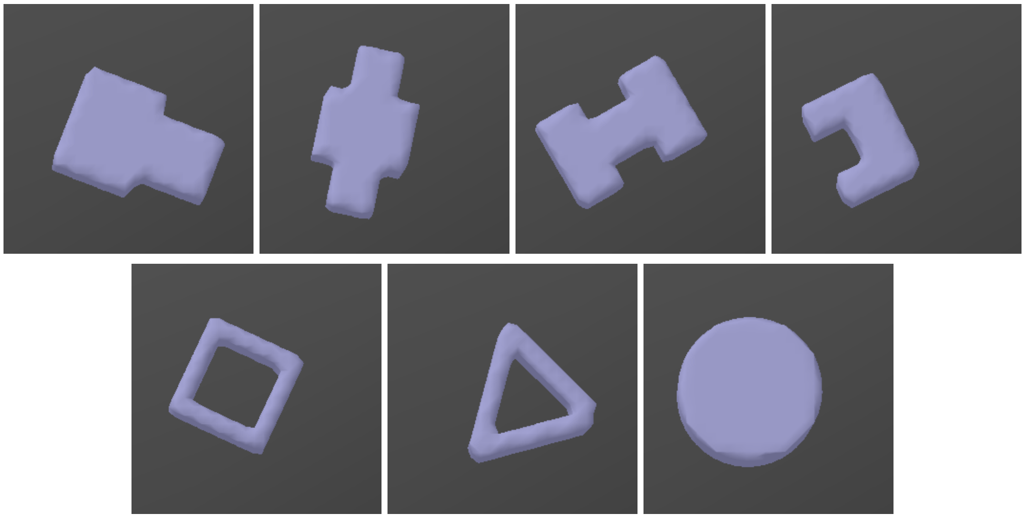

Figure 7: Tested object shapes for out-of-distribution shape generalization experiment for pushing scenario.

		x [mm]	y [mm]	α [°]
SDF	ϕ	2.3 ± 2.2	2.3 ± 2.3	0.81 ± 0.87
	$\phi(\cdot; I)$ (with image encoder)	2.4 ± 2.3	2.4 ± 2.4	0.85 ± 0.92
occupancy measure	ψ_s with $b = 100$	3.0 ± 3.2	3.2 ± 3.3	0.99 ± 1.06
	ψ_s with $b = 1000$	3.2 ± 3.5	3.3 ± 3.5	1.02 ± 1.06
	ψ	3.0 ± 3.3	3.4 ± 3.6	1.06 ± 1.09
point-cloud	pointnet	4.9 ± 4.5	5.1 ± 4.7	0.99 ± 0.97
	pointnet++	4.5 ± 4.9	4.6 ± 4.8	0.77 ± 0.78

Table 6: Shape generalization experiment. Comparison of Δq^1 prediction error between models learned on SDF, occupancy measure and point-cloud object representations.

Note that for the pushing scenario it is possible to write down an analytic physics-based model on the mesh representation that is suitable for planning. However, for the hanging scenario, this is not directly possible. In [45], a mesh-based dynamic model is proposed, but there the focus is on learning a passive simulator and not learning models suitable for planning.

C Generalization Experiments for Pushing Scenario

C.1 Generalization to Out-of-Distribution Shapes

The training data for the pushing scenario consists of boxes and L-shaped objects of different sizes. The test and evaluation scenes use different random seeds, but sample the object shapes from the same distribution. Since the learned model takes the actual geometry of the objects as input, it could, in principle, generalize to arbitrary shapes. This experiment investigates whether this generalization works.

In order to do so, we consider 7 different objects, shown in Fig. 7, which are clearly out-of-distribution compared to the training dataset in terms of shape type and topology.

C.1.1 Forward Prediction Error

Tab. 6 shows the forward prediction error in Δq^1 space for the out-of-distribution shapes. To obtain these results, we apply random pushes on the out-of-distribution shapes with pushers of different sizes in the simulator and record the ground-truth rigid transformations Δq^1 of the pushed object.

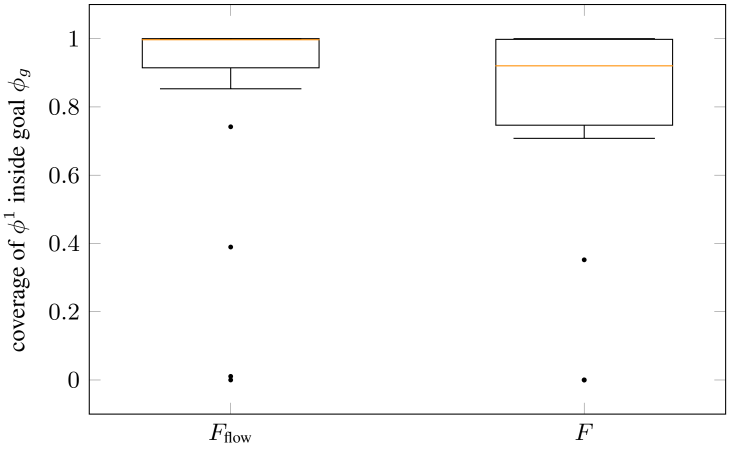

Figure 8: Pushing performance on scenes containing out-of-distribution shapes in terms of the amount of ϕ^1 that is inside of the goal region at the end of the execution. A value of 1 means that the object is fully contained in the goal region.

In terms of absolute numbers, the SDF based model significantly outperforms the other object representations.

Comparing these results with the forward prediction error evaluated on the on-distribution evaluation dataset (see Tab. 5), one can see that the one-step forward prediction error in x, y-transformation increases by only 9.5% for the proposed model learned on top of SDF object representations (9% with the learned image conditioned SDF), compared to 48.5% for a model learned with pointnet (66.7% increase for pointnet++). For the occupancy measure, the increase is about 14% (depending on the exact variant), which is not as much as for the point-cloud based models, but also more than with SDFs.

This shows that models learned in SDF space generalize better (and well in terms of absolute numbers) to out-of-distribution shapes.

C.1.2 Planning and Execution Performance

Fig. 8 shows the performance when executing the planning results open-loop in the simulator for each of the 7 shapes shown in Fig. 7 tested in 3 different environments (i.e. different start configurations and goal region locations), leading to 21 experiments in total. No changes in the methodology were required to achieve these results. The initialization options (cf. sec. I.2) are also the exact same as for the other experiments.

With a median coverage of 99.7% of the amount of ϕ^1 that is inside of the goal region at the end of the execution with the learned F_{flow}, the model generalizes very well to out-of-distribution scenes. Not surprisingly, the median coverage with the direct F of 92.0% is lower, since there we ask F to not only predict the change of the input SDF ϕ^1_{t-1} to ϕ^1_t, but the whole ϕ^1_t directly, which is more challenging especially for out-of-distribution shapes.

This experiment shows that the learned model is capable of generalizing quite out-of-distribution not only with respect to its prediction error, but also when utilized within the planning framework.

C.2 Generalization to three Interacting Objects

The pushing dynamics model we propose and learn is object-centric, i.e. it models the interaction between two objects. Compared to other approaches that model the dynamics on a scene level as in [33], we can train on only two objects and then generalize to situations where not only more objects are in the scene, but also multiple objects have to interact to solve the task, without having to relearn a new model. In Fig. 9 we show such a scenario where three objects have to interact to solve the task.

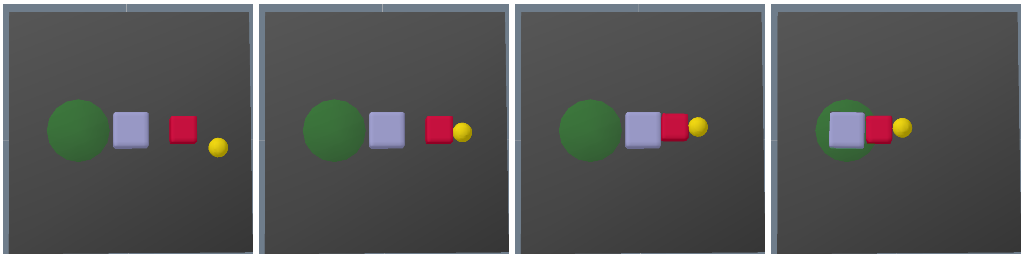

Figure 9: Generalization experiment to three objects that have to interact to solve the task. The yellow pusher is moved such that it pushes the red box such that the red box pushes the blue object to the green goal region. No new model has been learned for this experiment, it can be solved with the model trained on the interaction between two objects.

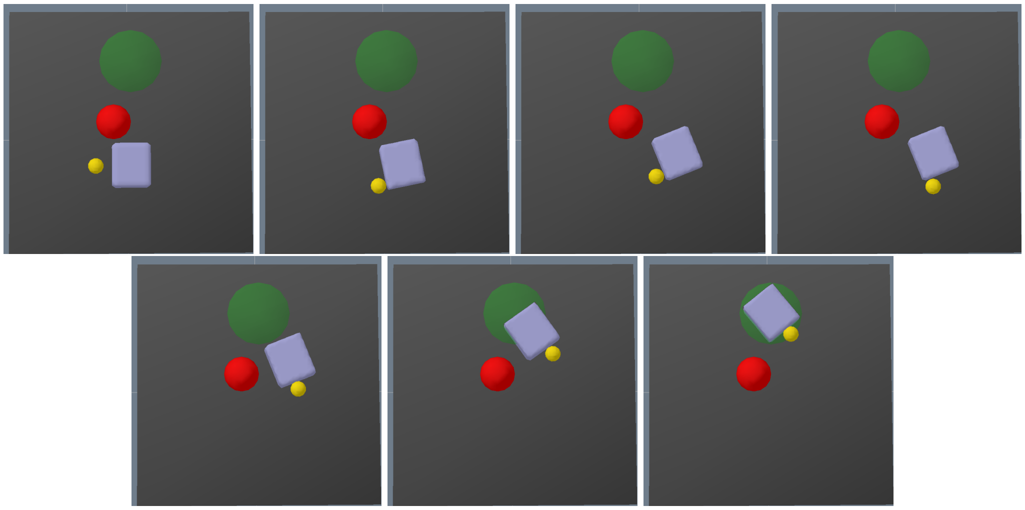

Figure 10: Generalization experiment to a scene that contains an obstacle (red). Three push phases have been determined to solve the task.

The same learned dynamics model constraint H_F is active here twice, once between the yellow sphere and the red box for the whole length of the trajectory. Second between the red box and the blue box for the last phase of the trajectory. Note that we here ask for additional generalization, since in the training data only spheres have been used as the pusher object, but now the model has to predict the dynamics for two boxes that interact.

While the capabilities of the framework are important to be applied to scenes with more objects than being trained on, a case where two objects push one other object simultaneously cannot be realized directly with a model learned on only pair-interactions. However, our general formalism could handle this case as well with a model of the form $\phi_t^1 = F(\phi_{t-1}^1, \phi_{t-1}^2, \phi_t^2, \phi_{t-1}^3, \phi_t^3)$, which would require relearning such a model. Nevertheless, this experiment again shows the advantages of our framework of both being object-centric and embedded into an optimization problem.

C.3 Generalization to Scenes with Obstacles

As mentioned in the last section, due to the (learned) functionals being object-centric, we can apply a model that has been trained on scenes containing only two objects to scenarios with more objects. Fig. 10 shows a scenario where there is an obstacle in the scene between the object that should be pushed and the goal region. The optimization problem then tries to find a pushing trajectory consistent with the learned dynamics model while trying to avoid collision with the obstacle. The found solution for the scene in Fig. 10 consists of three push phases. There are pair-wise collision constraint functionals H_{coll} between the objects in the scene.

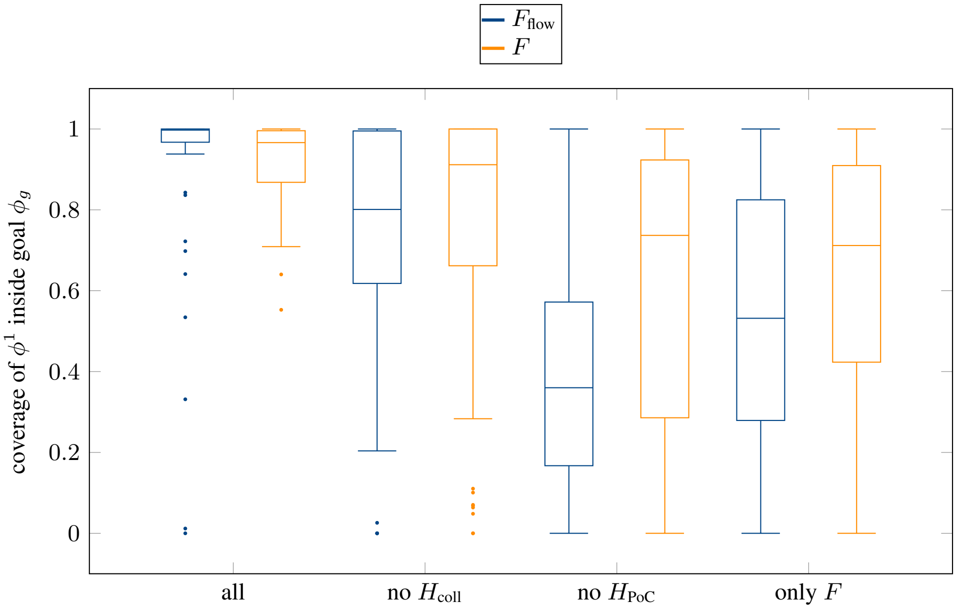

Figure 11: Ablation study for pushing experiment: removal of additional constraints H_{coll} and H_{PoC}.

Scaling this to scenes that contain many obstacles, the non-convexity of collision avoidance becomes an issue. However, this is *not* primarily caused by the fact that we represent objects as SDFs and that we plan with learned models on top of them. The same issue applies for mesh-based representations and analytic dynamic models.

D Ablation Study for Pushing Scenario

In sec. 7.2.3 and sec. I we describe the additional objectives H_{PoC} to encourage the trajectory to establish contact and H_{coll} to avoid collisions (cf. sec. 6 and sec. G, too). Here we investigate the importance of these, also in comparison with using F and H_g as the only constraints. Fig. 11 shows the performance on a dataset of 20 box pushing scenarios, both for F (orange) and F_{flow} (blue), where we remove H_{coll}, H_{PoC} or both (only F). The main reason for the significantly decreased performance when not using H_{PoC} (only F, no H_{PoC}) is that, in some cases, the pusher is not moved at all by the optimization problem. A forward model alone, or more precisely its gradients, simply does not contain enough information for long-horizon tasks to succeed. Therefore, H_{PoC} helps the optimization problem to establish contact, where then the model locally provides sufficient information to solve the task. H_{PoC} only models that there should be contact, and the exact contact locations are then subject to the model. However, there is still a large number of cases where planning/execution works even without H_{PoC}, since the SDF based dynamics model F, compared to point-cloud or occupancy measure based models, still provides somehow useful gradients, even without the help of H_{PoC}. When removing the collision constraint, the performance also drops. This is caused mainly by the fact that the optimized trajectory without the collision constraint sometimes is in collision for a very short moment, which then leads to a failed open-loop execution.

E Pushing Experiment with (Multiple) Robots

A major advantage of our proposed framework compared to other approaches is that the class of models we propose is object-centric, i.e. they model the interaction between the pusher and the object directly, instead of abstract actions. This allows us to directly integrate the model in optimization problems for more complex scenarios, for example where robots should push or otherwise interact with the objects, although the training process did not involve any robots. The learned model F acts as a constraint H_F on the possible object/pusher trajectories during the phase of the motion where the constraint is active.

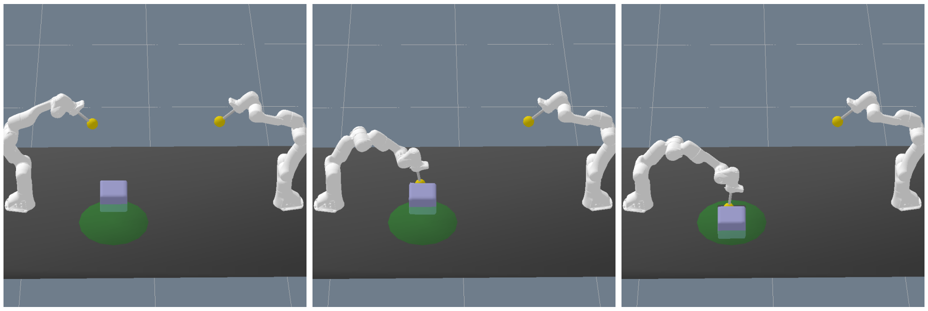

Figure 12: Pushing scenario with robots. Here the goal region (green) is located such that the left robot arm can push the object to the goal region.

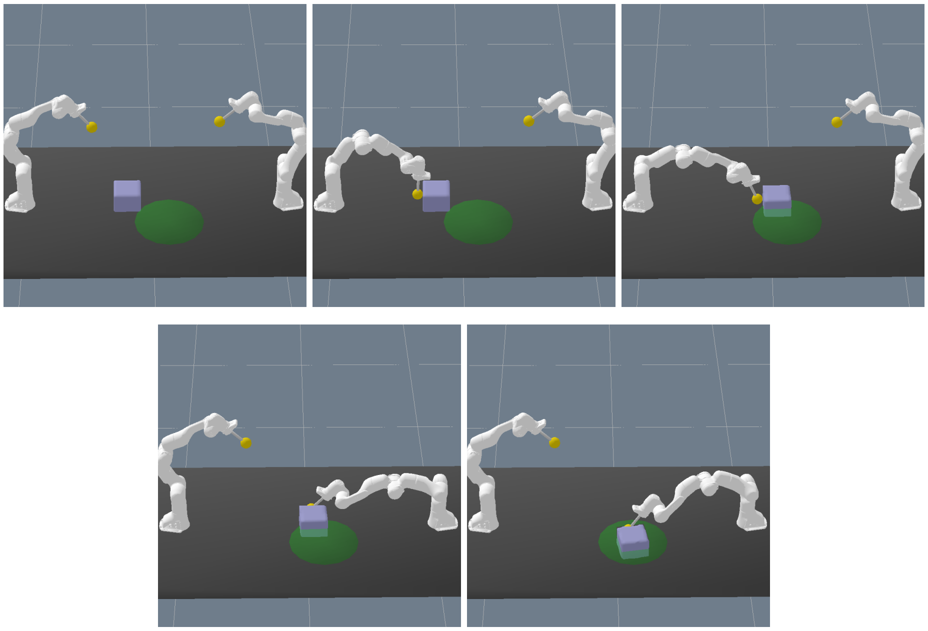

Figure 13: Pushing scenario with robots. Here the goal region (green) is located such that the left robot arm cannot push the object to the goal region. Hence, our framework finds a solution where both robot arms collaborate to push the object into the goal region. The movements of the robot arms are optimized jointly.

In Fig. 12 and Fig. 13, we show two different scenarios where two robots are in the scene. At the end-effectors of each robot a sphere is mounted, which serves as the pusher object. The goal is to push the light blue object to two different goal regions (green). In Fig. 12, the goal region is located such that the left robot arm can push the object directly into the goal region. In contrast, for the scenario shown in Fig. 13, the left robot arm cannot push the object into the goal region, since its kinematic limits do not allow for the necessary movements. Therefore, our framework finds a solution where both robot arms are involved in the pushing maneuver. There is no intermediate goal location specified or similar, the whole motions of the robots are optimized jointly. The discrete decisions of the optimization problem (2) now involve which robot arm to use at which phase of the motion, which implies when and between which objects the dynamic model constraint is active.

Goal regions are not the only way a goal can be specified. In Fig. 14, a scenario is shown where the goal is that the right robot arm should touch the object. Since the object is out of reach for the right robot arm, our framework plans a trajectory where the left robot arm pushes the object towards the right robot arm until it is able to touch the object. Note that in this case there is no notion of a goal region, goal position or similar. The fact that the object is moved in the direction of the other robot to the right is found by the optimizer trying to find a trajectory that is globally and jointly consistent

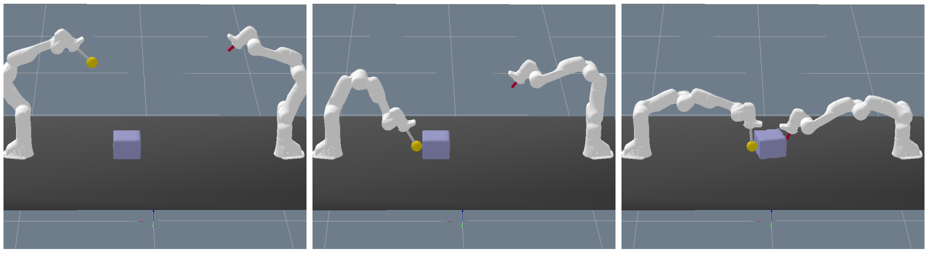

Figure 14: Scenario where the goal is that the right robot arm should touch the object, which is out of reach for the right arm. Therefore, our proposed framework plans a trajectory where the left arm pushes the object towards the right arm until it can reach it. Note how both arms move at the same time, since the optimization problem plans the robot/object motions jointly.

with all constraints. The goal constraint in this case is the contact establishment functional H_{PoC} (see sec. 6.3 and (9)) between the object and the end-effector of the right robot arm (both represented with SDFs).

Note that in none of these cases, we had to relearn a new model or change the methodology. The optimization problem now optimizes over both the joint angles of the robots and the rigid transformations of the SDF representing the object. Since the pusher object is attached to the end-effectors of the robots, the SDF of the pusher object is transformed in space via the movements of the robots. The dynamic model constraint then makes the motions consistent with the learned physical model and the goal.

These experiments show the versatility and advantages of our problem formulation.

F Loss Function for Kinematic Success Model

To account for the fact that we want to use H as a constraint, we use the loss function

$$L(\theta) = \sum_{i=1}^{n} y^i H\Big(\big(\phi^j\big)_{j\in\mathcal{I}_i}; \theta\Big)^2 + (1-y^i)\exp\Big(-H\Big(\big(\phi^j\big)_{j\in\mathcal{I}_i}; \theta\Big)\Big) \tag{21}$$

to train the kinematic success models from sec. 5.2. When $y^i = 1$, this loss function brings the value of H closer to zero, while for $y^i = 0$, the value of H is being pushed up.

G Contact Establishment Functional

The contact establishment functional H_{PoC} defined in (9) in sec. 6.3 has the property that, if the two objects are in contact, the global minimum of its optimization problem (9) is zero. Therefore, to integrate H_{PoC} into (2) without having nested optimizations, we can add $p \in \mathbb{R}^3$ as a decision variable to (2) and replace H_{PoC} with the two constraints $\phi_1(p) = 0$ and $\phi_2(p) = 0$. If these constraints are fulfilled, then a minimizer of (9) is found. This formulation of H_{PoC}, as mentioned, does not only need no nested optimizations, but also provides informative gradients for the optimization problem, especially since the gradients of SDFs point towards their zero level set.

Note that (9) alone does not prevent objects from overlapping. It only models that there exists a point where the distance to both objects is zero at the same time.

H Additional Experimental Details for Mug-Hanging Scenario

Fig. 16 shows the set $\mathcal{X}_h$ (red box) for the functional H_{hang}. The hooks are centered in the x, y-plane of $\mathcal{X}_h$, not in z-direction (cf. also Fig. 15). The bounding box $\mathcal{X}_h \in \mathbb{R}^{40\times40\times40}$ in which the SDFs are queried has dimensions of $40 \times 40 \times 40$ cm, i.e. the resolution is 1 cm. Note that ϕ queried at the bounding box grid points contains the information about the distance to the object surface and

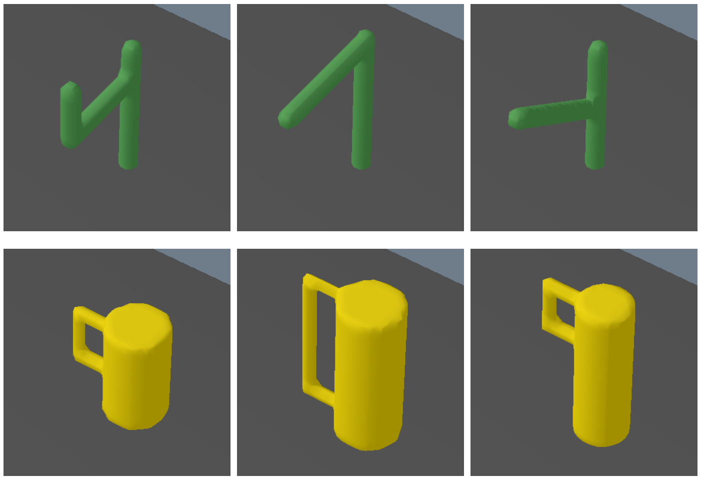

Figure 15: Different mug and hook shapes in evaluation dataset. SDFs meshed with marching cubes.

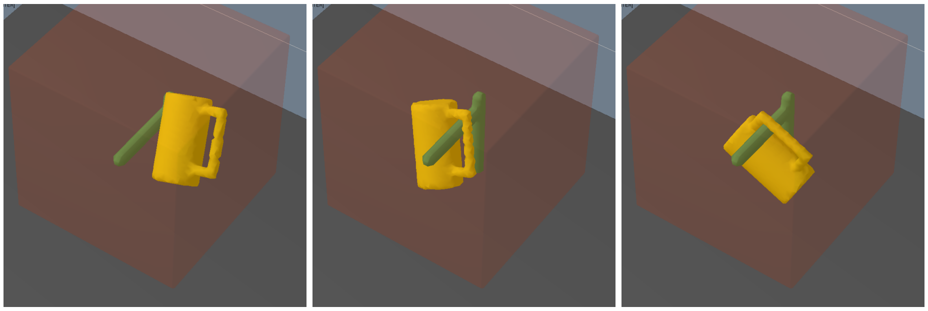

Figure 16: Left: sampled initial configuration from which the optimizer is started. Middle: found solution. Right: configuration after dropping the mug. Transparent red box is $\mathcal{X}_h$.

hence not only whether there is an object or not at the grid point (compare to the occupancy measure representation). Therefore, a 1 cm grid resolution turned out to be sufficient in our experiments.

As mentioned in sec. 7.1, to generate the training data, we uniformly sample the position and orientation of the mugs inside $\mathcal{X}$. We then run a forward simulation to check if the mug is stable when being dropped from the sampled configuration. In case the mug has not fallen onto the ground, we apply an impulse to the mug (see supplementary video) to further check the stability. The dataset $D = \left\{(\phi_i^1(\mathcal{X}_h), \phi_i^2(\mathcal{X}_h), y^i)\right\}_{i=1}^n$ then consists of the evaluated SDFs $\phi^2(\mathcal{X}_h)$ of the hook and $\phi^1(\mathcal{X}_h)$ of the mug in a configuration from which it either leads to a stable configuration ($y^i = 1$) or not ($y^i = 0$) when being dropped. During training, the functional forms of ϕ^j are not needed, only their values on $\mathcal{X}_h$. When using the model within the optimization problem, then the ϕ^j are utilized as functions. The same impulse is also applied when evaluating the performance of the model.

The radius, height and three parameters of the handle (position relative to the height of the mug, extend in two directions) of the mugs are uniformly sampled. The hook either consists of two or three parts, whose lengths and angle are sampled uniformly. See the supplementary video for visualizations of these mugs and hooks in the evaluation dataset. The training, test, and evaluation dataset utilize the same data generation technique, but all with a different random seed.

The parameter a of (7) for the collision constraint H_{coll} is $a = 1000$.

H.1 Network Architecture

The network architecture of H_{hang} consists of three 3D convolutional layers, followed by an MLP of 3 dense layers. The input SDFs (evaluated on $\mathcal{X}_h$) of both the mug and the hook are stacked into a $2 \times 40 \times 40 \times 40$-dimensional input per training sample. The convolutional layers have 3, 5, and 5 output channels with kernel sizes of 3, 5, and 5 and strides of 0, 2, and 2, respectively. The convolutions use ReLU activation functions. After the convolutions, a fully connected linear layer creates a 200 dimensional feature vector. This feature vector is followed by an MLP with three layers with hidden size of 300 each and ReLU activations. The output is 1 dimensional. The batch size is 32 with a learning rate of 0.0001 utilizing the ADAM optimizer.

I Additional Experimental Details for Pushing Scenario

During data generation, random movements of the pusher biased towards the object are applied. Every 20th timestep, the SDFs of the object and the pusher at two consecutive timesteps are extracted at $\mathcal{X}_h$, i.e. the dataset $D = \left\{\left(\phi_t^1(\mathcal{X}_h), \phi_{t-1}^1(\mathcal{X}_h), \phi_t^2(\mathcal{X}_h), \phi_{t-1}^2(\mathcal{X}_h)\right)_i\right\}_{i=1}^n$ consists of these SDF evaluations only. This means that for training, no other information is needed, in particular no actions/velocities/relative transformations etc., just the SDF values queried on the grid points $\mathcal{X}_h$. For planning with the learned model, the SDFs as functions are required, but not for training.

The bounding box is the 2D set $\mathcal{X}_h \in \mathbb{R}^{140\times 140}$ which covers the whole table. The table has dimensions of 1.4 m × 1.4 m, leading to a resolution of 1cm (as with the mug hanging experiment).

The trajectory of rigid transformations in the optimization problem (2) is discretized in time by $T = 10$ steps per phase. If there is a single push phase, then $K = 2$ or $K = 4$ for two push phases.

When the optimized trajectories are executed in the simulator open-loop, we interpolate linearly with 20 steps between two of the optimized consecutive rigid transformations of ϕ^2. This linear interpolation sometimes leads to collisions during the open-loop execution. For example, if two of the optimized pusher configurations are next to a corner of the object, the linear interpolation could collide with the corner, although the optimized configurations are not in collision.

The cost function c penalizes accelerations of the part of q that corresponds to the pusher ϕ^2. There are no such cost terms for the motion of ϕ^1, which is only influenced by the dynamics model functional H_F. In case H_{PoC} is added to the optimization problem, there is also an acceleration regularization for p. Since computing accelerations requires at least 3 consecutive timesteps in discrete time, $l = 2$, although the dynamics model F is only quasi static ($l = 1$). The prefix is $q_{-2} = q_{-1} = q_0$.

The parameter a in (7) for the collision constraint functional H_{coll} is $a = 500$ and $a = 100$ in (8) for the goal region functional.

Note that the training data for the forward model F is generated by a physical simulator. This means a situation where ϕ_t^1 and ϕ_t^2 overlap can never be contained in the data. Hence, there is no reason to believe that the model will predict anything useful in such a region [46]. This is another advantage of adding the collision constraint functional H_{coll} to the optimization problem, since it can prevent the optimization procedure to query the model in such regions where the model has seen no data.

The test and evaluation set for computing the prediction errors in Tab. 2 and 4 consists of 597 scenes. The evaluation scenes for 7.2.3 and 7.2.4 are yet another dataset.

For the evaluation scenes, we assume the object to be roughly (but not perfectly) in the middle of $\mathcal{X}$. This has no particular reason. Since the functionals are translational invariant, one could also move the bounding box such that the object is roughly centered in it. The bounding box, however, is never rotated. Nevertheless, the training data was not biased at all to have objects centered in $\mathcal{X}$, hence also the prediction errors in Tab. 2 are for object positions/orientations in all $\mathcal{X}$. The initial configuration of the pusher is sampled on a circle around the object. The goal regions are sampled randomly of a set of 8 positions round the object with added Gaussian noise to their exact center positions.

I.1 Network Architecture of F and F_{flow}

Each SDF ϕ_{t-1}^1, ϕ_{t-1}^2, ϕ_t^2, after being evaluated on $\mathcal{X}_h \in \mathbb{R}^{140\times 140}$, is encoded into a 200 dimensional feature vector by the same encoder with shared weights. This encoder consists of three 2D convolutional layers and a linear dense layer at the end to produce the 200 dimensional feature vector. The convolutions have 3, 5, and 5 output channels with kernel sizes of 3, 5, and 5 and strides of 0, 2, and 2, respectively. The input query point is encoded with one fully connected layer into a

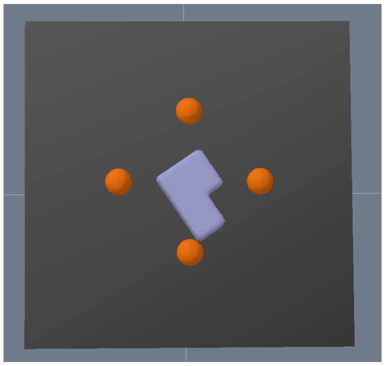

Figure 17: Visualization of the 4 initialization options (orange) for the trajectory optimization problem of the pusher around the object. These initialization options are always the same, independent from the shape/orientation of the object, the goal region position or the initial configuration of the pusher as given by the scene.

100 dimensional feature vector. The feature vectors of the 3 encoded SDFs and of the query point encoder are stacked into one 700-dimensional feature vector, which is then processed by an MLP with 2 hidden layers and 300 hidden units each. The output is one dimensional. All hidden units use ReLU activations. The batch size is 32 with a learning rate of 0.0001 utilizing the ADAM optimizer.

I.2 Initialization of Optimization Problem

As mentioned in sec. 7.2.3, directly solving (2) on the pushing scenario often leads the optimizer to converge to an infeasible local minimum, since the optimization problem is highly non-convex. This especially holds true in scenarios where the pusher has to go around the object to achieve the goal. Such non-convexities of nonlinear trajectory optimization (especially with respect to collision avoidance) are not unique to our approach, but many planning methods that rely on trajectory optimization, even with fully analytical models.

To mitigate this issue, we initialize the pusher trajectory for the optimization problem, i.e. the rigid transformations for ϕ^2, on four different positions around the object that is being pushed. Fig. 17 shows these four initialization options in orange. These initialization options are always the same, no matter of the shape, size, orientation of the object ϕ^1. The task planning part of (2) now not only decides on the number of pushing phases, but also the initialization of the optimization problem (since the initialization comes from a finite set). This leads to 4 optimization problems with one push phase and 16 optimization problems with two push phases. For each scene, we solve all those 20 optimization problems and then choose the one as the solution for the scene where the optimization problem converged best in terms of constraint violations and costs. Note that we believe that these initialization options are a rather weak prior. As can be seen in the supplementary video, even though sometimes the pusher has been initialized differently at two different phases, the optimizer sometimes chooses to let the pusher push the object two times on the same face. Further, these initialization options also do not already establish contact with the object. For the larger objects in the dataset, depending on the orientation of the object, those initialization options can also sometimes be in collision, which is taking care of during optimization. Between the initial given configuration of the pusher in the scene and the chosen initialization points as the initialization of the trajectory optimization problem, the trajectory of rigid transformations of the pusher is interpolated on a circle. Note that the initial configuration of the pusher has to be distinguished by these initialization options. The first is given by the scene that should be solved, while the latter corresponds to the initial guess for the trajectory of the optimization problem.