Reinforcement Learning with Neural Radiance Fields

Danny Driess* TU Berlin
Ingmar Schubert* TU Berlin
Pete Florence Google
Yunzhu Li MIT
Marc Toussaint TU Berlin

Abstract

It is a long-standing problem to find effective representations for training reinforcement learning (RL) agents. This paper demonstrates that learning state representations with supervision from Neural Radiance Fields (NeRFs) can improve the performance of RL compared to other learned representations or even low-dimensional, hand-engineered state information. Specifically, we propose to train an encoder that maps multiple image observations to a latent space describing the objects in the scene. The decoder built from a latent-conditioned NeRF serves as the supervision signal to learn the latent space. An RL algorithm then operates on the learned latent space as its state representation. We call this NeRF-RL. Our experiments indicate that NeRF as supervision leads to a latent space better suited for the downstream RL tasks involving robotic object manipulations like hanging mugs on hooks, pushing objects, or opening doors.
Video: https://dannydriess.github.io/nerf-rl

1 Introduction

The sample efficiency of reinforcement learning (RL) algorithms crucially depends on the representation of the underlying system state they operate on [1, 2, 3, 4, 5, 6, 7]. Sometimes, a low-dimensional (direct) representation of the state, such as the positions of the objects in the environment, is considered to make the resulting RL problem most efficient [2].

However, such low-dimensional, direct state representations can have several disadvantages. On the one hand, a perception module, e.g., pose estimation, is necessary in the real world to obtain the representation from raw observations, which often is difficult to achieve in practice with sufficient robustness. On the other hand, if the goal is to learn policies that generalize over different object shapes [8], using a low-dimensional state representation is often impractical. Such scenarios, while challenging for RL, are common, e.g., in robotic manipulation tasks.

Therefore, there is a large history of approaches that consider RL directly from raw, high-dimensional observations like images (e.g., [9, 10]). Typically, an encoder takes the high-dimensional input and maps it to a low-dimensional latent representation of the state. The RL algorithm (e.g., the Q-function or the policy network) then operates on the latent vector as state input. This way, no separate perception module is necessary, the framework can extract information from the raw observations that are relevant for the task, and the RL agent, in principle, may generalize over challenging environments, in which, e.g., object shapes are varied. While these are advantages in principle, jointly training encoders capable of processing high-dimensional inputs from the RL signal alone is challenging. To address this, one approach is to *pretrain* the encoder on a different task, e.g., image reconstruction [1, 4, 11], multi-view consistency [6], or a time-constrastive task [3]. Alternatively, an auxiliary loss on the latent encoding can be added *during* the RL procedure [5].

In both cases, the choice of the actual (auto-)encoder architecture and associated (auxiliary) loss function has a significant influence on the usefulness of the resulting latent space for the downstream

*equal contribution. Correspondence: danny.driess@gmail.com

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

RL task. Especially for image data, convolutional neural networks (CNNs) are commonly used for the encoder [12]. However, 2D CNNs have a 2D (equivariance) bias, while for many RL tasks, the 3D structure of our world is essential. Architectures like Vision Transformers [13, 14] process images with no such direct 2D bias, but they often require large scale data, which might be challenging in RL applications. Additionally, although multiple uncalibrated 2D image inputs can be used with generic image encoders [15], they do not benefit from 3D inductive biases, which may help for example in resolving ambiguities in 2D images such as occlusions and object permanence.

Recently, Neural Radiance Fields (NeRFs) [16] have shown great success in learning to represent scenes with a neural network that enables to render the scene from novel viewpoints, and have sparked broad interest in computer vision [17]. NeRFs exhibit a strong 3D inductive bias, leading to better scene reconstruction capabilities than methods composed of generic image encoders (e.g., [18]).

In the present work, we investigate whether incorporating these 3D inductive biases of NeRFs into learning a state representation can benefit RL. Specifically, we propose to train an encoder that maps multiple RGB image views of the scene to a latent representation through an auto-encoder structure, where a (compositional) NeRF decoder provides the self-supervision signal using an image reconstruction loss for each view.

In the experiments, we show for multiple environments that supervision from NeRF leads to a latent representation that makes the downstream RL procedure more sample efficient compared to supervision via a 2D CNN decoder, a contrastive loss on the latent space, or even hand-engineered, perfect low-level state information given as keypoints. Commonly, RL is trained on environments where the objects have the same shape. Our environments include hanging mugs on hooks, pushing objects on a table, and a door opening scenario. In all of these, the objects' shapes are not fixed, and we require the agent to generalize over all shapes from a distribution.

To summarize our main contributions: (i) we propose to train state representations for RL with NeRF supervision, and (ii) we empirically demonstrate that an encoder trained with a latent-conditioned NeRF decoder, especially with an object-compositional NeRF decoder, leads to increased RL performance relative to standard 2D CNN auto-encoders, contrastive learning, or expert keypoints.

2 Related Work

Neural Scene/Object Representations in Computer Vision, and Applications. To our knowledge, the present work is the first to explore if neural scene representations like NeRFs can benefit RL. Outside of RL, however, there has been a very active research field in the area of neural scene representations, both in the representations themselves [19, 20, 21, 22] and their applications; see [23, 24, 17] for recent reviews. Within the family of NeRFs and related methods, major thrusts of research have included: improving modeling formulations [25, 26], modeling larger scenes [26, 27], addressing (re-)lighting [28, 29, 30], and an especially active area of research has been in improving speed, both of training and of inference-time rendering [31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]. In our case, we are not constrained by inference-time computation issues, since we do not need to render images, and only have to run our latent-space encoder (with a runtime of approx. 7 ms on an RTX3090). Additionally of particular relevance, various methods have developed latent-conditioned [42, 43, 44] or compositional/object-oriented approaches for NeRFs [45, 46, 47, 48, 49, 50, 51, 52, 53], although they, nor other NeRF-style methods to our knowledge, have been applied to RL. Neural scene representations have found application across many fields (i.e., augmented reality and medical imaging [54]) and both NeRFs [55, 56, 57, 58] and other neural scene approaches [59, 60, 61, 62] have started to be used for various problems in robotics, including pose estimation [55], trajectory planning [56], visual foresight [11, 53], grasping [59, 57], and rearrangement tasks [60, 61, 58].

Learning State Representations for Reinforcement Learning. One of the key enabling factors for the success of deep RL is its ability to find effective representations of the environment from high-dimensional observation data [10, 63]. Extensive research has gone into investigating different ways to learn better state representations using various auxiliary objective functions. Contrastive learning is a common objective and has shown success in unsupervised representation learning in computer vision applications [64, 65]. Researchers built upon this success and have shown such learning objectives can lead to better performance and sample efficiency in deep RL [66, 67], where the contrasting signals could come from time alignment [68, 3], camera viewpoints [69], and different sensory modalities [70], with applications in real-world robotic tasks [6, 71]. Extensive efforts

have investigated the role of representation learning in RL [72], provided a detailed analysis of the importance of different visual representation pretraining methods [73], and shown how we can improve training stability in the face of multiple auxiliary losses [74]. There is also a range of additional explorations on pretraining methods with novel objective functions (e.g., bisimulation metrics [75] and temporal cycle-consistency loss [76]) and less-explored data sources (e.g., in-the-wild images [77] and action-free videos [78]). Please check the survey for more related work in this direction [79]. Our method is different in that we explicitly utilize a decoder that includes strong 3D inductive biases provided by NeRFs, which we empirically show improves RL for tasks that depend on the geometry of the objects.

3 Background

3.1 Reinforcement Learning

This work considers decision problems that can be described as discrete-time Markov Decision Processes (MDPs) $M = \langle \mathcal{S}, \mathcal{A}, T, \gamma, R, P_0 \rangle$. $\mathcal{S}$ and $\mathcal{A}$ are the sets of all states and actions, respectively. The transition probability (density) from s to s' using an action a is $T(s' \mid s, a)$. The agent receives a real-valued reward $R(s, a, s')$ after each step. The discount factor $\gamma \in [0, 1)$ trades off immediate and future rewards. $P_0 : \mathcal{S} \to \mathbb{R}_0^+$ is the distribution of the start state. RL algorithms try to find the optimal policy $\pi^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_0^+$, where $\pi^* = \operatorname{argmax}_\pi \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s_{t+1} \sim T(\cdot|s_t, a_t),\, a_t \sim \pi(\cdot|s_t),\, s_0 \sim P_0} \left[R(s_t, a_t, s_{t+1}) \right]$. Importantly, in this work, we consider RL problems where the state s encodes both the position and the shape of the objects in the scene. We require the RL agent to generalize over all of these shapes at test time. We can therefore think of the state as a tuple $s = (s_p, s_s)$, where s_p encodes positional information, and s_s encodes the shapes involved. We focus the experiments on sparse reward settings, meaning $R(s, a, s') = R_0 > 0$ for $s' \in \mathcal{S}_g$ and $R(s, a, s') = 0$ for $s \in \mathcal{S} \backslash \mathcal{S}_g$, where the volume of $\mathcal{S}_g \subset \mathcal{S}$ is much smaller than the volume of $\mathcal{S}$. The state space $\mathcal{S}$ usually is low-dimensional or a minimal description of the degrees of freedom of the system. In this work, we consider that the RL algorithm has only access to a (high-dimensional) observation $y \in \mathcal{Y}$ of the scene (e.g., RGB images). In particular, this means that the policy has observations as input $a \sim \pi(\cdot \mid y)$. Since we assume that the underlying state $s = (s_p, s_s)$ is fully observable from y, we can treat y like a state for an MDP.

Reinforcement Learning with Learned Latent Scene Representations. The general idea of RL with learned latent scene representations is to learn an *encoder* Ω that maps an observation $y \in \mathcal{Y}$ to a k-dimensional *latent vector* $z = \Omega(y) \in \mathcal{Z} \subset \mathbb{R}^k$ of the scene. The actual RL components, e.g., the Q-function or policy, then operate on z as its state description. For a policy π, this means that the action $a \sim \pi(\cdot \mid z) = \pi(\cdot \mid \Omega(y))$ is conditional on the latent vector z instead of the observation y directly. The dimension k of the latent vector is typically (much) smaller than that of the observation space $\mathcal{Y}$, but larger than that of the state space $\mathcal{S}$.

3.2 Neural Radiance Fields (NeRFs)

The general idea of NeRF, originally proposed by [16], is to learn a function $f = (\sigma, c)$ that predicts the emitted RGB color value $c(x) \in \mathbb{R}^3$ and volume density $\sigma(x) \in \mathbb{R}_{\geq 0}$ at any 3D world coordinate $x \in \mathbb{R}^3$. Based on f, an image from an arbitrary view and camera parameters can be rendered by computing the color $C(r) \in \mathbb{R}^3$ of each pixel along its corresponding camera ray $r(\alpha) = r(0) + \alpha d$ through the volumetric rendering relation

$$C(r) = \int_{\alpha_n}^{\alpha_f} T_f(r, \alpha) \sigma(r(\alpha)) c(r(\alpha)) \, \mathrm{d}\alpha \qquad \text{with} \qquad T_f(r, \alpha) = \exp\left(- \int_{\alpha_n}^{\alpha} \sigma(r(u)) \, \mathrm{d}u \right). \quad (1)$$

Here, $r(0) \in \mathbb{R}^3$ is the camera origin, $d \in \mathbb{R}^3$ the pixel dependent direction of the ray and $\alpha_n, \alpha_f \in \mathbb{R}$ the near and far bounds within which objects are expected, respectively. The camera rays are determined from the camera matrix K (intrinsics and extrinsics) describing the desired view.

4 Learning State Representations for RL with NeRF Supervision

This section describes our proposed framework, in which we use a latent state space for RL that is learned from NeRF supervision. For learning the latent space, we use an encoder-decoder where the

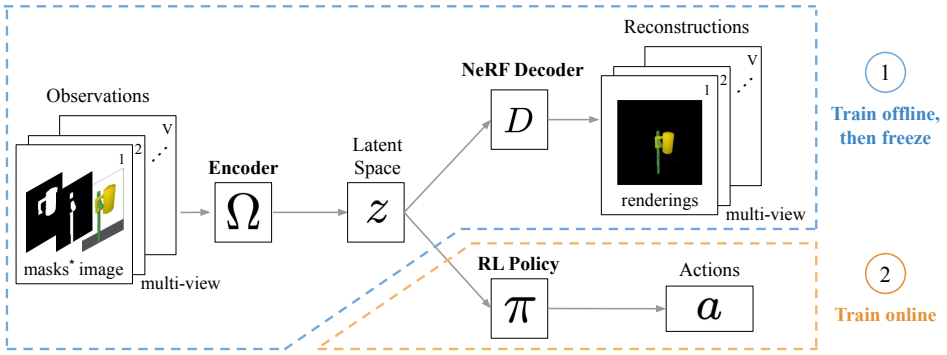


Figure 1: State representation learning for RL with NeRFs. First, the encoder and NeRF decoder are trained with supervision from a multi-view reconstruction loss on an offline dataset. Then, the encoder's weights are frozen, and the latent space is used as state input to train a policy with RL. *Masks of individual objects are only required for the compositional variant of our encoder.

decoder is a latent-conditioned NeRF, which may either be a global [42, 43, 44] or a compositional NeRF decoder [53]. To our knowledge, no prior work has used such NeRF-derived supervision for RL. In Sec. 4.1 we describe this proposition, Sec. 4.2 provides an overview of the encoder-decoder training, Sec. 4.3 and Sec. 4.4 introduce options for the NeRF decoder and encoder, respectively.

4.1 Using Latent-Conditioned NeRF for RL

We propose the state representation z on which an RL algorithm operates to be a latent vector produced by an encoder that maps images from multiple views to a latent z, which is trained with a (compositional) latent-conditioned NeRF decoder. As will be verified in experiments, we hypothesize that this framework is beneficial for the downstream RL task, as it produces latent vectors that represent the actual 3D geometry of the objects in the scene, can handle multiple objects well, as well as fuse multiple views in a consistent way to deal with occlusions by providing shape completion, all of which is relevant to solve tasks where the geometry is important. There are two steps to our framework, as shown in Fig. 1. First, we train the encoder + decoder from a dataset collected by random interactions with the environment, i.e., we do not yet need a trained policy. Second, we take the encoder trained in the first step, which we leave frozen, and use the latent space to train an RL policy. Note that we investigate two variants of the auto-encoder framework, a global one, where the whole scene is represented by one single latent vector, and a compositional one, where objects are represented by their own latent vector. For the latter, objects are identified by masks in the views.

4.2 Overview: Auto-Encoder with Latent-Conditioned NeRF Decoder

Assume that an observation $y = \left(I^{1:V}, K^{1:V}, M^{1:V}\right)$ of the scene consists of RGB images $I^i \in \mathbb{R}^{3\times h\times w}$, $i = 1, \dots, V$ taken from V many camera views, their respective camera projection matrices $K^i \in \mathbb{R}^{3\times 4}$ (including both intrinsics and extrinsics), and per-view image masks $M^{1:V}$. For a *global* NeRF decoder, these are global non-background masks $M^i_{\text{tot}} \in \{0,1\}^{h\times w}$, and for a *compositional* NeRF decoder as in [53], these are sets of binary masks $M^i_j \in \{0,1\}^{h\times w}$ that identify the objects $j = 1, \dots, m$ in the scene in view i. The global case is equivalent to $m = 1$, $M^i_{j=1} = M^i_{\text{tot}}$. The encoder Ω maps these posed image observations from the multiple views into a set of latent vectors $z_{1:m}$, where each z_j represents each object in the scene separately in the compositional case, or the single z_1 all objects in the scene. This is achieved by querying Ω on the masks $M^{1:V}_j$, i.e.,

$$z_j = \Omega\left(I^{1:V}, K^{1:V}, M^{1:V}_j\right) \in \mathbb{R}^k \tag{2}$$

for object j. The supervision signal to train the encoder is the image reconstruction loss

$$\mathcal{L}^i = \left\| I^i \circ M^i_{\text{tot}} - D\left(\Omega\left(I^{1:V}, K^{1:V}, M^{1:V}_{1:m}\right), K^i\right)\right\|^2_2 \tag{3}$$

on the input view i where the decoder D renders an image $I = D(z_{1:m}, K)$ for arbitrary views specified by the camera matrix K from the set of latent vectors $z_{1:m}$. Both the encoder and decoder

are trained end-to-end at the same time. The target images for the decoder are the same in both the global and compositional case: the global-masked image $I^i \circ M^i_{\text{tot}}$ ($\circ$ is the element-wise product). In the compositional case this can be computed with $M^i_{\text{tot}} = \bigvee_{j=1}^{m} M^i_j$. By fusing the information from multiple views of the objects into the latent vector from which the decoder has to be able to render the scene from multiple views, this auto-encoder framework can learn latent vectors that represent the 3D configurations (shape and pose) of the objects in the scene.

4.3 Latent-Conditioned NeRF Decoder Details

Global. The original NeRF formulation [16] learns a fully connected network f that represents one single scene (Sec. 3.2). In order to create a decoder from NeRFs within an auto-encoder to learn a latent space, we condition the NeRF $f(\cdot, z)$ on the latent vector $z \in \mathbb{R}^k$ [42, 43, 44]. While approaches such as [42, 43, 44] use the latent code to represent factors such as lighting or category-level generalization, in our case the latent code is intended to represent the scene variation, i.e., shape *and* configuration of objects, such that a downstream RL agent may use this as a state representation.

Compositional. In the compositional case, the encoder produces a set of latent vectors $z_{1:m}$ describing each object $j = 1, \dots, m$ individually, this leads to m many NeRFs $(\sigma_j(x), c_j(x)) = f_j(x) = f(x, z_j)$, $j = 1, \dots, m$ with their associated volume density σ_j and color value c_j. Note that while one could use different networks f_j with their own network weights for each object, we have a single network f for all objects. This means that both the object's pose as well as its shape and type are represented through the latent code z_j. In order to force those conditioned NeRFs to learn the 3D configuration of each object separately, we compose them into a global NeRF model with the composition formulas (proposed e.g., by [80, 81]): $\sigma(x) = \sum_{j=1}^{m} \sigma_j(x)$, $c(x) = \frac{1}{\sigma(x)} \sum_{j=1}^{m} \sigma_j(x) c_j(x)$. As this composition happens in 3D space, the latent vectors will be learned such that they correctly represent the actual shape and pose of the objects in the scene with respect to the other objects, which we hypothesize may be useful for the downstream RL agent.

4.4 Encoder Details

The encoder Ω operates by fusing multiple views together to estimate the latent vector for the RL task. Since the scientific question of this work is to investigate whether a decoder built from NeRFs to train the encoder end-to-end is beneficial for RL, we consider two different encoder architectures. The first one is a 2D CNN that averages feature encodings from the different views, where each encoding is additionally conditioned on the camera matrix of that view. The second one is based on a learned 3D neural vector field that incorporates 3D biases by fusing the different camera views in 3D space through 3D convolutions and camera projection. This way, we are able to distinguish between the importance of 3D priors incorporated into the encoder versus the decoder.

Per-image CNN Encoder ("Image encoder"). For the global version, we utilize the network architecture from [11] as an encoder choice. In order to work with multiple objects in the compositional case, we modify the architecture from [11] by taking the object masks into account as follows. For each object j, the 2D CNN encoder computes

$$z_j = \Omega_{\text{CNN}}\left(I^{1:V}, K^{1:V}, M_j^{1:V}\right) = h_{\text{MLP}}\left(\frac{1}{V}\sum_{i=1}^{V} g_{\text{MLP}}\left(E_{\text{CNN}}\left(I^i \circ M_j^i\right), K^i\right)\right). \tag{4}$$

E_{CNN} is a ResNet-18 [82] CNN feature extractor that determines a feature from the masked input image $I^i \circ M^i_j$ of object j for each view i, which is then concatenated with the (flattened) camera matrix. The output of the network g_{MLP} is hence the encoding of each view, including the camera information, which is averaged and then processed with h_{MLP}, to produce the final latent vector. Note that in the global case, we set $m = 1$, $M^i_{j=1} = M^i_{\text{tot}}$ such that Ω_{CNN} produces a single latent vector.

Neural Field 3D CNN Encoder ("Field encoder"). Several authors [43] have considered to incorporate 3D biases into learning an encoder by computing pixel-aligned features from queried 3D locations of the scene to fuse the information from the different camera views directly in 3D space. We utilize the encoder architecture from [53], where the idea is to learn a neural vector field $\phi\left[I^{1:V}, M_j^{1:V}\right] : \mathbb{R}^3 \to \mathbb{R}^E$ over 3D space, conditioned on the input views and masks. The features of ϕ are computed from projecting the query point into the camera coordinate system from the respective view. To turn ϕ into a latent vector, it is queried on a workspace set $\mathcal{X}_h \in \mathbb{R}^{d_\mathcal{X} \times h_\mathcal{X} \times w_\mathcal{X}}$ (a 3D

grid) and then processed by a 3D convolutional network, i.e., $z_j = E_{\text{3D CNN}}\left(\phi\left[I^{1:V}, M_j^{1:V}\right](\mathcal{X}_h)\right)$. This method differs from [43, 83, 60] by computing a latent vector from the pixel-aligned features.

5 Baselines / Alternative State Representations

In this section, we briefly describe alternative ways of training an encoder for RL, which we will investigate in the experiments as baselines and ablations. For details, refer to the appendix.

Conv. Autoencoder. This baseline uses a standard CNN decoder based on deconvolutions instead of NeRF to reconstruct the image from the latent representation, similar to [1]. Therefore, with this baseline we investigate the influence of the NeRF decoder relative to CNN decoders. We follow the architecture of [11] for the deconvolution part for the global case. In the compositional case, we modify the architecture to be able to deal with a set of individual latent vectors instead of a single, global one. The image $I = D_{\text{deconv}}(g_{\text{MLP}}(\frac{1}{m}\sum_{j=1}^{m} z_j), K)$ is rendered from $z_{1:m}$ by first averaging the latent vectors and then processing the averaged vector with a fully connected network g_{MLP}, leading to an aggregated feature. This aggregated feature is concatenated with the (flattened) camera matrix K describing the desired view and then rendered into the image with D_{deconv}. In the experiments, we utilize this decoder as the supervision signal to train the latent space produced by the 2D CNN encoder from Sec. 4.4. In the compositional version, the 2D CNN encoder (4) use the same object masks as the compositional NeRF-RL variant.

Contrastive Learning. As an alternative to learning an encoder via a reconstruction loss, the idea of contrastive learning [84] is to define a loss function directly on the latent space that tries to pull latent vectors describing the same configurations together (called positive samples) while ones representing different system states apart (called negative samples). A popular approach to achieve this is with the InfoNCE loss [85, 64]. Let y_i and $\tilde{y}_i$ be two *different* observations of the *same* state. Here, $\tilde{\cdot}$ denotes a perturbed/augmented version of the observation. For a mini-batch of observations $\{(y_i, \tilde{y}_i)\}_{i=1}^{n}$, after encoding those into their respective latent vectors $z_i = \Omega(y_i)$, $\tilde{z}_i = \Omega(\tilde{y}_i)$ with the encoder Ω, the loss for that batch would use $(z_i, \tilde{z}_i)$ as a positive pair, and $(z_i, \tilde{z}_{\neq i})$ as a negative pair, or some similar variation. A crucial question in contrastive learning is how the observation y is perturbed/augmented into $\tilde{y}$ to generate positive and negative training pairs, described in the following.

CURL. In CURL [5], the input image is randomly cropped to generate y and $\tilde{y}$. We closely follow the hyperparameters and design of [5]. CURL operates on a single input view and we choose a view for this baseline from which the state of the environment can be inferred as best as possible (Fig. 17).

Multi-View CURL. This baseline investigates if the neural field 3D encoder (Sec. 4.4) can be trained with a contrastive loss. As this encoder operates on multiple input views we *double* the number of available camera views. Half of the views are the same as in the other experiments, the other half are captured from sightly perturbed camera angles. We use the same loss as CURL, but with different contrastive pairs – rather than from augmentation, the contrastive style is taken from TCN [68]: the positive pairs come from different views but at the same moment in time, while negative pairs come from different times. Therefore, this baseline can be seen as a multi-view adaptation of CURL [5].

Direct State / Keypoint Representations. Finally, we also consider a direct, low-dimensional representation of the state. Since we are interested in generalizing over different object shapes, we consider multiple 3D keypoints that are attached at relevant locations of the objects by expert knowledge and observed with a perfect keypoint detector [8]. See Fig. 2b for a visualization of those keypoints. The keypoints both provide information about object shape and its pose. Furthermore, as seen in Fig. 2b, they have been chosen to reflect those locations in the environment relevant to solve the task. Additionally, we report results where the state is represented by the poses of the objects – as this cannot represent object shape, in this case we use a constant object shape for training and test.

6 Experiments

We evaluate our proposed method on different environments where the geometry of the objects in the scene is important to solve the task successfully. Please also refer to the video `https://dannydriess.github.io/nerf-rl`. Commonly, RL is trained and evaluated on a single environment, where only the poses are changed, but the involved object shapes are kept constant. Since latent-conditioned NeRFs have been shown to be capable of generalizing over geometry [43],

we consider experiments where we require the RL agent to generalize over object shapes within some distribution. Answering the scientific question of this work requires environments with multi-view observations — and for the compositional versions object masks as well. These are *not provided in standard RL benchmarks*, which is the reason for choosing the environments investigated in this work. We use PPO [86] as the RL algorithm and four camera views in all experiments. Refer to the appendix for more details about our environments, parameter choices, network architectures, and training times.

6.1 Environments

Mug on Hook. In this environment, adopted from [87] and visualized in Fig. 2b, the task is to hang a mug on a hook. Both the mug and the hook shape are randomized. The actions are small 3D translations applied to the mug. This environment is challenging as we require the RL agent to generalize over mug and hook shapes and the tolerance between the handle opening and the hook is relatively small. Further, the agent receives a sparse reward only if the mug has been hung stably. This reward is calculated by virtually simulating a mug drop after each action. If the mug does not fall onto the ground from the current state, a reward of one is assigned, otherwise zero.

Planar Pushing. The task in this environment, shown in Fig. 3b, is to push yellow box-shaped objects into the left region of the table and blue objects into the right region with the red pusher that can move in the plane, i.e., the action is two dimensional. This is the same environment as in [53] with the same four different camera views. Each run contains a single object on the table (plus the pusher). If the box has been pushed inside its respective region, a sparse reward of one is received, otherwise zero. The boxes in the environment have different sizes, two colors and are randomly initialized. In this environment, we cannot use keypoints for the multi-shape setting, as the reward depends on the object color; we evaluate the keypoints baseline only in the single shape case (Appendix).

Door Opening. Fig. 4b shows the door environment, where the task is to open a sliding door with the red end-effector that can be translated in 3 DoFs as the action. To solve this task, the agent has to push on the door handle. As the handle position and size is randomized, the agent has to learn to interact with the handle geometry accordingly. Interestingly, as can be seen in the video in the supplementary material, the agent often chooses to push on the handle only at the beginning, as, afterwards, it is sufficient to push the door itself at its side. The agent receives a sparse reward if the door has been opened sufficiently, otherwise, zero reward is assigned.

6.2 Results

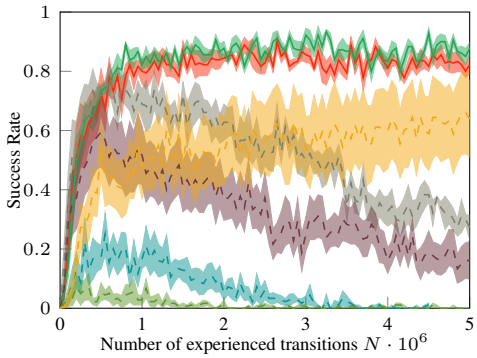
Figs 2a, 3a, 4a show success rates (averaged over 6 independent experiment repetitions and over 30 test rollouts per repetition per timestep) as a function of training steps. Also shown are the 68% confidence intervals. These success rates have been evaluated using randomized object shapes and initial conditions, and therefore reflect the agent's ability to generalize over these.

In all these experiments, a latent space trained with compositional NeRF supervision as the decoder consistently outperformed all other learned representations, both in terms of sample efficiency and asymptotic performance. Furthermore, our proposed framework with compositional NeRF even outperforms the expert keypoint representation. For the door environment, the 3D neural field encoder plus NeRF decoder (NeRF-RL comp. + field) reaches nearly perfect success rates. For the other two environments, the compositional 2D CNN encoder plus NeRF decoder (NeRF-RL comp. + image) was slightly better than with the neural field encoder but not significantly. This shows that the *decoder* built from compositional NeRF is relevant for the performance, not so much the choice of the encoder.

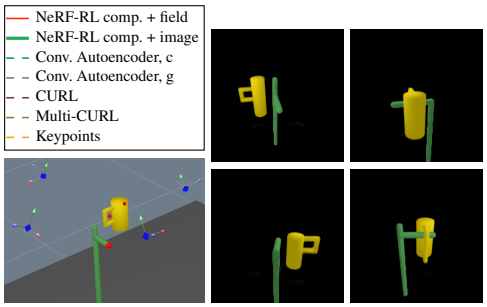
Training the 3D neural field encoder with a contrastive loss as supervision signal for different camera views as positive/negative training pairs is not able to achieve significant learning progress in these scenarios (Multi-CURL). However, the other contrastive baseline, CURL, which has a different encoder and uses image cropping as data augmentation instead of additional camera views, is able to achieve decent performance and sample efficiency on the door environment, but not for the pushing environment. In the mug environment, CURL initially is able to make learning progress comparable to our framework, but never reaches a success rate above 59% and then becomes unstable. Similarly, the global CNN autoencoder baseline shows decent learning progress initially on the mug and pushing scenario (not for the door), but then becomes unstable (mug) or never surpasses 50% success rate

		encoder	decoder	comp.	NeRF	loss
NeRF-RL (ours)	**comp.+field**	3D CNN	comp. 3D NeRF	✓	✓	image reconstr.: L2
	comp.+image	2D CNN	comp. 3D NeRF	✓	✓	image reconstr.: L2
	global+image	2D CNN	global 3D NeRF	✗	✓	image reconstr.: L2
	Conv. Autoencoder, *c*	2D CNN	comp. 2D CNN	✓	✗	image reconstr.: L2
	Conv. Autoencoder, *g*	2D CNN	2D CNN	✗	✗	image reconstr.: L2
	CURL	2D CNN	-	✗	✗	contrast: InfoNCE
	Multi-CURL	3D CNN	-	✓	✗	contrast: InfoNCE
	Keypoints	chosen by expert knowledge and perfect extraction				

Table 1: Overview of the different state representation learning frameworks.

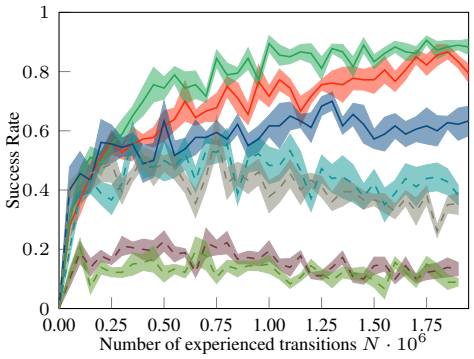


(a) Learning curve.

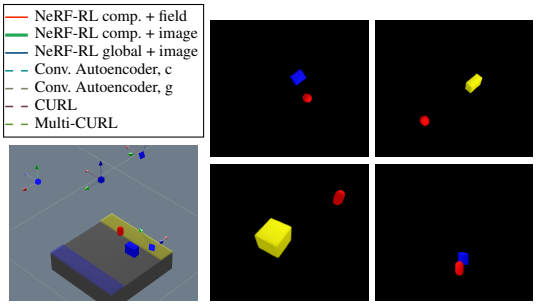


(b) Left: Blue coordinate frames denote the four camera poses. Red points are the expert keypoints. Right: NeRF renderings (different scenes).

Figure 2: Mug on hook environment. (b) shows an example scene and NeRF renderings

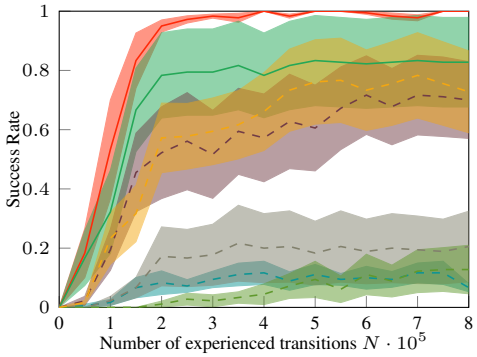


(a) Learning curve.

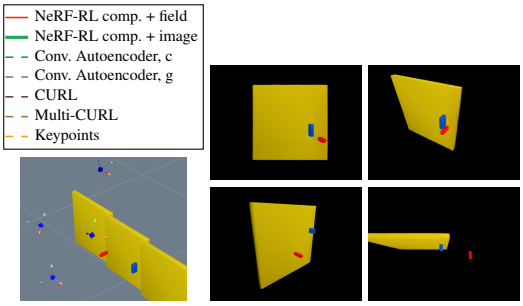


(b) Left: Blue coordinate frames denote the four camera poses. Yellow and blue areas are goal regions where the objects should be pushed to. Right: NeRF renderings.

Figure 3: Pushing environment. (b) shows NeRF renderings for different scenes.



(a) Learning curve.



(b) Left: Blue coordinate frames denote the four camera poses. Right: NeRF renderings.

Figure 4: Door environment. (b) shows NeRF renderings for different scenes.

(pushing). Such variations in performance or instable learning across the different environments have not been observed with our method, which is stable in all cases.

The compositional variant (NeRF-RL comp.) of our framework achieves the highest performance. Since the conv. comp. autoencoder baseline has worse performance than its global variant, compositionality alone is not the sole reason for the better performance of our state representation. Indeed, the global NeRF-RL + image variant in the pushing env. is also better than all other baselines.

In the appendix Sec. A.1, we find a positive correlation between NeRF reconstruction quality and RL performance. Furthermore, it turns out that the performance of our framework is not significantly affected when we pretrain the encoder with less data (Sec. A.2). In Sec. A.3, we investigate the influence of the number of input views on the RL performance. In the pushing scenario, only two or even one input view are sufficient for good performance. However, for tasks that require more 3D understanding such as the mug scenario, we observe a drop in performance when reducing the number of views from 4 to 2.

7 Discussion

Why NeRF provides better supervision. The NeRF training objective (1) strongly forces each $f(\cdot, z_j)$ to represent each object in its actual 3D configuration and relative to other objects in the scene (compositional case), including their shape. This implies that the latent vectors z_j have to contain this information, i.e., they are trained to determine the object type, shape and pose in the scene. In the global case, z_1 has to represent the geometry of the whole secne. As the tasks we consider require policies to take the geometry of the objects into account, we hypothesize that a latent vector that is capable of parameterizing a NeRF to reconstruct the scene in the 3D space has to contain enough of the relevant 3D information of the objects also for the policy to be successful.

Masks. In order for the auto-encoder framework to be compositional, it requires object masks. We believe that instance segmentation has reached a level of maturity [88] that this is a fair assumption to make. As we also utilize the individual masks for the compositional conv. autoencoder and the multi-view CURL baseline, which do not show good performance, it indicates that the masks are not the main reason that our state representation achieves higher performance. This is further supported by the fact that the global NeRF-RL variant which does not rely on individual object masks on the pushing scenario achieved a performance higher than all baselines, i.e., masks will increase the performance of NeRF-RL as they enable the compositional version, but they do not seem essential.

Offline/Online. In this work, we focused on pretraining the latent representation offline from a dataset collected by random actions. During RL, the encoder is fixed and only the policy networks are learned. This has the advantage that the same representation can be used for different RL tasks and the dataset to train the representation not necessarily has to come from the same distribution. However, if a policy is needed to explore reasonable regions of the state space, collecting a dataset offline to learn a latent space that covers the state space sufficiently might be more challenging for an offline approach. This was not an issue for our experiments where data collection with random actions was sufficient. Indeed, we show generalization over different starting states of the same environment and with respect to different shapes (within distribution). Future work could investigate NeRF supervision in an online setup. Note that the reconstruction loss via NeRF is computationally more demanding than via a 2D CNN deconv. decoder or a contrastive term, making NeRF supervision as an auxiliary loss at each RL training step costly. One potential solution for this is to apply the auxiliary loss not at every RL training step, but with a lower frequency. Regarding computational efficiency, this is where contrastive learning has an advantage over our proposed NeRF-based decoder, as the encoding with CURL can be trained within half a day, whereas the NeRF auto-encoder took up to 2 days to train for our environments. However, when using the encoder for RL, there is no difference in inference time.

Multi-View. The auto-encoder framework we propose can fuse the information of multiple camera views into a latent vector describing an object in the scene. This way, occlusions can be addressed and the agent can gain a better 3D understanding of the scene from the different camera angles. Having access to multiple camera views and their camera matrices is an additional assumption we make, although we believe the capability to utilize this information is an advantage of our method.

8 Conclusion

In this work, we have proposed the idea to utilize Neural Radiance Fields (NeRFs) to train latent spaces for RL. Our environments focus on tasks where the geometry of the objects in the scene is relevant for successfully solving the tasks. Training RL agents with the pretrained encoder that maps multiple views of the scene to a latent space consistently outperformed other ways of learning a state representation and even keypoints chosen by expert knowledge. Our results show that the 3D prior present in compositional NeRF as the decoder is more important than priors in the encoder.

Broader Impacts. Our main contribution is a method to learn representations that improve the efficiency of vision-based RL, which could impact automation. As such, our work inherits general ethical risks of AI, like the question of how to address the potential of increased automation in society.

Acknowledgments

The authors thank Russ Tedrake for initial discussions; Jonathan Tompson and Jon Barron for feedback on drafts; Vincent Vanhoucke for encouraging latent NeRFs.

This research has been supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2002/1 "Science of Intelligence" – project number 390523135. Danny Driess thanks the International Max-Planck Research School for Intelligent Systems (IMPRS-IS) for the support. Ingmar Schubert acknowledges support by the German Academic Scholarship Foundation. Yunzhu Li acknowledges support by Amazon.com Services LLC, PO# #2D-06310236 and the Wistron Corporation.

References

[1] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

[2] R. Jonschkowski, R. Hafner, J. Scholz, and M. Riedmiller. Pves: Position-velocity encoders for unsupervised learning of structured state representations. *arXiv preprint arXiv:1705.09805*, 2017.

[3] D. Dwibedi, J. Tompson, C. Lynch, and P. Sermanet. Learning actionable representations from visual observations. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1577–1584. IEEE, 2018.

[4] T. D. Kulkarni, A. Gupta, C. Ionescu, S. Borgeaud, M. Reynolds, A. Zisserman, and V. Mnih. Unsupervised learning of object keypoints for perception and control. *Advances in neural information processing systems*, 32, 2019.

[5] M. Laskin, A. Srinivas, and P. Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pages 5639–5650. PMLR, 2020.

[6] L. Manuelli, Y. Li, P. Florence, and R. Tedrake. Keypoints into the future: Self-supervised correspondence in model-based reinforcement learning. *arXiv preprint arXiv:2009.05085*, 2020.

[7] M. Vecerik, J.-B. Regli, O. Sushkov, D. Barker, R. Pevceviciute, T. Rothörl, C. Schuster, R. Hadsell, L. Agapito, and J. Scholz. S3k: Self-supervised semantic keypoints for robotic manipulation via multi-view consistency. *arXiv preprint arXiv:2009.14711*, 2020.

[8] L. Manuelli, W. Gao, P. Florence, and R. Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. *arXiv preprint arXiv:1903.06684*, 2019.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[11] Y. Li, S. Li, V. Sitzmann, P. Agrawal, and A. Torralba. 3d neural scene representations for visuomotor control. In *Conference on Robot Learning*, pages 112–123. PMLR, 2022.

[12] S. Lange and M. Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 2010.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[14] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[15] I. Akinola, J. Varley, and D. Kalashnikov. Learning precise 3d manipulation from multiple uncalibrated cameras. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4616–4622. IEEE, 2020.

[16] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[17] F. Dellaert and L. Yen-Chen. Neural volume rendering: Nerf and beyond, 2021.

[18] S. A. Eslami, D. Jimenez Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor, et al. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018.

[19] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[20] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019.

[21] 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*, pages 5939–5948, 2019.

[22] V. Sitzmann, M. Zollhöfer, and G. Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *Advances in Neural Information Processing Systems*, 32, 2019.

[23] Y. Xie, T. Takikawa, S. Saito, O. Litany, S. Yan, N. Khan, F. Tombari, J. Tompkin, V. Sitzmann, and S. Sridhar. Neural fields in visual computing and beyond. *arXiv preprint arXiv:2111.11426*, 2021.

[24] A. Tewari, J. Thies, B. Mildenhall, P. Srinivasan, E. Tretschk, Y. Wang, C. Lassner, V. Sitzmann, R. Martin-Brualla, S. Lombardi, et al. Advances in neural rendering. *arXiv preprint arXiv:2111.05849*, 2021.

[25] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021.

[26] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *arXiv preprint arXiv:2111.12077*, 2021.

[27] M. Tancik, V. Casser, X. Yan, S. Pradhan, B. Mildenhall, P. P. Srinivasan, J. T. Barron, and H. Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. *arXiv preprint arXiv:2202.05263*, 2022.

[28] M. Boss, R. Braun, V. Jampani, J. T. Barron, C. Liu, and H. Lensch. NeRD: Neural reflectance decomposition from image collections. *https://arxiv.org/abs/2012.03918*, 2020.

[29] P. Srinivasan, B. Deng, X. Zhang, M. Tancik, B. Mildenhall, and J. T. Barron. NeRV: Neural reflectance and visibility fields for relighting and view synthesis. *https://arxiv.org/abs/2012.03927*, 2020.

[30] X. Zhang, P. P. Srinivasan, B. Deng, P. Debevec, W. T. Freeman, and J. T. Barron. Nerfactor: Neural factorization of shape and reflectance under an unknown illumination. *https://arxiv.org/abs/2106.01970*, 2021.

[31] L. Liu, J. Gu, K. Z. Lin, T.-S. Chua, and C. Theobalt. Neural sparse voxel fields. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

[32] D. Lindell, J. Martel, and G. Wetzstein. AutoInt: Automatic integration for fast neural volume rendering. *https://arxiv.org/abs/2012.01714*, 2020.

[33] D. Rebain, W. Jiang, S. Yazdani, K. Li, K. M. Yi, and A. Tagliasacchi. DeRF: Decomposed radiance fields. *https://arxiv.org/abs/2011.12490*, 2020.

[34] T. Neff, P. Stadlbauer, M. Parger, A. Kurz, J. H. Mueller, C. R. A. Chaitanya, A. S. Kaplanyan, and M. Steinberger. DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks. *Computer Graphics Forum*, 40(4), 2021. ISSN 1467-8659. doi: 10.1111/cgf.14340. URL https://doi.org/10.1111/cgf.14340.

[35] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin. Fastnerf: High-fidelity neural rendering at 200fps. *https://arxiv.org/abs/2103.10380*, 2021.

[36] C. Reiser, S. Peng, Y. Liao, and A. Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps, 2021.

[37] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *arXiv*, 2021.

[38] S. Lombardi, T. Simon, G. Schwartz, M. Zollhoefer, Y. Sheikh, and J. Saragih. Mixture of volumetric primitives for efficient neural rendering, 2021.

[39] A. Yu, S. Fridovich-Keil, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa. Plenoxels: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2021.

[40] V. Sitzmann, S. Rezchikov, W. T. Freeman, J. B. Tenenbaum, and F. Durand. Light field networks: Neural scene representations with single-evaluation rendering. In *arXiv*, 2021.

[41] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. doi: 10.1145/3528223.3530127. URL https://doi.org/10.1145/3528223.3530127.

[42] R. Martin-Brualla, N. Radwan, M. S. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7210–7219, 2021.

[43] A. Yu, V. Ye, M. Tancik, and A. Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4578–4587, 2021.

[44] Q. Wang, Z. Wang, K. Genova, P. P. Srinivasan, H. Zhou, J. T. Barron, R. Martin-Brualla, N. Snavely, and T. Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4690–4699, 2021.

[45] K. Zhang, G. Riegler, N. Snavely, and V. Koltun. NERF++: Analyzing and improving neural radiance fields. *https://arxiv.org/abs/2010.07492*, 2020.

[46] M. Niemeyer and A. Geiger. GIRAFFE: Representing scenes as compositional generative neural feature fields. *https://arxiv.org/abs/2011.12100*, 2020.

[47] M. Guo, A. Fathi, J. Wu, and T. Funkhouser. Object-centric neural scene rendering. *https://arxiv.org/abs/2012.08503*, 2020.

[48] W. Yuan, Z. Lv, T. Schmidt, and S. Lovegrove. Star: Self-supervised tracking and reconstruction of rigid objects in motion with neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13144–13152, 2021.

[49] Z. Wang, T. Bagautdinov, S. Lombardi, T. Simon, J. Saragih, J. Hodgins, and M. Zollhöfer. Learning compositional radiance fields of dynamic human heads. *https://arxiv.org/abs/2012.09955*, 2020.

[50] J. Ost, F. Mannan, N. Thuerey, J. Knodt, and F. Heide. Neural scene graphs for dynamic scenes. *https://arxiv.org/abs/2011.10379*, 2020.

[51] H.-X. Yu, L. J. Guibas, and J. Wu. Unsupervised discovery of object radiance fields. *arXiv preprint arXiv:2107.07905*, 2021.

[52] B. Yang, Y. Zhang, Y. Xu, Y. Li, H. Zhou, H. Bao, G. Zhang, and Z. Cui. Learning object-compositional neural radiance field for editable scene rendering. In *International Conference on Computer Vision (ICCV)*, October 2021.

[53] D. Driess, Z. Huang, Y. Li, R. Tedrake, and M. Toussaint. Learning multi-object dynamics with compositional neural radiance fields. *arXiv preprint arXiv:2202.11855*, 2022.

[54] H. Zhang, R. Wang, J. Zhang, C. Li, G. Yang, P. Spincemaille, T. Nguyen, and Y. Wang. Nerd: Neural representation of distribution for medical image segmentation. *arXiv preprint arXiv:2103.04020*, 2021.

[55] L. Yen-Chen, P. Florence, J. T. Barron, A. Rodriguez, P. Isola, and T.-Y. Lin. iNeRF: Inverting neural radiance fields for pose estimation. *IROS*, 2021.

[56] M. Adamkiewicz, T. Chen, A. Caccavale, R. Gardner, P. Culbertson, J. Bohg, and M. Schwager. Vision-only robot navigation in a neural radiance world. *IEEE Robotics and Automation Letters*, 7(2):4606–4613, 2022.

[57] J. Ichnowski, Y. Avigal, J. Kerr, and K. Goldberg. Dex-nerf: Using a neural radiance field to grasp transparent objects. *arXiv preprint arXiv:2110.14217*, 2021.

[58] L. Yen-Chen, P. Florence, J. T. Barron, T.-Y. Lin, A. Rodriguez, and P. Isola. NeRF-Supervision: Learning dense object descriptors from neural radiance fields. In *IEEE Conference on Robotics and Automation (ICRA)*, 2022.

[59] K. Karunratanakul, J. Yang, Y. Zhang, M. J. Black, K. Muandet, and S. Tang. Grasping field: Learning implicit representations for human grasps. In *2020 International Conference on 3D Vision (3DV)*, pages 333–344. IEEE, 2020.

[60] J.-S. Ha, D. Driess, and M. Toussaint. Learning neural implicit functions as object representations for robotic manipulation. *arXiv preprint arXiv:2112.04812*, 2021.

[61] A. Simeonov, Y. Du, A. Tagliasacchi, J. B. Tenenbaum, A. Rodriguez, P. Agrawal, and V. Sitzmann. Neural descriptor fields: Se (3)-equivariant object representations for manipulation. *arXiv preprint arXiv:2112.05124*, 2021.

[62] Y. Wi, P. Florence, A. Zeng, and N. Fazeli. Virdo: Visio-tactile implicit representations of deformable objects. *arXiv preprint arXiv:2202.00868*, 2022.

[63] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[64] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[65] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[66] A. Srinivas, M. Laskin, and P. Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*, 2020.

[67] B. You, O. Arenz, Y. Chen, and J. Peters. Integrating contrastive learning with dynamic models for reinforcement learning from images. *Neurocomputing*, 2022.

[68] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1134–1141. IEEE, 2018.

[69] A. Kinose, M. Okada, R. Okumura, and T. Taniguchi. Multi-view dreaming: Multi-view world model with contrastive learning. *arXiv preprint arXiv:2203.11024*, 2022.

[70] K. Chen, Y. Lee, and H. Soh. Multi-modal mutual information (mummi) training for robust self-supervised deep reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4274–4280. IEEE, 2021.

[71] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022.

[72] A. Stooke, K. Lee, P. Abbeel, and M. Laskin. Decoupling representation learning from reinforcement learning. In *International Conference on Machine Learning*, pages 9870–9879. PMLR, 2021.

[73] S. Parisi, A. Rajeswaran, S. Purushwalkam, and A. Gupta. The unsurprising effectiveness of pre-trained vision models for control. *arXiv preprint arXiv:2203.03580*, 2022.

[74] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus. Improving sample efficiency in model-free reinforcement learning from images. *arXiv preprint arXiv:1910.01741*, 2019.

[75] A. Zhang, R. McAllister, R. Calandra, Y. Gal, and S. Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

[76] K. Zakka, A. Zeng, P. Florence, J. Tompson, J. Bohg, and D. Dwibedi. Xirl: Cross-embodiment inverse reinforcement learning. In *Conference on Robot Learning*, pages 537–546. PMLR, 2022.

[77] T. Xiao, I. Radosavovic, T. Darrell, and J. Malik. Masked visual pre-training for motor control. *arXiv preprint arXiv:2203.06173*, 2022.

[78] Y. Seo, K. Lee, S. James, and P. Abbeel. Reinforcement learning with action-free pre-training from videos. *arXiv preprint arXiv:2203.13880*, 2022.

[79] T. Lesort, N. Díaz-Rodríguez, J.-F. Goudou, and D. Filliat. State representation learning for control: An overview. *Neural Networks*, 108:379–392, 2018.

[80] M. Niemeyer and A. Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[81] K. Stelzner, K. Kersting, and A. R. Kosiorek. Decomposing 3d scenes into objects via unsupervised volume segmentation. *arXiv preprint arXiv:2104.01148*, 2021.

[82] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[83] S. Saito, Z. Huang, R. Natsume, S. Morishima, A. Kanazawa, and H. Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2304–2314, 2019.

[84] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[85] A. Van den Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv e-prints*, pages arXiv–1807, 2018.

[86] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[87] D. Driess, J.-S. Ha, M. Toussaint, and R. Tedrake. Learning models as functionals of signed-distance fields for manipulation planning. In *Conference on Robot Learning (CoRL)*, 2021.

[88] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[89] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

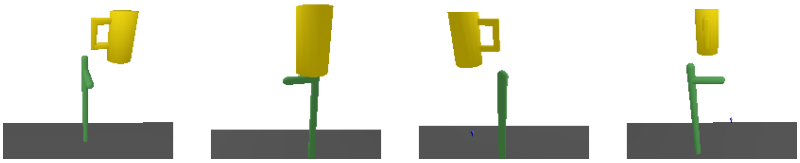
Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See website.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See plots in main paper.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See appendix.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] In the paper and in the code.
   (b) Did you mention the license of the assets? [Yes] In the code.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See website.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] In the code.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

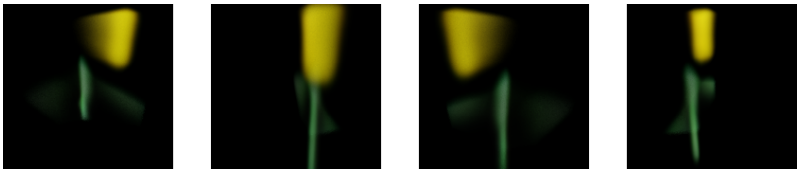
A Additional Experiments

A.1 Influence of Reconstruction Capabilities

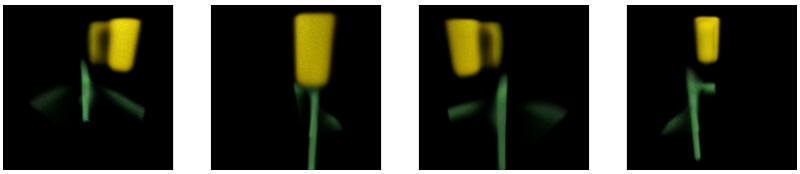
In this section, we investigate for the mug environment how the reconstruction quality of the NeRF-RL auto-encoder influences the downstream RL performance. The state representations of all experiments reported in the paper have been trained for a maximum computational budget of 5 days. The encoder network that has the lowest image reconstruction loss or contrastive loss is then chosen for the RL training procedure. Now, for the NeRF-RL compositional + image variant, we deliberately choose 3 more networks that are trained for fewer epochs and hence have a lower reconstruction quality. Fig. 5 qualitatively shows the reconstruction quality for these networks. As one can see in Fig. 5b, for a network trained on only 10 epochs, the reconstruction is still very rough and the handle is not reconstructed properly yet. After 30 epochs (Fig. 5c), the shape of the mug body and of the hook are reasonably reconstructed, but the handle is still reconstructed as a "mean" handle shape. In Fig. 5d (100 epochs), the handle is reasonably reconstructed, but still a bit imprecise. Finally, Fig. 5e shows the network after 405 epochs, which is also the one used for the main experiment, where the reconstruction, especially with respect to the handle, has become quite precise.



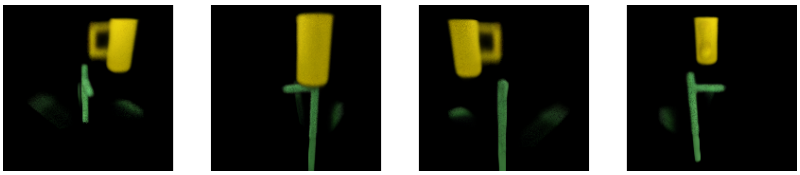
(a) Input views.



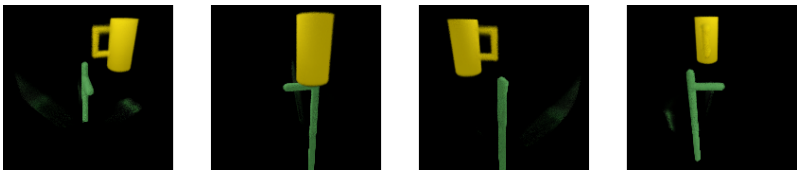
(b) Reconstructions with network trained for 10 epochs.



(c) Reconstructions with network trained for 30 epochs.



(d) Reconstructions with network trained for 100 epochs.



(e) Reconstructions with network trained for 405 epochs (used for main experiments).

Figure 5: Investigation of how the reconstruction quality of the learned state representation with NeRF supervision (NeRF-RL comp. + image) influences the downstream RL performance. The reconstruction quality monotonically increases from (b) – (e).

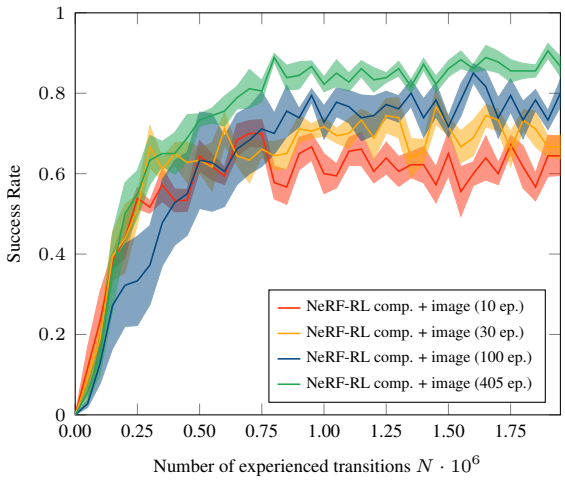


Figure 6: Learning curves investigating the influence of the reconstruction quality of the learned state representation on RL performance for mug on hook environment.

Fig. 6 compares the learning curves in the mug environment for NeRF-RL comp. + image with state representation networks trained for different numbers of epochs. As one can see, there is a correlation between asymptotic performance and reconstruction quality. The better the reconstruction quality, the higher the performance. An explanation for this is that the encoder network trained for only 10 epochs (see Fig. 5b) is unable to reconstruct the mug handle precisely, hence the latent vector likely does not contain the necessary information about the handle shape either. As the mug shape distribution also contains handles that have larger openings, even without representing the handle shape precisely, an agent can have some success (red curve in Fig. 6). However, it never reaches the performance that is possible if the state representation is capable of precisely reconstructing the handle shape (green curve), as for mugs with a small handle size, the policy needs this geometrical information in order to be successful.

A.2 Influence of Data Size for Encoder Pre-Training

In this section we investigate, for the mug and pushing environment, the effect on the RL performance if the auto-encoder is trained with less data.

In Fig. 7 one can see that for the pushing scenario, having 10 times less data available (10,000 instead of 100,000) to train the encoder offline does not affect the performance with statistical significance. Similarly, for the mug on hook scenario, as shown in Fig. 8, halving the dataset from 10,000 to 5,000 samples does not affect performance either. Note that, as described in Sec. B.1, the parameter space from which the mug environment is sampled is 11 dimensional. Despite neither 5,000 (this experiment) nor 10,000 (original experiment) data points cover this space at all, the auto-encoder can be trained to learn a latent space that allows the RL policy to effectively generalize to all possible shapes within the distribution. Generally, the offline data required to train our latent state representation is orders of magnitudes smaller than the number of samples during online RL.

A.3 Influence of Number of Views

In order for an MDP to be well-defined, the relevant state information (in the first-order Markov sense) has to be given to the policy. If only one single view is used, and relevant parts of the environment are, e.g., occluded, this assumption can be broken. This problem becomes less severe if multiple views of the scene are readily available, which is often the case in robotics applications. In these scenarios, NeRF-RL enables the fusion of multiple views into one single state representation.

On the other hand, it is useful to know to which extent NeRF-RL *relies* on having access to multiple views. In all experiments so far, we utilized 4 views of the environment as input to the encoder. As can be seen from Fig. 7, the performance of NeRF-RL is largely unaffected by using two views or even one single view in the pushing environment. The single-view setup for the pushing environment

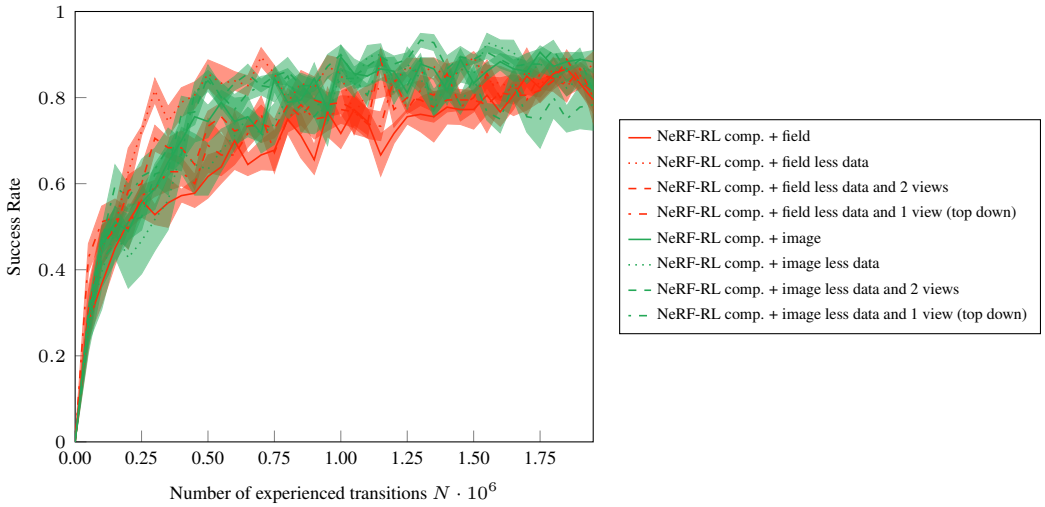


Figure 7: Learning curves for the pushing environment investigating the influence of the amount of data to train the latent state representation and of the number of input views for the encoder on the RL performance. Neither reducing the dataset size by a factor of 10, nor additionally reducing the number of input views to only a single one has a statistically significant effect on the RL performance.

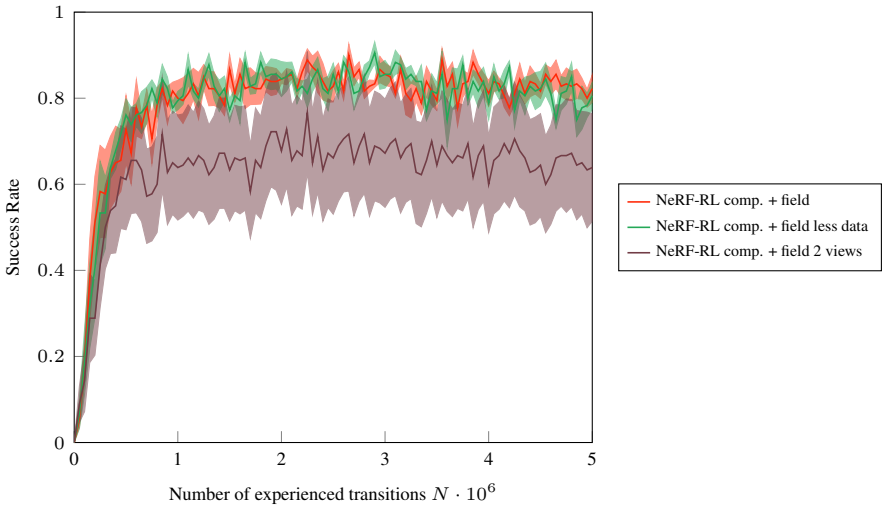


Figure 8: Learning curves for the mug environment investigating the influence of the amount of data to train the latent state representation and the number of input views for the encoder on the RL performance. Reducing the dataset size by a factor of 2 has no statistically significant effect on the RL performance.

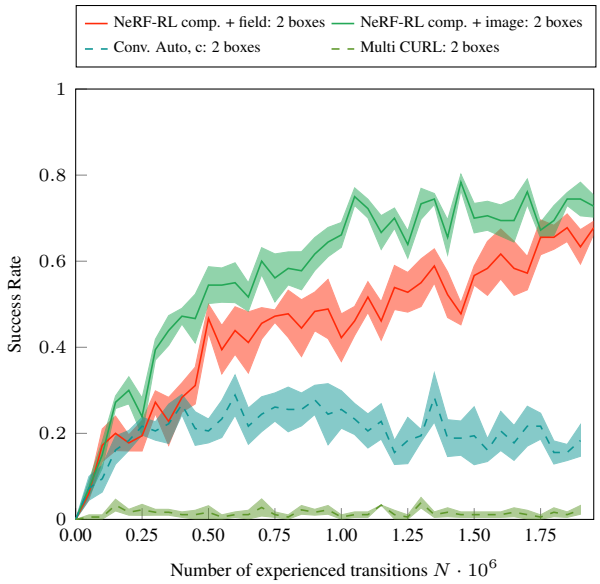


Figure 9: Learning curves for pushing scenario with two objects in the scene. The policy is directly applied to these scenes without further training, i.e. the learning curves show the evaluations on the two object scene environments for policies that were trained for different numbers of timesteps on the original environment with only one object.

uses a single top-down view. Since in this environment, the movement of the object happens mostly on the table, this setup represents a 2D task. Therefore, these results show that NeRF-RL can also be applied to scenes that are 2D in nature. However, the focus of this work is on 3D scenes, where the 3D geometry, as opposed to the pushing scenario, is crucial for the policy to be successful. Therefore, in the more complicated mug environment, as shown in Fig. 8, the performance drops if we use 2 instead of 4 views, but still leads to acceptable performance.

A.4 Generalization to More Objects (Pushing Scenario)

The compositional variants of the auto-encoder are able to generalize to different numbers of objects in the scene without retraining. Therefore, we can take a policy trained on a scenario with a single object and run it on scenes containing more than one object.

Fig. 9 reports learning curves for the pushing environment with two objects in the scene. These learning curves show the evaluations on the two object scene environments for policies that were trained for different numbers of time-steps on the original environment with only one object. Our approach generalizes well in this case, while the (compositional) baselines fail. In the supplementary material, we show a video of a NeRF-RL policy solving a scene containing two objects.

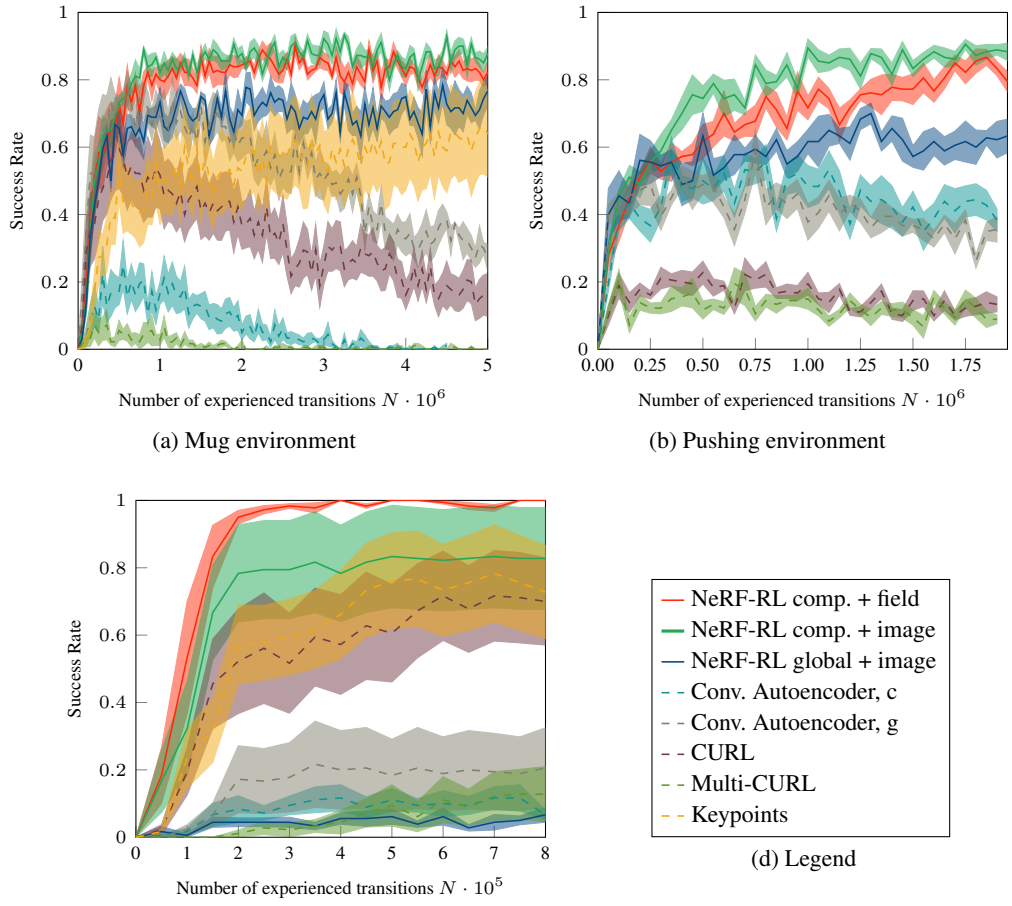
A.5 Additional Learning Curves

In this section, we report learning curves for all environments that include the global version of the NeRF-RL framework (NeRF-RL global + image) as well as an extended keypoint version for the mug environment.

As can be seen in Figures 10a and 10b (dark blue curve), although slightly worse than the compositional variant, the global version of NeRF-RL (NeRF-RL global + image) outperforms all other baselines where the encoder is not trained with NeRF supervision. As discussed in Sec. 7, this indicates that the masks are not the main reason for the improved performance of NeRF-RL. Further, both the Conv. Autoencoder, c and Multi-CURL baseline also use the masks in the same way as NeRF-RL comp. However, as shown in Fig. 10c, for the door environment, the global NeRF-RL version does not learn anything significant. This can be explained by the fact that for the door environment, the auto-encoder NeRF-RL global + image did not learn a correct reconstruction of

the pusher, therefore the latent code presumably does not represent the pusher's position, which is a necessary prerequisite to solve the task.

Fig. 12 shows additional keypoints baselines ("keypoints+", "keypoints larger network", "keypoints++") for the mug on hook environment. For the "keypoints+" baseline, we choose a larger number of keypoints, with additional ones located at the corner points of the mug handle, and on the opposite side of the mug (visualized in Fig. 11). For "keypoints++", we additionally project the "keypoints+" locations into a total of 128 values with a fixed matrix, leading to the same input size to the policy/value network as for all other baselines that use a latent state representation. Hence, in this case, the policy/value network has the same number of parameters as for NeRF-RL or the other latent space baselines. Similarly, for "keypoints larger network", we increase the number of hidden units in the first layer of the policy/value networks such that these networks have the same number of parameters, despite the input dimension being smaller. Note that, as mentioned above, for the other baselines, the policy/value networks have the same number of parameters as NeRF-RL. The performance with more keypoints ("keypoints+", "keypoints++") is slightly better than the "keypoints" baseline with less, but is still surpassed by compositional NeRF-RL. The "keypoints larger network" baseline is able to reach similar asymptotic performance as NeRF-RL, but learns slower than NeRF-RL.



(a) Mug environment

(b) Pushing environment

(c) Door environment

(d) Legend

Figure 10: Learning curves.

A.6 Mask Perturbations

For the compositional variants of the auto-encoders (NeRF-RL global + image, Conv. Autoencoder, g), we assumed to have access to object masks, cf. discussion in Sec. 7. In this experiment, we

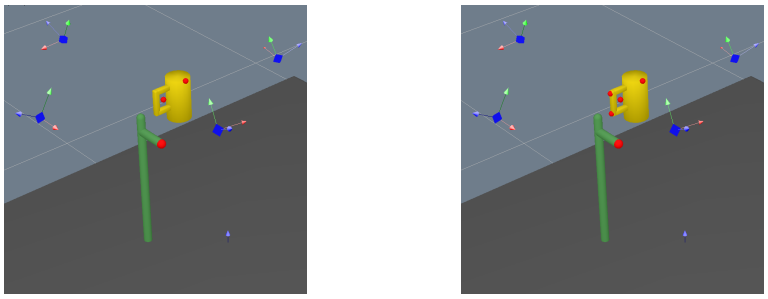


Figure 11: Visualization of the keypoint locations for the mug on hook environment. Left: Keypoints corresponding to "keypoints" in Fig. 10. Right: Keypoints corresponding to "more keypoints" in Fig. 10.

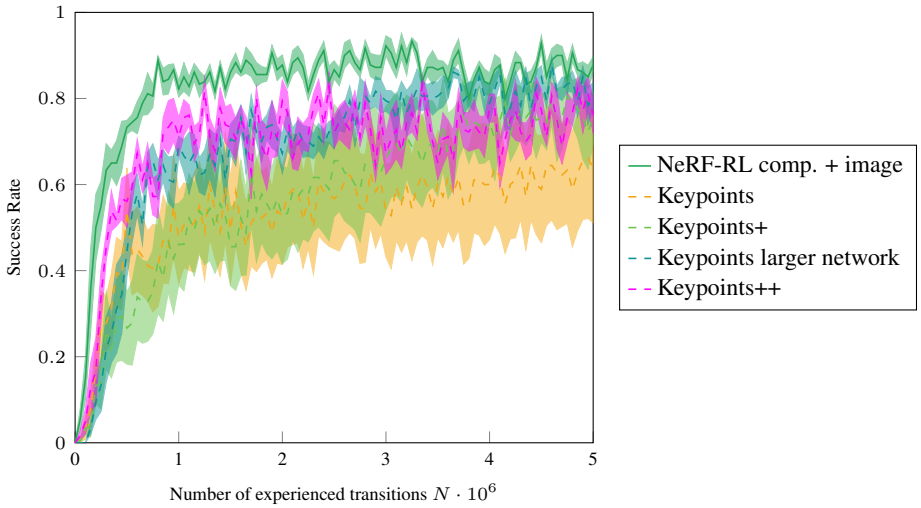


Figure 12: Learning curves investigating multiple variants of keypoints baselines for the mug on hook scenario.

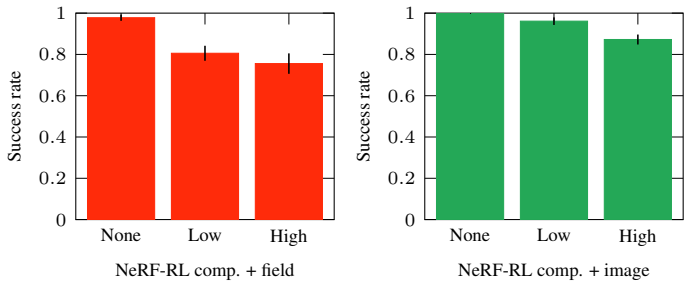
investigate how the performance of an agent that was trained with perfect masks is deteriorated when the masks are perturbed. This measures the robustness of the framework to mask perturbations, in scenarios where an agent is not allowed to adapt to these mask perturbations.

For each version of NeRF-RL, we train 6 independent agents on unperturbed input, create checkpoints at intervals of 50,000 steps, evaluate these checkpoints by running 30 test rollouts, and then for each experiment select the checkpoint with the highest average success rate on these test rollouts. We then test the success rate of these trained agents if we apply mask perturbations to the input before feeding them into the encoder. We randomly remove 2 (low perturbation) or 6 (high perturbation) patches from the object masks of each object in each view. As mentioned before, the agents are trained using unperturbed masks only.

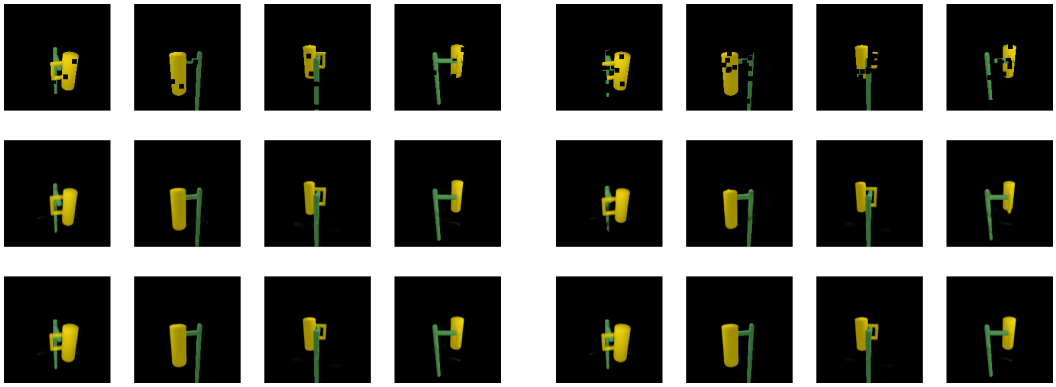
In the mug hanging environment (Fig. 13), we find that the performance of the NeRF-RL agents is very robust even against quite strong perturbations in the input masks. As can be seen in Fig. 13b, the perturbations remove part of the handle from the mug mask, but the model is still able to reconstruct the handle shape.

In box pushing (Fig. 14) and door opening (Fig. 15), the size of the red pusher in the image is small compared to the severeness of the mask perturbations. Therefore, already the low perturbation very often removes the majority of the red pusher from the input to the encoder, hence the corresponding latent vector representing the red pusher has nearly no information about its position in space. Consequently, the agent's performance deteriorates faster compared to the mug scenario. Nevertheless,

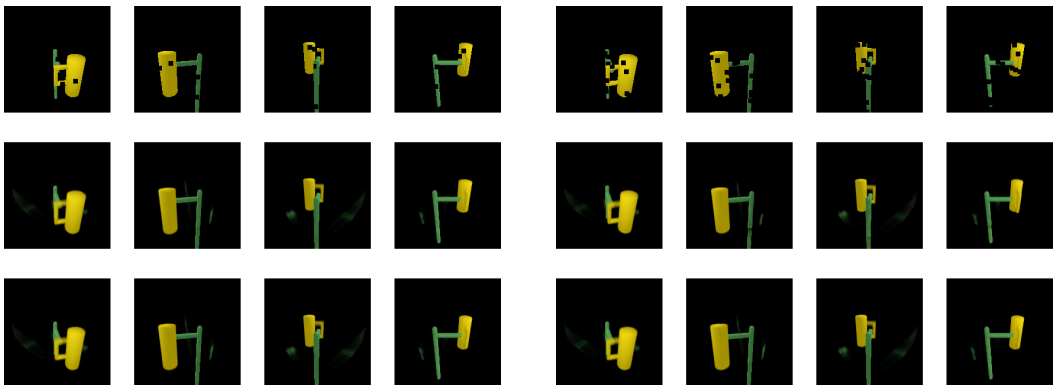
especially the NeRF-RL comp. + image variant can achieve reasonable performance even when the red pusher (and the box for the pushing scenario) is nearly invisible due to the mask perturbations.



(a) Performance of the NeRF-RL agents after applying mask perturbations at three different levels (no perturbation, low perturbation, high perturbation).

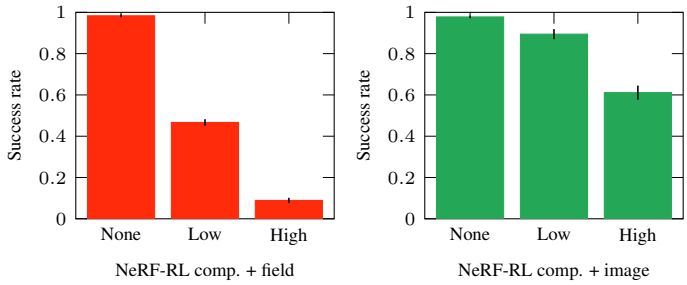


(b) Image reconstructions for NeRF-RL comp. + field with mask perturbations. Left: low perturbation, right: high perturbation. The first line shows perturbed mask inputs, second line shows reconstructions from these, third line shows reconstructions from unperturbed input.

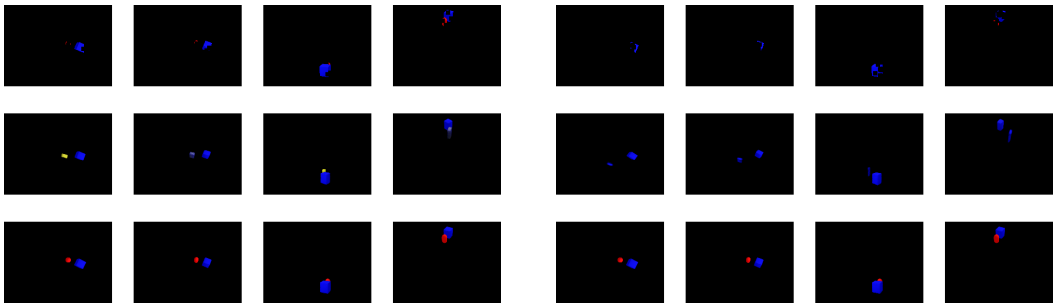


(c) Image reconstructions for NeRF-RL comp. + image with mask perturbations. Left: low perturbation, right: high perturbation. The first line shows perturbed mask inputs, second line shows reconstructions from these, third line shows reconstructions from unperturbed input.

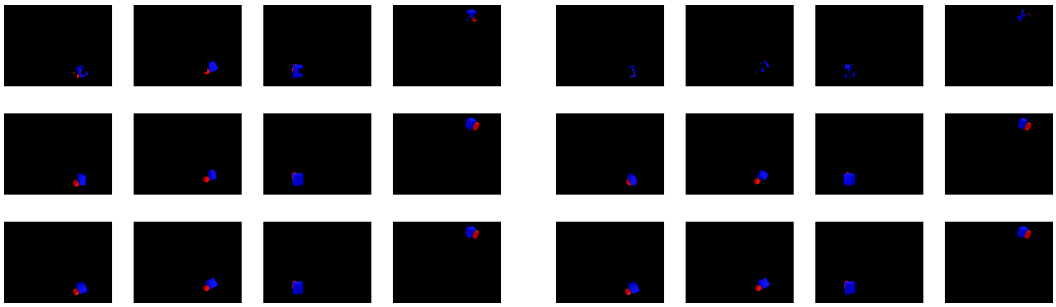
Figure 13: Mask perturbation experiments for mug on hook.



(a) Performance of the NeRF-RL agents after applying mask perturbations at three different levels (no perturbation, low perturbation, high perturbation).

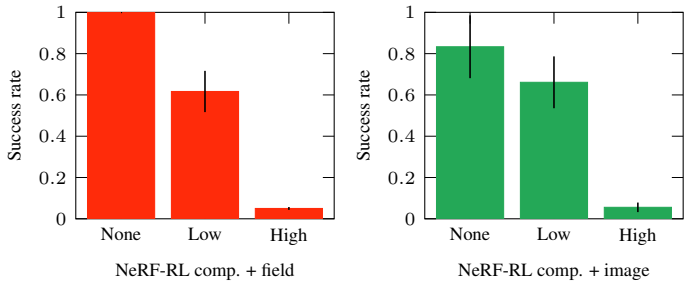


(b) Image reconstructions for NeRF-RL comp. + field with mask perturbations. Left: low perturbation, right: high perturbation. The first line shows perturbed mask inputs, second line shows reconstructions from these, third line shows reconstructions from unperturbed input.

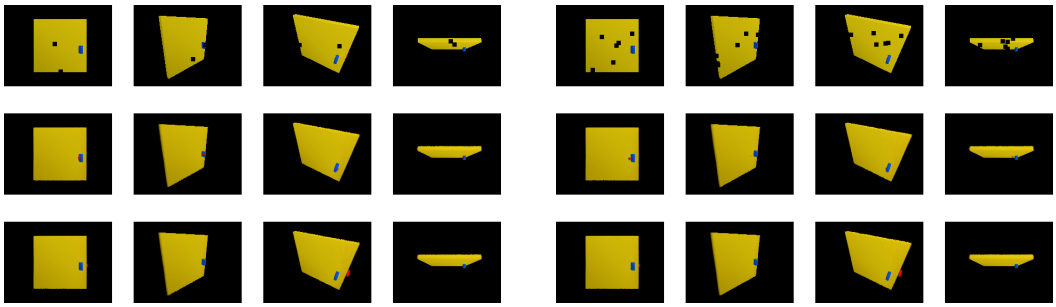


(c) Image reconstructions for NeRF-RL comp. + image with mask perturbations. Left: low perturbation, right: high perturbation. The first line shows perturbed mask inputs, second line shows reconstructions from these, third line shows reconstructions from unperturbed input.

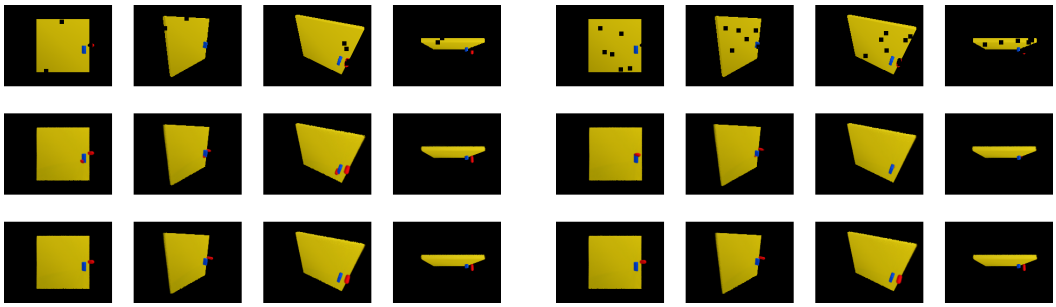
Figure 14: Mask perturbation experiments for box pushing.



(a) Performance of the NeRF-RL agents after applying mask perturbations at three different levels (no perturbation, low perturbation, high perturbation).



(b) Image reconstructions for NeRF-RL comp. + field with mask perturbations. Left: low perturbation, right: high perturbation. The first line shows perturbed mask inputs, second line shows reconstructions from these, third line shows reconstructions from unperturbed input.



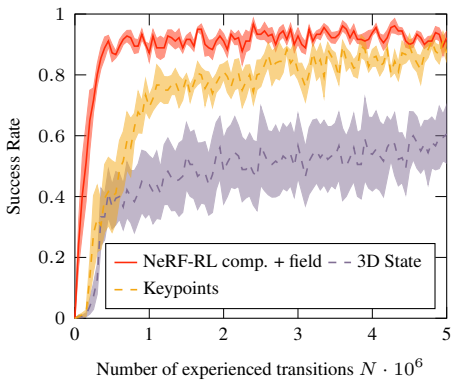
(c) Image reconstructions for NeRF-RL comp. + image with mask perturbations. Left: low perturbation, right: high perturbation. The first line shows perturbed mask inputs, second line shows reconstructions from these, third line shows reconstructions from unperturbed input.

Figure 15: Mask perturbation experiments for door opening.

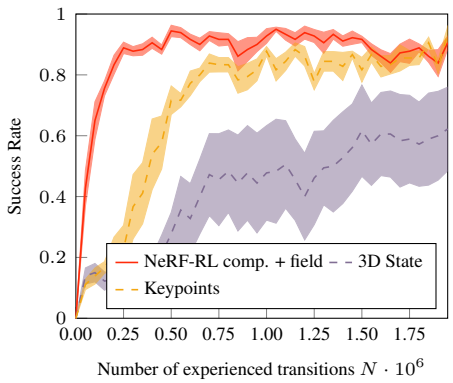
A.7 Single Shape

In this experiment, we consider the same environments as in Sec. 6.1 but with a fixed object shape. This means that only the initial configuration is randomized in each run, but the same object shape is used throughout training and evaluation. Hence, the agents are not required to generalize over object shapes. This allows us to consider low-dimensional pose-like state input (3D state) as a baseline. In case of the mug environment, this is the 3D input of the position of the mug, for the pushing environment a 8D input (2D position of the pusher, 2D position of the box, 4D quaternion for rotation of box), and 4D for the door environment (3D position of the red pusher, 1D opening of the door).

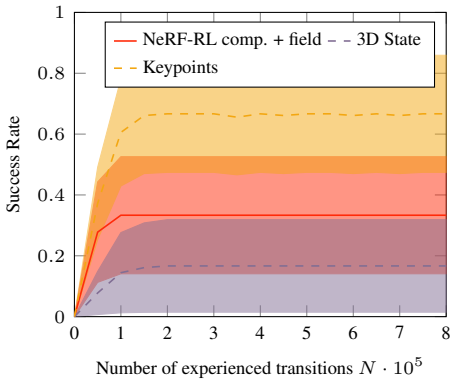
As can be seen in Figures 16a and 16b, for the mug and pushing environment, even in the single shape case, the NeRF-RL state representation outperforms both direct 3D state and keypoint representations, not only in terms of asymptotic performance, but especially in terms of sample efficiency. The keypoint representation works better than the (lower dimensional) 3D state. Fig. 16c shows that in case of the door environment, the keypoint representation leads to higher performance compared to NeRF-RL comp. + field. We hypothesize that when using a single shape, there is not enough randomness in the door environment for the agent to achieve a sufficient success signal by exploration (sparse reward setup), which can also be seen by the much larger confidence intervals. Nevertheless, the overall best performance achieved in the single shape door case is lower than what our method (NeRF-RL comp. + field and NeRF-RL comp. + image) is able to achieve in the more challenging multi-shape door environment task, see Fig. 4a.



(a) Mug on hook environment

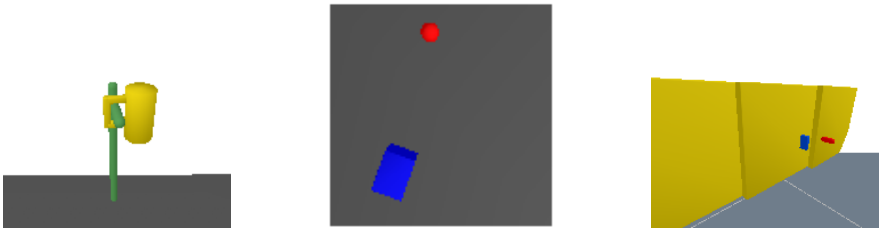


(b) Box pushing environment



(c) Door opening environment

Figure 16: Single shape experiments. We compare NeRF-RL with using 3D state as input, and with using keypoints chosen by expert knowledge. NeRF-RL outperforms 3D state in all environments, and outperforms expert keypoints in all environments except door opening.



(a) Mug environment (b) Pushing environment (c) Door environment

Figure 17: Views used for the CURL baseline. They have been chosen such that a single view represents the state of the system as best as possible.

B Experiment/Environment Details

This section provides details about our environments, parameter choices and training times. Please refer to the provided code for details about the exact network architectures.

We use PPO [86] as the RL algorithm from the stable-baselines-3 [89] implementation for all experiments. The hyperparameters of the PPO algorithm are the default ones from [89] for all environments and all experiments. The dimension of the latent vector is $k = 64$ for each object, or in case a global framework is used, it is 64 times the number of objects in the environment. This way, the total dimensionality of the latent vector describing the scene is always the same, independently from whether a compositional- or global-style framework is used. All comparison methods utilize the same amount of training data, as well as, where applicable, use the same network sizes. For CURL, we used the same architecture as proposed in the original publication, which resulted in NeRF-RL having larger network sizes for the mug and door environment, but a smaller network size in the box scenario than the CURL baseline.

All experiments utilize four camera views as input to the encoder. Please refer to the supplementary video to see a visualization of these input views.

B.1 Mug on Hook

In this environment, adopted from [87] and visualized in Fig. 2b, the task is to hang a mug on a hook. Both the mug and the hook shape are randomized. More specifically, the height, radius, handle size, and handle position of the mug vary. This leads to an 8 dimensional parameter space for the shape of the objects in the scene. The actions are small 3D translations applied to the mug.

This environment is challenging as we require the RL agent to generalize over mug and hook shapes and the tolerance between the handle opening and the hook is relatively small. Further, the agent receives a sparse reward only if the mug has been hung stably. This reward is calculated by virtually simulating a mug drop after each action. If the mug would not fall onto the ground from the current state, a reward of one is assigned, otherwise zero.

To collect the data for training the state representations we uniformly sample the mug position within the workspace region and only add those samples to the dataset where the mug and the hook are not in collision. In total, the dataset contains 10,000 configurations. As this includes not only the position of the mug, but also variations in shape of the mug and the hook, this is not a dense coverage of the 11 dimensional (8 shape + 3 initial mug position) parameter space from which the scenes are sampled. Note that as the training data for the auto-encoder is collected by randomly sampling mug configurations, there are no trajectories in the dataset. Furthermore, it is much less likely to sample a mug configuration that is stable. Hence the data distribution the RL agent generates online potentially differs from the offline dataset to train the state representation. Please refer to the video for a visualization of the offline data generation procedure.

Fig. 2b shows the 4 camera poses (blue coordinate frames), which are also visualized in the video. In the compositional case, the mug and the hook are represented by their own latent vector, in the global case both the mug and the hook have one single latent vector together.

The keypoint baseline utilizes 4 keypoints, three on the mug and one on the hook. Fig. 2b visualizes these keypoints as red dots, one keypoint on the mug is not visible (center of the mug). As can be seen, those keypoints have been chosen to reflect the relevant geometry to solve the task, in particular the center position of the mug and the hook position.

B.2 Planar Pushing

The task in this environment, shown in Fig. 3b, is to push yellow box-shaped objects into the left region of the table and blue objects into the right region, using the red pusher that can move in the plane, i.e., the action is two dimensional. This is the same environment as in [53] with the same four different camera views. Each scenario contains a single object on the table (plus the pusher). The boxes in the environment have different sizes, two different colors (blue, yellow) and are randomly initialized (shape, color, and pose). The box and the pusher are represented by a separate latent vector for the compositional case, and a single latent vector for both in the global case. If the box has been pushed inside its respective region, a sparse reward of one is received, otherwise zero. These regions are shown only for visualization purposes in Fig. 3b and not part of the input views for the network.

We use the same dataset as in [53] to train the auto-encoder, which has been collected by applying movements of the red pusher in a randomly chosen direction, biased towards the object center until either the pusher leaves the workspace in which case a new direction is chosen or the box is pushed from the table in which case the environment is reset with a new randomly chosen box shape, color and initial configuration of the pusher and box. Note that the offline training data in particular does not contain trajectories that push boxes in goal directed ways. Please refer to the video for a visualization of the offline data generation procedure.

In this environment, we cannot use keypoints for the multi-shape setting, as the reward depends on the object color. Hence, we evaluate the keypoints baseline only in the single shape case, cf. Sec. A.7.

B.3 Door Opening

Fig. 4b shows the door environment, where the task is to open a sliding door with the red end-effector that can be translated in 3 DoFs as the action. To solve this task, the agent has to push on the door handle. As the handle position and size are randomized, the agent has to learn to interact with the handle geometry accordingly. Interestingly, as can be seen in the video in the supplementary material, the agent often chooses to push on the handle only at the beginning, as, afterwards, it is sufficient to push the door itself at its side. The agent receives a sparse reward if the door has been opened sufficiently, otherwise, zero reward is assigned.

In total, the dataset to train the latent space contains 50,000 configurations, which is collected by randomly sampling a handle configuration (size and position on the door), door opening, and the initial height of the end-effector. The end-effector then moves in a random direction until it either cannot move anymore or it leaves the workspace, in which case a new door handle, door opening, and starting position of the end-effector is sampled. Please refer to the video for a visualization of the offline data generation procedure. These random directions are in a 2D plane of the sampled initial pusher height. Therefore, during RL, the agent has to apply actions to solve the task leading to trajectories that are not part of the offline dataset used to train the state representation.

Similar to the other environments, we use four different camera views. There are two latent vectors in the compositional case, one representing the red end-effector, and the other the yellow door including the blue handle. The keypoint baseline utilizes 5 keypoints, three on the door handle, one on the door and one on the red pusher.

B.4 Computational Resources

The majority of our experiments have been performed on a GPU server containing 7 NVIDIA RTX3090 GPUs, 1 TB RAM and two AMD EPYC 7452 CPUs. Training the NeRF-based auto-encoder on a single such GPU takes 2-3 days. Training the convolutional auto-encoder baselines on a

single GPU takes 3-5 days. Training the CURL baselines takes half a day on a single GPU. After the representations have been learned, RL training on a single GPU takes 4 to 5 days in the mug hanging environment, roughly 1 day in the box pushing environment, and 1 to 2 days in the door opening environment. The longer RL training time for the mug environment is mainly caused by an inefficient collision check in the respective simulator.