Kinematic Morphing Networks for Manipulation Skill Transfer

Peter Englert and Marc Toussaint

***Abstract*— The transfer of a robot skill between different geometric environments is non-trivial since a wide variety of environments exists, sensor observations as well as robot motions are high-dimensional, and the environment might only be partially observed. We consider the problem of extracting a low-dimensional description of the manipulated environment in form of a kinematic model. This allows us to transfer a skill by defining a policy on a prototype model and morphing the observed environment to this prototype. A deep neural network is used to map depth image observations of the environment to morphing parameter, which include transformations and configurations of the prototype model. Using the concatenation property of affine transformations and the ability to convert point clouds to depth images allows to apply the network in an iterative manner. The network is trained on data generated in a simulator and on augmented data that is created with its own predictions. The algorithm is evaluated on different tasks, where it is shown that iterative predictions lead to a higher accuracy than one-step predictions.**

I. Introduction

Modern robots are equipped with sensors (e.g., camera, laser scanner) that allow them to perceive their environment. For many policy representations, raw sensor signals are not directly usable as an input since they are too abstract and high-dimensional. Therefore, algorithms are necessary to extract a representation that is a suitable input for such policies. We propose to use kinematic models of the environment as such a representation. The main objective of this work is to extract parameters of such kinematic models from an observed environment.

In this paper, we consider the scenario where a robot observes a manipulation environment with a depth sensor. The observation is taken from a specific viewpoint, which often results in a measurement that only covers certain parts of the environment. The goal is to learn a deep neural network that maps observations to parameters, which are the input to a manipulation policy. We assume to have the kinematic model structure of the manipulated environment and a simulator that can create large amounts of supervised training data. In the following, we describe the different components of our approach.

A. Environment Parametrization

We represent the manipulated environment with a kinematic model that consists of rigid bodies and joints. Figure 1 shows such a kinematic model for a door environment with its parametrization (e.g., width, position, handle location). A key element of our approach is a *prototype* that serves as a reference to describe a model. The models are parametrized by *morphing parameters* that define the mapping of each point of a model to a corresponding point on the prototype. The parameters of this morphing include 3D transformation parameters as well as configuration parameters of the prototype, such as the height of a door handle. We assume that if a policy for the prototype model and the morphing parameters of an observed environment are known, then the policy can be transferred from the prototype to the observed environment. Specifically, we will use a trajectory optimization method to define costs and constraints depending on the morphing parameter. These costs and constraints describe how the robot should interact with the environment (e.g., contacts) in order manipulate it into a desired state.

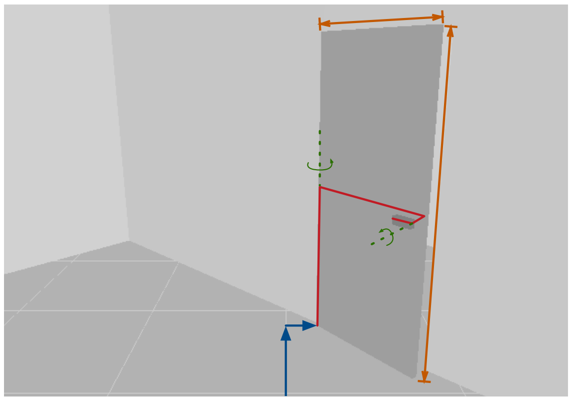


Fig. 1: Kinematic model of a door that we want to extract from depth images and use to transfer manipulation skills.

B. Kinematic Morphing Model

The goal of this work is to extract the morphing parameters from sensor observations. The proposed *kinematic morphing model* is defined as a convolutional neural network that maps depth images to morphing parameters. We propose a data augmentation method that uses the network predictions to generate more data. This augmentation is done by applying the predicted morphing parameters on the input point cloud and generating a new depth image from it. We use the same mechanism to make predictions with the network in an iterative manner. This can be viewed as a controller that changes the inputs in multiple steps until a steady point is reached. In our case the steady point is the prototype model and the goal is to transform all observed models to this prototype. The advantage is that the model does not have to predict the morphing parameters in a single step. We show

This work was supported by the DFG project Exploration Challenge TO 409/9-1.

A python implementation of the proposed algorithm is available at `https://github.com/etpr/kinematic_morphing_network`

Peter Englert and Marc Toussaint are with the Machine Learning & Robotics Lab, University of Stuttgart, Germany.
Email: englertpr@gmail.com

in our experiments that this results in an increase of the prediction accuracy.

C. Data Generation in Simulation

Training the parameters of a deep neural network requires a large amount of data, which is difficult to collect in the real world since the labels would have to be provided by hand. In this work, we follow a recent trend to generate synthetic data with simulators (see Section II-B). A kinematic engine and OpenGL renderer are used to create 3D representations of environments for a given set of morphing parameters. This allows to generate a large supervised dataset that consists of point clouds, depth images, and morphing parameters. In this paper, we focus on the morphing of the kinematic models and therefore assume that the background has already been removed in the data. The kinematic morphing model is trained on this data and on augmented data that is created with network predictions.

Combining all these ingredients provides us with a tool to extract a compact representation from high dimensional sensor data, which can be used to transfer robot skills between different environments. We will demonstrate these transfer abilities in the experimental section, where the same skill policy is used to manipulate doors of different shapes and locations. The main contribution of this work is the kinematic morphing network that is trained iteratively by augmenting the data with its own predictions.

II. Related Work

A. Extracting 3D Models from Data

Point set registration methods try to find the transformation between two observations of a model [1], [2]. Iterative Closest Point (ICP) is a widely used algorithm to align two point clouds [1]. ICP iterates between the steps: 1) Matching the points between the two point clouds by finding the closest pairs; 2) Computing a transformation that minimizes the distances between the point pairs; 3) Applying the transformation on a point cloud and continuing with step 1. The advantage of these kind of methods is that they do not require an expensive training procedures like neural networks. However, they still have open parameters that influence the performance and the initial estimate of the transformation is important. The main difference to our approach is that we can also handle kinematic model parameters beyond affine transformations. We compare our approach to ICP in the experiment in Section V-A.

An alternative strategy is to extract models from motions of the environment [3], [4]. Martin-Martin *et al.* [4] do a feature based approach by tracking the motion of different feature points and extracting a kinematic model from them. Sturm *et al.* [3] follow a probabilistic approach to extract a kinematic model from pose trajectories of rigid bodies. The objective of these algorithms is similar to our work. The main difference to our work is that we do not require object motions, which is difficult to produce in an automated way when the environment is initially unknown.

Zhou *et al.* [5] follow a similar approach to ours and learn a deep neural network that predicts the configuration of a kinematic hand model from depth images. The forward kinematic function is integrated as final layer into the network and outputs the location of each joint. The loss function is defined on the location of these joints and an additional loss term ensures that the predicted parameters fulfil physical constraints. In our approach, we directly compute the error in the parameter space and also train the network to predict the model structure (e.g., finger length).

B. Learning in Simulated Environments

Using simulation environments is a way to bypass the lack of large datasets in robotics. However, bridging the gap between simulated and real sensor data (e.g., images) is still an open research question. In [6], a robot grasping skill is improved by using synthetic data generated with a simulator. The proposed approach uses domain adaptation techniques [7], [8] that map synthetic images to realistically looking images. They use generative adversarial networks [9] to learn this mapping by using two networks that are trained adversarial. The generated data is used to train a network that maps RGB images and actions to grasp success probabilities [10]. Their results show that the use of simulation data leads to a consistent improvement of the overall grasp success rate. Rusu *et al.* [11] transfer policies from simulated to real robots by using Progressive Neural Networks [12]. The first column of the progressive neural network is trained in simulation. All further layers are trained on the real robot while the first column is kept fixed. The training is done with the Asynchronous Advantage Actor-Critic [13] method. The input to the network is an RGB image and the outputs are a discrete velocity signal for each joint plus a value function. Instead of following an end-to-end approach, we propose a more structured way of transferring skills by extracting a representation that is suitable as input for standard planning methods.

Mitash *et al.* [14] propose an object detection algorithm based on a physics simulation and a real-world self learning mechanism. The physics simulator uses CAD models of the objects to generate realistic scenes. Each scene is rendered from multiple perspectives and the produced RGB-D images are used to train a convolutional neural network. The trained network is used to generate more labeled training data in a real world environment. Thereby, a robot is used to arrange the scene and take multiple images from different perspectives. The detected objects with high confidence are used to create corresponding labels for all perspectives. Their results show that the simulation part of the algorithm provide the policy with a good starting point for the real-world self learning.

C. Integrating 3D Geometry in Neural Networks

There are different ways of how neural networks can be used with 3D sensor data and how transformations or rendering operations can be represented within a network. A problem that often occurs is that the observations are taken from a specific viewpoint, which leads to only partial observations of objects. Eitel *et al.* [15] do object detection based on RGB-D data. The model consists of a two-stream convolutional neural network with an RGB image input and a depth image input. The output is a probability of how likely an object is in the image. They compare different ways to represent depth images and to transfer pre-trained networks trained on RGB data. Byravan *et al.* [16] learn rigid body motions with deep neural networks based on depth data. The model inputs are a point cloud shaped as an XYZ image and a force vector. The network combines both inputs in a late fusion architecture and outputs a transformed point cloud image. The transformation is done by predicting a fixed amount of object masks and corresponding rigid body transformations.

A Spatial Transformer Network (STN) is a network module to transform an input feature map to an output feature map [17]. STN can be used as a layer in a network and are differentiable. The parameters of the affine transformation are predicted based on the input feature map. Using STN leads to invariance regarding translation, scale and rotation. Rezende *et al.* [18] use STN to extract 3D structure from images. A conditional latent variable model with a low dimensional codec is used to map observed data to an abstract code that a decoder maps to volume representations. Similar to our approach, they also use an OpenGL renderer to convert from a 3D representation to image. In our case, the low dimensional representation are interpretable kinematic parameters that can be used for planning methods. Discretizing the 3D space might be suitable for some tasks like object detection. However, the achieved accuracy strongly depends on the resolution of the grid.

III. Model & Policy Representation

A. Kinematic Model of the Manipulated Environment

We introduce a parametrized kinematic model of the environment $\boldsymbol{m}(\boldsymbol{\theta}, \boldsymbol{\gamma}) \subset \mathbb{R}^3$, which is defined as a set of points of the manipulated environment (e.g., a door) with parameters $\boldsymbol{\theta} \in \mathbb{R}^n$ and $\boldsymbol{\gamma} \in \mathbb{R}^m$. A prototype model $\boldsymbol{m}(\boldsymbol{\theta}_0, \boldsymbol{\gamma}_0)$ is defined as a reference for other models, where $\boldsymbol{\gamma}_0$ and $\boldsymbol{\theta}_0$ are usually set to $\boldsymbol{0}$. In this paper, the term *morphing* is used to describe a mapping of a model $\boldsymbol{m}(\boldsymbol{\theta}, \boldsymbol{\gamma})$ to the prototype $\boldsymbol{m}(\boldsymbol{\theta}_0, \boldsymbol{\gamma}_0)$. The parameters of the kinematic model are:

1) The *transformation parameters* $\boldsymbol{\theta}$ of an affine transformation $\boldsymbol{T}_{\boldsymbol{\theta}} \in \text{Aff}(3)$ that describes the linear mapping between two models. The parameters $\boldsymbol{\theta}$ are specific rotation, translation, and scale parameters around or along a certain axis.
2) The *configuration parameters* $\boldsymbol{\gamma}$ describe the nonlinear mapping between two models that cannot be represented with an affine transformation of the complete model. An example is the relative position between two bodies of the environment.

The affine transformation of the morphing operation for a given configuration $\boldsymbol{\gamma}$ is

$$\boldsymbol{m}(\boldsymbol{\theta}_0, \boldsymbol{\gamma}) = \boldsymbol{T}_{\boldsymbol{\theta}}^{-1} \boldsymbol{m}(\boldsymbol{\theta}, \boldsymbol{\gamma}) \ . \tag{1}$$

This equation describes how all points of a model paramerized by $\boldsymbol{\theta}$ transform onto a prototype $\boldsymbol{\theta}_0$. The goal of this paper is to predict model parameters $(\boldsymbol{\theta}, \boldsymbol{\gamma})$ from sensor observations of the model. These parameters relate the current observed model to the prototype model, which will be used to adapt the policy from the prototype to the observed model.

B. Constrained Trajectory Optimization Policies

Our policy representation is a constrained trajectory optimization problem consisting of costs and constraints that describe how the robot should interact with the environment. We use k-order Markov optimization [19] that finds trajectories $\bar{\boldsymbol{x}}_{0:T} \in \mathbb{R}^{Q \times (T+1)}$ by solving the problem

$$\begin{aligned} \bar{\boldsymbol{x}}_{0:T}^{\star} = \operatorname*{arg\,min}_{\bar{\boldsymbol{x}}_{0:T}} & \ \boldsymbol{w}^\top \boldsymbol{\Phi}^2(\bar{\boldsymbol{x}}_{0:T}, \boldsymbol{m}) \\ \text{s.t.} & \ \ \boldsymbol{g}(\bar{\boldsymbol{x}}_{0:T}, \boldsymbol{m}) \leq \boldsymbol{0} \\ & \ \ \boldsymbol{h}(\bar{\boldsymbol{x}}_{0:T}, \boldsymbol{m}) = \boldsymbol{0} \ . \end{aligned} \tag{2}$$

$\boldsymbol{\Phi}$ are cost features (e.g., endeffector position/orientation) of the trajectory and $\boldsymbol{w}$ are feature weights. Inequality constraints $\boldsymbol{g}$ and equality constraints $\boldsymbol{h}$ are used to define further properties (e.g., contacts, collision avoidance) of the motion. The kinematic model $\boldsymbol{m}$ is an input to the cost and constraint functions, which generalizes the skill between different models.

IV. Kinematic Morphing Networks

In this section, we propose an approach to extract morphing parameters from environment observations. The robot observes an instance of the kinematic model $\boldsymbol{m}(\boldsymbol{\theta}, \boldsymbol{\gamma})$ in form of a depth image $\boldsymbol{D} \in \mathbb{R}^{W \times H}$ and corresponding point cloud $\boldsymbol{P} \in \mathbb{R}^{3 \times (WH)}$. We define the function

$$f: \ \mathbb{R}^{W \times H} \rightarrow \mathbb{R}^n \times \mathbb{R}^m \tag{3}$$

that maps depth images $\boldsymbol{D}$ to morphing parameters $(\boldsymbol{\theta}, \boldsymbol{\gamma})$ (see Section III-A). In this paper, $f(\boldsymbol{D}; \boldsymbol{\beta})$ is represented as a neural network with parameters $\boldsymbol{\beta} \in \mathbb{R}^B$. In the proposed approach, data of the form $\mathcal{D} = \{(\boldsymbol{D}^{(i)}, \boldsymbol{P}^{(i)}, \boldsymbol{\theta}^{(i)}, \boldsymbol{\gamma}^{(i)})\}_{i=1}^N$ is used to optimize the network parameters $\boldsymbol{\beta}$. In the following sections, we describe the network prediction, training, and architecture.

A. Iterative Network Predictions

We introduce an iterative network prediction mechanism that applies the network in Equation (3) repeatedly. For a given input $(\boldsymbol{D}, \boldsymbol{P})$, the predictions are computed by iterating:

1) Predicting parameters $\boldsymbol{\theta}$ for $\boldsymbol{D}$ with Equation (3).
2) Applying the transformation $\boldsymbol{T}_{\boldsymbol{\theta}}$ to the point cloud $\boldsymbol{P}$.
3) Rendering a new depth image $\boldsymbol{D}$ from $\boldsymbol{P}$.

These three steps are repeated until a fixed point is reached. The resulting point cloud after t iterations is

$$\boldsymbol{P}_t = \boldsymbol{T}_{\boldsymbol{\theta}_t}^{-1} \dots \boldsymbol{T}_{\boldsymbol{\theta}_2}^{-1} \boldsymbol{T}_{\boldsymbol{\theta}_1}^{-1} \boldsymbol{P} \ . \tag{4}$$

Algorithm 1: Multi-step Network Predictions

```
*function* predict($\boldsymbol{D}, \boldsymbol{P}, \boldsymbol{\theta}, \boldsymbol{\beta}, N$) :
    **for** $d = 1 : N$
        $(\bar{\boldsymbol{\theta}}, \boldsymbol{\gamma}) = f(\boldsymbol{D}; \boldsymbol{\beta})$
        $\boldsymbol{P} = \boldsymbol{T}_{\bar{\boldsymbol{\theta}}}^{-1} \boldsymbol{P}$
        $\boldsymbol{D}$ = pointCloudToDepth($\boldsymbol{P}$)
        $\boldsymbol{\theta} = \bar{\boldsymbol{\theta}}^{-1} \circ \boldsymbol{\theta}$
    **end**
    **return** $(\boldsymbol{D}, \boldsymbol{P}, \boldsymbol{\theta}, \boldsymbol{\gamma})$
```

The idea is that in each step the point cloud is transformed a bit closer towards the prototype. After convergence, this point cloud should overlay with the prototype $\boldsymbol{\theta}_0$. The necessary steps are summarized in Algorithm 1. The inputs are a depth image $\boldsymbol{D}$, a point cloud $\boldsymbol{P}$, a previous transformation $\boldsymbol{\theta}$ (by default $\boldsymbol{\theta}_0$), network parameters $\boldsymbol{\beta}$ and number of predictions N. In the first step of the loop, the network predicts transformation parameters for the given depth image. Afterwards, the corresponding point cloud is transformed with the predicted transformation, which is then mapped to a new depth image. Finally, the predicted transformations are concatenated that we express with the symbol $\circ$. This procedure is repeated N times. The output are a new depth image and point cloud with the corresponding morphing parameter. An alternative to the fixed number of iterations N would be to repeat the steps until the network predicts a transformation that is close to the identity transformation, which indicates convergence. The algorithm requires a converting functionality *pointCloudToDepth* that renders a depth image $\boldsymbol{D}$ from the transformed point cloud $\boldsymbol{P}$. Since the configuration parameters $\boldsymbol{\gamma}$ cannot be predicted in an iterative manner, only the last prediction of the network is used.

B. Data Generation and Network Training

Algorithm 2 shows the combined data generation and network training. Throughout the training, the current state of the network is used to augment the training data by using Algorithm 1. The inputs are the parametrized model of the environment $\boldsymbol{m}(\boldsymbol{\theta}, \boldsymbol{\gamma})$, the neural network $f(\boldsymbol{D}; \boldsymbol{\beta})$, the initial dataset size N_{data}, the augmented dataset size N_{aug}, and the limits of the model parameters $(L^{\text{up}}, L^{\text{low}})$. In the first part of the algorithm, a dataset is generated with an OpenGL renderer that creates for a given model $\boldsymbol{m}(\boldsymbol{\theta}, \boldsymbol{\gamma})$ a depth image $\boldsymbol{D}$ and a point cloud $\boldsymbol{P}$. The parameters are thereby sampled uniformly in the feasible parameter range defined by L^{up} and L^{low}. Afterwards, the network is trained on the generated dataset. In the second part, the trained model is taken to augment the dataset by applying the model on a subset N_{aug} of the initial data. Thereby, the iterative network predictions with N_{pred} predictions from Algorithm 1 is used to generate a new datapoint. The augmented data is appended to D and the network is retrained. This second part can be seen as a fine-tuning of the network parameters for data points that are close the prototype. The data generation and retraining procedure is repeated until there is no further change in network parameters $\boldsymbol{\beta}$.

Algorithm 2: Data Generation and Network Training

```
**Input**
    Kinematic model $\boldsymbol{m}(\boldsymbol{\theta}, \boldsymbol{\gamma})$
    Neural network $f(\boldsymbol{D}, \boldsymbol{\beta})$
    Initial dataset size $N_{\text{data}}$
    Augmented dataset size $N_{\text{aug}}$
    Upper and lower parameter limits $(L^{\text{up}}, L^{\text{low}})$

Initialize dataset $\mathcal{D} = \emptyset$
**// Generate an initial dataset**
**for** $d = 1 : N_{\text{data}}$
    $(\boldsymbol{\theta}, \boldsymbol{\gamma}) \sim \mathcal{U}(L^{\text{up}}, L^{\text{low}})$
    $(\boldsymbol{D}, \boldsymbol{P})$ = render($\boldsymbol{m}(\boldsymbol{\theta}, \boldsymbol{\gamma})$)
    $\mathcal{D} = \mathcal{D} \cup \{(\boldsymbol{D}, \boldsymbol{P}, \boldsymbol{\theta}, \boldsymbol{\gamma})\}$
**end**
**// Train network**
$\boldsymbol{\beta} = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{|\mathcal{D}|} \left|\left|(\boldsymbol{\theta}^{(i)}, \boldsymbol{\gamma}^{(i)}) - f(\boldsymbol{D}^{(i)}; \boldsymbol{\beta})\right|\right|^2$
**// Generate data with model predictions**
$N_{\text{pred}} = 1$
**repeat**
    **for** $d = 1 : N_{\text{aug}}$
        $(\boldsymbol{D}, \boldsymbol{P}, \boldsymbol{\theta}, \boldsymbol{\gamma})$ = predict($\boldsymbol{D}^{(d)}, \boldsymbol{P}^{(d)}, \boldsymbol{\theta}^{(d)}, \boldsymbol{\beta}, N_{\text{pred}}$)
        $\mathcal{D} = \mathcal{D} \cup \{(\boldsymbol{D}, \boldsymbol{P}, \boldsymbol{\theta}, \boldsymbol{\gamma}^{(d)})\}$
    **end**
    **// Retrain network**
    $\boldsymbol{\beta} = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{|\mathcal{D}|} \left|\left|(\boldsymbol{\theta}^{(i)}, \boldsymbol{\gamma}^{(i)}) - f(\boldsymbol{D}^{(i)}; \boldsymbol{\beta})\right|\right|^2$
    $N_{\text{pred}} = N_{\text{pred}} + 1$
**until** no change in $\boldsymbol{\beta}$

**Output:**
    Optimal network weights $\boldsymbol{\beta}^{\star}$
```

C. Network Architecture

The function f is parametrized as a multi-layer convolutional neural network. The data consists of depth images with width $W = 640$ and height $H = 480$ as well as corresponding point clouds with $W \cdot H$ points. The depth images are downsampled to a resolution of 128×96 before using them as an input to the network while the point cloud dimensionality is kept the same. This downsampling has two benefits: 1) A reduction of the amount of network parameters $\boldsymbol{\beta}$; 2) The conversion from point clouds to depth images is better since there are fewer holes that might occur through scaling or rotation operations. The depth values are normalized between a value of 0 and 1, where a depth value of 0 belongs to the background and all other values to the environment. The basic structure of the network consists of 5 convolution layers where each layer is followed by a max-pooling layer. We use a rectified linear unit activation function in the convolutional layers and a kernel size of 3×3. The number of channels of the convolutional layers is chosen dependent on the model complexity. The last layer of the network is a linear layer that outputs the parameters $(\boldsymbol{\theta}, \boldsymbol{\gamma})$.

Scenario + Parameter	L^{low}	L^{up}	N_{data}	**Network architecture**	**Baseline (Train)**	**Baseline (Test)**	**KMN (Train)**	**KMN (Test)**	**ICP (Train)**	**ICP (Test)**
box A			40000	[2 4 6 8 10]	4.521e-03	4.512e-03	**1.429e-03**	**1.417e-03**	1.668e-02	1.681e-02
$\boldsymbol{\theta}$:										
x translation	-0.40	0.40			2.124e-03	2.075e-03	6.870e-04	7.007e-04	7.419e-03	7.183e-03
y translation	-0.40	0.40			2.397e-03	2.437e-03	7.417e-04	7.165e-04	9.258e-03	9.623e-03
box B			60000	[2 4 6 8 10]	7.125e-02	6.968e-02	**7.335e-03**	**7.345e-03**	6.370e-01	6.281e-01
$\boldsymbol{\theta}$:										
x translation	-0.40	0.40			9.530e-03	9.559e-03	1.590e-03	1.629e-03	1.023e-02	1.057e-02
y translation	-0.40	0.40			1.161e-02	1.129e-02	1.790e-03	1.763e-03	1.281e-02	1.332e-02
z axis rotation	-1.05	1.05			5.010e-02	4.883e-02	3.955e-03	3.953e-03	6.140e-01	6.042e-01
box C			100000	[4 8 10 12 14]	1.886e-01	1.911e-01	**4.242e-02**	**4.295e-02**	-	-
$\boldsymbol{\theta}$:										
x translation	-0.40	0.40			1.348e-02	1.345e-02	2.904e-03	2.807e-03	-	-
y translation	-0.40	0.40			1.307e-02	1.345e-02	2.835e-03	2.727e-03	-	-
z axis rotation	-1.05	1.05			8.080e-02	8.430e-02	9.675e-03	1.094e-02	-	-
length scaling	-0.40	0.40			4.213e-02	3.956e-02	1.383e-02	1.298e-02	-	-
height scaling	-0.50	1.50			3.913e-02	4.029e-02	1.318e-02	1.350e-02	-	-
door			100000	[2 4 8 16 32]	1.385e-01	1.394e-01	**5.253e-02**	**5.307e-02**	-	-
$\boldsymbol{\theta}$:										
x translation	-0.80	0.80			1.217e-02	1.238e-02	2.954e-03	2.940e-03	-	-
y translation	-0.80	0.80			8.174e-03	8.471e-03	3.236e-03	3.192e-03	-	-
z axis rotation	-1.05	1.05			1.976e-02	1.948e-02	5.275e-03	5.512e-03	-	-
$\boldsymbol{\gamma}$:										
door height	-0.40	0.20			1.504e-02	1.473e-02	1.199e-02	1.177e-02	-	-
door width	-0.20	0.20			1.662e-02	1.686e-02	4.539e-03	4.474e-03	-	-
handle y	-0.04	0.04			1.858e-02	1.903e-02	1.058e-02	1.068e-02	-	-
handle z	-0.10	0.10			4.818e-02	4.849e-02	1.395e-02	1.449e-02	-	-

Fig. 2: Results of experiment V-A.

The loss function is the mean squared error that is minimized with the Adam optimizer [20]. A python implementation of the proposed algorithm can be found in the supplementary material.

V. Experiments

The proposed approach is evaluated based on three experiments. In the first experiment, the performance is compared to alternative strategies on different tasks with varying complexity. The second experiment shows the adaption to real world sensor data and the third experiment demonstrates the transfer of a policy between different simulated door environments.

A. Evaluation of Kinematic Morphing Networks

The prediction accuracy is evaluated on two tasks:

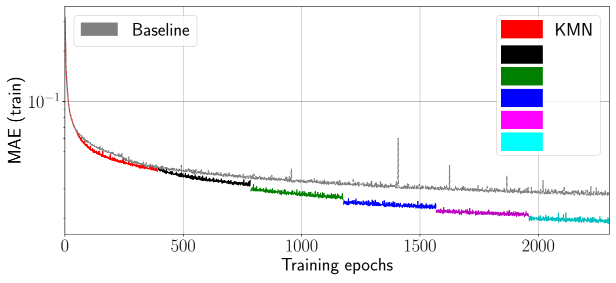
1) **box:** Three different box parametrizations are defined with varying complexity: A) $n = 2$: the box is only translated along the horizontal x and y direction; B) $n = 3$: the box is additionally rotated around the z axis; C) $n = 5$: the box is additionally scaled in its width and height.
2) **door:** This environment has $n = 3$ transformation parameters and $m = 4$ configuration parameters. The transformation parameters $\boldsymbol{\theta}$ are, similar to the box, the translation along x and y and the rotation around the z axis. The configuration parameters $\boldsymbol{\gamma}$ are the size of the door and the location of the door handle. The parametrization is sketched in Figure 1.

Figure 5 shows several depth images of both tasks. The prototype is, in both tasks, defined at $\boldsymbol{\theta}_0 = \mathbf{0}$ and $\boldsymbol{\gamma}_0 = \mathbf{0}$, which corresponds to the configuration where the object is directly in front of the robot. We compare different algorithms on both environments that predict the morphing parameters $(\boldsymbol{\theta}, \boldsymbol{\gamma})$ from depth images and point clouds. The algorithms are:

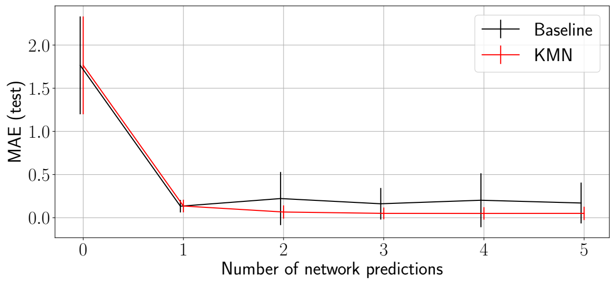
- **Kinematic Morphing Network (KMN):** This is the method proposed in this paper (see Section IV) with $N_{\text{pred}} = 5$ number of predictions.
- **Baseline:** Uses the same neural network as KMN. However, the network is only trained on the initially generated data without retraining and applied with a single prediction step ($N_{\text{pred}} = 1$).
- **Iterative Closest Point (ICP):** The iterative closest point algorithm [1] with 100 iterations.

The results of this experiment are shown in the table in Figure 2. The table lists the environment parameters, model architectures, and prediction error for all tasks and algorithms. The network architecture describes the number of channels in the 5 convolutional layers. The dataset is split into 80% train and 20% test data. The initially generated amount of data N_{data} and the network architecture is chosen heuristically according to the complexity of the task. The number of augmented data points N_{aug} is set to 20% of N_{data}. The reported error metric is the mean absolute error over 1000 data points and reported on the train and test set for each algorithm.

The results indicate that the proposed approach KMN achieves the lowest prediction errors on all tasks. The difference between KMN and Baseline comes through the iterative prediction and retraining mechanisms, since both variants use the same network architecture. Figure 3a shows the training error of the Baseline and KMN variants on the box environment C. A different color denotes a new iteration of the KMN retraining loop in Algorithm 2. The



(a)



(b)

Fig. 3: The graphs in (a) show the training error of the Baseline and KMN variant. The alternating colors denote a new retraining iteration of Algorithm 2. The plot in (b) shows the prediction error with standard deviation over multiple predictions with each network.

data augmentation of KMN leads to a faster decrease of the training error. Figure 3b shows the prediction error with standard deviation of Baseline and KMN over number of predictions on the test set. The first prediction of both networks achieves a similar error. However, the KMN method improves the prediction by applying the network multiple times. After 3 iterations there is no significant change in the accuracy anymore. The prediction error of the baseline increases since it only was trained on the initial dataset. This shows that the retraining mechanism is necessary in order to apply the network iteratively. The ICP algorithm was applied on box A and box B since they do not have any configuration parameter. ICP achieves a reasonable performance on the translation parameters of both environments. However, ICP had difficulties on the rotation parameter since it sometimes could not detect the correct rotation direction or led to rotations that flipped the box.

Figure 5 shows different samples of the dataset (top row) with the corresponding network prediction (bottom row) separated in best and worst predictions. The worst predictions occurred when the box or door were only partially observed. This makes sense since it is difficult to estimate the height of a door when it is not fully visible. The samples also show that the transformation of point clouds and the subsequent rendering can lead to holes in the depth image. However, since the KMN network also has such data points in the training phase, it could handle them better than the Baseline. The morphing transformations of the door and box task are shown in the appended video.

B. Evaluation on Real Sensor Data

In this experiment, we evaluate how the KMN model performs on real sensor data. The trained model of the box C task is applied on data recorded with a Kinect v1 camera. The point clouds are recorded with the IR depth-finding camera of the Kinect. We tried to reproduce the simulated environment in the real world (e.g., same camera pose). The point clouds are preprocessed by transforming the points from camera into world frame and removing the points that do not belong to the object. We put the box at 15 different locations inside the field of view of the camera. The network was able to predict morphing parameters for all 15 samples that transform the observed box close to the prototype. The achieved accuracy was lower and the number of network predictions N_{pred} until convergence was slightly higher in comparison to simulation data. Figure 6 shows different point clouds overlaid with their predicted box location (green box).

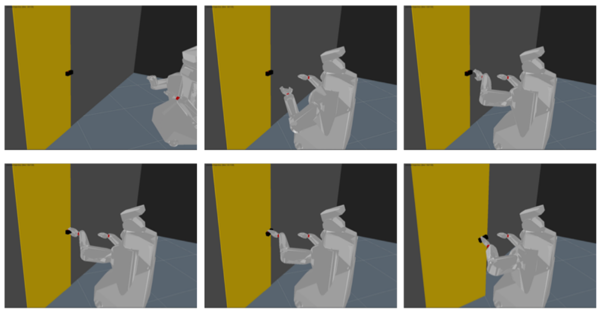


Fig. 4: Sequence of a door opening motion.

C. Skill Transfer on the Door Task

We use the trained KMN model to transfer a skill policy between different doors. The policy is defined on the kinematic model $\boldsymbol{m}(\boldsymbol{\theta}, \boldsymbol{\gamma})$ as a constrained optimization problem [21]. The robot is a PR2 and the trajectory $\bar{\boldsymbol{x}}_{0:T}$ consists of $T = 200$ configurations of the robot base (3 dof), left arm (7 dof), gripper (1 dof), and door (2 dof). The features $\boldsymbol{\Phi}$ of the cost function are defined as the base pose in front of the door, the pre-grasp pose of the gripper in front of the handle as well as the target states of the handle and door joint. The equality constraint $\boldsymbol{h}$ consists of a feature that describes the contact between door handle and gripper. Specifically, two points are defined on the door handle and on the robot gripper. The constraint measures the difference between a point pair that should be zero during the manipulation. Further constraints are defined to avoid collisions and to fix joints when they are not being manipulated. Figure 4 shows a sequence of the skill on an instance of the door environment.

VI. Conclusion

We introduced kinematic morphing networks to transfer manipulation skills between different environments. Kinematic morphing networks extract parameters from depth images and are trained on data generated with a simulator. The conversion between point clouds and depth images allows to apply the network in an iterative manner, which increases the overall accuracy. We demonstrated the network performance on real sensor data and the transfer of a skill with a motion planning method.

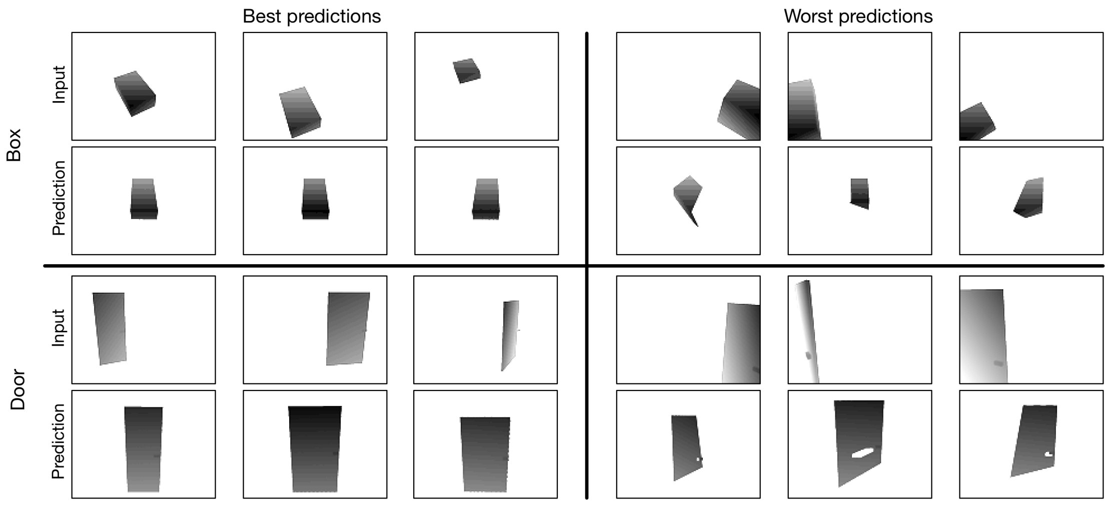


Fig. 5: In the top row are the depth image inputs and in the bottom row are the corresponding transformed depth images from the network predictions.

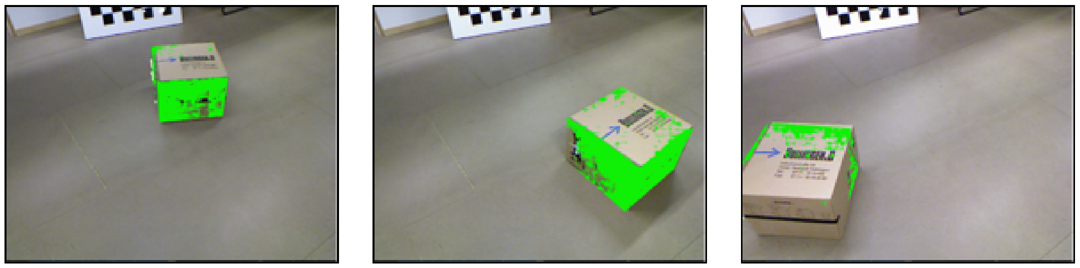


Fig. 6: Kinematic morphing network predictions (green box) overlaid with point clouds from a Kinect camera.

References

[1] Y. Chen and G. Medioni, “Object modelling by registration of multiple range images,” *Image and vision computing*, vol. 10, no. 3, pp. 145–155, 1992.

[2] S. Gold, A. Rangarajan, C.-P. Lu, S. Pappu, and E. Mjolsness, “New algorithms for 2d and 3d point matching: pose estimation and correspondence,” *Pattern Recognition*, vol. 31, no. 8, pp. 1019 – 1031, 1998.

[3] J. Sturm, C. Stachniss, and W. Burgard, “A Probabilistic Framework for Learning Kinematic Models of Articulated Objects,” *Journal of Artificial Intelligence Research*, vol. 41, pp. 477–526, 2011.

[4] R. Martin-Martin and O. Brock, “Online interactive perception of articulated objects with multi-level recursive estimation based on task-specific priors,” in *Proceedings of the International Conference on Intelligent Robots and Systems*, 2014.

[5] X. Zhou, Q. Wan, W. Zhang, X. Xue, and Y. Wei, “Model-based deep hand pose estimation,” in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2016.

[6] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, and V. Vanhoucke, “Using simulation and domain adaptation to improve efficiency of deep robotic grasping,” *arXiv:1709.07857*, 2017.

[7] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan, “Unsupervised pixel-level domain adaptation with generative adversarial networks,” in *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2017.

[8] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, “Domain-adversarial training of neural networks,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems*, 2014.

[10] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, “Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection,” in *International Symposium on Experimental Robotics*, 2016.

[11] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, “Sim-to-real robot learning from pixels with progressive nets,” in *Proceedings of the Conference on Robot Learning*, 2017.

[12] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, “Progressive neural networks,” *arXiv:1606.04671*, 2016.

[13] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *Proceedings of the International Conference on Machine Learning*, 2016.

[14] C. Mitash, K. Bekris, and A. Boularias, “A self-supervised learning system for object detection using physics simulation and multi-view pose estimation,” in *Proceedings of the International Conference on Intelligent Robots and Systems*, 2017.

[15] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard, “Multimodal deep learning for robust rgb-d object recognition,” in *International Conference on Intelligent Robots and Systems*, 2015.

[16] A. Byravan and D. Fox, “Se3-nets: Learning rigid body motion using deep neural networks,” in *Proceedings of the International Conference on Robotics and Automation*, 2017.

[17] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, “Spatial transformer networks,” in *Advances in Neural Information Processing Systems*, 2015.

[18] D. J. Rezende, S. A. Eslami, S. Mohamed, P. Battaglia, M. Jaderberg, and N. Heess, “Unsupervised learning of 3d structure from images,” in *Advances in Neural Information Processing Systems*, 2016.

[19] M. Toussaint, “A tutorial on Newton methods for constrained trajectory optimization and relations to SLAM, Gaussian Process smoothing, optimal control, and probabilistic inference,” in *Geometric and Numerical Foundations of Movements*. Springer, 2017.

[20] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *Proceedings of the International Conference on Learning Representations*, 2014.

[21] P. Englert and M. Toussaint, “Learning manipulation skills from a single demonstration,” *International Journal of Robotics Research*, vol. 37, no. 1, pp. 137–154, 2018.