Predicting Pushing Action Effects on Spatial Object Relations by Learning Internal Prediction Models

Fabian Paus, Teng Huang and Tamim Asfour

***Abstract*— Understanding the effects of actions is essential for planning and executing robot tasks. By imagining possible action consequences, a robot can choose specific action parameters to achieve desired goal states. We present an approach for parametrizing pushing actions based on learning internal prediction models. These pushing actions must fulfill constraints given by a high-level planner, e. g., after the push the brown box must be to the right of the orange box. In this work, we represent the perceived scenes as object-centric graphs and learn an internal model, which predicts object pose changes due to pushing actions. We train this internal model on a large synthetic data set, which was generated in simulation, and record a smaller data set on the real robot for evaluation. For a given scene and goal state, the robot generates a set of possible pushing action candidates by sampling the parameter space and then evaluating the candidates by internal simulation, i. e., by comparing the predicted effect resulting from the internal model with the desired effect provided by the high-level planner. In the evaluation, we show that our model achieves high prediction accuracy in scenes with a varying number of objects and, in contrast to state-of-the-art approaches, is able to generalize to scenes with more objects than seen during training. In experiments on the humanoid robot ARMAR-6, we validate the transfer from simulation and show that the learned internal model can be used to manipulate scenes into desired states effectively.**

I. Introduction

Predicting how things might be, not only now, but also in the future are a fundamental part of human cognitive abilities which allow us to choose how to act [1]. A cognitive system, which uses an internal model to reason about action consequences in the world and learns from experience, should also be able to explain why and what it is doing [2]. To this end, simulations have been used in combination with logic-based reasoning to acquire common sense knowledge [3].

A robot interacting with the world needs the ability to reason about the effects of its actions [4]. Humans can predict the effects of actions based on low-level motor control as well as high-level reasoning [5], [6]. When comparing physics engines with intuitive physics models based on probabilistic simulations of interactions between objects and their relations, humans tend to give results more consistent with the latter [7]. Taking inspiration from this idea, we want to enable a robot to predict action effects based on objects and the spatial relations between them. This results in two challenges. First, a scene can contain an arbitrary number of objects. Second, training a prediction model requires many samples, which are hard to obtain on a real robot. We address the first challenge by learning a model which is invariant to the number of objects and their order. By using a physics simulation, we address the second challenge.

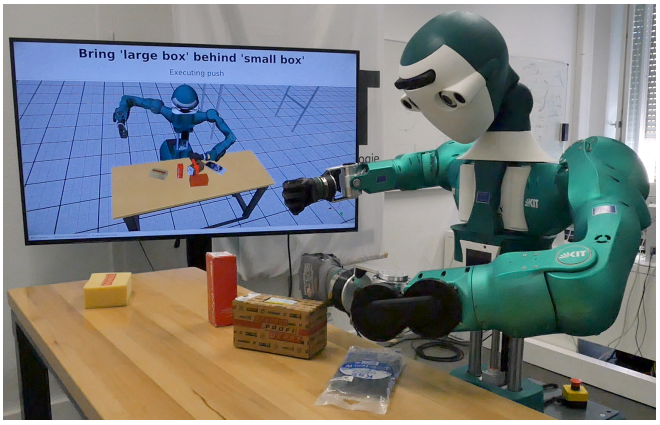


Fig. 1. The humanoid robot ARMAR-6 has the goal of moving the large brown screw box behind the small orange box. The robot samples the parameter space of possible pushes and evaluates them according to the learned internal prediction model. We choose push parameters, which are predicted to produce the desired outcome. On the monitor, the red arrow indicates the chosen push to be executed and the red box shows the predicted pose of the brown box after the push.

In this paper, we focus on non-prehensile manipulation, in particular, pushing actions. Section II discusses related work on the topic of prediction for pushing actions and compares them to our approach. In section III, we first explain the data generation in simulation and then present our approach for learning an internal prediction model. The training data for learning push effects is generated by executing pushes in randomized scenes using the physics simulation MuJoCo [24]. Each scene is represented as an attributed directed graph, in which nodes store object properties and edges contain relative spatial information between object pairs. To formulate an optimization problem on these scene graphs, we use graph networks [25], which provide machine learning building blocks with graphs as input and output. The evaluation in section IV shows a high prediction accuracy of our internal model in simulation and on real data. We demonstrate that the learned model enables goal-oriented manipulation for table-top scenes in section V by conducting experiments on the humanoid robot ARMAR-6 [26]. The paper concludes in section VI with a discussion of the results.

The research leading to these results has received funding from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) Project Number 146371743 TRR 89 Invasive Computing and from the European Unions Horizon 2020 Research and Innovation programme under grant agreement No. 731761 (IMAGINE).

The authors are with the Institute for Anthropomatics and Robotics, Karlsruhe Institute of Technology, Karlsruhe, Germany. {paus, asfour}@kit.edu

TABLE I
COMPARISON OF THE RELATED WORK WITH OUR APPROACH

Reference	Method	Explicit	Object Count	Dim.	Input	Output
Mason, 1986 [8]	Model-based	Yes	Single	3D	Push velocity	Direction of rotation
Lynch et al., 1992 [9]	Model-based	Yes	Single	2D	Push velocity	Object velocity
Hogan and Rodriguez, 2016 [10]	Model-based	Yes	Single	2D	End-effector motion	Object velocity
Zhou et al, 2018. [11]	Hybrid	Yes	Single	2D	End-effector motion	Object velocity
Kloss et al., 2017 [12]	Hybrid	Yes	Single	2D	Depth image and action vector	Object pose
Omrcen et al., 2009 [13]	Data-driven	Yes	Single	2D	Mask	Object velocity
Kopicki et al., 2011 [14]	Data-driven	Yes	Single	3D	Mask	Relative pose change
Elliott et al., 2016 [15]	Data-driven	Yes	Single	2D	Object pose and action vector	New object pose
Byravan et al., 2017 [16], [17]	Data-driven	Yes	Fixed	3D	Point cloud and action vector	Point cloud
Finn et al., 2016 [18]	Data-driven	Yes	Fixed	2D*	Image	Image
Janner et al., 2019 [19]	Data-driven	Yes	Multiple	2D*	Image	Image
Li et al., 2018 [20]	Data-driven	No	Single	2D	Masks and action vector	Similarity score
Eitel et al., 2017 [21]	Data-driven	No	Multiple	2D*	Image and action vector	Success probability
Agrawal et al., 2016 [22]	Data-driven	No	Multiple	2D*	Images	Action vector
Zeng et al., 2018 [23]	Data-driven	No	Multiple	3D	Height map	Dense expected reward
Our Approach	Data-driven	Yes	Multiple	3D	Object Poses	Object Poses

Figure 1 shows the humanoid robot ARMAR-6 in front of a table with multiple objects. The robot has the task of bringing the large brown screw box to the left of the small orange screw box. The image shows the robot executing one of the pushes required to reach the goal state.

The main contributions of this paper are as follows:

1) A data-driven method for push effect prediction with high prediction accuracy while being faster than executing a physics simulation. In contrast to state-of-the-art approaches, our method is neither limited to planar object movement nor a fixed number of objects.
2) We show that the push effect prediction enables goal-oriented manipulation tasks due to the efficient evaluation of many possible pushes and their effects.

II. RELATED WORK

Non-prehensile manipulation is an active research area [27]. Prediction of push effects has many applications including singulation of objects ([21], [23]), rearrangement of scenes ([22], [20]), pre-grasp manipulation ([13], [28]), and object segmentation ([29]). We have compiled a comparison overview (see Table I) illustrating the differences between related approaches regarding the following aspects:

- **Method**: *Model*-based approaches use an analytical physics model to describe and predict object interactions and motions. In contrast, *data*-driven approaches learn a model from training data and do not rely on an explicit physics model. If major parts of an approach require a physics model, but other parts are learned, we call it a *hybrid* method.
- **Explicit**: An *explicit* prediction model produces the state after action execution as an output, whereas an *implicit* approach learns an internal prediction model, which does not produce human-interpretable results.
- **Object Count**: Many approaches are designed to predict the motion of a *single* object. If the approach can handle more than one object but still limits the number of objects it can handle a priori, either by design or during training time, we categorize the object count as *fixed*. If the maximum amount of objects is not inherently limited, we say that such an approach can handle *multiple* objects.
- **Dim.**: We call an approach *2D* if it constrains objects to planar motion. If an approach explicitly models non-planar motions, we categorize it as *3D*. Note, that this does not require the approach to model full 6D transformations. Some approaches only process 2D images, which are nonetheless able to handle specific non-planar cases. We label such approaches *2D**.
- **Input** and **Output**: Common representations for the input and output of an approach include images (color and/or depth), object segmentation masks (short: *mask*), object poses and pose changes. We use the term *action vector* to refer to the encoding for parametrizing an action, e. g. , start point, direction, and length of a push.

A. Model-based and Hybrid Methods

Early work on analytical models for pushing has been done, predicting the object motion based on frictional forces [8], [9]. Current model-based approaches [11], [10], [30] are focused on planar pushing and predict object motion based on end-effector motion. In this work, we do not expect the physical properties required to build analytical models to be accurately known.

Hybrid approaches incorporate learning for different parts

of a physics model. In [11], the authors learn parameters of an even-degree homogeneous polynomial as a friction model. Another approach trains a convolutional neural network to estimate object poses from depth images [12]. An analytical model then uses this object position to predict the effect of a pushing action. Both model-based and hybrid methods are mostly limited to planar object motions, while we consider also non-planar object motions.

B. Data-driven Methods

Early data-driven approaches predict the motion of a single object using binary segmentation masks of the target object as input [13], [14]. Models can support a finite but fixed amount of objects, e. g. , by training a model which uses a constant amount of image masks [16], [18]. By working only on images, the inherent limitations of a fixed number of objects can be circumvented [19], [21], [22]. While these approaches are able to learn efficient models for predicting effects on images, including non-planar object interactions, they are still limited to the perceived 2D image plane. Zeng et al. use a height map with color information as input and can, therefore, incorporate depth as well [23]. Our approach considers scenes with multiple objects and does not limit the number of involved objects inherently. In contrast to the related work, we do not use images or height maps as scene representation. Instead, we represent a scene as a graph, whose nodes contain object properties.

Recent works have applied implicit models effectively to determine action parameters using start and goal state as input. Discriminative approaches rate the fitness of a specific action vector to transform a given start state into the desired goal state. An action then can be chosen by sampling and maximizing this score [20], [21], [23]. Contrary, a regression approach predicts the action vector itself given start and goal state [22]. While these approaches have been proven to achieve their goals successfully, they lack explainability. An explicit model has the advantage that a human can interpret the model's prediction. We believe that the ability to explain why the robot took a certain action is crucial for reasoning about action effects and failure analysis. Therefore, our approach uses an explicit prediction model.

Janner et al. learn an object-centric physics model [19]. Their work is similar to ours, in that they train their model using a physics simulation. They take as input a rendered 2D image of blocks, where one block is suspended in the air. Then, they predict how the image will look after the block has fallen. One difference is the representation of input and output since they use images and we use object properties directly. Furthermore, they only consider falling blocks, and no action by a robot is involved. In their experiment, the robot lets a block fall onto other blocks, mimicking the simulations. Instead, our approach considers where and how the robot pushes into the scene.

III. Push Effect Prediction

Given a scene as a set of objects $\mathcal{O}$, the goal of our approach is to predict pose changes for these objects caused by a pushing action. We represent an object $o = (\mathbf{t}, R, \mathbf{s}) \in \mathcal{O}$ as a tuple consisting of global position $\mathbf{t} \in \mathbb{R}^3$, global rotation $R \in \mathbb{R}^{3\times3}$, and oriented bounding box size $\mathbf{s} \in \mathbb{R}^3$. A pushing action $a = (\mathbf{d}, \mathbf{e}) \in \mathcal{A}$ is composed of a direction $\mathbf{d} \in \mathbb{R}^3$ and an endpoint $\mathbf{e} \in \mathbb{R}^3$ which specifies where the end-effector stops after executing the action. The goal is to learn a prediction model M which, given an initial scene $\mathcal{O}_{\text{before}}$ and a push action $a \in \mathcal{A}$, outputs the scene $\mathcal{O}_{\text{after}}$ after executing the push.

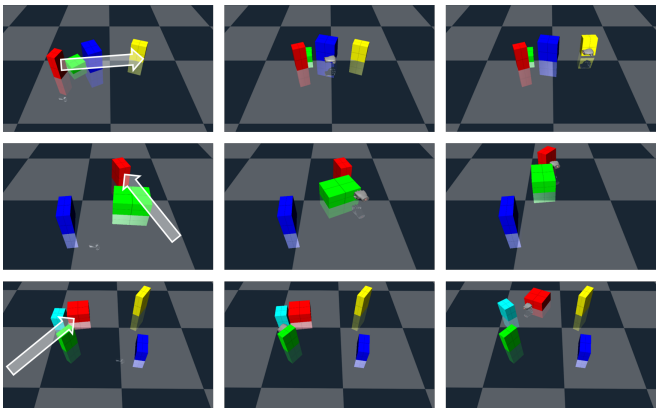


Fig. 2. Examples of randomized scenes in simulation. Each row represents a push executed in a generated scene. The first column shows the initial scene state and the sampled push direction as a white arrow. The second column shows the scene during the execution of the push action. In the third column, the final scene state after the push is shown.

$$\mathcal{O}_{\text{after}} = M(\mathcal{O}_{\text{before}}, a)$$

We approach the problem by generating randomized scenes and executing random pushes in simulation. After preprocessing the training data to a local representation, we employ graph networks [25] as a learning model.

A. Generating Training Data in Simulation

The MuJoCo physics simulator [24] is used to execute pushes in simulation. Figure 2 shows examples of generated scenes and pushes. The following parameters have been randomized during scene generation using a uniform distribution over all possible values:

- Object count: Scenes contain one to five objects
- Object position: The center point of each object is chosen inside a rectangular region with size $1\text{m} \times 1\text{m}$.
- Object rotation: Only the rotation around the z-axis is randomized to ensure that the objects can stand upright.
- Object size: Width, height, and depth of the boxes can be in the interval $[0.05\text{m}, 0.20\text{m}]$.

Push parameters were chosen by first selecting a target object. Then, a local offset is sampled around the target's center point, taking into account the size of the object. The endpoint of the push is chosen to be the target's center point shifted by the local offset. This way, we ensure that most pushes are executed close to objects where relevant interactions are happening. Then, the push direction is sampled, allowing only pushes parallel to the ground.

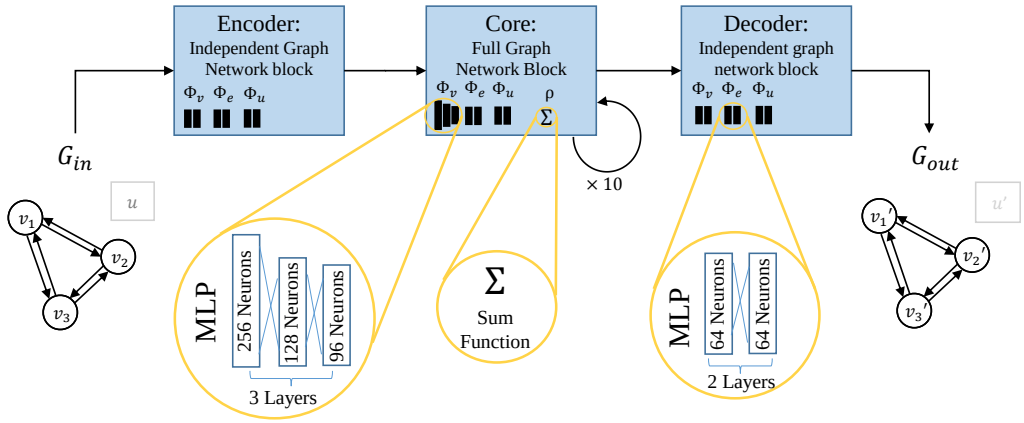


Fig. 3. The architecture of the prediction model is an encode-process-decode graph network. The input graph G_{in} is encoded into a latent space using a graph independent block, i. e. , nodes, edges and global attributes are transformed individually. We execute a full graph network block ten times as the core processing step of our model. The latent representation is then transformed into the output graph G_{out} using a graph independent block as a decoder.

During data generation, we first load the model of the ARMAR-6 hand. As a second step, we generate a random scene as described above. In this scene, we execute 200 random pushes, after which we proceed with the next scene. The object poses before and after the push, as well as the push parameters, are saved to train the prediction model.

B. Preprocessing the Training Data

Since object poses and push directions depend on an arbitrarily chosen global coordinate system, we first transform the scene data to a local representation. First, the push endpoint $\mathbf{e} \in \mathbb{R}^3$ becomes the origin of the new local coordinate frame. Second, we rotate the scene so that the push direction $\mathbf{d} \in \mathbb{R}^3$ is aligned with the positive x-axis of the local coordinate system while keeping the z-axis pointing upwards. Since the local origin is the push endpoint and the local x-axis is the push direction, no explicit push parameters are needed during training.

To use a graph network model [25], the input and output data need to be represented as a directed attributed multi-graph $G = (V, E, \mathbf{u})$, where V is a set of vertices, E a set of edges, and $\mathbf{u}$ a global feature vector. After transforming the scene to the local coordinate system, we encode the object features as a stacked vector of position $\mathbf{t} \in \mathbb{R}^3$, orientation $R \in \mathbb{R}^{3\times3}$, and size $\mathbf{s} \in \mathbb{R}^3$, resulting in a node feature vector $\mathbf{v} \in \mathbb{R}^{15}$. We found that representing the orientation as a redundant 3×3 matrix gives better results when compared to quaternions or Euler angles. Additionally, we create edges between each node pair using the relative position difference as the feature vector $\mathbf{f} \in \mathbb{R}^3$. A global feature vector $\mathbf{u}$ is not needed, as discussed above.

C. Learning a Prediction Model using Graph Networks

We chose an encode-process-decode structure for the graph network (see Figure 3). First, the input graph G_{in} is encoded where nodes, edges, and globals are expanded into a latent representation. The process step consists of a full graph network block which processes the latent representation 10 times. The process block uses Multi-layer perceptrons (MLPs) with batch normalization as update functions:

- Node update Φ_v: 3-layer MLP with sizes $[256, 192, 96]$
- Edge update Φ_e: 2-layer MLP with sizes $[128, 96]$
- Global update Φ_u: 2-layer MLP with sizes $[64, 64]$

We choose element-wise sum for all aggregation functions $\rho_{e\to v}$, $\rho_{v\to u}$, and $\rho_{e\to u}$. The encoder and decoder both use a graph independent block with two-layer MLPs of size $[64, 64]$ for all update functions. All the MLPs contain parameters, which need to be optimized during training. We summarize them into the model parameter vector θ to define a parametrized prediction model M_θ. As the loss function $\mathcal{L}$, we use the mean squared error over the node attributes $\mathbf{v} \in \mathbb{R}^{15}$, excluding the bounding box size $\mathbf{s}$, since it cannot change during a push, and edge attributes $\mathbf{f} \in \mathbb{R}^3$ from the predicted and ground truth scene.

The goal is to find a model parameter vector θ which minimizes the loss over all input and ground-truth output graphs, G_{in} and G_{GT}: $\arg\min_\theta \mathcal{L}(M_\theta(G_{in}), G_{GT})$

The dataset of 2,000,000 simulated pushes is split into training (70%), validation (20%), and test set (10%). During training, the model is optimized based on the training set using a batch size of 256. We use the Adam optimizer with a learning rate of 0.001. The layer sizes have been determined by a hyperparameter search using the validation set.

IV. Evaluation

First, we evaluate the performance of our approach quantitatively by looking at the position and orientation accuracy, combinatorial generalization, and computation time in simulation. Then, we presents results for evaluation experiments on the real robot.

A. Results in Simulation

We evaluate the accuracy of our learned model in simulation by comparing position and orientation errors in the training, validation, and test set. Table III shows the average position and orientation errors (Mean) as well as the standard

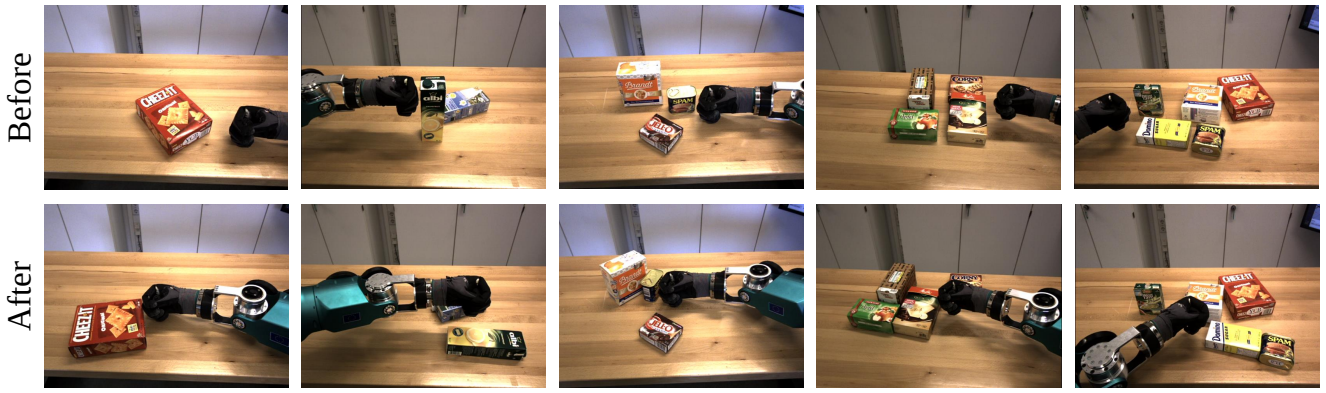


Fig. 4. Scenes from the data set recorded on the real robot. The top row contains images from the robot's cameras right before the push action begins. The bottom row shows images after the push has been executed. Each row shows a scene with an increasing number of objects.

TABLE II

DATA SETS: POSITION AND ORIENTATION CHANGES

	Position Change in $[cm]$		**Orientation Change in $[°]$**	
	Mean	**Stddev.**	**Mean**	**Stddev.**
Simulation	8.45	15.92	18.61	34.69
Real	7.90	11.67	19.18	31.30

TABLE III

POSITION AND ORIENTATION ERROR OF THE LEARNED MODEL

	Position Error in $[cm]$		**Orientation Error in $[°]$**	
	Mean	**Stddev.**	**Mean**	**Stddev.**
Training	0.83	0.57	4.21	6.54
Validation	0.86	0.61	5.35	7.77
Test	0.87	0.60	5.24	7.60
Real	1.84	2.66	14.82	23.83

deviation (Stddev.). The mean position and orientation changes in the data set are included in table II. Our model achieves an average position error of less than one centimeter and an orientation error of around $5°$.

Furthermore, we investigated the ability of our model to cope with an unseen number of objects, i. e. , combinatorial generalization. We trained a model only on scenes containing $2-3$ objects and tested the prediction errors in scenes with 1, 4 and 5 objects. In Table IV, we can see that scenes with fewer objects are no problem. Also, scenes with 4 or 5 objects only incur a minor increase in position and orientation error. The whole data set was used for this evaluation.

Regarding the runtime, we compare our approximate model with the full physics simulation. Executing a single push in MuJoCo takes on average 2.27s (with rendering disabled). Processing a batch of 1000 scenes with our prediction model takes around 1.16s. We conclude that our model is much faster than using the full physics simulation. All measurements were done on an Intel Core i7-5820K CPU with 3.30GHz. This runtime efficiency enables us to test many possible pushes and find push parameters which achieve a certain manipulation goal efficiently.

TABLE IV

PERFORMANCE ON AN UNSEEN NUMBER OF OBJECTS

Number of Objects	**Position Error in $[cm]$**		**Orientation Error in $[°]$**	
	Mean	**Stddev.**	**Mean**	**Stddev.**
$2-3$	0.79	0.48	4.17	5.90
1	0.76	0.45	4.02	5.87
4	0.91	0.75	7.03	8.05
5	0.98	0.81	7.61	8.46

B. Results on the Robot

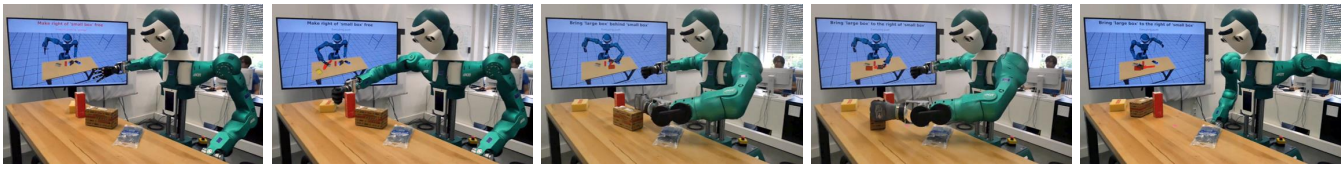
In order to evaluate the performance of our model on real data, we execute 185 pushes in 20 different scenes on the robot ARMAR-6. The involved objects were taken from the KIT and YCB object sets [31], [32]. For ground truth data generation, we localize the involved objects before and after execution and store their poses and the push parameters accordingly. Table II contains the variance in position and orientation changes for the real data set as well the simulation data set. The recorded scenes contain between one and five objects. Figure 4 shows camera images from the recorded data set.

Table III compares the results for our model, trained in simulation, and applied on the real data set to the simulation results. For the position, we still achieve a small prediction error less than two centimeters. However, the orientation prediction is considerably worse than in simulation indicating the need for further work on the transfer to the real world.

Instead of modeling the complete physics involved, our goal was to build an internal model based on intuitive physics. Although there are some cases, in which the prediction model makes minor mistakes, it is still able to predict plausible outcomes of the push actions. Thus, we will demonstrate that this push prediction system can be used on a robot to estimate action effects in the next section.

V. EXPERIMENT

We tested our push effect prediction in a goal-oriented manipulation task. The robot ARMAR-6 has the task to move



Initial Scene | Pushing the Sponge away from the Small Box | Pushing the Large Box behind the Small Box | Pushing the Large Box to the right of the Small Box | Goal Scene

Fig. 5. ARMAR-6 has the goal of bringing the large brown box to the right of the small red box. The leftmost image shows the initial scene state while the rightmost image shows the final scene state after executing three pushes. Execution of each push is shown in the middle images. The robot chooses the right arm for the first push the left arm for the two following pushes. The monitor shows the world state and the predicted push effects as transparent boxes.

the large brown screw box to the right of the small orange screw box. In order to fulfill this task, the robot first pushes the sponge away, so that there is a free space next to the small box. Then the large box is moved to the left of the small box in two pushes. The robot derives this high-level plan from the goal description, but still needs the concrete push parameters (direction and endpoint) for each action. We sample from the possible push parameters and run our internal prediction model, to predict how the push will affect the current scene. The robot chooses one of the sampled push parameters which produces the desired effect without affecting other objects. Figure 5 shows the different steps during the execution of the experiment. We recommend watching the video of the full experiment attached to this paper and available online (`https://youtu.be/_eEQnEUJrYY`).

A. Integration into the Software Architecture

Our internal prediction model needs object poses and bounding boxes. Therefore, we localize objects in a scene based on 6D pose estimation for known objects [33] or by fitting geometric primitives to unknown objects [34]. Object poses along with the robot pose are tracked in the working memory of the robot. In the first step, our method transforms the input objects into the local coordinate system and builds a graph from them as described in subsection III-B. The next step uses the internal model to predict an output graph, from which the local object poses after the push can be extracted. The orientation matrices produced by the network are not necessarily valid. Therefore, we find the nearest orthogonal matrix by using singular value decomposition.

To achieve a certain goal scene configuration, we sample push parameters by generating push endpoints in the vicinity of the target object's center. The direction is chosen as a random rotation around the global z-axis. We sample 200 push parameters and evaluate for which of the pushes the internal model predicts the desired scene state. Once the push parameters have been chosen, the robot needs to execute the specified push.

B. Push Execution

The humanoid robot ARMAR-6 has two 8-DoF arms which we use for push execution. We will use a cartesian controller, which utilizes the two-dimensional nullspace to avoid joint limits. Given the push parameters as endpoint $\mathbf{e} \in \mathbb{R}^3$ and direction $\mathbf{d} \in \mathbb{R}^3$, we want to select which arm to use and calculate a cartesian push trajectory. The trajectory begins at a defined initial pose, goes through a start point 10cm before the object, pushes along the direction $\mathbf{d}$ until the endpoint $\mathbf{e}$ is reached and the reverses the trajectory until it arrives at the initial pose again. Since not all trajectories are executable with arbitrary hand orientation, we generate trajectories with hand rotations around the z-axis in the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Now, we can evaluate the minimal distance $\Delta q_{\min}$ to the joint limits for each generated trajectory $t \in T$ and each arm $side \in \{\text{left}, \text{right}\}$: $\Delta q_{\min}(t, side) = \text{minDistanceToLimits}(t, side)$. We choose the trajectory t^* and the arm $side^*$ which maximize $\Delta q_{\min}$:

$$t^*, side^* = \arg\max_{t \in T, side \in \{\text{left,right}\}} (\Delta q_{\min}(t, side))$$

Finally, the chosen trajectory t^* will be executed by the cartesian controller for the arm $side^*$. Figure 6 shows an example optimization, where a trajectory with the left arm and a rotation of around $-1.396\text{rad} \approx -80°$ will be selected.

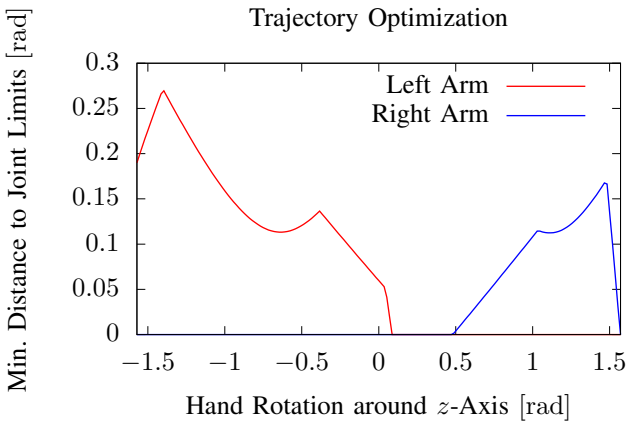


Fig. 6. Trajectory optimization for both arms showing the minimal distance to joint limits for varying hand orientations.

VI. Conclusion

In this work, we presented an approach to predict push action effects in the form of object pose changes in the scene. Our approach achieves efficient and plausible predictions enabling internal simulation for goal-oriented manipulation tasks. Furthermore, with the use of graph networks, we learn a model that achieves combinatorial generalization, i. e. , it does not rely on a fixed number of objects or their order.

REFERENCES

[1] A. Berthoz, *The brain's sense of movement*, vol. 10. Harvard University Press, 2000.

[2] R. J. Brachman, "Systems that know what they're doing," *IEEE Intelligent Systems*, vol. 17, no. 6, pp. 67–71, 2002.

[3] B. Johnston and M.-A. Williams, "Comirit: Commonsense reasoning by integrating simulation and logic," *Frontiers in Artificial Intelligence and Applications*, vol. 171, p. 200, 2008.

[4] N. Krüger, C. Geib, J. Piater, R. Petrick, M. Steedman, F. Wörgötter, A. Ude, T. Asfour, D. Kraft, D. Omrčen, A. Agostini, and R. Dillmann, "Object-action complexes: Grounded abstractions of sensorimotor processes," *Robotics and Autonomous Systems*, vol. 59, pp. 740–757, 2011.

[5] J. Moore and P. Haggard, "Awareness of action: Inference and prediction," *Consciousness and Cognition*, vol. 17, no. 1, pp. 136 – 144, 2008.

[6] A. Sato, "Both motor prediction and conceptual congruency between preview and action-effect contribute to explicit judgment of agency," *Cognition*, vol. 110, no. 1, pp. 74 – 83, 2009.

[7] P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum, "Simulation as an engine of physical scene understanding," *Proceedings of the National Academy of Sciences*, vol. 110, no. 45, pp. 18327–18332, 2013.

[8] M. T. Mason, "Mechanics and planning of manipulator pushing operations," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 53–71, 1986.

[9] K. M. Lynch, H. Maekawa, and K. Tanie, "Manipulation and active sensing by pushing using tactile feedback.," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 1, 1992.

[10] F. R. Hogan and A. Rodriguez, "Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics," *arXiv preprint arXiv:1611.08268*, 2016.

[11] J. Zhou, M. T. Mason, R. Paolini, and D. Bagnell, "A convex polynomial model for planar sliding mechanics: theory, application, and experimental validation," *The International Journal of Robotics Research*, vol. 37, no. 2-3, pp. 249–265, 2018.

[12] A. Kloss, S. Schaal, and J. Bohg, "Combining learned and analytical models for predicting action effects," *CoRR*, vol. abs/1710.04102, 2017.

[13] D. Omrčen, C. Böge, T. Asfour, A. Ude, and R. Dillmann, "Autonomous acquisition of pushing actions to support object grasping with a humanoid robot," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, (Paris, France), pp. 277–283, 2009.

[14] M. Kopicki, S. Zurek, R. Stolkin, T. Mörwald, and J. Wyatt, "Learning to predict how rigid objects behave under simple manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5722–5729, 2011.

[15] S. Elliott, M. Valente, and M. Cakmak, "Making Objects Graspable in Confined Environments through Push and Pull Manipulation with a Tool," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4851–4858, May 2016.

[16] A. Byravan and D. Fox, "SE3-nets: Learning rigid body motion using deep neural networks," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 173–180, 2017.

[17] A. Byravan, F. Leeb, F. Meier, and D. Fox, "SE3-pose-nets: Structured deep dynamics models for visuomotor planning and control," *CoRR*, vol. abs/1710.00489, 2017.

[18] C. Finn, I. Goodfellow, and S. Levine, "Unsupervised learning for physical interaction through video prediction," in *Advances in neural information processing systems*, pp. 64–72, 2016.

[19] M. Janner, S. Levine, W. T. Freeman, J. B. Tenenbaum, C. Finn, and J. Wu, "Reasoning about physical interactions with object-centric models," in *International Conference on Learning Representations*, pp. 1–12, 2019.

[20] J. Li, W. Sun Lee, and D. Hsu, "Push-net: Deep planar pushing for objects with unknown physical properties," in *Robotics: Science and Systems XIV*, pp. 1–9, Robotics: Science and Systems Foundation, 2018.

[21] A. Eitel, N. Hauff, and W. Burgard, "Learning to singulate objects using a push proposal network," in *International Syposium on Robotics Research (ISRR)*, (Puerto Varas, Chile), pp. 1–15, 2017.

[22] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to Poke by Poking: Experiential Learning of Intuitive Physics," in *Advances in Neural Information Processing Systems 29*, pp. 5074–5082, 2016.

[23] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning Synergies Between Pushing and Grasping with Self-Supervised Deep Reinforcement Learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4238–4245, 2018.

[24] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026–5033, 2012.

[25] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *CoRR*, vol. abs/1806.01261, 2018.

[26] T. Asfour, M. Wächter, L. Kaul, S. Rader, P. Weiner, S. Ottenhaus, R. Grimm, Y. Zhou, M. Grotz, and F. Paus, "Armar-6: A high-performance humanoid for human-robot collaboration in real world scenarios," *IEEE Robotics & Automation Magazine*, vol. 26, no. 4, pp. 108–121, 2019.

[27] F. Ruggiero, V. Lippiello, and B. Siciliano, "Nonprehensile Dynamic Manipulation: A Survey," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 3, pp. 1711–1718, 2018.

[28] D. Kappler, L. Y. Chang, N. S. Pollard, T. Asfour, and R. Dillmann, "Templates for pre-grasp sliding interactions," *Robotics and Autonomous Systems*, vol. 60, pp. 411–423, Mar. 2012.

[29] H. Van Hoof, O. Kroemer, H. B. Amor, and J. Peters, "Maximally informative interaction learning for scene exploration," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5152–5158, IEEE, 2012.

[30] K.-T. Yu, M. Bauza, N. Fazeli, and A. Rodriguez, "More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 30–37, IEEE, 2016.

[31] A. Kasper, Z. Xue, and R. Dillmann, "The KIT object models database: An object model database for object recognition, localization and manipulation in service robotics," *The International Journal of Robotics Research*, vol. 31, no. 8, pp. 927–934, 2012.

[32] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The YCB object and model set: Towards common benchmarks for manipulation research," in *International Conference on Advanced Robotics (ICAR)*, pp. 510–517, 2015.

[33] P. Azad, T. Asfour, and R. Dillmann, "Accurate shape-based 6-dof pose estimation of single-colored objects," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2690–2695, Oct 2009.

[34] R. Kartmann, F. Paus, M. Grotz, and T. Asfour, "Extraction of physically plausible support relations to predict and validate manipulation action effects," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 4, pp. 3991–3998, 2018.