Combining learned and analytical models for predicting action effects

Alina Kloss[1], Stefan Schaal[1,2] and Jeannette Bohg[3]

***Abstract*— One of the most basic skills a robot should possess is predicting the effect of physical interactions with objects in the environment. This enables optimal action selection to reach a certain goal state. Traditionally, these dynamics are described by physics-based analytical models, which may however be very hard to find for complex problems. More recently, we have seen learning approaches that can predict the effect of more complex physical interactions directly from sensory input. However, it is an open question how far these models generalize beyond their training data. In this work, we analyse how analytical and learned models can be combined to leverage the best of both worlds. As physical interaction task, we use planar pushing, for which there exists a well-known analytical model *and* a large real-world dataset. We propose to use a neural network to convert the raw sensory data into a suitable representation that can be consumed by the analytical model and compare this approach to using neural networks for both, perception and prediction. Our results show that the combined method outperforms the purely learned version in terms of accuracy and generalization to push actions not seen during training. It also performs comparable to the analytical model applied on ground truth input values, despite using raw sensory data as input.**

I. Introduction

We approach the problem of predicting the consequences of physical interaction with objects in the environment. Traditionally, these dynamics are described by a physics-based analytical model [1, 2, 3]. This approach has the advantage that the underlying function and the input parameters have physical meaning and can therefore be transferred to problems with variations of these parameters. They also make the underlying assumptions in the model transparent. However, defining such models for complex scenarios and extracting the required input values from raw sensory data may be very hard.

More recently, we have seen approaches that simultaneously learn a representation of the sensory input data and the associated dynamics from large amounts of training data, e.g. [4, 5, 6, 7]. They have shown impressive results in predicting the effect of physical interactions. In [5], the authors argue that a network may benefit from the freedom to choose its own representation of the raw input data. The underlying function of the dynamics and input variables are however not intuitively understandable and cannot be mapped to physical quantities. Therefore it is unclear how these models could be transferred to similar problems. Additionally, neural networks often have the capacity to memorize their training data [8] and basically learn a mapping from inputs to outputs instead of the underlying function. This behaviour can make perfect sense if the training data covers the whole problem domain. However, when training data is sparse (e.g. because a robot learns by experimenting), the question of how to generalize beyond the training data becomes more important.

Our hypothesis is that using prior knowledge from existing physics-based models can provide a way to ensure good generalization. In this paper, we thus investigate using neural networks for extracting a suitable representation from raw sensory data that can then be consumed by an analytical model for prediction. We compare this combined approach to using a neural network for both perception and prediction.

As example physical interaction task, we choose planar pushing. For this task, a well-known physical model [2] is available as well as a large, real-world data set [1] that we augmented with simulated images. Given a depth image of a tabletop scene with one object and the starting position and movement of the pusher, our models need to predict the object's position in the given image and its movement due to the push. We chose this problem, because on the one hand the state-space of the object is rather low-dimensional (2D position plus orientation). On the other hand, pushing is already a quite involved manipulation problem: The system is under-actuated and the relationship between the movement of the pusher and the object's movement is highly non-linear. The pusher can for example slide along the object and the dynamics display sharp changes when it transitions between sticking and sliding-contact or makes and breaks contact.

Our experiments show that using the analytical model not only speeds up training but also that the combination of neural networks and analytical models outperforms a learned dynamics model in terms of accuracy and generalizability to different pushing velocities. Despite having to extract the input values to the model from raw sensory data, the combined approach almost reaches the performance of the analytical model applied to the ground truth parameters.

In summary, the contributions of this paper are: (i) We combine a neural network for perception with an analytical model of planar pushing and train it end-to-end. (ii) We demonstrate the advantages of this approach over using a neural network for learning both, perception and prediction. (iii) We augment an existing dataset of planar pushing with depth and rgb images and annotate it with additional

[1] Autonomous Motion Department at the MPI for Intelligent Systems, Germany Email: first.lastname@tue.mpg.de

[2] Computational Learning and Motor Control lab at the University of Southern California, USA

[3] Department of Computer Science, Stanford University, USA

This research was supported in part by National Science Foundation grants IIS-1205249, IIS-1017134, EECS-0926052, the Office of Naval Research, the Okawa Foundation, German Research Foundation and the Max-Planck-Society. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding organizations.

information. This data set will be publicly available.

II. Related Work

Many recent approaches in reinforcement learning aim to solve manipulation problems using neural networks. They mostly treat the so called "pixels to torque" problem where the network processes input images to extract a representation of the current state and then returns the required action to achieve a certain predefined task [9, 10].

Jonschkowski and Brock [11] argue that the state-representation learned by such methods can be improved by enforcing *robotic priors* on the extracted state. Those priors may include e.g. temporal coherence of the states or repeatability. This is an alternative way of including knowledge about the basic principles of physics in a learning approach compared to what we propose in this work.

All these methods learn a direct mapping from observations to actions and thus work without acquiring an explicit model for predicting the consequences of an action. We are interested in learning a predictive model, as it enables optimal action selection (potentially over a larger time horizon) to reach a certain goal state. The following related papers share this aim.

Agrawal et al. [5] consider a learning-based approach to the problem of pushing objects. Their network takes as input a pair of images (before and after a push) as well as the pushing action. After encoding the images, two different network streams attempt to (i) predict the encoding of the second image given the first encoding and the action and (ii) predict the action necessary to transform the first encoding into the second one. Simultaneously training for both tasks improves the results on action prediction. The authors do not enforce any physical models or robotic priors. As the learned models directly operate on image encodings instead of physical quantities, we cannot compare the accuracy of the forward prediction part (i) to our results.

SE3-Nets [4] process dense 3D point clouds and a force applied to one object in the scene to predict the point cloud at the next timestep. For each object in the scene, the network predicts a segmentation mask and the parameters of an SE3 transform (linear velocity, rotation angle and axis). The output is obtained by transforming all input pixels according to the transform for the object they correspond to. The resulting predictions are very sharp and the network is shown to correctly segment the objects and determine which are affected by the action. An evaluation of the generalization ability of SE3-Nets to new objects or forces was however not performed in this work.

Our own network architecture is based on this work and we use a simplified variant of SE3-Nets to compare it to our combination of neural networks and analytical model (see Sec. III-B). We however define the loss directly on the predicted movement of the object and omit predicting the next observation. Therefore, we also drop the segmentation part, as it is only required to predict the next observation. We also add an additional input to our network that consists of an image patch around the robot's end-effector.

Finn et al. [7] is similar to [4] and explores different possibilities to predicting the next frame of a sequence of actions and rgb images using recurrent neural networks.

Visual Interaction Networks [6] also take temporal information into account. Three consecutive rgb frames serve as input at each timestep. A convolutional neural network extracts a sequence of encoded states (position and velocity) for all objects. The dynamics prediction part is a recurrent network that considers pairs of objects to predict the next state of each object.

The idea of using analytical models of dynamics in combination with learning algorithms has previously been explored e.g./ in the following works: Degrave et al. [12] implement a differentiable physics engine for rigid body dynamics in Theano and demonstrate how it can be used to train a neural network controller for various tasks. The work however does not cover visual perception. Wu et al. [13] use a physics engine to infer object properties from videos using MCMC. The so obtained object properties are then used as labels to train a neural network to infer these properties from static images of the video, thereby providing better initialization for the MCMC algorithm.

III. Predicting the Effects of Pushing Actions

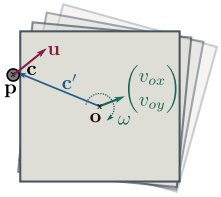
The aim of this paper is to analyse the benefits of combining learned models with analytical models. We compare to models that exclusively rely on either approach. As a test-bed, we use the process of planar pushing, for which a well-known analytical model is available as well as a real-world data set. In the next section, we introduce the analytical model, followed by the different network architectures.

A. An Analytical Model of Planar Pushing

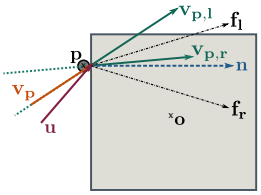
We use the analytical model of quasi-static planar pushing that was devised in [2]. It predicts the resulting object movement given the pusher velocity $\mathbf{u}$, the contact point $\mathbf{c}$ and associated surface normal $\mathbf{n}$ as well as two friction-related parameters l and μ. The problem is illustrated in Figure 1 which also contains a list of symbols. Note that this model is still approximate and far from perfectly modelling the stochastic process of planar pushing [1].

Predicting the effect of a push with this model has two stages: First, it needs to be determined whether the push is stable ("sticking contact") or whether the pusher will slide along the object ("sliding contact"). In the first case, the velocity of the object at the contact point will be the same as the velocity of the pusher. In the sliding case however, the pusher's movement can be almost orthogonal to the resulting motion at the contact point. We call the motion at the contact point "effective push velocity" $\mathbf{v_p}$, as it describes the effect that the pusher's movement has on the object. It is the output of the first stage.

Given $\mathbf{v_p}$ and the contact point, the second stage then determines the resulting translational and rotational velocity of the object $\mathbf{v}_o = [v_{ox}, v_{oy}, \omega]$.



$\mathbf{o}$	position of the object
$\mathbf{v_o}$	linear and angular object velocity
$\mathbf{v_p}$	linear velocity at the contact point - *effective push velocity*
$\mathbf{p}$	position of the pusher
$\mathbf{u}$	linear pusher velocity - *action*
$\mathbf{c}$	contact point (global)
$\mathbf{c}'$	contact point relative to $\mathbf{o}$
$\mathbf{n}$	surface normal at $\mathbf{c}$
l	ratio between maximal torsional and linear friction force
μ	friction coefficient pusher-object



$\mathbf{f}_b$	left or right boundary force of the friction cone
m_b	torques corresponding to the boundary forces
$\mathbf{v_{o,b}}$	object velocities resulting from boundary forces
$\mathbf{v_{p,b}}$	effective push velocities corresponding to the boundary forces
$b = l, r$	placeholder for left or right boundary
s	contact indicator, $s \in [0, 1]$
$\mathbf{k}$	rotation axis

Fig. 1: Overview and illustration of the terminology for pushing.

Stage 1: Determining the contact type and computing $\mathbf{v}_p$: To determine the contact type (slipping or sticking), we first have to find the left and right boundary forces $\mathbf{f_l}$, $\mathbf{f_r}$ of the friction cone (i.e. the forces for which the pusher will just not start sliding along the object) and the corresponding torques m_l, m_r. The opening angle α of the friction cone is defined by the given friction coefficient μ between the pusher and the object. These quantities are computed by

$$\alpha = \arctan(\mu) \tag{1}$$

$$\mathbf{f}_l = \mathbf{R}(-\alpha)\mathbf{n} \qquad \mathbf{f}_r = \mathbf{R}(\alpha)\mathbf{n} \tag{2}$$

$$m_l = c'_x f_{ly} - c'_y f_{lx} \qquad m_r = c'_x f_{ry} - c'_y f_{rx} \tag{3}$$

where $\mathbf{R}(\alpha)$ denotes a rotation matrix given α and $\mathbf{c}' = \mathbf{c} - \mathbf{o}$ is the contact point relative to the object's centre of mass.

To relate the forces to object velocities, [2] uses an ellipsoidal approximation to the limit surface. To simplify notation, we use subscript b to refer to quantities associated with either the left l or right r boundary forces. $\mathbf{v_{o,b}}$ and $\omega_{o,b}$ denote linear and angular object velocity, respectively. $\mathbf{v_{p,b}}$ denotes the velocities of the pusher that would create the boundary forces.

$$\mathbf{v_{o,b}} = \frac{\omega_{o,b} l^2}{m_b} \mathbf{f_b} \tag{4}$$

$$\mathbf{v_{p,b}} = \omega_{o,b} \left(\frac{l^2}{m_b} \mathbf{f_b} + \mathbf{k} \times \mathbf{c}' \right) \tag{5}$$

$\omega_{o,b}$ acts as a scaling factor. Since we are only interested in the direction of $\mathbf{v_{p,b}}$ and not in its magnitude, we set $\omega_{o,b} = m_b$, which yields

$$\mathbf{v_{p,b}} = l^2 \mathbf{f_b} + m_b \mathbf{k} \times \mathbf{c}' \tag{6}$$

To compute the effective push velocity $\mathbf{v_p}$ at the contact point, we need to determine the contact case and which boundary velocity is closer to the push velocity. If the angle between the pusher's movement and one of the boundary velocities is greater than the angle between the two boundary velocities, the push lies outside of the motion cone. The contact will slip. The resulting effective push velocity then acts in the direction of the boundary velocity $\mathbf{v_{p,b}}$ which is closer to the push direction:

$$\mathbf{v_p} = \frac{\mathbf{u} \cdot \mathbf{n}}{\mathbf{v_{p,b}} \cdot \mathbf{n}} \mathbf{v_{p,b}} \tag{7}$$

Otherwise contact is sticking and we can use the pusher's velocity as effective push velocity $\mathbf{v_p} = \mathbf{u}$. When the norm of $\mathbf{n}$ is zero (due to e.g. a wrong prediction of the perception neural network), we set the output $\mathbf{v_{p,b}}$ to zero.

Stage 2: Using $\mathbf{v_p}$ to predict the object's motion: Given the effective push velocity $\mathbf{v_p}$ and the contact point $\mathbf{c}'$ relative to the object's centre of mass, we can compute the object's linear and angular velocity $\mathbf{v_o} = [v_{ox}, v_{oy}, \omega]$.

The object will of course only move if the pusher is in contact with the object. To use the model also in cases where no force acts on the object, we introduce the contact indicator variable s. It takes values between zero and one and is multiplied with $\mathbf{v_p}$ to switch off responses when there is no contact. We allow s to be continuous instead of binary to give the model a chance to react to the pusher making or breaking contact during the interaction.

$$v_{ox} = \frac{(l^2 + c'^2_x) s v_{px} + c'_x c'_y s v_{py}}{l^2 + c'^2_x + c'^2_y} \tag{8}$$

$$v_{oy} = \frac{(l^2 + c'^2_y) s v_{py} + c'_x c'_y s v_{px}}{l^2 + c'^2_x + c'^2_y} \tag{9}$$

$$\omega = \frac{c'_x v_{oy} - c'_y v_{ox}}{l^2} \tag{10}$$

Discussion of Underlying Assumptions: The analytical model is built on three simplifying assumptions: (i) quasi-static pushing, i.e. the force applied to the object is big enough to move the object, but not to accelerate it (ii) the pressure distribution of the object on the surface is uniform and the limit-surface of frictional forces can be approximated by an ellipsoid (iii) the friction coefficient between surface and object is constant.

The analysis performed in [1] shows that assumption (ii) and (iii) are violated frequently by real world data. Assumption (i) holds for push velocities below 50 $\frac{mm}{s}$. In addition, the contact situation may change during pushing (as the pusher may slide along the object and even lose contact), such that the model's predictions become increasingly inaccurate the longer ahead it needs to predict in one step.

B. Combining Neural Networks and Analytical Models

We now introduce the three network variants that we will analyse in this paper. All architectures feature a first network stage that processes raw input depth images and produces a lower-dimensional encoding of them. Given this encoding and a pushing action (the pusher's movement $\mathbf{u}$ and its position $\mathbf{p}$), the second part of these models predicts the initial object position $\mathbf{o}$ and its linear and angular

velocity $\mathbf{v}_o$[1]. This predictive part differs for the three network variants. While two of them (*simple, full*) use variants of the analytical dynamics model established above, the third (*neural*) has to learn the dynamics with a neural network. In all three variants, the prediction part has about 7 million trainable parameters. The model-based variants have slightly more parameters than *neural*, but also predict more output values.

We implement all our networks as well as the analytical model in tensorflow [14], which allows us to propagate gradients through the analytical models just like any other neural network layer.

1) Perception: The architecture of the network part that processes the image is depicted in Figure 2. The image is processed by five layers of convolution with ReLu non-linearity, each followed by max-pooling and batch normalization [15]. The output of this part is the flattened feature-maps from the last convolutional layer.

A second stream exploits that a robot usually knows where its end-effector is. Given this knowledge, we extract a small (80×80 pixel) image patch ("glimpse") around the tip, that is processed by three convolutional layers as described above. This helps the networks to focus on the most relevant part of the image for predicting the object velocity.

For the two model-based network variants *simple* and *full*, using the glimpse has an extra advantage: All inputs to the analytical model except for the position of the object can be extracted from the encoding of the glimpse, which has a higher resolution than the same excerpt in the encoding of the full image. It also facilitates predicting the contact point: The contact point can be predicted locally in the glimpse, yielding a prediction that is relative to the pusher. This is then made global by adding the pusher's position.

2) Prediction:

a) Neural Network only (neural): Figure 3 a) shows the prediction part of the variant *neural*, which uses a neural network to learn the dynamics of pushing. It concatenates the output from perception with the action and processes this input with three fully-connected layers before predicting the location of the object $\mathbf{o}$ and its velocity $\mathbf{v_o}$. All intermediate fully-connected layers use ReLu non-linearities. The output layers do not apply a non-linearity. Together with the perception part, this is basically a simplified version of SE3-Nets [4] without predicting the next raw observation, i.e. a depth image. Instead of an SE3 transform, our network predicts an SE2 transform for the object. In contrast to SE3-Nets, we also added an additional input (the glimpse) and do not preprocess the action before concatenation.

b) Full analytical model (full) : This network-based variant uses the complete analytical model as described in Section III-A. Several fully-connected layers are used to extract the necessary input values from the image encoding as shown in Figure 3 b). These are the object's position $\mathbf{o}$, the contact point $\mathbf{c}$, the surface normal $\mathbf{n}$ and the contact indicator s. The layer that predicts s uses a sigmoidal non-linearity to limit the predicted values to $[0, 1]$.

[1]We do not predict the object's initial orientation, since it is not relevant to describe the object's motion (in contrast to the object's position, which is the centre of its rotation).

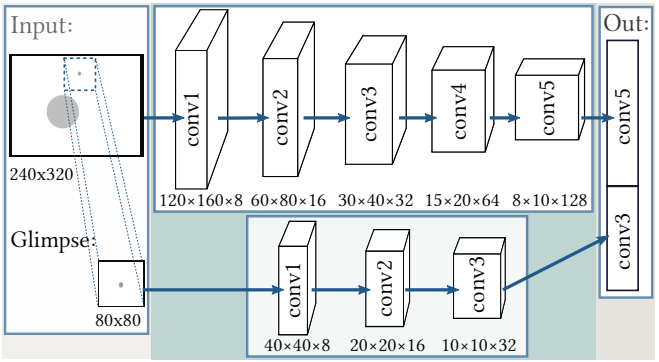


Fig. 2: Architecture of the perception part of all network variants.

c) Simplified analytical model (simple): Variant *simple* (Figure 3 c), uses only the second stage of the analytical model. As for *full*, a neural network part extracts the model inputs (effective push velocity $\mathbf{v_p}$, object position $\mathbf{o}$ and contact point $\mathbf{c}$), from the encoded observation.

We use this variant as a middle ground between the two other options: It still contains the main mechanics of how an effective push at the contact point moves the object, but leaves it to the neural network to deduce the effective push velocity from the scene and the action. This gives the model more freedom to correct for possible shortcomings of the analytical model. We expect these to manifest mostly in the first stage of the model (as small errors can have a big effect when they change whether the contact is classified as sticking or slipping). As the second stage of the analytical model does not specify how the input action relates to the object's movement, *simple* also allows us to evaluate the importance of this particular aspect of the analytical model.

3) Training: For training we use Adam optimizer [16] with a learning rate of 1^{-4} and a batch-size of 32 for 150 epochs. The loss L penalizes the Euclidean distance between the predicted and the real object position in the first image (*pos*), the Euclidean error of the predicted linear movement (*trans*), the error in the predicted magnitude of linear velocity (*mag*) and the error in predicted angular movement in degree (*rot*). Additionally, we use weight decay with $\lambda = 0.1$.

Let $\hat{\mathbf{v}}_\mathbf{o}$ and $\hat{\mathbf{o}}$ denote the predicted and $\mathbf{v_o}$, $\mathbf{o}$ the real object movement and position. w are the network weights and $\nu_\mathbf{o} = [v_{ox}, v_{oy}]$ denotes linear object velocity.

$$L(\hat{\mathbf{v}}_\mathbf{o}, \hat{\mathbf{o}}, \mathbf{v_o}, \mathbf{o}) = trans + mag + rot + pos + \lambda \sum_w \| w \|$$

$$trans = \|\hat{\nu}_\mathbf{o} - \nu_\mathbf{o}\| \qquad mag = |\|\hat{\nu}_\mathbf{o}\| - \|\nu_\mathbf{o}\||$$

$$rot = \tfrac{180}{\pi}|\omega - \hat{\omega}| \qquad pos = \| \mathbf{o} - \hat{\mathbf{o}} \|$$

Training the networks that use the analytical model is slightly harder than training a pure neural network, since the estimated parameters interact in the computation of the solution and can, to some extend, compensate for each other. Therefore, the networks can easily get stuck in local minima and careful initialization of the network weights is important.

When using the variant *full*, a major challenge is the contact indicator s: In the beginning of training, the direction of the predicted object movement is mostly wrong. s therefore

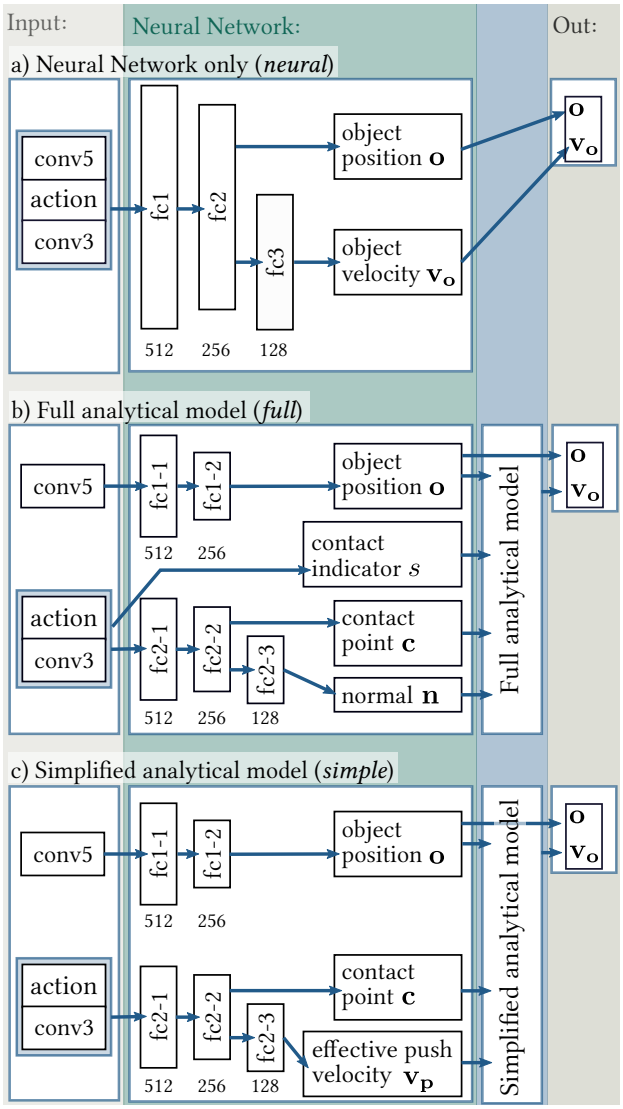


Fig. 3: Architecture of the prediction parts of the three network variants *neural*, *full* and *simple*.

receives a strong negative gradient, causing it to decrease quickly. Since the predicted object motion is effectively multiplied by s, a low s results in the other parts of the network receiving smaller gradients and thus greatly slows down training. We therefore explicitly add the error in the magnitude of the predicted velocity to the loss to prevent s from decreasing too far.

IV. Data

We use the MIT Push Dataset [1] for our experiments. It contains object pose and force recordings from real robot experiments, where eleven different planar objects are pushed on four different surface materials. For each object-surface combination, the dataset contains about 6000 examples of pushes that vary in the robot's manipulator's ("pusher") velocity and acceleration, the point on the object where the pusher makes contact and the angle between the object's surface and the pusher's movement direction. All pushes have a total length of 5 cm and data was recorded at 250 Hz.

As this dataset does not contain images or depth measurements, we render rbg and depth images using OpenGL and

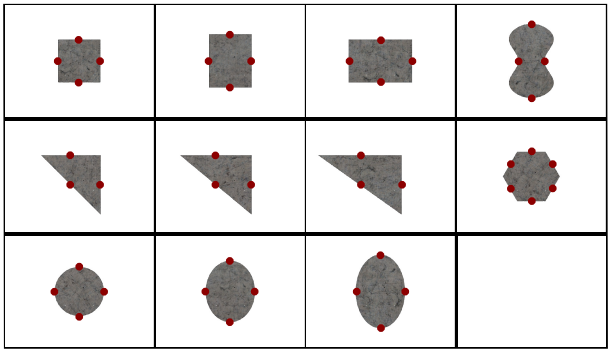


Fig. 4: Rendering of all objects of the Push Dataset [1]: *rect1-3, butter, tri1-3, hex, ellip1-3*. Red dots indicate the contact points we use to collect a testset with held-out pushes.

Fig. 5: Examples of rendered rgb and depth images on surfaces *plywood* and *delrin*.

the mesh-data supplied with the dataset. In this work, we only use the depth images, rgb will be considered in future work. A rendered scene consists of a flat surface with one of four textures (representing the four different materials in the Push Dataset), on which one of the eleven possible objects from the dataset is placed. The pusher is represented by a vertical cylinder, with no arm attached. We use a top-down view of the scene, such that the object can only move in the image plane and the z-coordinate of all scene components remains constant. This simplifies the application of the analytical model by removing the need for introducing an additional transform between the camera and the table. Figures 4 and 5 show the different objects and example images.

To evaluate the performance of the analytical pushing model, we also annotate the data with additional information: Whether the pusher is in contact with the object at each timestep, and if so, the contact point and the normal to the object's surface there. We also estimate the friction parameter l for each object-surface combination by solving the analytical model for it and averaging results over all timesteps.

We compose our own datasets for training and testing from a subset of the Push Dataset: As we use an analytical model that does not take acceleration of the pusher into account, we only work with push variants that have zero acceleration. Pushes with the highest pushing velocity ($500\frac{mm}{s}$) are also exclude because they do not last long enough for us to extract a reasonable amount of datapoints. We use all surface materials except for polyurethane, which was shown to widely violate common assumptions about friction [1]. Estimating friction is an aspect of the system that we do not address in this work, as friction can only be inferred from sequences of observations, not from static images. We therefore postpone more detailed analysis of errors due to friction to future work.

Our datasets vary in the objects they contain, included pushing angle and contact points, and in the speed of the

pusher. Section V contains more details about the datasets used for each experiment. One datapoint consists of a pair of depth images before and after a push, the object's position in the starting image and the pushers initial position and movement. The frames are 0.5 seconds apart in all datasets to ensure a notable amount of movement.

We use multiple frame-pairs from each push sequence in the Push Dataset. This results in some of the datapoints containing shorter push-motions than others, as the pusher starts moving with some delay or ends its movement during the 0.5 seconds time-window. To achieve more visual variance, we sample a number of transforms of the scene relative to the camera for each datapoint. In addition, about a third of our dataset consists of examples where we moved the pusher away from the object, such that the object is not affected by the push movement.

V. Experiments and Results

In this section, we test our hypothesis that using an analytical model for prediction together with a neural network for perception will lead to better generalization than using neural networks for perception and prediction. In detail, we evaluate how well the different architectures generalize

- to pushes with new pushing angles and contact points,
- to push velocities not used for training, and
- to unseen objects.

We also evaluated two different prediction horizons and found no significant effect on the performance of the different networks. For space reasons, we do not include this results.

A. Metrics

For evaluation, we compute the average Euclidean distance between the predicted and the ground truth object translation (*trans*) and position (*pos*) in millimetres as well as the average error on object rotation (*rot*) in degree. As our datasets differ in the overall object movement, we report errors on translation and rotation normalized by the average motion in the dataset.

B. Baselines

We use three baselines in our experiments. All of them use ground truth data as input instead of images. The first is the average translation and rotation over the dataset. This is equal to the error when always predicting zero movement, and we therefore name it *zero*.

The second baseline (*model*) is the full analytical model evaluated for the ground truth parameters for each push. In cases where the pusher makes contact with the object during the push, but is not in contact initially, we use the contact point and normal from when contact is first made and shorten the action accordingly.

In addition to the networks described in Section III, we also train a neural network (*neural dyn*) that receives the same (ground truth) input values as the *model* baseline (action, contact point, surface normal, contact indicator and friction coefficients) and predicts the object's velocity. It allows us to evaluate how well the dynamics can be learned with a neural network if perception was perfect.

As none of the baselines includes a perception part, they do not predict the object's initial position.

TABLE I: Average test set error with standard deviation for evaluation on held-out pushes: Predicted object translation (*trans*) and rotation (*rot*) are reported as percentage of the average movement in the test set, given in the last row *zero*. *pos* denotes errors on the predicted object position.

	trans	rot	pos [mm]
neural	$56.7 \pm 60.5\,\%$	$83.0 \pm 176.3\,\%$	5.95 ± 24.8
simple	$26.7 \pm 29.4\,\%$	$53.4 \pm 166.6\,\%$	5.9 ± 25.5
full	$24.1 \pm 32.4\,\%$	$41.5 \pm 161.3\,\%$	$\mathbf{5.68 \pm 25.6}$
neural dyn	$31.5 \pm 69\,\%$	$67.7 \pm 338.4\,\%$	-
model	$\mathbf{23.6 \pm 36.2}\,\%$	$\mathbf{41.1 \pm 164.3}\,\%$	-
zero	$3.42 \pm 3.45\,\mathrm{mm}$	$2.24 \pm 4.17^\circ$	-

C. Generalization to new pushing angles and contact points

In this experiment, we evaluate the performance of the three network variants *neural*, *simple* and *full* when evaluating on held-out push variants.

Data: We train the networks on a dataset that contains all objects from the MIT Push dataset (*all*) and evaluate on a testset with held-out pushes: The test set contains data from all pushes

- with pushing angles $\pm 20^\circ$ and 0° to the surface normal (independent from the contact points).
- at a set of contact points illustrated in Figure 4 (independent from the pushing angle).

The rest of the data is split randomly between training and validation set, such that there is no systematic difference between them. We use the validation split to monitor the training process. The velocity of the pusher is 20 $\frac{mm}{s}$ for all pushes in the dataset and we predict 0.5 s into the future. The training split contains 8800 datapoints, the validation split 2848 and the test set has 10752 datapoints.

Results: Table I reports the average errors in the predicted object translation, rotation and object position. Among the trained networks, the network *full* which uses the full analytical model, performs best for predicting the object's velocity for pushes not seen during training. It also has the lowest position error, but given the high standard deviation (std), differences between the models in this aspect are not significant.

The standard deviation of the predicted movement also appears rather high, even for the $model$ baseline. This can be explained by two factors: First, the prediction error scales with the magnitude of the object's movement. As the ground-truth object movement varies greatly between the datapoints in the testset (see the values given for the *zero* baseline), this results in an increased variance of the prediction errors. Second, planar pushing is not a deterministic process and the same push can lead to different object movements. This means that fitting the data perfectly in all cases is not possible for any deterministic model.

The network that uses the simplified model (*simple*) is almost as good as *full* on predicting the object's translation, but much worse for predicting the rotation. *Neural* converges slowest and never reaches the performance of the other

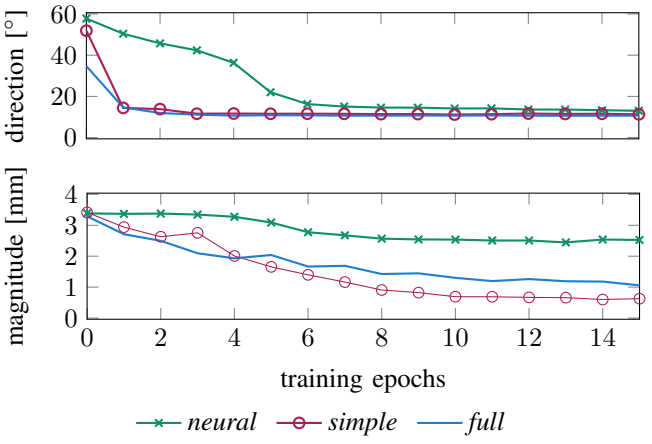


Fig. 6: Development of the magnitude and directional component of the translation error on the test-split of *all*. *Neural* has no difficulty learning to predict the direction of the object's translation but has trouble estimating the magnitude of the movement.

variants. The perception-free variant *neural dyn* performs better in terms of mean errors, but has a much higher standard deviation. It is also still worse than the model-based variants, especially for predicting rotations. As *neural dyn* receives the ground truth input values, we expected the quality of its predictions to be less variable than for the other networks, which use images as input. The high standard deviation could be explained by overfitting: The network approximates well the dynamics in the training data, but breaks down for some pushes not seen during training.

Comparing *neural* and *neural dyn* is hard, since *neural dyn* gets the ground truth parameters as input. The big difference between their performance in terms of mean values however implies that *neural* could not profit much from the freedom to pick its own state representation.

We take a closer look at the predicted object translation by splitting the error into an error on the direction of the movement (in degrees) and an error on its magnitude. Figure 6 reports the development of these errors on the testset for the first 15 training-epochs. While *neural* quickly learns to predict the translation's direction with the same accuracy as the other two variants, it struggles with the magnitude. For explaining this, we can refer the analytical model in Sec. III-A. We find that the magnitude of the movement scales with the action and also depends on the contact point's position relative to the object's centre of mass. The latter trades-off rotation and translation; the more the object rotates, the smaller its translation. Both are multiplicative relationships, which are hard to learn with fully-connected layers. This may explain why the model-based variants perform better at predicting the magnitude of the motion.

The difference between *full* and *simple* could thus be due to the simplified model not specifying the relationship between the input action and the object's motion.

Qualitative Example: We plot the predictions of our networks, the *model* baseline and the ground truth object motion for 200 repetitions of the same push configuration. All pushes have the same pushing angle ($0°$), velocity ($20\frac{mm}{s}$) and contact point, but we sample a different transformation of the whole scene for each push, such that the object's pose in the image varies between the pushes. The training data for the learned models does not contain pushes with angles similar to $0°$.

The results shown in Figure 7 illustrate that even the ground truth object motion varies greatly. As the ground truth contact point is not exactly the same for each push, the predictions by the analytical model shows some variance in the direction and length of the translation as well. In this particular example, the analytical model underestimates the rotation of the object and instead predicts a higher translation.

Neural shows the highest variance in its predictions, *full* the lowest. Interestingly, both *simple* and *full* predict more object rotation than the analytical model. As discussed before, the trade-off between translation and rotation in the analytical model is controlled by the position of the contact point relative to the object's centre of mass. Figures 7 (f)-(h) show the predicted and ground truth contact points. Both networks predict them to lie closer to the object's centre of mass in y direction, while retaining or even increasing the distance in x direction. This results in predicting a shorter translation, but higher rotation. The models have apparently learned to compensate for an inaccuracy of the analytical model. This is an advantage of training the perception networks end-to-end through the analytical model instead of on the ground truth input values to the analytical model.

D. Generalization to Different Push Velocities

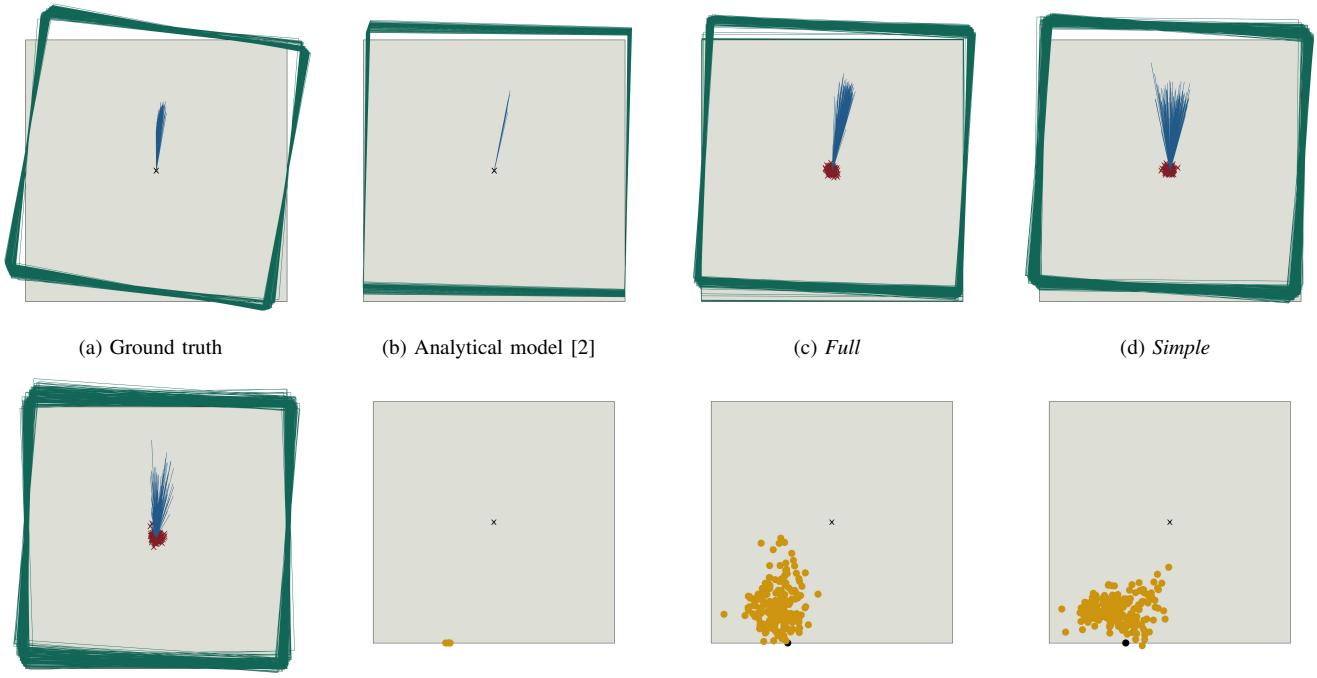
As neural networks are usually not good at extrapolating beyond their training domain, we expect that the model-based network variants will generalize better to push-velocities not seen during training.

Data: We use the networks that were trained in the previous experiment and evaluate them on five datasets with different push velocities from $10\,\frac{mm}{s}$ to $200\,\frac{mm}{s}$. To make the results comparable with the previous experiment, we only evaluate on test-splits of the datasets, that were constructed from a subset of the pushes as explained in Sec. V-C.

Results: Results are shown in Figure 8. Since the input action should (and does) not influence perception of the object's position, we only report the errors on the predicted object motion.

The performance of *full* is most constant over the different push velocities, declining only slightly more than the *model* baseline. As in the previous experiment, *neural* does not perform well in general. Both *simple* and *neural* become worse with increasing velocity, but also perform worse when predicting the object's rotation for pushes with a smaller velocity ($10\,\frac{mm}{s}$) as in the training set. The decline in the accuracy is stronger for *simple*, presumably because it performs much better than *neural* on the training velocity.

The large difference between *simple* and *full* suggests that the necessary ability for generalization to new velocities is scaling the output values according to the push velocity. As discussed before, classical neural networks cannot learn a general multiplication operation of their inputs. We suppose



(a) Ground truth (b) Analytical model [2] (c) *Full* (d) *Simple*

(e) *Neural* (f) Ground truth contact points (g) Contact points predicted by *full* (h) Contact points predicted by *simple*

Fig. 7: Qualitative evaluation on 200 repeated pushes with the same push configuration (angle, velocity, contact point). (a) - (e): The green rectangles show the (predicted) pose of the object after the push and the blue lines illustrate the object's translation (for better visibility, we upscaled the lines by factor 5). For the networks, the red crosses indicate the predicted initial object positions. Both *full* and *simple* correctly predict more object rotation than the analytical model, which indicates that the networks learn to compensate for errors of the analytical model. (f) - (h): Ground truth and predicted contact points. In (g) and (h), the black dot marks the average ground truth contact point. By altering the predicted contact point, the networks can increase the amount of rotation predicted by their internal analytical model.

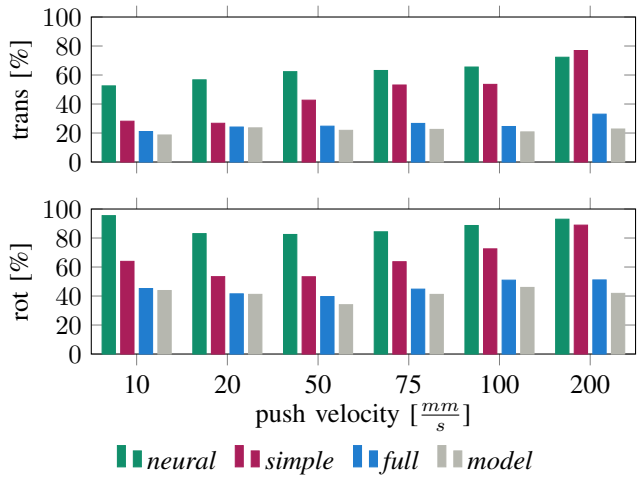


Fig. 8: Errors on predicted translation and rotation as percentage of the average movement for different push velocities. All models were trained on pushes with velocity 20 $\frac{mm}{s}$. The x-axis shows the push velocity of the different datasets used for testing. The errors of *neural* and *simple* increase with higher test velocities, whereas the performance of *full* remains more constant.

that the variants *simple* and *neural* instead learn a mapping from the actions seen during training to output, which would explain why they cannot well extrapolate to actions outside of their training data.

E. Generalization to Different Objects

In this experiment, we test how well the networks generalize to unseen object shapes and how many different objects the networks have to see during training to generalize well.

Data: We train the networks on three types of datasets: With one object (butter), two objects (butter and hex) and three objects (butter, hex and one of the ellipses or triangles). The datasets with fewer objects contain more augmented data, such that the number of datapoints is similar to data sets with more object shapes.

As testsets, we use one data set containing the three different ellipses and one containing all three triangles. The push velocity is 20 $\frac{mm}{s}$ and the prediction horizon is 0.5 seconds for all sets.

Results: The results in Figure 9 show that the model-based variants still perform comparable to the *model* baseline, even if they have only seen one object during training. Predicting the rotation of unseen objects seems to be generally harder than predicting their translation.

Neural has most trouble in predicting the object's motion and often even becomes worse when more different objects are present in the training set. This suggests that the state representation it has learned is sensitive to the shape of the test objects, whereas the representations enforced by using the analytical model are more generally applicable.

Localizing the objects in the input image generalizes rather well to the ellipses but not to triangles. This can be explained by the ellipses being similar in shape and size to the butter object, whereas the triangles are very different from all other objects in the Push Dataset. Since all architectures use roughly the same network structure to compute the position, the results do not differ significantly between them.

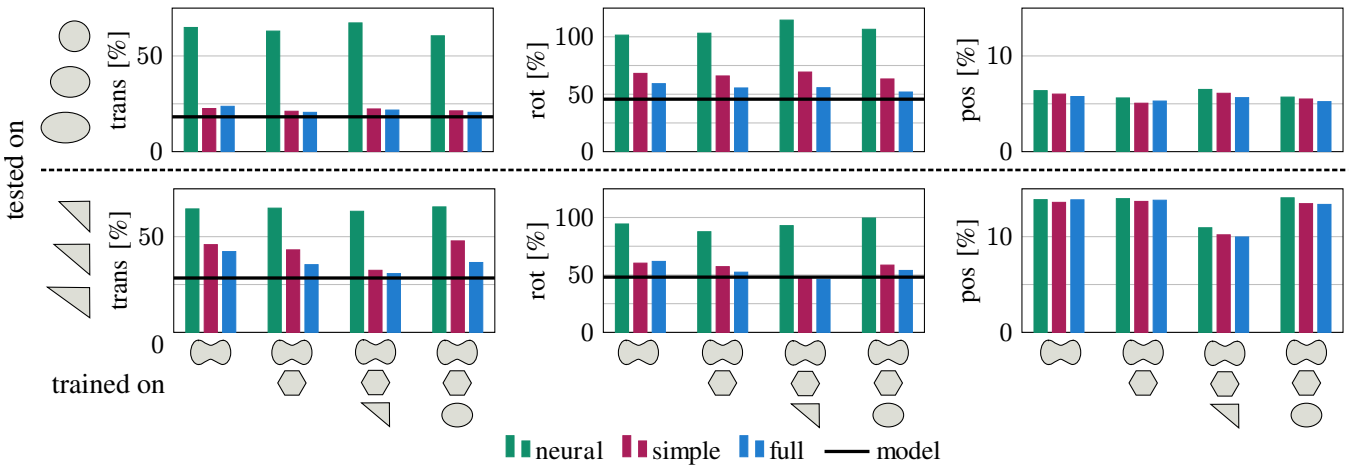


Fig. 9: Prediction results when testing on held-out objects: The x-axis shows the objects included in the training set. On the y-axis are the prediction errors in translation, rotation and object position relative to the average movement in the testset. Plots in the first row show results from evaluating on *ellipses*, the second row on *triangles*. The performance of the *model* baseline on each testset is marked by the black line. As it does not predict the position, there is no baseline for the last column. Even with only few objects in the training set, the model-based variants perform relatively well on unseen test objects. The performance of *neural* is again worse, and the network does not seem to profit from a higher number of different objects in the training dataset.

VI. Discussion and Conclusion

In this paper, we considered the problem of predicting the effect of pushing actions on a planar object on a plane surface.

Our experiments showed that combining an analytical dynamics model with a neural network for perception improves generalization and also makes learning faster as compared to a pure neural network architecture. While this might be different for other problems, our results provided no evidence that learning a state representation instead of using the one prescribed by the analytical model is beneficial. Combining learned and existing models may provide prior knowledge that helps to implicitly regularize the learned components.

A neural network might generally not be the best choice for learning dynamics models, as these often include multiplicative relationships between input values and output. Multiplication of inputs is hard to learn for classical neural networks, since multiplications are not expressible through weighted additions. Instead, the networks may have to memorize a mapping from inputs to output, which takes more network parameters the bigger the output range and the higher the desired accuracy. Avoiding to learn such operations by providing them to the network thus not only improves the results but also reduces the number of network parameters necessary to achieve a certain accuracy.

In perception on the other hand, the strengths of neural networks can be well exploited to extract the input parameters of the analytical model from raw sensory data. By training end-to-end through a given model, we can avoid the effort of labelling data with the ground truth input values for the analytical model. Our results also suggest that the networks can even learn to compensate for possible flaws in the model by adapting the predicted input values.

However, for some physical processes it may be hard to find an analytical model. Furthermore, not all existing analytical models may be suitable for our approach, as they for example need to be differentiable everywhere. Especially the switching dynamics encountered when the contact situation changes proved to be challenging and more work needs to be done in this direction. A pure neural network may also outperform analytical models when the underlying assumptions of the model are more heavily violated than in our experiments.

In the future, we would like to extend our work to more complex perception problems, like different camera viewpoints, rgb images as input or scenes with multiple objects. It would also be interesting to collect our own data, since the pushes in the MIT Push Dataset are comparably short and more variation in the appearance of objects would be desirable. Also, we would like to look at sequences instead of one-step predictions. This would allow us to enforce constraints on the temporal consistency of the estimated states, to infer latent variables like friction or to use temporal cues like optical flow as additional input.

References

[1] K. T. Yu *et al.*, "More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing," in *2016 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, Oct 2016, pp. 30–37.

[2] K. M. Lynch, H. Maekawa, and K. Tanie, "Manipulation and active sensing by pushing using tactile feedback," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, vol. 1, Jul 1992, pp. 416–421.

[3] L. Zhang and J. C. Trinkle, "The application of particle filtering to grasping acquisition with visual occlusion and tactile sensing," in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 3805–3812.

[4] A. Byravan and D. Fox, "Se3-nets: Learning rigid body motion using deep neural networks," in *Robotics and Automation (ICRA), 2017 IEEE Int. Conf. on*. IEEE, 2017, pp. 173–180.

[5] P. Agrawal *et al.*, "Learning to poke by poking: Experiential learning of intuitive physics," in *Advances in Neural Inform. Process. Syst.*, 2016, pp. 5074–5082.

[6] N. Watters *et al.*, "Visual Interaction Networks," *ArXiv e-prints*, Jun. 2017.

[7] C. Finn, I. Goodfellow, and S. Levine, "Unsupervised learning for physical interaction through video prediction," in *Advances in Neural Inform. Process. Syst. 29*, 2016, pp. 64–72.

[8] C. Zhang *et al.*, "Understanding deep learning requires rethinking generalization," *CoRR*, vol. abs/1611.03530, 2016. [Online]. Available: http://arxiv.org/abs/1611.03530

[9] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[10] S. Levine *et al.*, "End-to-end training of deep visuomotor policies," *J. Mach. Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[11] R. Jonschkowski and O. Brock, "Learning state representations with robotic priors," *Autonomous Robots*, vol. 39, no. 3, pp. 407–428, Oct 2015. [Online]. Available: https://doi.org/10.1007/s10514-015-9459-7

[12] J. Degrave *et al.*, "A differentiable physics engine for deep learning in robotics," *CoRR*, vol. abs/1611.01652, 2016. [Online]. Available: http://arxiv.org/abs/1611.01652
[13] J. Wu *et al.*, "Galileo: Perceiving physical object properties by integrating a physics engine with deep learning," in *Advances in Neural Inform. Process. Syst.*, 2015, pp. 127–135.
[14] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.
[15] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. on Machine Learning*, vol. 37, 07–09 Jul 2015, pp. 448–456. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html
[16] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.