



Physical Representation Learning and Parameter Identification from Video Using Differentiable Physics

Rama Krishna Kandukuri¹ · Jan Achterhold¹ · Michael Moeller² · Joerg Stueckler¹

Received: 25 January 2021 / Accepted: 16 June 2021 / Published online: 17 October 2021
© The Author(s) 2021

Abstract

Representation learning for video is increasingly gaining attention in the field of computer vision. For instance, video prediction models enable activity and scene forecasting or vision-based planning and control. In this article, we investigate the combination of differentiable physics and spatial transformers in a deep action conditional video representation network. By this combination our model learns a physically interpretable latent representation and can identify physical parameters. We propose supervised and self-supervised learning methods for our architecture. In experiments, we consider simulated scenarios with pushing, sliding and colliding objects, for which we also analyze the observability of the physical properties. We demonstrate that our network can learn to encode images and identify physical properties like mass and friction from videos and action sequences. We evaluate the accuracy of our training methods, and demonstrate the ability of our method to predict future video frames from input images and actions.

Keywords Physical scene understanding · Video representation learning · Differentiable physics

1 Introduction

Scene forecasting Mottaghi et al. (2016b, a) or vision-based control and planning Finn et al. (2016); Finn and Levine (2017); Hafner et al. (2019) require representations of image observations which facilitate prediction. In recent years, representation learning methods have emerged for this purpose that enable predictions directly on images or video Mottaghi et al. (2016a); Babaeizadeh et al. (2018); Zhu et al. (2019).

Many video prediction models encode images into a low dimensional latent scene representation which is predicted

forward in time – sometimes conditioned on actions – and that is decoded back into images. Neural representations for video prediction such as Srivastava et al. (2015); Finn et al. (2016); Babaeizadeh et al. (2018) perform these steps implicitly and typically learn a latent representation which cannot be directly interpreted for physical quantities such as mass, friction, position and velocity. This can limit explainability and generalization for new tasks and scenarios. Analytical models like Kloss et al. (2017); Degraeve et al. (2016); Belbute-Peres et al. (2018) in contrast structure the latent space as an interpretable physical parameterization and use analytical physical models to forward the latent state.

In this paper we investigate the use of differentiable physics for video representation learning. We examine supervised and self-supervised learning approaches which identify physical parameters of objects from video. Our approach learns to encode images into physical states and uses a differentiable physics layer Belbute-Peres et al. (2018) to forward the physical scene state based on latent physical scene parameters which are also learned from training video. Self-supervised learning is achieved by decoding the predicted physical states back into images using spatial transformers Jaderberg et al. (2015) (see Fig. 1) and assuming known object models.

Communicated by Zeynep Akata.

✉ Rama Krishna Kandukuri
rama.kandukuri@tue.mpg.de
Jan Achterhold
jan.achterhold@tue.mpg.de
Michael Moeller
michael.moeller@uni-siegen.de
Joerg Stueckler
joerg.stueckler@tue.mpg.de

¹ Max Planck Institute for Intelligent Systems, Tuebingen, Germany

² University of Siegen, Siegen, Germany

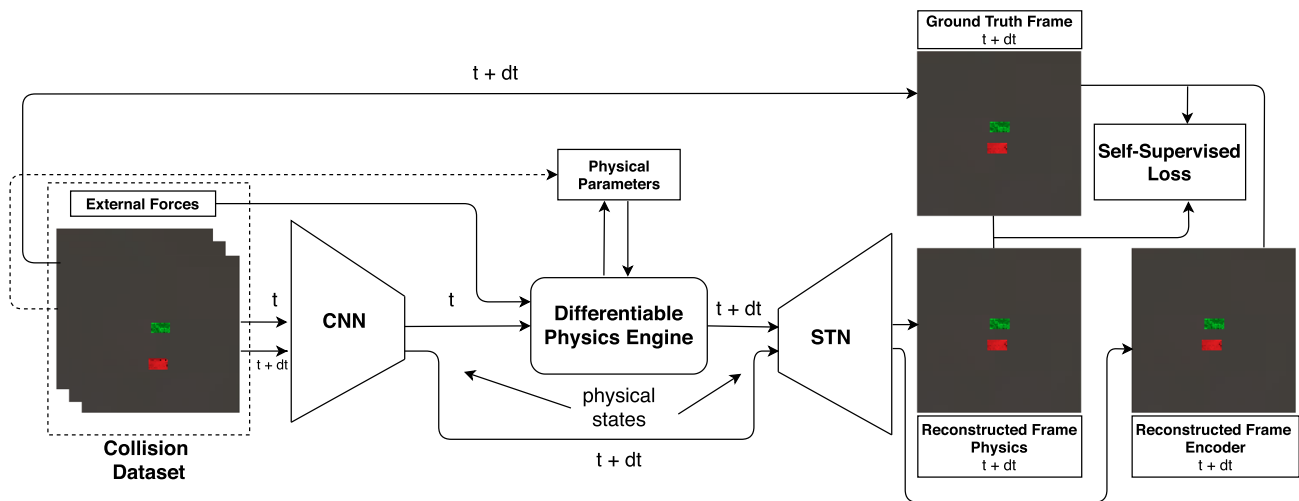


Fig. 1 We propose a self-supervised video representation learning approach that combines differentiable physics and spatial transformer layers to learn a physical state latent representation from videos and object interactions in object pushing and collision scenarios

We evaluate our supervised and self-supervised learning approaches in simulated object pushing, sliding and collision scenarios. We evaluate our approaches on simulated scenarios, assess their accuracy in estimating object pose and physical parameters, and compare to a purely neural network based model. We also analyze the observability of physical parameters in these scenarios. We demonstrate that physical scene encodings can be learned from video and interactions through our training approaches. Our approaches allow for identifying the observable physical parameters of the objects from the videos.

To summarize, we make the following contributions:

- We present approaches that learn to encode scenes into physical latent representations of objects in supervised and self-supervised ways. Our network architecture integrates a differentiable physics engine to predict next states. Self-supervised learning is achieved through spatial transformers which decode the states back into images. Our learning approaches simultaneously identify the observable physical parameters while learning the network parameters of the encoder.
- Our physics-based approach outperforms a pure neural network based baseline model in terms of prediction accuracy and generalizes better to forces which are unseen during training.
- We analyse the observability of physical parameters in object pushing, sliding and collision scenarios.

This article extends our previous conference publication Kandukuri et al. (2020) with further details and back-

ground of our approach and additional baseline results, comparisons and analysis.

1.1 Related Work

1.1.1 Neural Video Prediction

Several video prediction models learn to recurrently embed video frames into a latent representation using neural network layers such as convolutions, non-linearities, and recurrent units. The recurrent structure is used to predict the latent state forward in time. Srivastava et al. (2015) recurrently encode images into a latent representation using long short term memory (LSTM Hochreiter and Schmidhuber (1997)) cells. The latent representation is decoded back into images using a convolutional decoder. Video prediction is achieved by propagating the latent representation of the LSTM forward using predicted frames as inputs. Finn et al. (2016) also encode images into a latent representation using successive LSTM convolutions Shi et al. (2015). The decoder predicts motion kernels (5×5 pixels) and composition masks for the motion layers which are used to propagate the input images.

A typical problem of such architectures is that they cannot capture multi-modal distributions on predicted frames well, for example, in the case of uncertain interactions of objects, which leads to blurry predictions. Babaeizadeh et al. (2018) introduce a stochastic latent variable which is inferred from the full sequence at training time and sampled from a fixed prior at test time. Visual interaction networks explicitly model object interactions using graph neural networks in a recurrent video prediction architecture Watters et al. (2017). However, these approaches do not learn a physically inter-

pretable latent representation and cannot be used to infer physical parameters. To address these shortcomings, Ye et al. (2018) train a variational autoencoder based architecture in a conditional way by presenting training data with variation in each single specific physical property while holding all but a few latent variables fixed. This way, the autoencoder is encouraged to represent this property in the corresponding part of the latent vector. The approach is demonstrated on videos of synthetic 3D scenes with colliding shape primitives. Zhu et al. (2019) combine disentangled representation learning based on total correlation Chen et al. (2018) with partial supervision of physical properties. These purely deep learning based techniques still suffer from sample efficiency and require significant amounts of training data.

1.1.2 Physics-based Prediction Models

Several works have investigated differentiable formulations of physics engines which could be embedded as layers in deep neural networks. In Degraeve et al. (2016) an impulse-based velocity stepping physics engine is implemented in a deep learning framework. Collisions are restricted to sphere shapes and sphere-plane interactions to allow for automatic differentiation. The method is used to tune a deep-learning based robot controller but neither demonstrated for parameter identification nor video prediction.

Belbute-Peres et al. (2018) propose an end-to-end differentiable physics engine that models friction and collisions between arbitrary shapes. Gradients are computed analytically at the solution of the resulting linear complementarity problem (LCP) Amos and Kolter (2017). They demonstrate the method for including a differentiable physics layer in a video prediction network for modelling a 2D bouncing balls scenario with 3 color-coded circular objects. Input to the network are the color segmented images and optical flow estimated from pairs of frames. The network is trained in a supervised way using ground-truth positions of the objects. We propose to use spatial transformers in the decoder such that the network can learn a video representation in a self-supervised way. We investigate 3D scenarios that include pushing, sliding, and collisions of objects and analyze observability of physical parameters using vision and known forces applied to the objects. A different way of formulating rigid body dynamics has been investigated in Greydanus et al. (2019) using energy conservation laws. The method is demonstrated for parameter identification, angle estimation and video prediction for a 2D pendulum environment using an autoencoder network. Similar to our approach, Jaques et al. (2020) also uses spatial transformers for the decoder. However, differently the physics engine only models gravitational forces between objects and does not investigate full 3D rigid body physics with collision and friction modelling and parameter identification.

Recently, Runia et al. (2020) demonstrated an approach for estimating physical parameters of deforming cloth in real-world scenes. The approach minimizes distance in a contrastively learned embedding space which encodes videos of the observed scene and rendered scenes generated with a physical model based on the estimated parameters. In our approach, we train a video embedding network with the physical model as network layer and identify the physical parameters of observed rigid objects during training.

2 Background

2.1 Unconstrained and Constrained Dynamics

The governing equation of unconstrained rigid body dynamics in 3D can be written as

$$\mathbf{f} = \mathbf{M}\dot{\boldsymbol{\xi}} + \text{Coriolis forces} \quad (1)$$

where $\mathbf{f} : [0, \infty[\rightarrow \mathbb{R}^{6N}$ is the time-dependent torque-force vector stacking individual torques and forces for the N objects. The matrix $\mathbf{M} \in \mathbb{R}^{6N \times 6N}$ is the mass-inertia matrix and $\dot{\boldsymbol{\xi}} : [0, \infty[\rightarrow \mathbb{R}^{6N}$ is the time-derivative of the twist vector so that $\boldsymbol{\xi}_i = (\boldsymbol{\omega}_i^\top, \mathbf{v}_i^\top)^\top$ stacks rotational and linear velocities $\boldsymbol{\omega}, \mathbf{v} : [0, \infty[\rightarrow \mathbb{R}^3$ Cline (2002) of the i -th object. The twist vector itself represents the time derivative of the poses $\mathbf{x} \in SE(3)^N$ of the N objects which are elements of the Special Euclidean Group $SE(3)$.

In our experiments we do not consider rotations between two or more frames of reference, therefore we do not have any Coriolis forces. Most of the real world rigid body motions are constrained. To simulate those behaviors we need to constrain the motion with joint, contact and frictional constraints Cline (2002).

The force-acceleration based dynamics which we use in equation (1) does not work well for collisions since there is a sudden change in the direction of velocity in infinitesimal time Cline (2002). Therefore we use impulse-velocity based dynamics, where even the friction is well-behaved Cline (2002), i.e., equations have a solution at all configurations. We discretize the acceleration using the forward Euler method as $\dot{\boldsymbol{\xi}} = (\boldsymbol{\xi}_{t+h} - \boldsymbol{\xi}_t)/h$, where $\boldsymbol{\xi}_{t+h}$ and $\boldsymbol{\xi}_t$ are the velocities in successive time steps at times $t+h$ and t , and h is the time-step size. Equation (1) now becomes

$$\mathbf{M}\boldsymbol{\xi}_{t+h} = \mathbf{M}\boldsymbol{\xi}_t + \mathbf{f} \cdot h. \quad (2)$$

2.1.1 Constrained Dynamics

Joint constraints are equality constraints $g_e(\mathbf{x}) = 0$ in the poses of two objects. They restrict degrees of freedom of the rigid bodies. By deriving the pose constraints for time, this

can be written as $\mathbf{J}_e \dot{\boldsymbol{\xi}}_{t+h} = 0$ where \mathbf{J}_e is the Jacobian of the constraint function which gives the directions in which the motion is restricted. The joint constraints exert constraint forces which are solved using Euler-Lagrange equations by solving for the joint force multiplier λ_e .

The contact constraints $g_c(\mathbf{x}) \geq 0$ are inequality constraints which prevent bodies from interpenetration. This ensures that the minimum distance between two bodies is always greater than or equal to zero. The constraint equations can be written using Newton's impact model Cline (2002) as $\mathbf{J}_c \dot{\boldsymbol{\xi}}_{t+h} \geq -k\mathbf{J}_c \dot{\boldsymbol{\xi}}_t$. The term $k\mathbf{J}_c \dot{\boldsymbol{\xi}}_t$ can be replaced with \mathbf{c} which gives $\mathbf{J}_c \dot{\boldsymbol{\xi}}_{t+h} \geq -\mathbf{c}$, where k is the coefficient of restitution, \mathbf{J}_c is the Jacobian of the contact constraint function at the current state of the system and λ_c is the contact force multiplier. Since it is an inequality constraint we introduce slack variables \mathbf{a} , which also gives us complementarity constraints Mattingley and Boyd (2012).

The friction is modeled using a maximum dissipation energy principle since friction damps the energy of the system. In this case we get two inequality constraints in the object twist vectors since frictional force depends on normal force Anitescu and Potra (1997); Stewart (2000). They can be written as $\mathbf{J}_f \dot{\boldsymbol{\xi}}_{t+h} + \mathbf{E}\boldsymbol{\gamma} \geq 0$ and $\mu\lambda_c \geq \mathbf{E}^T \boldsymbol{\lambda}_f$ where μ is the friction coefficient, \mathbf{J}_f is the Jacobian of the friction constraint function $g_f(\mathbf{x})$ at the current state of the system, \mathbf{E} is a binary matrix which ensures linear independence between equations at multiple contacts, and $\boldsymbol{\lambda}_f$ and $\boldsymbol{\gamma}$ are frictional force multipliers. Since we have two inequality constraints we have two slack variables $\boldsymbol{\sigma}$, $\boldsymbol{\zeta}$ and two complementarity constraints.

In summary, all the constraints that describe the dynamic behavior of the objects we consider in our scene can be written as the following linear complementarity problem (LCP),

$$\begin{pmatrix} 0 \\ 0 \\ \mathbf{a} \\ \boldsymbol{\sigma} \\ \boldsymbol{\zeta} \end{pmatrix} - \begin{pmatrix} \mathbf{M} & -\mathbf{J}_e^T & -\mathbf{J}_c^T & -\mathbf{J}_f^T & 0 \\ \mathbf{J}_e & 0 & 0 & 0 & 0 \\ \mathbf{J}_c & 0 & 0 & 0 & 0 \\ \mathbf{J}_f & 0 & 0 & 0 & \mathbf{E} \\ 0 & 0 & \mu & -\mathbf{E}^T & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\xi}_{t+h} \\ \lambda_e \\ \lambda_c \\ \boldsymbol{\lambda}_f \\ \boldsymbol{\gamma} \end{pmatrix} = \begin{pmatrix} -\mathbf{M}\dot{\boldsymbol{\xi}}_t - h\mathbf{f}_{ext} \\ 0 \\ \mathbf{c} \\ 0 \\ 0 \end{pmatrix},$$

$$\text{subject to } \begin{pmatrix} \mathbf{a} \\ \boldsymbol{\sigma} \\ \boldsymbol{\zeta} \end{pmatrix} \geq 0, \begin{pmatrix} \lambda_c \\ \boldsymbol{\lambda}_f \\ \boldsymbol{\gamma} \end{pmatrix} \geq 0, \begin{pmatrix} \mathbf{a} \\ \boldsymbol{\sigma} \\ \boldsymbol{\zeta} \end{pmatrix}^T \begin{pmatrix} \lambda_c \\ \boldsymbol{\lambda}_f \\ \boldsymbol{\gamma} \end{pmatrix} = 0. \quad (3)$$

The above LCP is solved using a primal-dual algorithm as described in Mattingley and Boyd (2012). It is embedded

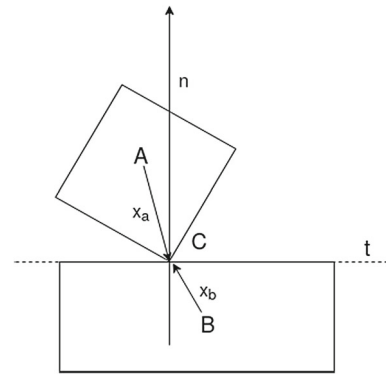


Fig. 2 Contact handling

in our deep neural network architecture in a similar way as in Amos and Kolter (2017) and Belbute-Peres et al. (2018), which facilitates backpropagation of gradients at its solution.

2.2 3D Contact Handling

We implement contact handling for 3D shapes such as planes and cubes using the open source implementation of the Open Dynamics Engine Smith (2008). The contact handling step consists of two sub-steps: collision detection and contact resolution. The collision detection step verifies if two or more bodies are under collision, i.e., the shortest distance between two objects is below a threshold value. If a collision is detected then for each pair of bodies, the contact resolution step is applied, which gives out the contact points on each body, the contact normal and the penetration distance.

The contact Jacobian (\mathbf{J}_c) and the frictional Jacobian (\mathbf{J}_f) are constructed from the outputs of contact resolution step. The entries for contact Jacobian are calculated from the directions of contact normal and the contact points on the bodies.

From Fig. 2, we can write the equation for the contact constraint as $g_c(\mathbf{x}) = \mathbf{n}^T \cdot (\mathbf{x}_a - \mathbf{x}_b) - \epsilon$, where \mathbf{n} is the contact normal, ϵ the distance threshold for collisions and \mathbf{x}_a and \mathbf{x}_b are the contact points on the bodies A and B respectively.

The contact Jacobians are derived from

$$\dot{g}_c(\mathbf{x}) = \mathbf{n}^T \cdot (\dot{\mathbf{x}}_a - \dot{\mathbf{x}}_b) \quad (4)$$

$$= \mathbf{n}^T \cdot ((\mathbf{v}_a + \boldsymbol{\omega}_a \times \mathbf{x}_a) - (\mathbf{v}_b + \boldsymbol{\omega}_a \times \mathbf{x}_b)) \quad (5)$$

$$= \underbrace{((\mathbf{x}_a \times \mathbf{n})^T \mathbf{n}^T)}_{\mathbf{J}_a} \underbrace{\begin{pmatrix} \boldsymbol{\omega}_a \\ \mathbf{v}_a \end{pmatrix}}_{\boldsymbol{\xi}_a} \quad (6)$$

$$+ \underbrace{(-(\mathbf{x}_b \times \mathbf{n})^T \mathbf{n}^T)}_{\mathbf{J}_b} \underbrace{\begin{pmatrix} \boldsymbol{\omega}_b \\ \mathbf{v}_b \end{pmatrix}}_{\boldsymbol{\xi}_b} \quad (7)$$

Table 1 Convolutional layers in our encoder architecture

Layer	Type	Dim in	Dim out	Kernel	Stride
1	Conv	4	64	5	2
2	Conv	64	128	5	2
3	Conv	128	256	5	2
4	Conv	256	256	5	2
5	Conv	256	128	3	1

Table 2 Fully connected layers in our encoder architecture

Layer	Type	Input size	Output size
5	FC	3*3*128	50
6	FC	50	50
7	FC	50	latent size

$$= \underbrace{\begin{pmatrix} \mathbf{J}_a & 0 \\ 0 & \mathbf{J}_b \end{pmatrix}}_{\mathbf{J}_c} \underbrace{\begin{pmatrix} \xi_a \\ \xi_b \end{pmatrix}}_{\xi} = \mathbf{J}_c \xi. \tag{8}$$

The frictional Jacobian is calculated in a similar way using the directions from the contact resolution step but the main difference is that the contact normal has only one direction and the direction of friction is perpendicular to the normal, which is the whole contact plane C. For this reason to get a finite number of directions we have to discretize the plane. Theoretically, the higher the number of the directions, the better the accuracy in calculation, but increasing the number also comes with larger computational cost. For simplicity in calculation and also considering the range of object motion, in this paper we consider four perpendicular directions.

3 Method

We develop a deep neural network architecture which extracts physical states $\mathbf{s}_i = (\mathbf{x}_i^\top, \xi_i^\top)^\top$ from images, where $\mathbf{x}_i = (\mathbf{q}_i^\top, \mathbf{p}_i^\top)^\top$ is the pose of object i with orientation $\mathbf{q}_i \in \mathbb{S}^3$ as unit quaternion and position $\mathbf{p}_i \in \mathbb{R}^3$. We propagate the state using the differentiable physics engine which is integrated as layer on the encoding in the deep neural network. For self-supervised learning, a differentiable decoder subnetwork generates images back from the integrated state representation of the objects.

We aim to learn the system’s dynamics by regressing the state trajectories and learning the physical parameters of the objects. These parameters can be the masses of the bodies and the coefficient of friction between two bodies. We initialize the objects at certain locations in the scene with some velocity and start the simulation by applying forces. In the following, we will detail our network architecture and training losses.

3.1 Network Architecture

3.1.1 Encoder

For supervised learning experiments, we use convolutional layers followed by fully connected layers with exponential linear units (ELU) Clevert et al. (2016) to encode poses from images. The encoder receives the image I_t and is encoded as pose \mathbf{x}_t . We need at least two images to infer velocities from images. For self-supervised learning experiments, we use an encoder with the same base architecture as in the supervised case to encode the input image I_t to pose \mathbf{x}_t . Details on the layers and parametrization are given in Tables 1 and 2.

We use three images so that we can average out the velocities in case of collisions when the two frames are collected just before and after collision. We use the difference in poses to estimate velocity instead of directly training the network to output velocities. This gives us the average velocity, not the final velocity. For example in 1D, when a block of mass m is acting under an effective force f_{eff} between times t_0 and t_1 , the velocity at time t_1 is given by

$$v(t_1) = \underbrace{\frac{p(t_1) - p(t_0)}{t_1 - t_0}}_{\text{average velocity}} + \frac{1}{2} \frac{f_{\text{eff}}}{m} (t_1 - t_0) \tag{9}$$

If we would let the network learn the velocities, it would require to implicitly learn the physics which we want to avoid by the use of the differentiable physics engine. The encoded states are provided as input to the differentiable physics engine.

3.1.2 Trajectory Integration

We integrate a trajectory of poses from the initial pose estimated by the encoder and the velocity estimates by the differentiable physics engine. In each time step, we calculate the new pose of each object $\mathbf{x}_t = (\mathbf{q}_t^\top, \mathbf{p}_t^\top)^\top$ where $\mathbf{q} \in \mathbb{S}^3$ is a unit quaternion representing rotation and $\mathbf{p} \in \mathbb{R}^3$ is the position from the resulting velocities of the LCP $\xi_t = (\omega_t^\top, \mathbf{v}_t^\top)^\top$ by

$$\begin{aligned} \mathbf{p}_t &= \mathbf{p}_{t-1} + \mathbf{v}_t \cdot h \\ \mathbf{q}_t &= \mathbf{q}_{t-1} \times \text{quat}(e^{0.5\omega_t h}) \end{aligned} \tag{10}$$

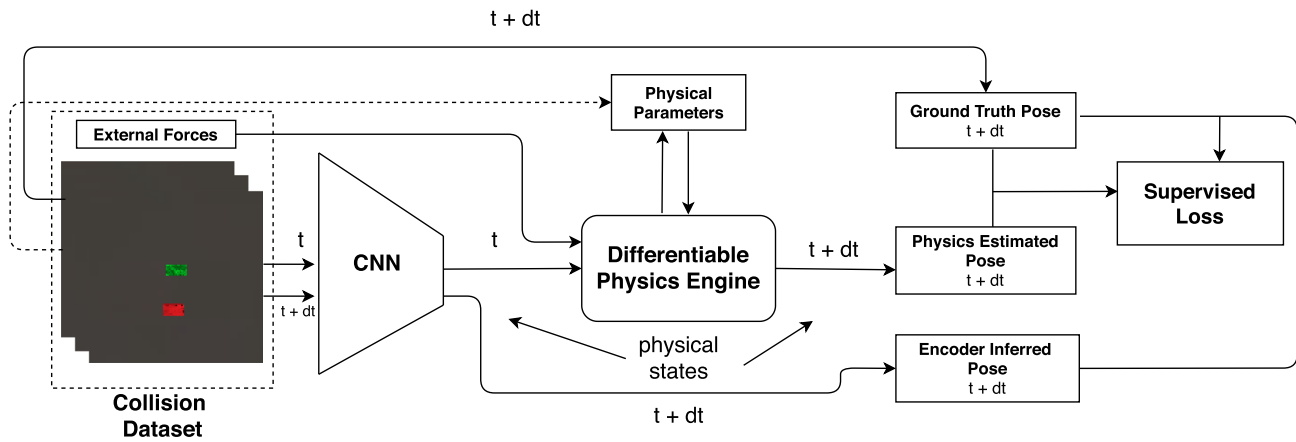


Fig. 3 Loss calculation for supervised video representation learning. An encoder predicts physical states which are forwarded in time using differentiable physics and compared with ground truth trajectories

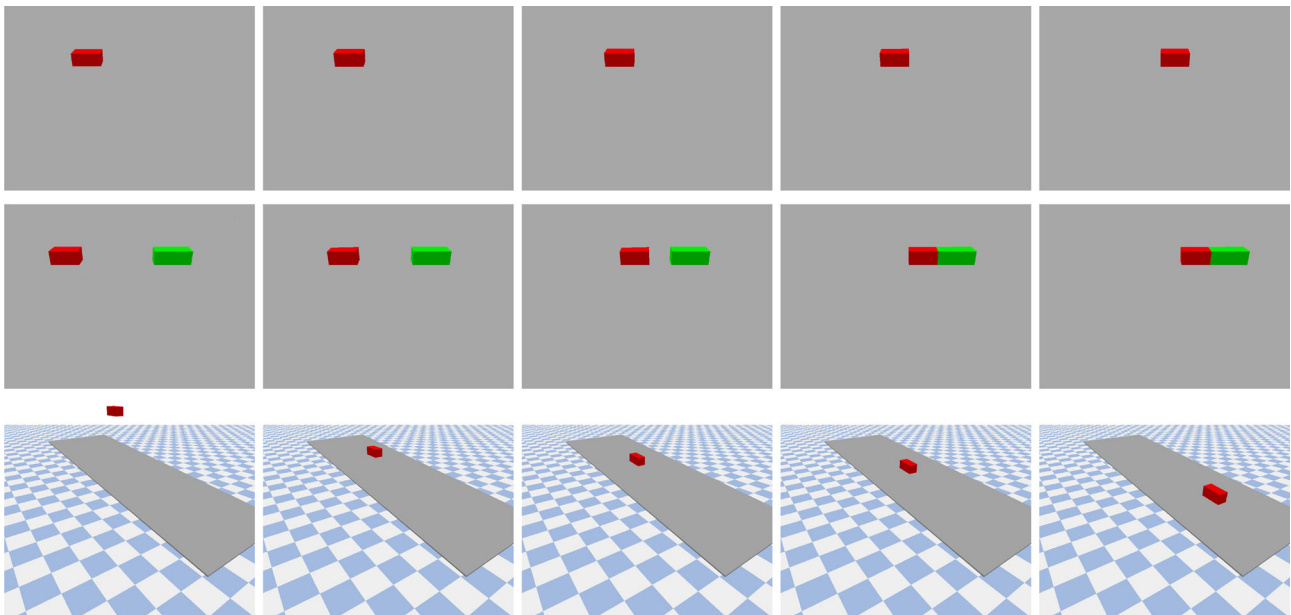


Fig. 4 3D visualization of the simulated scenes. Top: block pushed on a flat plane. Middle: block colliding with another block. Bottom: block falling and sliding down on an inclined plane

where $\text{quat}(\cdot)$ is an operator which converts a rotation matrix into a quaternion.

3.1.3 Decoder

To train the pipeline in a self-supervised way, we need a decoder which interprets the output of the physics engine layer and renders the objects at the estimated poses. For that purpose, we use a spatial transformer network (STN Jaderberg et al. (2015)) layer. Since we use only the STN layer for rendering, we do not have any learnable parameters for the decoder.

We use the absolute poses predicted by the physics engine layer to render the images. We assume known shape and appearance of the object and extract a content image for each

object from the first image of the sequence using ground-truth segmentation masks. The predicted poses are converted to image positions and in-plane rotations of the rectangular shape of the object in the top-down view assuming known camera intrinsics and extrinsics. A spatial transformer network layer renders the object's content image at the predicted image position and rotation. Finally, the prediction is reconstructed by overlaying the transformed object images onto the known background.

3.2 Training Losses

3.2.1 Supervised Learning

For supervised learning, we initialize the physics engine with inferred poses $\mathbf{x}_{1:N,i}^{enc}$ for each object i from the encoder where N is the (video) sequence length. Estimated poses $\hat{\mathbf{x}}_{1:N,i}$ by the physics engine as well as the inferred poses by the encoder are compared with ground truth poses $\mathbf{x}_{1:N,i}^{gt}$ to infer physical parameters,

$$L_{supervised} = \sum_i e(\mathbf{x}_{1:N,i}^{gt}, \mathbf{x}_{1:N,i}^{enc}) + \alpha e(\mathbf{x}_{1:N,i}^{gt}, \hat{\mathbf{x}}_{1:N,i}),$$

$$e(\mathbf{x}_1, \mathbf{x}_2) := \frac{1}{N} \sum_{t=1}^N \left\| \ln(\mathbf{q}_{2,t}^{-1} \mathbf{q}_{1,t}) \right\|_2^2 + \left\| \mathbf{p}_{2,t} - \mathbf{p}_{1,t} \right\|_2^2, \tag{11}$$

where α is a weighting constant, t is the time step and i indexes objects. In the warm up phase of the training (see 4.2.2), we use $\alpha = 0.0$ to pre-train the encoder. This avoids ill-posedness and stabilizes training because the physics engine needs a good initialization for position and velocity from the encoder. We then continue training the encoder along with the physics engine using $\alpha = 0.1$. We use the quaternion geodesic norm to measure differences in rotations. The loss calculation is visualized in Fig. 3.

3.2.2 Self-supervised Learning

For self-supervised learning, we initialize the physics engine with inferred poses $\mathbf{x}_{1:N}^{enc}$ from the encoder. Both estimated poses by the physics engine and the inferred poses are reconstructed into images $\hat{I}_{1:N}^{rec}$ and $I_{1:N}^{rec}$ using our decoder, respectively. The images are compared to input frames $I_{1:N}^{gt}$ to identify the physical parameters and train the network. Figure 1 illustrates the loss formulation.

$$L_{self-supervised} = \frac{1}{N} \left\| I_{1:N}^{gt} - I_{1:N}^{rec} \right\|_2^2 + \frac{\alpha}{N} \left\| I_{1:N}^{gt} - \hat{I}_{1:N}^{rec} \right\|_2^2 \tag{12}$$

3.2.3 System Identification

For reference, we also directly optimize for the physical parameters based on the ground-truth trajectories $\mathbf{p}_{1:N}^{gt}$ without the image encoder. For this we use the first state as an input to the differentiable physics engine. In this case, the loss function is $L_{sys-id} = \sum_i e(\mathbf{x}_i^{gt}, \hat{\mathbf{x}}_i)$.

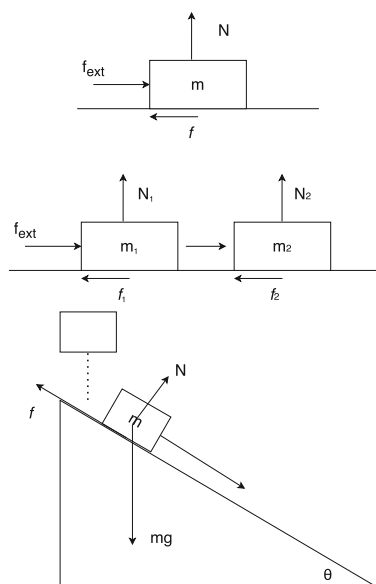


Fig. 5 1D/2D sketches of scenarios. Top left: block pushed on a flat plane. Bottom left: block colliding with another block. Right: block sliding down on an inclined plane

4 Experiments

We evaluate our approach in 3D simulated scenarios including pushing, sliding and collision of objects (see Fig. 4).

4.1 Simulated Scenarios and Observability Analysis

In this section, we discuss and analyze the different scenarios for the observability of physical parameters. To this end, we simplify the scenarios into 1D or 2D scenarios where dynamics equations are simpler to write.

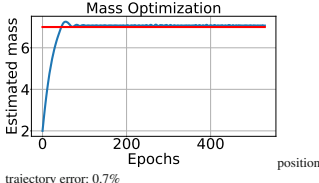
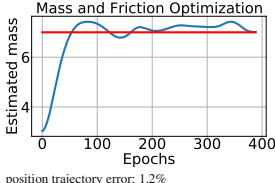
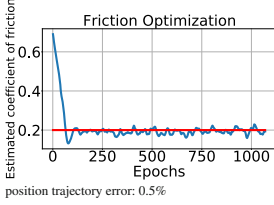
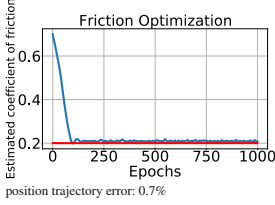
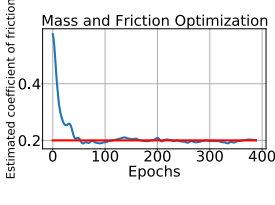
4.1.1 Block Pushed on a Flat Plane

In this scenario, a block of mass m , lying on a flat plane is pushed with a force \mathbf{f}_{ext} at the center of mass as shown in Fig. 5 (top left). In this 1D example, since we only have a frictional constraint we can use Eq. (2) in combination with the frictional force $f = \mu N$ to describe the system, where μ is the coefficient of friction, $g = 9.81m/s^2$ is the acceleration due to gravity and $N = mg$ is the normal force since the body has no vertical motion. The velocity v_{t+h} in the next time step hence is

$$v_{t+h} = v_t + \frac{f_{ext}}{m}h - \mu gh \tag{13}$$

We observe that only either one of mass or friction can be inferred at a time. Thus, in our experiments we fix one of the parameters and learn the other.

Table 3 System identification results (blue lines) for the 3 scenarios. Ground truth is shown as red lines

Inference	Block pushed on a flat plane	Block sliding down the inclined plane	Block colliding with another block
Mass	 <p>trajectory error: 0.7%</p>	Not feasible	 <p>position trajectory error: 1.2%</p>
Coefficient of friction	 <p>position trajectory error: 0.5%</p>	 <p>position trajectory error: 0.7%</p>	

4.1.2 Block Colliding With Another Block

To learn both mass and coefficient of friction simultaneously, we introduce a second block with known mass (m_2) made of the same material as the first one. This ensures that the coefficient of friction (μ) between the plane and the two blocks is same. Since we are pushing the blocks, after collision, both blocks move together. In the 1D example in Fig. 5 (bottom left), when applied an external force (f_{ext}), the equation to calculate the linear velocities $v_{1/2,t+h}$ of both objects in the next time step becomes

$$v_{1,t+h} = v_{1_t} + \frac{f_{\text{ext}}}{m_1}h - \mu gh, \quad v_{2,t+h} = v_{2_t} + \frac{f'}{m_2}h - \mu gh, \quad (14)$$

where μgm_1 and μgm_2 are frictional forces acting on each block and f' is the equivalent force on the second body when moving together. Now, in our experiments we can learn both mass and coefficient of friction together given the rest of the parameters in the equation.

4.1.3 Block Freefall and Sliding Down on an Inclined Plane

In this scenario the block slides down the inclined plane after experiencing a freefall as shown in Fig. 5 (right). In the 1D example, since the freefall is unconstrained (ignoring air resistance), the velocity update is given by $v_{t+h} = v_t + gh$. For block sliding down on an inclined plane, the equation to calculate velocity in the next time is

where θ is the plane inclination. We can see that we can only infer the coefficient of friction μ and due to the free fall we do not need to apply additional forces.

4.2 Results

We simulated the scenarios in 3D using the bullet physics engine using PyBullet.¹ Note that the bullet physics engine is different to the LCP physics engine in our network and can yield qualitatively and numerically different results. The bodies are initialized at random locations to cover the whole workspace. Random forces between 5 and 20N are applied at each time step. These forces are applied in $+x$, $-x$, $+y$ and $-y$ directions which are chosen at random but kept constant for a single trajectory while the magnitude of the forces randomly varies in each time step. In total, 1000 different trajectories are created with 300 time steps each for each scenario. We render top-down views at 128×128 resolution. Training and test data are split with ratio 9 : 1.

For evaluation we show the evolution of the physical parameters during the training. We also give the average relative position error by the encoder which is the average of the difference between ground truth positions and estimated poses divided by object size.

4.2.1 System Identification Results

As a baseline result, system identification (see Sect. 3.2.3) can be achieved within 200 epochs with an average position error for all the scenarios between 0.7 and 1.2%. Table 3 plots the estimates of the physical parameters. They reach nominal values with high accuracy.

¹ <https://pybullet.org>

Table 4 Supervised learning results for the 3 scenarios (smoothed over epochs with a Gaussian filter with $\sigma = 5$). The physical parameters are well identified (blue lines) close to the ground truth values (red lines)

Inference	Block pushed on a flat plane	Block sliding down the inclined plane	Block colliding with another block
Mass	<p>Mass Optimization</p> <p>position inference error: 4%</p>	<p>Not feasible</p>	<p>Mass and Friction Optimization</p> <p>position inference error: 8%</p>
Coefficient of friction	<p>Friction Optimization</p> <p>position inference error: 2%</p>	<p>Friction Optimization</p> <p>position inference error: 5%</p>	<p>Mass and Friction Optimization</p> <p>position inference error: 8%</p>

4.2.2 Supervised Learning Results

We train our network using the supervised loss in Sect. 3.2.1. We warm up the encoder by pre-training with ground truth poses so that when optimizing for physics parameters the training of the encoder is stable. We then continue training the encoder on the full supervised loss. From Table 4, we observe that all the learned physical parameters (in blue) slightly oscillate around the ground truth values (in red). The average inferred position error for all the scenarios is between 2 and 8% and the average inferred rotation error for the collision scenario is 8°. Our experiments show that this level of accuracy in the estimated initial states suffices for robust parameter learning.

4.2.3 Self-Supervised Learning Results

Now, we train the network in a self-supervised way (see Sect. 3.2.2). In this experiment, we generate sequences where the objects start at random locations with zero initial velocity, since the initial velocity estimate is ambiguous for our self-supervised learning approach. We obtain average velocities from the estimated poses (Eq. (9)). Since the pose estimation error is high in self-supervised experiments, the accuracy in velocity especially at the beginning of training is not sufficient for self-supervised learning. We pre-train the encoder in an encoder-decoder way so that when optimizing for physics parameters the training is stable. We continue training the encoder on the full self-supervised loss. To provide the network with gradients for localizing the objects, we use Gaussian smoothing on the input and reconstructed images starting from kernel size 128 and standard deviation 128, and reducing it to kernel size 5 and standard deviation 2 by the end of training. From Table 5, we observe that our approach can still recover the physical parameters at good accuracy. Expectably, they are less accurate than in the supervised learning experiment. The average inferred position error for all the scenarios is between 9 and 15% and the average inferred rotation error for the collision scenario is 10°. Through the use of spatial transformers our approach is limited to rendering top-down views and can not handle 3D translation and rotation in our third scenario.

4.3 Qualitative Video Prediction Results

The learned model in Sect. 4.2.3 can be used for video prediction. The images in the top row in Fig. 6a and b are the ground truth, the images in the middle row are the reconstructions from the predicted trajectories by our network and the images in the bottom row are the difference images. We roll out a 180 frame (3 seconds) trajectory. We can observe that the positions of the objects are well predicted by our approach,

Table 5 Self-supervised learning results for the pushing and collision scenarios (smoothed over epochs with a Gaussian filter with $\sigma = 5$). While the encoder error is slightly higher than in the supervised learn-

ing case, the physical parameters are identified (blue lines) close to the ground truth values (red lines)

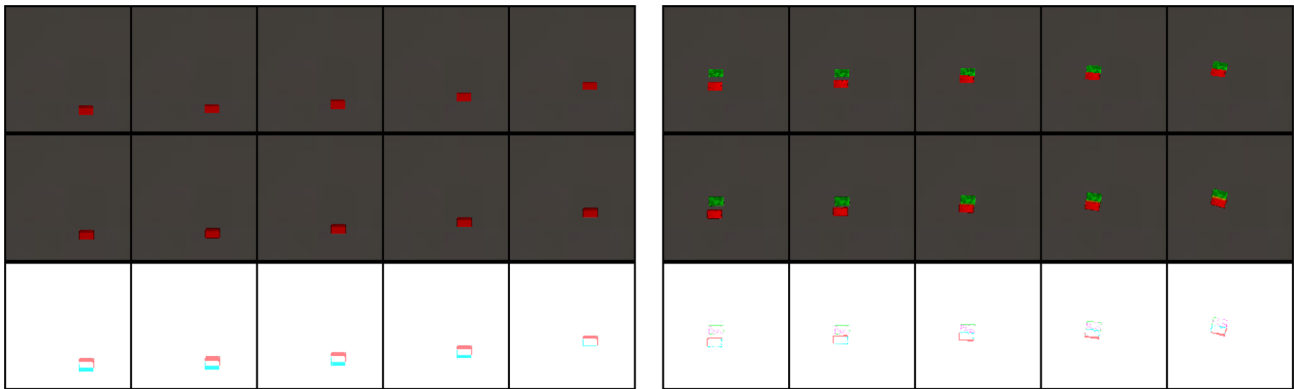
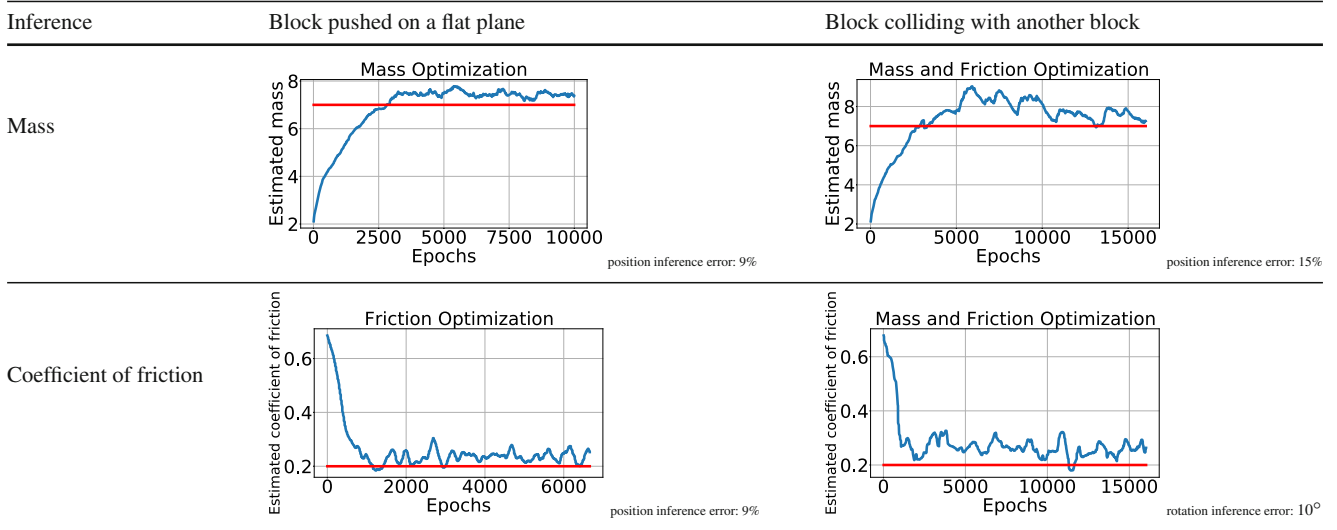


Fig. 6 Qualitative video prediction results for block pushing (left) and collision scenarios (right) with our method. Top: simulated images (from left to right frames 0, 30, 60, 120, 180) Middle: predicted images by our approach. Bottom: difference images.

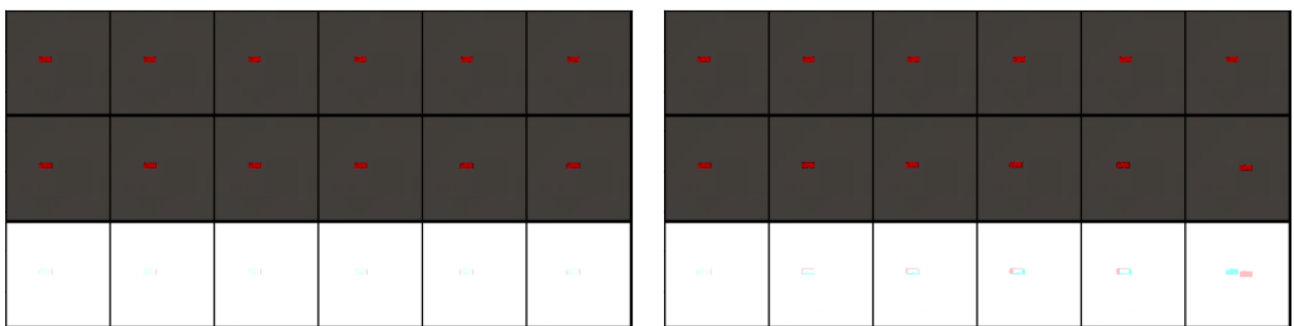


Fig. 7 Qualitative video prediction results of our method (left) and the MLP baseline (right) for the pushing scenario. Top: simulated images (from left to right frames 0, 20, 40, 60, 80, 120). Middle: predicted images. Bottom: difference images

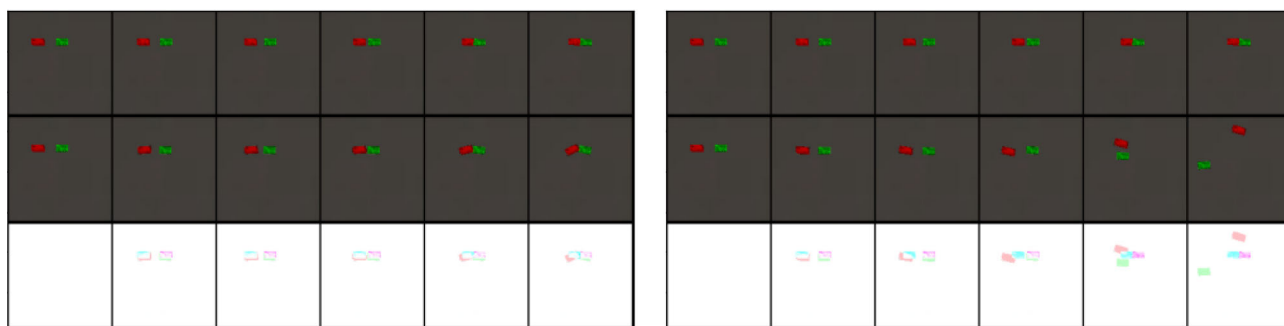


Fig. 8 Qualitative video prediction results of our method (left) and the MLP baseline (right) for the collision scenario. Top: simulated images (from left to right frames 0, 20, 40, 60, 80, 120). Middle: predicted images. Bottom: difference images

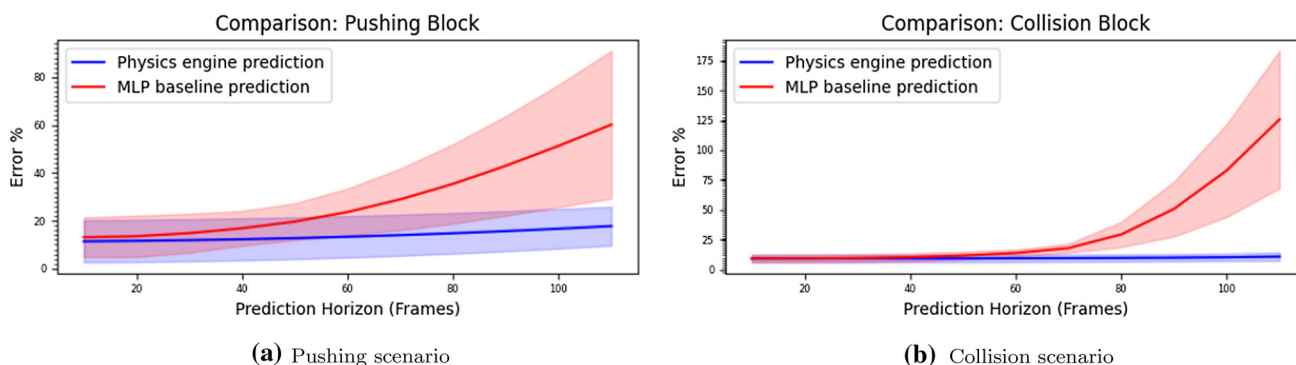


Fig. 9 Comparison of pose error (relative to object size in percent, mean: lines, std. var.: shaded area) for varying prediction horizons for the MLP baseline (red) and our approach (blue) in the block pushing (left) and collision scenarios (right). (a) Pushing scenario (b) Collision scenario

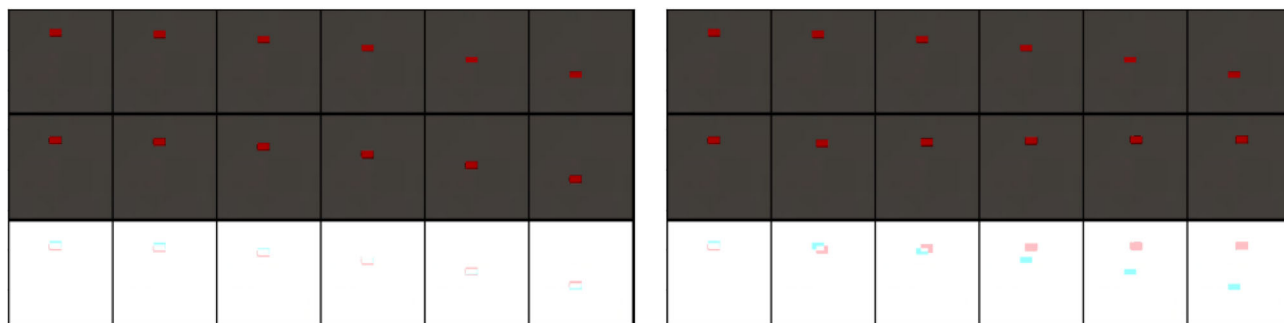


Fig. 10 Qualitative video prediction results of our method (left) and the MLP baseline (right) for the pushing scenario for new force distribution. Top: simulated images (from left to right frames 0, 12, 24, 36, 48, 60). Middle: predicted images. Bottom: difference images

while the approach yields small inaccuracies in predicting rotations which occur after the collision of the objects.

4.4 Baseline MLP Prediction

We compare our approach against a baseline method in which the physics engine is replaced by a multi-layer perceptron (MLP) network with three hidden layers (10 features each for pushing and 20 features each for collision scenarios), while retaining the rest of the architecture. The MLP network takes the concatenated vector of current pose (x_t), current

velocity (ξ_t) and the applied force vector (f_t) and outputs the change in pose and velocity ($\Delta x_t, \Delta \xi_t$). Since the output of the network does not guarantee that the objects lie inside the frame after a forward roll out, we stabilize the training by an additional penalty term to the outputs of the MLP to keep the objects in the frame in the following loss function

$$L_{\text{self-supervised-MLP}} = \frac{1}{N} \|I_{1:N}^{gt} - I_{1:N}^{rec}\|_2^2 + \frac{\alpha}{N} \|I_{1:N}^{gt} - \hat{I}_{1:N}^{rec}\|_2^2$$

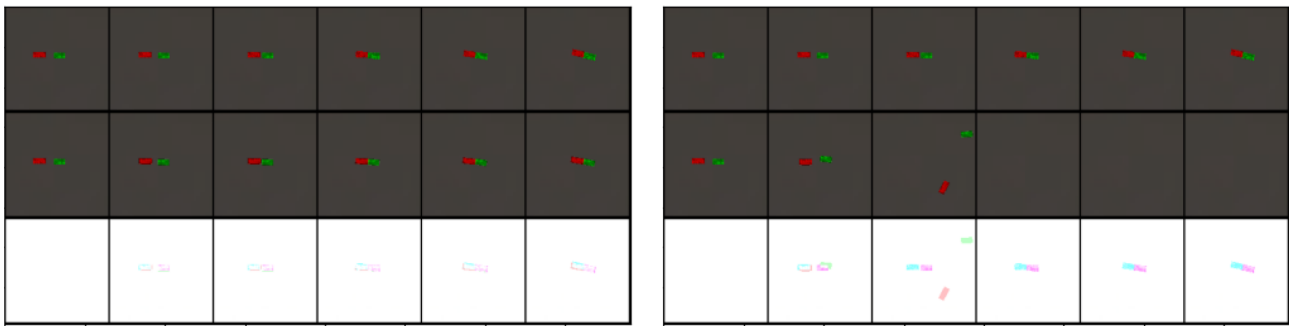


Fig. 11 Qualitative video prediction results of our method (left) and the MLP baseline (right) for the collision scenario for new force distribution. Top: simulated images (from left to right frames 0, 12, 24, 36, 48, 60). Middle: predicted images. Bottom: difference images

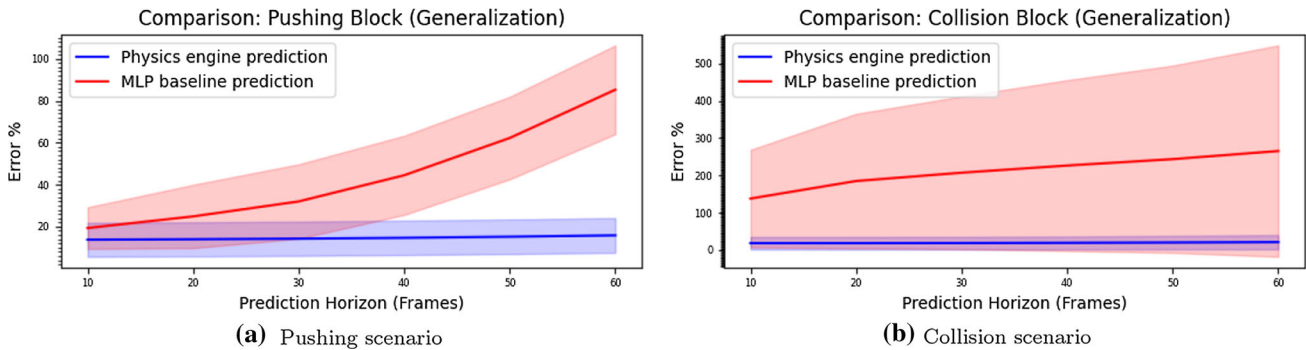


Fig. 12 Generalization to different force distribution. Comparison of relative pose error (relative to object size in percent, mean: lines, std. var.: shaded area) for varying prediction horizons for the MLP baseline

(red) and our approach (blue) in the block pushing (left) and collision scenarios (right). (a) Pushing scenario (b) Collision scenario

$$\begin{aligned}
 & + \gamma \sum_{i=1}^N \left(\left\| \max(|x_i^{rec}| - x_{max}, 0) \right\|_2^2 \right. \\
 & \left. + \left\| \max(|y_i^{rec}| - y_{max}, 0) \right\|_2^2 \right)
 \end{aligned}$$

where I^{gt} is the ground truth frame, I^{rec} is the reconstructed frame from encoder inferred pose, \hat{I}^{rec} is the reconstructed frame from MLP predicted pose, γ is the weight at which the MLP is penalized, x_i^{rec} and y_i^{rec} are the poses predicted by MLP and x_{max} and y_{max} are the maximum limits in the frame.

The roll out size in the training set is 30 frames and in the evaluation set is 120 frames. For training we employ a similar procedure like for self-supervised experiments. We warm-up the encoder by training it standalone for 7000 epochs and then start training the MLP together with the encoder for 50000 more epochs or until convergence.

We initialize our method and the MLP baseline with the output of their encoder and predict 120 frames. Figures 7 and 8 show the qualitative video prediction results for prediction with our method and the MLP baseline for pushing and collision scenarios, respectively. We see that the error growth in our method is smaller when compared to the MLP baseline. The results are shown quantitatively in Fig. 9a and

b. For the prediction error plots, we sample 20 trajectories and calculate mean and standard deviation per frame. In the collision scenario, Fig. 8, we see that the MLP network is not able to capture the more complex dynamics involved in collisions. Note that the MLP baseline does not infer physical parameters such as mass or friction.

This also becomes evident when we apply the models to unseen forces. We perform roll outs for 60 frames on a dataset with different force distribution than that was used during training. For these generalization experiments we use a uniform force distribution between 30N and 40N which is distinct from the forces used for training. Figures 10 and 11 show the qualitative video prediction results for prediction with our method and baseline method for pushing and collision scenarios, respectively. From the prediction error over 20 sample trajectories in Fig. 12 we can see that the error of the MLP baseline shoots up even for a 60 frame roll out, while our physics-based approach generalizes clearly better.

4.5 Discussion and Limitations

Our approach achieves physical parameter identification and CNN parameter learning by supervised and self-supervised learning in the evaluated scenarios. It outperforms an MLP

baseline model in terms of accuracy and generalization. We have studied observability and feasibility of learning physical parameters and video embedding by our approach. At its current stage, our architecture makes several assumptions on the scenes which could be addressed in future research. Our approach for using 2D spatial transformers for image generation restricts the self-supervised learning approach to known object shape and appearance and top down views. For real scenes our methods needs information about the applied forces which can be obtained from a known dynamics model of the interacting device (e.g. of a robot) or force sensors. Additionally, our model currently assumes that the forces are directly applied at the center of the object. It would require extensions, for instance, to also estimate the point of contact and direction of the applied forces using the CNN encoder. For self-supervised learning, methods for bridging the sim-to-real domain gap have to be investigated.

5 Conclusion

In this article we study supervised and self-supervised deep learning approaches which learn image encodings and identify physical parameters. Our deep neural network architecture integrates differentiable physics with a spatial transformer network layer to learn a physical latent representation of video and applied forces. For supervised learning, an encoder regresses the initial object state from images. Self-supervised learning is achieved through the implementation of a spatial transformer which decodes the predicted positions by the encoder and the physics engine back into images. This way, the model can also be used for video prediction with known actions by letting the physics engine predict positions and velocities conditioned on the actions. We evaluate our approach in scenarios which include object pushing, sliding and collisions and compare our results with an MLP baseline. We analyze the observability of physical parameters and assess the quality of the reconstruction of these parameters using our learning approaches. In future work we plan to investigate further scenarios including learning the restitution parameter and extend our self-supervised approach to real scenes and full 3D motion of objects.

Acknowledgements We acknowledge support from Cyber Valley, the Max Planck Society, and the German Federal Ministry of Education and Research (BMBF) through the Tuebingen AI Center (FKZ: 01IS18039B). The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Jan Achterhold.

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adap-

tation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Amos, B., Kolter, J.Z. (2017) Optnet: Differentiable optimization as a layer in neural networks. In: International Conference on Machine Learning, p. 136–145
- Anitescu, M., & Potra, F. A. (1997). Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, *14*, 231–247.
- de Avila Belbute-Peres, F., Smith, K., Allen, K., Tenenbaum, J., Kolter, J.Z. (2018) End-to-end differentiable physics for learning and control. In: Advances in Neural Information Processing Systems, pp. 7178–7189
- Babaeizadeh, M., Finn, C., Erhan, D., Campbell, R., Levine, S. (2018) Stochastic variational video prediction. In: Proceedings of the International Conference on Learning Representations
- Chen, R.T.Q., Li, X., Grosse, R.B., Duvenaud, D.K. (2018) Isolating sources of disentanglement in variational autoencoders. In: Advances in Neural Information Processing Systems, pp. 2610–2620
- Clevert, D.A., Unterthiner, T., Hochreiter, S. (2016) Fast and accurate deep network learning by exponential linear units (elus). In: Proceedings of the International Conference on Learning Representations
- Cline, M.B. (2002) Rigid body simulation with contact and constraints. Master's thesis. <https://doi.org/10.14288/1.0051676>
- Degrave, J., Hermans, M., Dambre, J., & Wyffels, F. (2016). A differentiable physics engine for deep learning in robotics. *Frontiers in Neurobotics*. <https://doi.org/10.3389/fnbot.2019.00006>
- Finn, C., Goodfellow, I.J., Levine, S. (2016) Unsupervised learning for physical interaction through video prediction. In: Advances in Neural Information Processing Systems, pp. 64–72
- Finn, C., Levine, S. (2017) Deep visual foresight for planning robot motion. In: International Conference on Robotics and Automation, pp. 2786–2793
- Greydanus, S., Dzamba, M., Yosinski, J. (2019) Hamiltonian neural networks. In: Advances in Neural Information Processing Systems, pp. 15379–15389
- Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., Davidson, J. (2019) Learning latent dynamics for planning from pixels. In: International Conference on Machine Learning, pp. 2555–2565
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*, 1735–80.
- Jaderberg, M., Simonyan, K., Zisserman, A., Kavukcuoglu, K. (2015) Spatial transformer networks. In: Advances in Neural Information Processing Systems, pp. 2017–2025
- Jaques, M., Burke, M., Hospedales, T.M. (2020) Physics-as-inverse-graphics: Joint unsupervised learning of objects and physics from video. Proceedings of the International Conference on Learning Representations
- Kandukuri, R., Achterhold, J., Moeller, M., Stueckler, J. (2020) Learning to identify physical parameters from video using differentiable

- physics. In: Proceedings of the 42th German Conference on Pattern Recognition (GCPR)
- Kloss, A., Schaal, S., & Bohg, J. (2017). Combining learned and analytical models for predicting action effects. *CoRR* abs/1710.04102.
- Mattingley, J., & Boyd, S. (2012). Cvxgen: A code generator for embedded convex optimization. *Optimization and Engineering*. <https://doi.org/10.1007/s11081-011-9176-9>
- Mottaghi, R., Bagherinezhad, H., Rastegari, M., Farhadi, A. (2016) Newtonian scene understanding: Unfolding the dynamics of objects in static images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition
- Mottaghi, R., Rastegari, M., Gupta, A., Farhadi, A. (2016) “What happens if...” learning to predict the effect of forces in images. In: European Conference on Computer Vision
- Runia, T.F.H., Gavriluyk, K., Snoek, C.G.M., Smeulders, A.W.M. (2020) Cloth in the wind: A case study of estimating physical measurement through simulation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition
- Shi, X., Chen, Z., Wang, H., Yeung, D.Y., Wong, W.k., Woo, W.c. (2015) Convolutional lstm network: A machine learning approach for precipitation nowcasting. In: Advances in Neural Information Processing Systems, pp. 802–810
- Smith, R.: Open dynamics engine (2008). <http://www.ode.org/>.
- Srivastava, N., Mansimov, E., Salakhutdinov, R. (2015) Unsupervised learning of video representations using lstms. In: International Conference on Machine Learning
- Stewart, D. (2000). Rigid-body dynamics with friction and impact. *SIAM Rev*, 42, 3–39. <https://doi.org/10.1137/S0036144599360110>
- Watters, N., Zoran, D., Weber, T., Battaglia, P., Pascanu, R., Tacchetti, A. (2017) Visual interaction networks: Learning a physics simulator from video. In: Advances in Neural Information Processing Systems
- Ye, T., Wang, X., Davidson, J., Gupta, A. (2018) Interpretable intuitive physics model. In: European Conference on Computer Vision
- Zhu, D., Munderloh, M., Rosenhahn, B., Stückler, J. (2019) Learning to disentangle latent physical factors for video prediction. In: German Conference on Pattern Recognition

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.