# iPOKE: Poking a Still Image for Controlled Stochastic Video Synthesis

—

# Supplementary Material

## Preliminaries

For all of our reported datasets in the main paper, we provide additional video material resulting in 14 videos in total. Analogous to the main paper, the direction of the control input is indicated by a red arrow starting at the poke location $l$ and the target location is marked by a red dot, if not stated otherwise. For some examples were either the arrow depicting the user input or the target location are small, we additionally overlay them with larger arrows and dots. We loop each video three times and replay it with three fps. The file structure of the videos is as follows:

```
04284-supp
      |
      +--A-Additional_Visualizations
          |
          +--A1-Controlled Stochastic
                Video Synthesis
             |
             +--A1.1-Samples_PP.mp4
             |
             +--A1.2-Samples_Iper.mp4
             |
             +--A1.3-Samples_Human36m.mp4
             |
             +--A1.4-Samples_TaiChi.mp4
          |
          +--A2-Kinematics_Transfer
          |
          +--A3-Control_Sensitivity
          |
      +--B-Control Inputs
           of Human Users
          |
          +--B1-Human_Control_PP.mp4
          |
          +--B2-Human_Control_Iper.mp4
          |
          +--B3-GUI-Demo_1.mp4
          |
          +--B3-GUI-Demo_2.mp4
          |
      +--C-Qualitative Comparison
           with Baselines
          |
          +--C1-Controllable
                Video Synthesis
             |
             +--C1.1-Cmp_Hao_PP.mp4
             |
             +--C1.2-Cmp_Hao_Iper.mp4
```

```
    |
+--C2-Stochastic
     Video Synthesis
    |
    +--C2.1-Cmp_IVRNN_PP.mp4
    |
    +--C2.2-Cmp_IVRNN_Iper.mp4
    |
```

In Section A, we show and discuss qualitative video results of iPOKE on the four considered datasets for the tasks of controllable stochastic video synthesis (Section A.1) and transfer of kinematics between two unrelated object instances (Section A.2), which are the two main tasks considered in our paper. Our model iPOKE, however, also allows human users to animate still images by poking, as demonstrated by examples shown in Section B, which further contains examples of interactions of our model with human users via a graphical user interface. For the quantitative evaluations conducted in the main paper, we additionally provide qualitative comparisons in Section C. A derivation of our objective function is given in Section D. Moreover, we also provide the implementation details of our model in Section E and these of all considered baselines in Section F. Finally, describe the metrics used for evaluation in more detailed in Section G.

## A. Additional Visualizations

Here we show and discuss additional video material showing the results of iPOKE on all considered datasets for our main task of controllable stochastic video synthesis. Moreover we also provide additional examples of successfully transferred kinematics from a source sequence to a given still image frame on the iPER [42] dataset.

### A.1. Controllable Stochastic Video Synthesis

For each individual dataset presented in the main paper we show a video containing results of iPOKE for multiple object instances. Within each video, the first column depicts the ground truth video sequences starting with the same image frame $x_0$ which is used as part of the conditioning for our model. The remaining columns contain five generated video sequences corresponding to different samples drawn from the residual kinematics prior $q(r)$ for the same conditioning $(x_0, c)$. The user control $c$ is visualized as a red arrow starting at the controlled location $l = c_{3:4}$ and pointing to the intended target location, which is depicted by a red dot and plotted after the sequence has ended. As the arrows are sometimes tiny, we additionally visualize larger arrows indicating the direction of the user control input in the first column as an overlay over the ground truth sequences. For generating the user control, we used the procedure explained in Section 3.3 in the main paper. Subsequently, we discuss each video in detail.

**A1.1-PokingPlants.** For the PokingPlants dataset we show the results of iPOKE for five plants with substantially varying shapes and appearances in the video sequence *A1.1-Samples_Plants.mp4*. Despite these large variances, iPOKE manages to generate appealing video sequences showing plausible reactions to the user control $c$. In all cases, the intended end position of the controlled object part is reached. The remainder of the object, however, shows unique, plausible motion for each individual sample from $q(r)$ for the same conditioning. In summary, for the PokingPlants dataset, iPOKE is capable of rendering realistic video sequences by locally controlling the motion of a given part of the shown object instances. Moreover, diverse, plausible motion is generated for the remaining object parts.

**A1.2-iPER.** In the video *A1.2-Samples_Iper.mp4* we visualize results for the test set of iPER [42], thus showing persons previously unseen by iPOKE. We clearly see that our model generalizes to unseen appearances. Moreover, iPOKE generates realistic video sequences showing the poked human body parts approaching their intended target location, while retaining significant motion variance for these parts not directly affected by the user control $c$.

**A1.3-Human3.6m.** To also evaluate the capabilities of iPOKE to generate human motion not contained in iPER, i.e. behavior such as walking and sitting, we additionally use the Human3.6m [30] dataset. The resulting synthesized output of our model can be seen in the video *samples_human36m.mp4*. Also in this case, iPOKE achieves to generate realistic looking video sequences showing such complex human motion for control inputs ranging from very small (e.g. third row) to large (e.g. second and last rows) control inputs. Unfortunately, iPOKE changes the appearance of the depicted person if the generated motion is large. However, this is a common issue, when using the Human3.6m test set [53, 70, 76, 24], as the train set only contains five distinct appearances and, thus, generalization of appearance is very hard to achieve.

**A1.4-Tai-Chi** To further evaluate the capabilities of iPOKE to synthesize human motion for in-the-wild settings we use the Tai-Chi dataset [64]. This is a very challenging dataset due to diverse video backgrounds, which also contains motion.

The human motion contained in the Tai-Chi, however, is not as large and diverse as for the iPER and Human3.6m datasets. Nevertheless, our model is able to synthesize realistic looking sequences as visualized in *A1.4-Samples_Tai-Chi.mp4*.

### A.2. Kinematics Transfer

In the video *A2-Kinematics_Transfer_Iper.mp4* we show the ability of iPOKE to transfer kinematics from a source sequence (top row) to an image showing a person of arbitrary appearance exhibiting a similar starting posture (mid row). For comparison, we synthesized sequences in which we induce random object kinematics by sampling from our residual kinematics prior $q(r)$ (bottom row). For a further description on transfer of kinematics, cf. the respective paragraph in Section 4.2 in the main paper. The implications arising from observing these video examples are twofold: Firstly we can see that our model, due to its conditionally invertible nature, is capable of transferring kinematics from a given sequence to a still image in a fine-grained manner, as subtle details of the motion shown in the source sequence are preserved. Secondly, these experiments show that our residual kinematics representation $r$ is indeed independent of the conditioning factors $(x_0, c)$ as *i)* our model only transfers kinematics from the source sequence to the target image and *not* the appearance of the depicted person and *ii)* the sequences obtained from drawing random samples from $q(r)$ exhibit significantly different motion compared to the corresponding transferred sequences *except* for the controlled location, which is correctly approaching its target location in both cases. Thus, these two factors, are not influenced by the residual kinematics representation but only the kinematics of the body parts not directly influenced by the controlled location, due to this independence. This empirical evidence of the independence of $r$ from $(x_0, c)$ is further backed by theoretical arguments which are discussed in Section D.

### A.3. Control Sensitivity

In this paragraph, we directly verify that iPOKE also generates plausible object responses to control inputs $c$ at the same pixel location $c_{3:4}$ with different shift vectors $c_{1:2}$ for a given initial image frame $x_0$. To this end, we sample the initial frame $x_0$ of a sequence $\boldsymbol{X}$ from the iPER [42] test set and simulate a vector $c$ based on the procedure introduced in Sec. 3.3 from which we only take the respective interaction location $c^*_{3:4}$. To obtain different shift vectors at this location we randomly sample $L$ directions from a uniform distribution $\mathcal{U}(0, 2\pi)$ and magnitudes from $\mathcal{U}(\mu_{|F|}, \max_{|F|})$, where $\mu_{|F|}$ and $\max_{|F|}$ denote the mean and maximum optical flow magnitudes in the optical flow map $F$ between $x_0$ and the end frame $x_T$ of $\boldsymbol{X}$. Thus, we obtain $L$ different control inputs $\{(c^l, x_0) : c^l_{3:4} = c^*_{3:4}\}^L_{l=1}$ at the same location for a given $x_0$. Drawing different residual samples $\{r_l\}^L_{l=1}$ and applying the forward transformation results in video representations $z_l = \tau_\theta(r_l|x_0, c_l)$ which can afterwards be decoded to object responses by our autoencoding framework (cf. Sec. 3.2 and E.1). Each row of the video *A2-Control_Sensitivity_Iper.mp4* shows such generated object responses of iPOKE for $L = 4$ different shift vectors for the same static input image $x_0$ and compares these to the ground truth video starting with $x_0$. From observing these results, we see that the the poked body part always approaches its intended target location which is defined by the respective shift vector $c^l_{1:2}$, thus indicating the sensitivity of our model with respect to the control input.

## B. Control Inputs of Human Users

So far, we have only considered simulated control inputs as described in Section 3.3 in the main paper. However, as our goal is not only to provide a model for controlled stochastic video synthesis, but also to enable human users to directly animate a still image frame, we now additionally show the results of our model for inputs of human users for the object categories of plants on the PP dataset in Sec. B.1 and humans on iPER in Sec. B.2. Each video contains three unique object instances shown in the rows of the respective video *B1-Human_Control_PP.mp4 B2-Human_Control_Iper.mp4*. In each row, we show the ground truth sequence on the left, followed by an example using a simulated control input as described in Sec. 3.3 in the main paper. In the remaining three rows, we show examples of control inputs by human users. The results were obtained via our graphical user interface (GUI) which is presented in the videos *GUI_Demo_[1,2].mp4* in Section B.3. The GUI will be open sourced together with the project code upon publication.

### B.1. PokingPlants

The three plants in the video *B1-Human_Control_PP.mp4* again show that iPOKE is capable of generating plausible object kinematics as a reaction to the control. In particular, these examples highlight the accurate object reactions generated by our model even for fine-grained user control as, for instance, shown in the third column of the first row, where a control input with a small magnitude at an outer trunk results very subtle motion, leaving most of the controlled plant mainly unaffected. Large pokes, on the other hand, which are depicted in the remaining columns of the first row, result in large amounts of motion, thus corresponding to what we humans would expect from such a manipulation. Moreover, the results visualize the generalization abilities of iPOKE, as not all of these pokes from humans are within the data distribution.

### B.2. iPER

The video *B2-Human_Control_Iper.mp4* contains similar results to these discussed in the last section for the iPER dataset. The synthesized videos are realistic looking for inputs defined by human users and the controlled body parts are accurately moved to the intended target location. Moreover it is shown that our model learns to separate the pixels comprising the object from those in the background, as emerging from observing the fourth column in the top row, where a part of the background is controlled. In this case, our model does not generate human kinematics, but renders a still video sequence, where the entire person remains unaffected. This is caused by by our training procedure where we explicitly generate control signals in the background to teach the iPOKE to ignore such inputs. For a detailed description, refer to Section E.3.

### B.3. Graphical User Interface

Finally we recorded two videos (*B3-GUI_Demo_[1,2].mp4*) showing iPOKE interactively generating kinematics as defined by humans using our graphical user interface. The videos were recorded without cuts and, thus, clearly indicate the robustness of iPOKE. Moreover it can be seen that our model is capable of processing multiple control inputs one after another without generating heavy artifacts or implausible kinematics, thus enabling an interactive experience for human users. We plan to make this GUI publicly available and will release it together with our code.

## C. Qualitative Comparison with Competing Methods

To provide visual evidence for the quantitative evaluation presented in the main paper, we now show visual comparisons to Hao et al. [27] for the task of controllable video synthesis and to IVRNN [10] for the task of stochastic video synthesis. The comparisons with these methods are conducted in the iPER and PokingPlants (PP) datasets.

### C.1. Controllable Video Synthesis

Similar to Sec. A.1, in *C1.1-Comparison_Hao_PokingPlants.mp4* and *C1.2-Comparison_Hao_Iper.mp4* we show video generations for iPOKE and compare them to the approach of Hao et al [27] trained on sparse sets of random optical flow vectors and tracked keypoint representations (cf. Sec. 4.3 in main paper for details.). We observe that iPOKE produces fluent and realistic object kinematics on both datasets. In contrast, Hao et al. fails to capture and generate the complex, structured motion depicted in the iPER dataset due to only warping individual frames. We attribute the lower FVD scores obtained by this method, compared to ours, to this. On the PP dataset, the competing approach produces blurry outputs, as successive warping cannot represent coherent, fine-grained motion due to noisy optical flow estimates.

### C.2. Stochastic Video Synthesis

In *C2.1-Comparison_IVRNN_Plants.mp4* and *C2.2-Comparison_IVRNN_Iper.mp4* we compare video sequences generated by our model against sequences generated by IVRNN [10], a state-of-the-art approach for unconditional stochastic video synthesis. We chose this particular competitor as it was the best performing of the competing methods on these two datasets when considering video quality and diversity together (cf. Tab. 2 in the main paper). In our case, we generate diverse video sequences depicting different object kinematic realizations inferred by sampling from the residual representation $r \sim q(r)$ for different fixed user interactions $c$ on the iPER and PP dataset, thus following a similar setup as in Sec. A.1. For IVRNN [10] we show different video generations for two conditioning frames preceding the sequence to be generated. Our videos clearly show that our model iPOKE successfully synthesizes plausible video sequences which closely follow the user interaction $c$ while, on the other hand, the remaining object parts exhibit a diverse range of motion when varying $r$. In contrast, the videos samples of IVRNN hardly show visible differences across different video samples. Moreover, IVRNN fails to capture fine-grained motion, e.g. of individual leaves, on the PP dataset, thus generating blurry video sequences. This can be explained by the trade-off between quality and diversity, which IVRNN, as a VAE-based method, suffers from.

## D. Derivation and Implications of Learning Objective

### D.1. Independence of $r$ and $(x_0, c)$

We intend to learn $\tau$ defined in Eq. (4) such that the residual $r$ contains all information about $\boldsymbol{X}$ not present in $(x_0, c)$. A way to achieve this is to force independence between $r$ and the conditioning $(x_0, c)$. We now show that a minimizer of $\mathrm{KL}[p(r|x_0, c)\|q(r)]$, where $q(r)$ is some prior distribution independent of $(x_0, c)$, forces this independence. By the definition

of the Kullback-Leibler-Divergence, we have

$$\begin{aligned}\mathrm{KL}[p(r|x_0,c)\|q(r)] &= \int_{r,x_0,c} p(r,x_0,c)\log\left(\frac{p(r|x_0,c)}{q(r)}\right)\\ &= \int_{r,x_0,c} p(r,x_0,c)\log\left(p(r|x_0,c)\right) - \int_r p(r)\log\left(q(r)\right)\ .\end{aligned} \tag{10}$$

Following the argument of [63, 2] and using the fact that the Kullback-Leibler-Divergence is always positive, we have

$$\mathrm{KL}[p(r)\|q(r)] \geq 0 \Rightarrow \int_r p(r)\log\left(q(r)\right) \leq \int_r p(r)\log\left(p(r)\right)\ , \tag{11}$$

where $p(r)$ denotes the (unknown) true distribution of $r$. By inserting (11) into (10), we obtain

$$\begin{aligned}\mathrm{KL}[p(r|x_0,c)\|q(r)] &\geq \int_{r,x_0,c} p(r,x_0,c)\log\left(p(r|x_0,c)\right) - \int_r p(r)\log\left(p(r)\right)\\ &= \int_{r,x_0,c} p(r,x_0,c)\log\left(\frac{p(r|x_0,c)}{p(r)}\right)\\ &= \int_{r,x_0,c} p(r,x_0,c)\log\left(\frac{p(r,x_0,c)}{p(r)p(x_0,c)}\right) = \mathrm{MI}\left[r,(x_0,c)\right]\ .\end{aligned} \tag{12}$$

As (12) implies, $\mathrm{KL}[p(r|x_0,c)\|q(r)] \geq \mathrm{MI}\left[r,(x_0,c)\right]$ is an upper bound on the mutual information $\mathrm{MI}\left[r,(x_0,c)\right]$ and, thus, its minimizer also forces independence of $r$ and $(x_0,c)$.

### D.2. Derivation of Objective Function

Again, we start with the definition of the Kullback-Leibler-Divergence and follow [63]

$$\begin{aligned}\mathrm{KL}[p(r|x_0,c)\|q(r)] &= \int_r p(r|x_0,c)\log\left(\frac{p(r|x_0,c)}{q(r)}\right)\\ &= \int_{\boldsymbol{X}} p\left(\tau_\theta^{-1}(\boldsymbol{X}|x_0,c)|x_0,c\right)\cdot|\det J_{\tau_\theta^{-1}}(\boldsymbol{X}|x_0,c)|\cdot\log\left(\frac{p\left(\tau_\theta^{-1}(\boldsymbol{X}|x_0,c)|x_0,c\right)}{q\left(\tau_\theta^{-1}(\boldsymbol{X}|x_0,c)\right)}\right)\ ,\end{aligned} \tag{13}$$

where we have made a change of integration variable from $r$ to $\boldsymbol{X}$. Further, by rearranging Eq. (6), we have

$$p(\tau_\theta^{-1}(\boldsymbol{X}|x_0,c)|x_0,c) = \frac{p(\boldsymbol{X}|x_0,c)}{|\det J_{\tau_\theta^{-1}}(\boldsymbol{X}|x_0,c)|}\ . \tag{14}$$

Inserting (14) into (13) yields

$$\begin{aligned}\mathrm{KL}[p(r|x_0,c)\|q(r)] &= \int_{\boldsymbol{X}} p(\boldsymbol{X}|x_0,c)\log\left(\frac{p(\boldsymbol{X}|x_0,c)}{q\left(\tau_\theta^{-1}(\boldsymbol{X}|x_0,c)\right)|\det J_{\tau_\theta^{-1}}(\boldsymbol{X}|x_0,c)|}\right)\\ &= \mathbb{E}_{\boldsymbol{X}}\left[\log p(\boldsymbol{X}|x_0,c) - \log\left(q\left(\tau_\theta^{-1}(\boldsymbol{X}|x_0,c)\right)\right) - \log\left(|\det J_{\tau_\theta^{-1}}(\boldsymbol{X}|x_0,c)|\right)\right]\ .\end{aligned} \tag{15}$$

If we now choose $q(r) = \mathcal{N}(0,\boldsymbol{I})$ and insert this into (15), we obtain our final objective by taking the expectation over $(x_0,c)$, as

$$\begin{aligned}\mathrm{KL}[p(r|x_0,c)\|q(r)] &= \mathbb{E}_{\boldsymbol{X},x_0,c}\left[\frac{1}{2}\|\tau_\theta^{-1}(\boldsymbol{X}|x_0,c)\|_2^2 - \log\left(|\det J_{\tau_\theta^{-1}}(\boldsymbol{X}|x_0,c)|\right)\right] + H\\ &\propto \mathbb{E}_{\boldsymbol{X},x_0,c}\left[\|\tau_\theta^{-1}(\boldsymbol{X}|x_0,c)\|_2^2 - \log\left(|\det J_{\tau_\theta^{-1}}(\boldsymbol{X}|x_0,c)|\right)\right]\ ,\end{aligned} \tag{16}$$

where the constant entropy of the data $H = \mathbb{E}_{\boldsymbol{X},x_0,c}\left[\log p(\boldsymbol{X}|x_0,c)\right]$ can be neglected for optimizing $\tau_\theta$.
Summarizing the derived results, we can see, that minimizing $\mathrm{KL}[p(r|x_0,c)\|q(r)]$ not only yields a probabilistic generative

model capturing the data distribution, but also forces independence between the latent variable $r$ and the conditioning $(x_0, c)$. Interestingly, we can compare our model with cVAE-based models [38, 2, 61, 65] which have to balance between two contradicting loss terms where one ensures good reconstruction properties and the other induces independence between the latent variable and the conditioning. Thus, these models face the trade-off mentioned in the main paper. As derived above, minimizing $\mathrm{KL}[p(r|x_0, c)\|q(r)]$ induces both these properties without forcing the model to balance between them (for a broader discussion, see [63]).

## E. Implementation Details of iPOKE

In this Section, we provide detailed information regarding our employed model architectures and losses as well as specifications of the models' hyperparameters. Our model is implemented using PyTorch [57]. The corresponding code is available at https://bit.ly/36h8OKr.

### E.1. Video Autoencoding Framework

We now provide details regarding the video autoencoding framework which is used to retain tractable computational cost for training our conditionally invertible model $\tau_\theta$ (cf Section 3.2 in the main paper). The autoencoding framework is trained prior to optimizing $\tau_\theta$. In the following we will first describe how to train this model, before its implementation details are given.

**Model Overview and Losses.**

*Model Overview.* As stated in the main paper, we pre-train a video-autoencoder to obtain a compact, information-preserving video encoding $z = E(\boldsymbol{X}) \in \mathbb{R}^{h \times w \times d}$ as fixed input for our invertible model $\tau_\theta$, where $E$ is a 3D-ResNet-encoder [28]. While training the autoencoding framework, $z = \hat{z}_0$ constitutes the initial hidden state of the latent GRU [12] predicting the (temporally) subsequent states $\hat{z}_i = \mathrm{GRU}(\hat{z}_{i-1})\ ,\ i = 1, \dots T$ which are mapped back into the image domain by using a 2D-decoder, thus yielding the individual images frames $\hat{x}_i = D(z_i)$ of a predicted sequence $\hat{\boldsymbol{X}}$. The autoencoding framework is visualized together with our invertible model in Fig. 2.

*Training the Autoencoder.* Here we explain the individual terms of the overall loss function of this model (9) in more detail. The first term, $\mathcal{L}_{rec}$, is an average reconstruction loss between the individual image frames of the ground-truth and predicted sequences, consisting of an $L1$-loss in the pixel space and a perceptual loss $\ell^\phi$ [20, 33].

$$\mathcal{L}_{rec} = \frac{1}{T}\left[\sum_{i=1}^{T} \parallel x_i - \hat{x}_i \parallel_1 + \sum_{i=1}^{T} \ell^\phi(x_i, \hat{x}_i)\right] \tag{17}$$

Moreover, we follow previous work [13, 73] and employ a spatial discriminator $\mathcal{D}_S$ and a temporal discriminator $\mathcal{D}_T$. Both discriminators are optimized using the hinge formulation [41, 7]. For stabilizing the GAN training, gradient penalty [52, 26] is used for the temporal discriminator. Additionally, we add a feature matching loss [72] for each individual discriminator, thus yielding the adversarial objectives

$$\mathcal{L}_{\mathcal{D}_S} = \frac{1}{T} \cdot \sum_{i=1}^{T} \left[\mathcal{D}_S(\hat{x}_i) - \ell_{F_S}(x_i, \hat{x}_i)\right]\ , \tag{18}$$

and

$$\mathcal{L}_{\mathcal{D}_T} = \mathcal{D}_T(\hat{\boldsymbol{X}}) + \ell_{F_T}(\boldsymbol{X}, \hat{\boldsymbol{X}}) \tag{19}$$

for the generator, where $\ell_{F_T}$ and $\ell_{F_S}$ denote additional feature matching losses [72].

The counterfighting loss objective for the temporal discriminator can be written as

$$\mathcal{L}_{\mathcal{D}_T, \mathcal{D}} = \rho(1 - \mathcal{D}_T(\boldsymbol{X})) + \lambda_{gp} \parallel \nabla \mathcal{D}_T(\boldsymbol{X}) \parallel_2^2 + \rho(1 + \mathcal{D}_T(\hat{\boldsymbol{X}}))\ , \tag{20}$$

where $\parallel \nabla \mathcal{D}_T(X) \parallel_2^2$ denotes the gradient penalty [52, 26] with $\lambda_{gp}$ as its weighting factor and $\rho$ the ReLU activation function. For the spatial discriminator, the objective can be formulated as

$$\mathcal{L}_{\mathcal{D}_S, \mathcal{D}} = \frac{1}{T} \sum_{i=1}^{T} [\rho(1 - \mathcal{D}_S(x_i)) + \rho(1 + \mathcal{D}_S(\hat{x}_i))]\ . \tag{21}$$

**Implementation details.**

*Encoder $E$.* The encoder $E$ is a 3D-ResNet built up of layers following the structure proposed by [28]. GroupNorm [78]

is used as a normalization layer. The number of subsequently applied layers varies with the spatial size of the input video sequence $\boldsymbol{X}$, such that a latent encoding $z$ of spatial size $h = w = 8$ is obtained in all cases. The number of channels $d$ of $z$ is chosen as $d = 32$ for the iPER [42] and Tai-Chi [64] datasets and $d = 64$ for the PP and Human3.6m [30] datasets.

*GRU.* We use a convolutional-GRU with 4 hidden layers. The number of channels remains the same for each hidden layer. Within each GRU cell, we use common convolutions with kernel size $3 \times 3$. As stated above, $z$ initializes the hidden state for all hidden layers of the GRU model. Subsequent hidden states are re-fed into the network until the intended sequence length is obtained. The input state is a learned constant of the same shape than the hidden state for all time steps.

*Decoder D.* The decoder is of a sequence of ELU-activated [14] 2D-ResNet blocks [28] consisting of convolutions with $3 \times 3$ kernels, where all but the first ResNet blocks upsample their input feature maps by a factor of two until the intended output resolution is reached. If upsampling is applied, the first convolution and the skip connection of the respective ResNet-block are replaced by a transposed convolution. Spectral normalization is applied to each of the convolutional layers contained in the decoder. Further, we use a SPADE-normalization layer which gets the start frame $x_0$ the respective sequence as input after each upsampling ResNet block.

*Discriminators.* For the static discriminator, a patch discriminator [31] is used and for the temporal discriminator a 3D ResNet-18 [28]. Both discriminators use GroupNorm [78] as a normalization layer.

*Training Details.* We train our model to reconstruct videos of sequence length $T = 10$ comprising images of spatial size $64 \times 64$ for comparison with the stochastic video synthesis baselines and $128 \times 128$ for all other experiments which are considered within the main paper. When training on sequences of spatial size $64 \times 64$, we use batch size 16 on a single NVIDIA RTX 2080 TI, whereas in the latter case, we use a single NIVDIA A100 with batch size 20. In both cases, all hyperparameters which are subsequently reported were also equally. The models are trained using Adam [35] with a learning rate of $2 \cdot 10^{-4}$, $\beta_1 = 0.5$, $\beta_2 = 0.9$, weight decay of $10^{-5}$, and exponential learning rate decay. The weighting factor of the gradient penalty for $\mathcal{D}_T$ is chosen as $\lambda_{gp} = 1.2$.

## E.2. Conditional Invertible Neural Network

In the following, we provide additional implementation details for our conditional invertible model $\tau_\theta$ and the encoders $\Phi_c$ and $\Phi_{x_0}$ processing the individual components of $(x_0, c)$.

**Encoders $\Phi_c$ and $\Phi_{x_0}$.** To obtain descriptive representations capturing all details regarding the two conditioning variables $c$ and $x_0$ we pretrain the respective encoders $\Phi_c$ and $\Phi_{x_0}$. We train $\Phi_{x_0}$ and a subsequent decoder to reconstruct image frames by using a combination of a standard $L1$ reconstruction loss and an additional perceptual loss as a training objective. $\Phi_c$ is also combined with a decoder at training time and gets a simulated user control $c$ (cf. Section 3.3) and the related source image frame $x_0$ as inputs. The obtained model is trained to reconstruct the entire flow field between $x_0$ and $x_T$. Again, a combination of $L1$ and an additional perceptual loss is used as loss function. In both cases, the decoders are used only for training and discarded afterwards.

**Conditional INN $\tau_\theta$.**

*Model Architecture.* The cINN is implemened as a convolutional normalizing flow model [60, 47, 37, 56]. As stated in the main paper, it is built up of $K$ subsequently arranged flow blocks, where the $k$-th block comprises $N_k$ sub-blocks, followed by a modified glow block [37], consisting of ActNorm [37], affine coupling [18] and shuffling layers[1]. In each sub=block, we employ masked convolutions [48], which subdivide a convolutional kernel into four systematically rotated, efficiently invertible sub-kernels, whose Jacobian is tractable. These properties are crucial when training normalizing flow model [60, 47, 37, 56], as they guarantee tractable training times and efficient sampling from $q(r)$. Moreover, masked convolutions result in the same receptive field than common convolutions and, thus, show improved expressivity compared to widely used architectures such as affine coupling layers [17, 18]. The flow overall architecture and the sub-blocks are visualized in Figure 2 in the main paper. For more details regarding masked convolutions, we refer the reader to [48].

Within all models and datasets , we use $K = 15$ flow blocks. The number $N_k$ of sub-blocks contained in the the $k$-th flow block decreases with increasing $k$. The individual number of sub blocks per flow block can be summarized by the following vector

$$\boldsymbol{N}_{1:K} = [10, 5, 5, 4, 4, 4, 3, 3, 3, 2, 2, 2, 1, 1, 1]$$

where the $k$-th component of $\boldsymbol{N}_{1:K}$ denotes the number of sub-blocks within flow block $k$.

*Training Details.* We train our cINN model using Adam [36] optimizer with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$ and weight decay of $10^{-5}$. The learning rate is annealed from zero to $10^{-3}$ for the first 500 train steps and linearly decrease afterwards, until training ends. We train the cINN models on a single NVIDIA A100 using batches comprising 40 individual training

[1] We modify these blocks by using shuffling layers instead of invertible $1 \times 1$ convolutions, as originally proposed in glow [37].

examples.

### E.3. Further Training Details of iPOKE

**Foreground-Background-Separation** We do not only train our model to infer the implications of a localized poke onto the entire object, we also intend to learn iPOKE such that it only generates object kinematics for control inputs actually acting at the object itself. Hence a poking of a pixel in background areas unrelated to the object should be ignored. We now present an additional training strategy ensuring this property.

We assume parts of the background of the videos within the train set to be static and the foreground to obtain a sufficient amount of motion, indicated by a specific magnitude of optical flow. Thus, during training, we only consider those locations with an optical flow magnitude larger than the mean of magnitudes of the flow map $F$ for sampling the controlled location $l = c_{3:4}$ for a user control with the depicted object. However, as we also want our model to separate the pixels on the object surface from those in the background, we sample the twelfth of all controlled locations in each epoch out of background pixels and construct artificial user controls by sampling the magnitudes and angles at these locations from the locations within the foreground. If our model gets such an artificial poke as input, it is trained to reconstruct a still sequence obtained by repeating the source image $x_0$ $T$ times. This strategy is applied both to train the autoencoding framework described in Sec. E.1 and the cINN model $\tau_\theta$. Thus, the model learns to separate the pixels comprising the object from those in the background, as indicated by the video examples in Section B, where we show examples for such artificial user controls.

## F. Implementation Details of Baselines

**Video Prediction Models.** For comparison, we implemented the competing stochastic video prediction baselines [40, 10, 24] based on the officially provided code from Github [2][3][4]. As no pretrained models are available for the utilized datasets, we trained models from scratch for all competitors except for SRVP [24], which provide a pretrained model for the Human3.6m [30] dataset. All models are trained to predict sequences of length 10 and spatial size $64 \times 64$ based on two context frames. For models trained from scratch, we used the hyperparameters proposed in the respective publications. For SAVP [40] we further tried all individual hyperparameter-settings which are available in the official code repository[4], but could not manage to obtain any visual diversity.

**Controlled Video Synthesis Models.** Both controllable video synthesis baselines we compare iPOKE with [27, 6] are implemented based on the official code repositories[5][6] and the provided hyperparameters for generating videos of spatial size $128 \times 128$ on all used datasets. For [27], due to the lack of pretrained weights, we trained the new models on the considered datasets, contrasting II2V [6], where we entirely used the available pretrained models.

For the method of Hao et al [27] we follow their proposed procedure to predict videos $[x_0, \dots, x_T]$ of length $T$: we first sample $k$ flow vectors between $x_0$ and $x_1$. We then extend these initial vectors to discrete trajectories of length $T-1$ in the image space by tracking the initial points using consecutive flow vectors between $x_i$ and $x_{i+1}$. Videos are finally obtained by successively warping $x_0$ using the shift vectors from the trajectories. We follow the official protocol from [27] and set $k = 5$. The motion trajectories are generated from the same optical flow which was used to train our own model.

For the model trained on ground truth keypoints on the iPER dataset, we constructed the motion trajectories based on the shifts between the individual keypoints of the source frame and all remaining frames of the considered video sequence.

**Variational Counterpart of iPOKE.** We here report the implementation details of the cVAE-based counterpart of iPOKE, which we compared our model to in the ablation study in Section 4.3. The model is composed of the individual parts of the autoencoding framework of iPOKE as described in Sec. E.1. Thus, the cVAE-grounded baseline consists of a 3D-Encoder, a convolutional GRU and a 2D-decoder following the respective architectures presented in Section E.1. Additionally, we use two encoders to individually encode the components of the conditioning $(x_0, c)$ with the same architecture than $\Phi_{x_0}$ and $\Phi_c$ as described in Section E.2. All these components of the overall model have the same inputs and outputs as the corresponding components of iPOKE. The differences between the two models are *i)* the regularization of the video encoding $z$ towards the standard normal prior by using the reparameterization trick [38], as usually done in cVAEs [38, 61], instead of using a cINN operating on the latent level, and *ii)* the initialization of the hidden state of the latent GRU, which are here the stacked features of the reparametrized video encoding and the representations of the source image $x_0$ and the user control $c$. For iPOKE, in

[2] https://github.com/edouardelasalles/srvp
[3] https://github.com/facebookresearch/improved_vrnn
[4] https://github.com/alexlee-gk/video_prediction
[5] https://github.com/zekunhao1995/ControllableVideoGen
[6] https://github.com/CompVis/interactive-image2video-synthesis

contrast, the initial hidden state is only the video representation $z = \tau_\theta(r|x_0, c)$.
The losses for the model are exactly similar to those of our video autoencoding framework as summarized in Eq. (9), except for an additional KL-loss ensuring the mentioned regularization of the video representation towards the standard normal. We tried different options for the weighting factor $\lambda_{KL}$ of this KL-term, as its choice has a major influence on the performance of the trained model and, thus, needs to be selected carefully [11, 83]. The best performance was obtained by choosing $\lambda_{KL} = 0.1$. Additionally, to avoid posterior collapse [46], we use KL-annealing for 5 epochs, before the final value of $\lambda_{KL}$ is reached. Similar to our proposed autoencoding model for iPOKE, the VAE-baseline is trained using Adam [36] with a learning rate of $2 \cdot 10^{-4}$, $\beta_1 = 0.5$, $\beta_2 = 0.9$, weight decay of $10^{-5}$, and exponential learning rate decay. All remaining hyperparameters were chosen to be equal to those of the corresponding iPOKE-model for video sequences of spatial size $64 \times 64$.

## G. Evaluation Details

**Motion Consistency.** To compute the FVD-score [68] for a given model, we generated 1000 video sequences and sampled 1000 random videos of the same length from the ground truth data. Both the real and the generated examples are the input to the I3D [8] model which was pretrained on the Kinetics [9] dataset. Subsequently their distributions in the I3D feature space are compared resulting in the reported FVD-scores. For all models we concatenate the last conditioning frame (for our model the input frame) with the generated sequence.
**Synthesis Quality.** All reported accuracy metrics are based on 1000 predicted video sequences and the corresponding ground truth videos. As these metrics are calculated based on individual image frames, we compare each frame of a generated sequence with its corresponding frame in the ground truth sequence, resulting in $T$ scores for a predicted video of length $T$, which are subsequently averaged to obtain a scalar value per sequence.
**Motion Diversity.** To evaluate the diversity for each model, we generated 5 realizations for each video resulting in $5 \times 1000$ videos. The diversity is then computed between the different realizations of the video using LPIPS[82] and MSE in the pixel space. We did not notice any change in numbers when using more realizations.