KEYIN: Discovering Subgoal Structure with Keyframe-based Video Prediction

Karl Pertsch [*1] **Oleh Rybkin** [*2]
Jingyun Yang [1] **Konstantinos G. Derpanis** [3] **Joseph Lim** [1] **Kostas Daniilidis** [2] **Andrew Jaegle** [2]

Abstract

Real-world image sequences can often be naturally decomposed into a small number of frames depicting interesting, stochastic moments (its *keyframes*) and all the deterministic frames in between. In image sequences depicting trajectories to a goal, keyframes can be seen as capturing the *subgoals* of the sequence as they depict the high-variance moments of interest that ultimately led to the goal. In this paper, we introduce a video prediction model that discovers the keyframe structure of image sequences in an unsupervised fashion. We do so using a hierarchical Keyframe-Intermediate model (KEYIN) that stochastically predicts keyframes and their offsets in time and then uses these predictions to deterministically predict the intermediate frames. We propose a differentiable formulation of this problem that allows us to train the full hierarchical model using a sequence reconstruction loss. We show that our model is able to find meaningful keyframe structure in a simulated dataset of robotic demonstrations and that these keyframes can serve as subgoals for planning. Our model outperforms other hierarchical prediction approaches for planning on a simulated pushing task.

1. Introduction

The interesting structure in real-world image sequences can often be compactly described in terms of a sparser sequence of informative frames selected from such sequences. In animation, these informative frames are called *keyframes*. When creating a sequence, lead animators first draw keyframes depicting the important changes in the sequence and then pass the sparse keyframe sequence to other animators, knowing they can interpolate between the keyframes to flesh out the correct, finished animation. For the full sequence to be depicted faithfully, keyframes must be informative about the underlying dynamics of the sequence. For example, consider the settings depicted in Fig. 1. In domains like billiards or tennis, it is simple to reconstruct the trajectory of the ball given only a description of the moments when the ball strikes a surface. However, it would be difficult to reconstruct the trajectory given a description of the ball at other times. In this light, a natural strategy to predict the full trajectory of the ball is to first estimate the points of contact and then interpolate between them using an appropriate, simple model.

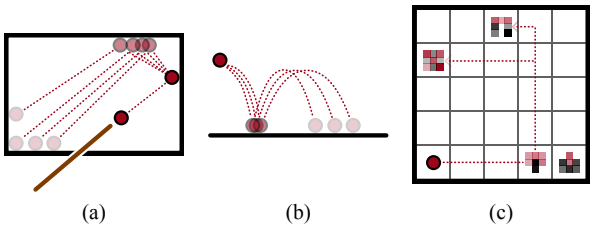

Figure 1. A variety of sequences exhibit an apparent keyframe structure. In billiards and tennis (a, b), the motion of the ball is well-defined and easily predictable in-flight, but the points of contact are difficult to model. These sequences can be compactly described in terms of the frames containing bounce points, because the motion between these points is largely deterministic. Similarly, in many sequences depicting behavior, the frames depicting the subgoals of a sequence are difficult to predict (c). In the example shown, an agent (denoted by the circle) must collect a subset of the objects in a gridworld environment. If the agent's subgoals are known the agents full trajectory can be recovered using an interpolation strategy.

In this work, we use keyframe-based video prediction to address the problem of discovering subgoal structure in image sequences depicting behavior. Our approach is motivated by the observation that a keyframe structure also exists in trajectories executed by intelligent agents performing temporally extended tasks, where keyframes are determined by the subgoal structure of the task (Fig. 1, right). Real-world demonstration sequences often take this form, and such sequences can be very effective at guiding learning once a subgoal sequence is discovered (Konidaris et al., 2012; Niekum et al., 2015; Kroemer et al., 2015; Hausman

[*]Equal contribution [1]University of Southern California [2]University of Pennsylvania [3]Ryerson University. Correspondence to: Karl Pertsch <pertsch@usc.edu>, Oleh Rybkin <oleh@seas.upenn.edu>.

Preprint. Supplementary video results can be found here: https://sites.google.com/view/keyin.

et al., 2017). Indeed, much of the difficulty in learning complex tasks lies in determining which subtask to solve, as the subpolicies that solve each subtask can often be more easily modeled individually (Teh et al., 2017; Ghosh et al., 2018). Sensory sequence analysis techniques, such as video prediction methods, have a key role to play in enabling this, especially if they can capture hierarchical temporal structure.

A large body of previous work has addressed the challenge of learning to predict image sequences (Villegas et al., 2017; Vondrick et al., 2016; Srivastava et al., 2015; Mathieu et al., 2016). However, most previous approaches predict trajectories one frame at a time without attempting to capture hierarchical temporal structure. There are several ways to model this structure, depending on the goal of prediction. Recently, Neitz et al. (2018); Jayaraman et al. (2019) proposed models that predict the future frames that have lowest uncertainty. The resulting models predict a small number of easily predictable "bottleneck" frames through which any possible prediction must pass. While finding the bottleneck frames may simplify the prediction task by ignoring inconsequential stochasticity between bottlenecks, it may also make it more difficult to reason about what happens between bottlenecks. For finding planning subgoals, it is often beneficial to select the frames that are more informative about the trajectory, as by describing the trajectory in between them such subgoals may provide richer guidance about how to reach them. Instead of predicting bottleneck frames exclusively, we propose to predict the keyframes that are best suited for the subsequent prediction of the entire video. In contrast to methods that predict the least uncertain frames, our model predicts the frames that lead to the largest information gain because they enable deterministic prediction of the entire sequence. We show that this design allows our method to be applied both to dense video prediction and to planning tasks.

We propose a neural network architecture for modeling video sequences that exploits the keyframe structure of image sequences by including two modules: one module predicts keyframes together with corresponding interframe offsets between keyframes, and the other deterministically predicts the intermediate frames between the keyframes. Using this structure, the changes in the sequence are first modeled in terms of a set of keyframes that summarizes the video, while temporally finer-scale predictions are filled in afterwards. We call this the Keyframe-Intermediate model or KEYIN. We propose a continuous relaxation of the temporally discrete prediction problem by allowing the predicted time offsets between the keyframes to be "soft" distributions over discrete time steps. The soft objective allows us to train the model with just the video reconstruction loss: the network is trained to identify the timesteps containing keyframes that best describe the entire video sequence, such that the intermediate predictions that use these keyframes achieve the lowest reconstruction loss.

As illustrated in Fig1, the keyframes of a sequence can be viewed as a summary of its key moments. We show that when our method is applied to videos, it produces keyframes summarizing the key moments of the sequence. Furthermore, we show that these keyframes can be treated as subgoals in a planning task. The subgoals generated by our model facilitate planning as they allow us to partition the planning task into shorter parts that correspond to single subtasks.

In summary, we formulate a keyframe-based approach to discovery of sequence structure using stochastic video prediction. To this end, we propose a differentiable loss that allows us to train the model using only a video reconstruction loss. We show that our model discovers meaningful temporal structure on a toy dataset with stochastic dynamics and on a simulated robotic manipulation dataset. Furthermore, we show that by using the keyframes as subgoals in a planning procedure, we outperform baselines that predict bottleneck-based subgoals or subgoals with constant offset.

2. Related work

Stochastic video prediction. Models for variational video prediction (Babaeizadeh et al., 2018; Denton & Fergus, 2018; Lee et al., 2018) learn to capture the multimodal distribution of future images given a sequence of past frames by using a recurrent conditional variational autoencoder (Chung et al., 2015), itself derived from the variational autoencoder (VAE, Kingma & Welling (2014); Rezende et al. (2014)). We build on variational inference methods and show how they can be used to model the distribution of possible keyframes in a video sequence.

Video prediction for planning and control. We build on recent work that explored applications of learned predictive models of videos to planning and control. Chiappa et al. (2017); Oh et al. (2015); Finn et al. (2016) propose models that predict the consequences of actions taken by an agent given its control output. Recent work Byravan et al. (2017); Hafner et al. (2018); Ebert et al. (2018) has shown that visual model predictive control based on such models can be applied to a variety of different settings. In this work, instead of learning a predictive model from random manipulation data, we learn to model the hierarchical structure of the dynamics of demonstration data, which allows us to plan more efficiently.

Discovering temporal structure Neitz et al. (2018); Jayaraman et al. (2019) propose models that find and predict "bottlenecks" in video — i.e. the frames that have to be passed to reach a goal image from an initial image. In con-

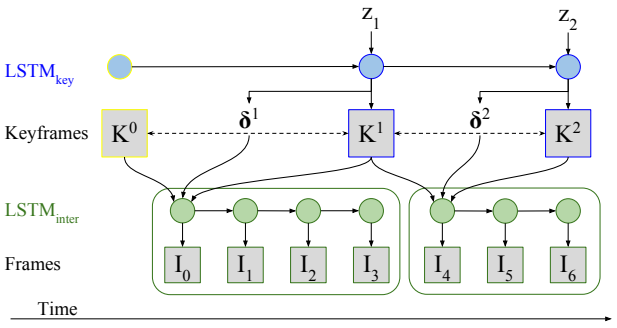

Figure 2. Overview of our keyframe-based prediction approach. The keyframe network LSTM$_{\text{key}}$ (shown in blue), predicts a set of keyframes $\hat{K}^t$ of the sequence and the length of the intervals between them δ^t. LSTM$_{\text{inter}}$ (shown in green) takes two consecutive keyframes and the interframe offset between them as input and predicts the intermediate frames I_i.

trast to these models, which predict the frames that are least uncertain and so most easily predicted, our model predicts keyframes that allow the entire video to be easily produced by deterministic neural interpolation. Our model is also designed to be able to predict full videos, instead of keyframes alone, and it can predict the times at which the keyframes occur. This makes it more suitable for planning and other tasks that benefit from reconstructing frame-by-frame dynamics.

In parallel to our work, Kipf et al. (2018) propose a strikingly similar method for sequence segmentation via variational inference with continuous relaxation of segment boundaries. Kipf et al. use this model to train an RL agent by recovering subtasks from demonstrations (in the spirit of the options framework for hierarchical RL (Sutton et al., 1999)), while we focus on leveraging models for video prediction and planning within a learned latent space.

3. Approach

3.1. Keyframe-based video prediction

Our model takes the form of a conditional variational autoencoder (Sohn et al., 2015) that learns to produce a distribution of possible future sequences $I_{0:N}$ (the indexing range notation includes all items up to but not including the last item) given a conditioning sequence $I_{-T:0}$ by dividing the produced sequence $\hat{I}_{0:N}$ into M segments and producing each segment individually.

To achieve this, we use several recurrent modules described in this section. We use a multilayer Long Short-Term Memory network (LSTM, Hochreiter & Schmidhuber (1997)) for each module. We encode the conditioning frames with LSTM$_{cond}$, which initializes the keyframe predictor LSTM$_{key}$. The approximate inference network LSTM$_{inf}$ observes and encodes the future sequence $I_{0:N}$, where N is the total number of frames in the future. At each step t, LSTM$_{key}$ observes a latent variable z^t sampled from the output of LSTM$_{inf}$ via an attention mechanism, and produces the next keyframe embedding $K^{e,t}$, where "e" stands for "embedding". Finally, an instance of LSTM$_{inter}$ is initialized for every pair of keyframes to produce the segment between the keyframes $(\hat{I}_i^{e,t})_i$.

The video frames are first processed with a convolutional encoder module $I_i^e = \text{CNN}_{enc}(I_i)$, and the frame embeddings are decoded with a convolutional decoder $\hat{I}_i^t = \text{CNN}_{dec}(\hat{I}_i^{e,t})$. The overall video prediction model is illustrated in Fig. 2.

Keyframe prediction. LSTM$_{key}$ predicts a sequence of keyframe embeddings $(\hat{K}^{e,t})_{t\leq M}$ and corresponding distributions of the interframe offsets $(\delta^t_{0:S})_{t\leq M}$, where M is the maximal possible number of keyframes and S is the maximum possible distance between two keyframes. We condition LSTM$_{key}$ on a sequence of T initial frames to let the model estimate the motion via LSTM$_{cond}$. LSTM$_{cond}$ produces the initial state of LSTM$_{key}$: $\text{Init}^t_{key} = \text{Final}^t_{cond}$. The first keyframe K^0 is given by the last frame in the input sequence.

Intermediate prediction. The output of LSTM$_{key}$ is passed to the intermediate LSTM, LSTM$_{inter}$, which predicts the frames that fill the gaps between keyframes. LSTM$_{inter}$ produces S frames $(\hat{I}_i^t)_{i\leq S}$ for each segment t. The initial state of LSTM$_{inter}$ is computed as a feedforward function of the corresponding segment information: $\text{Init}^t_{inter} = \text{MLP}_{init\_inter}(\hat{K}^{e,t-1}, \hat{K}^{e,t}, \hat{\delta}^t_{0:S})$. We reinitialize LSTM$_{inter}$ for each segment.

Stochastic prediction. As shown in (Babaeizadeh et al., 2018; Denton & Fergus, 2018; Lee et al., 2018), deterministic video prediction methods produce blurry, unrealistic images on stochastic data. To model the stochasticity of a dataset, we use the recurrent latent variable approach. By using a latent variable model, we can learn to predict the distribution of possible video continuations given the beginning of a video.

We condition the predictive model $\{\text{LSTM}_{key}, \text{LSTM}_{inter}\}$ on a sequence of latent variables $(z^t)_{t\leq M}$, which at training time are produced by an approximate inference network LSTM$_{inf}$. LSTM$_{inf}$ observes the full sequence of $M \times S$ frames and outputs $(K_j^{e,inf}, \tau_j)_{j\leq M\times S}$, where $K^{e,inf}$ is an embedding used to compute an attention weight and the τ_j are values to be attended over. We compute the posterior distribution over z^t using a key-value attention mechanism (Bahdanau et al., 2015; Luong et al., 2015):

$$a_{t,j} = \exp(d(\hat{K}^{e,t-1}, K_j^{e,inf})) \tag{1}$$

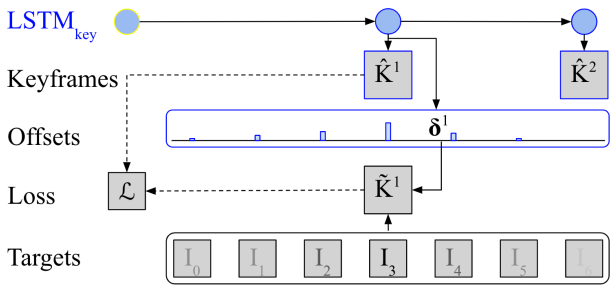

Figure 3. Soft loss shown for the first keyframe. The blue circles represent LSTM$_{key}$, which produces the keyframes $\hat{K}^t$ and the interframe offsets. The offset distribution δ^t specifies how likely a certain ground truth image is to be matched with the produced keyframe. The target image $\tilde{K}^t$ is composed of the ground truth images by linearly weighting them with this distribution. Finally, we use a reconstruction loss between the produced image $\hat{K}^t$ and the soft target $\tilde{K}^t$.

$$\mu^t, \sigma^t = (\sum_j a_{t,j}\tau_j) / \sum_j a_{t,j}. \tag{2}$$

The distance metric, d, is the inner product. We use a Gaussian approximate posterior and unit Gaussian prior distributions.

To produce a sequence given the output of the predictive model, we choose the interframe offset for keyframe t as $\hat{\delta}^t = \arg\max_i \delta_i^t$. We then compose the predicted sequence $\hat{I}_{0:N}$ as $(\hat{I}^t_{0:\hat{\delta}^t})_t$. Note that $N < M \times S$: if N is equal to $M \times S$, the network would be forced to always predict the maximum number of segments and frames in the segment. At training time, $\hat{I}_{0:N}$ is *not* used for the loss directly, as detailed in the next section.

3.2. Soft loss by linear interpolation in time

The temporal spacing of keyframes is not uniform within a sequence: some parts of the trajectory might contain more stochastic events than the others. We want the network to be able to adjust the keyframes it predicts to the sequence at hand by training it to dynamically predict the best offset δ^t. To allow this and still allow end-to-end differentiability, we propose a continuous relaxation of the reconstruction loss with discrete time offsets. To arrive at a continuous loss, we produce soft target frames $\tilde{K}^t$ by linearly interpolating between all possible targets for the predicted keyframes $\hat{K}^t$ weighted with the corresponding offset probabilities $\tilde{\delta}^t_j$, and soft target frames $\tilde{I}_j$ for ground truth frames I_j weighted with $\tilde{\delta}^t_{i,j}$ respectively.

Similar continuous relaxation strategies have been previously introduced for images by the Spatial Transformer Network (Jaderberg et al., 2015) and RNN outputs in the Adaptive Computation Time model (Graves, 2016). We note that a continuous relaxation might not result in a valid trained model if the network at convergence does not output distributions δ^t that are close to one-hot. In practice, we did not observe this problem for our experiments as our networks always converged to a valid solution. A continuous relaxation allows us to train the neural network efficiently without the need of sampling-based methods like REINFORCE (Williams, 1992). The intuition behind the loss is depicted in Fig. 3.

Keyframe targets. To produce a keyframe target, $\tilde{K}^t$, we first compute the discrete distribution $\tilde{\delta}^t$ over possible interframe offsets from the beginning of the predicted sequence by iteratively convolving the sequence of individual distributions δ^t. The values of this distribution are used to linearly interpolate between the ground truth images to produce the keyframe target: $\tilde{K}^t = \sum_j \tilde{\delta}^t_j I_j$. We further compute the probability that $\hat{K}^t$ is inside the predicted sequence $c^t = \sum_{j \le N} \tilde{\delta}^t_j$. We weigh the keyframe loss with this probability as we want to allow the network to leave further keyframes unused if the predicted sequence at hand has less than M segments. This allows the network to predict sequences with varying number of keyframes.

Keyframe Losses. We compute keyframe losses for both the keyframe embeddings $\hat{K}^{e,t}$ and for decoded keyframe estimates $\hat{K}^t$. Finally, we normalize the loss by the expected number of keyframes $\sum_t c^t$ predicted by the network, as we want the loss to be invariant to this quantity. The final keyframe objective combines the losses on the embedding targets, image targets, and prior divergence:

$$\begin{aligned}\mathcal{L}_{key} = (\sum_t & c^t \beta_{ki} ||\hat{K}^t - \tilde{K}^t||^2 \\ & + c^t \beta_{ke} ||\hat{K}^{e,t} - \tilde{K}^{e,t}||^2 \\ & + c^t \beta \mathrm{KL}[\mathcal{N}(\mu^t, \sigma^t) || \mathcal{N}(0, I)]) / \sum_t c^t.\end{aligned} \tag{3}$$

Interpolating frames. In a similar fashion, we define a soft loss for the generated interpolating frames $\hat{I}^t_i$. This time, however, instead of producing targets for the predicted frames, we want to produce a target image for each *ground truth* frame, as we want to predict all ground truth frames. Similarly to the previous section, the targets for ground truth images are given as an interpolation between generated images $\tilde{I}_j = (\sum_{t,i} \tilde{\delta}^t_{i,j} \hat{I}^t_i) / \sum_{t,i} \tilde{\delta}^t_{i,j}$. Here, $\tilde{\delta}^t_{i,j}$ is the probability of the i-th predicted image in segment t to have an offset of j from the beginning of the predicted sequence. Since such probabilities do not have to sum to one (e.g. if the network predicts a sequence that is too short), we normalize the target image by the cumulative probability, $\sum_{t,j} \tilde{\delta}^t_{i,j}$.

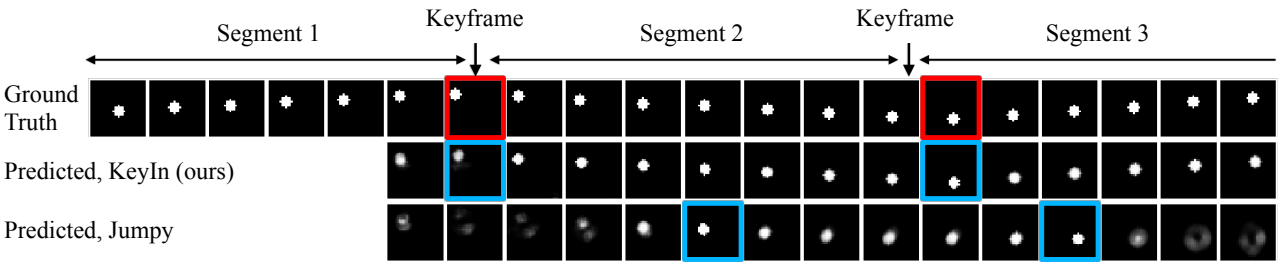

Figure 4. reconstructed sequences by KEYIN and Jumpy ablation with constant interframe offset. The prediction is conditioned on the first five ground truth frames. Only half of the predicted sequence is shown for clarity. Movement direction changes are marked with red in the ground truth sequence and predicted keyframes are marked in blue. We see that KEYIN can correctly reconstruct the motion as it selects an informative set of keyframes. The Jumpy model fails to reconstruct the motion as it cannot select the correct keyframes. The sequence the baseline predicts is missing both direction changes since they cannot be inferred from the boundary frames.

The final interpolation loss is:

$$\mathcal{L}_{inter} = \sum_{t,i} ||I_j - \tilde{I}_j||^2, \tag{4}$$

and the full loss for our model is:

$$\mathcal{L}_{total} = \mathcal{L}_{key} + \mathcal{L}_{inter}. \tag{5}$$

Training procedure. To train our model, we want the keyframe embeddings $\hat{K}^{e,t}$ to only describe the corresponding frame. However, we found that when training the keyframe and the intermediate predictor together, this rarely happens, as the network learns to embed information about the segment into $\hat{K}^{e,t}$. If $\hat{K}^{e,t}$ contains information about the entire segment, the positioning of the keyframe no longer matters since the segment can be correctly reconstructed from any position. To prevent this, we train our model with a two-stage procedure. First, the intermediate predictor LSTM$_{inter}$ is trained to interpolate between frames sampled with random offsets, thus learning interpolation strategies for a variety of different inputs. In the second stage, we freeze LSTM$_{inter}$ weights and only use it to backpropagate the error to the rest of the model. In this way, we can train the entire model to produce image sequences by using the trained interpolator LSTM$_{inter}$. We found this technique effective in preventing the network from embedding extra information in $\hat{K}^{e,t}$ since the interpolator is insensitive to such extra information. To let it cope with uncertainty when the randomly selected boundaries are not true keyframes, we train the interpolator in a stochastic manner, similarly to a seq2seq VAE model (Bowman et al., 2016). However, we want the keyframe predictor to find the keyframes that lead to deterministic interpolations. Accordingly, we sample the interpolator latent variable from the prior when training the keyframe predictor. In practice we found that intermediate predictions are only accurate if they are fully determined by the two keyframes and not dependent on other information that might be encoded in the seq2seq VAE latent.

4. Experiments

We evaluate our model on two datasets to test whether it can discover a keyframe structure and whether the discovered keyframes improve hierarchical planning. We train the interpolator on segments of two to eight frames for Structured Brownian motion data, and two to six frames for Pushing data. The KL-divergence term for the interpolator VAE is 10^{-3}. For training the keyframe predictor, we set $\beta_{ke} = 1, \beta_{ki} = 0, \beta = 510^{-2}$. We activate the produced images with sigmoid and use BCE loss to avoid saturation. The convolutional encoder and decoder both have three layers for the Structured Brownian motion dataset and four layers for the Pushing dataset. We use a simple two-layer LSTM with a 256-dimensional state in each layer for all recurrent modules. Each LSTM has a linear projection layer before and after it that projects the observations to and from the correct dimension. We use the Adam optimizer (Kingma & Ba, 2015) with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, batch size of 30, and a learning rate of $2e-4$. For more details please refer to the appendix.

Datasets. We use two simulated datasets to evaluate the ability of our model to discover keyframes and its application to planning. First, the *Structured Brownian motion* dataset consists of binary image sequences of resolution 32×32 in which a ball randomly changes directions after periods of straight movement. In the *Pushing Dataset* we use a rule-based policy to create 50k sequences of a robot arm pushing a puck towards a goal in a sequence of pushes to the other side of a wall. Both start and target position, and the placement of the wall are varying, and the individual demonstration pushes vary in length and direction, see Fig. 5 (left) for demonstration examples. The demonstrations are generated with the Mujoco simulator (Todorov et al., 2012) at a resolution of 64×64. For more details on the data generation process see the appendix.

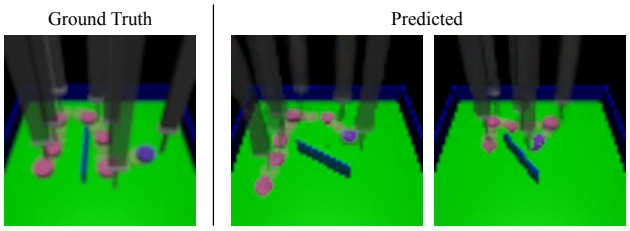

Figure 5. A training sequence and two samples from our model on the Pushing dataset. Each image shows an entire trajectory. Our model first samples the keyframes (shown in red), and then deterministically predicts the rest of the sequence. The start object position is colored purple and the robot arm is displayed only for the keyframes.

Table 1. F1 accuracy score for keyframe discovery on Structured Brownian motion and Pushing dataset, higher is better.

METHOD	BROWNIAN	PUSH
RANDOM	0.15	0.18
CONSTANT	0.17	0.23
STATIC	0.21	0.18
SURPRISE	0.73	0.10
KEYIN (OURS)	**0.84**	**0.30**

Baselines. To evaluate whether the flexible keyframe prediction is beneficial, we compare to baselines that do not use our proposed interframe offset predictor. First, we compare to *jumpy*, a model that predicts keyframes at a constant interframe offset, similar to (Buesing et al., 2018). For keyframe discovery, we compare to a *surprise* baseline based on (Denton & Fergus, 2018) that uses frame-by-frame sequential prediction to determine keyframes, and a *static* baseline that uses our proposed loss but cannot adapt the offset prediction to the sequence at hand. For planning, we also compare to TAP (Jayaraman et al., 2019) and the low-level planner that plans directly to the goal. We use the ground truth dynamics simulation for the low-level planning in all experiments.

4.1. Keyframe discovery

In this section, we validate that our model is able to detect meaningful keyframes, reconstruct the input sequence given the keyframe and model the distribution of possible video continuations given a beginning of a video.

4.1.1. STRUCTURED BROWNIAN MOTION DATA

We show the keyframe discovery results of our network on a Structured Brownian motion dataset. Videos in this dataset depict a ball that changes its angle of motion stochastically at certain points and then moves in this direction deterministically for six to eight frames until the next change. We designed the dataset to test whether our model can discover the keyframes that define the entire sequence. For our model, we set $N = 30$, $M = 6$ and $S = 10$, and condition on five input images. Note that we allow our model to use larger interframe offsets than necessary to test whether it will be able to find the correct structure. Similarly, we allow it to use a larger than necessary number of keyframes.

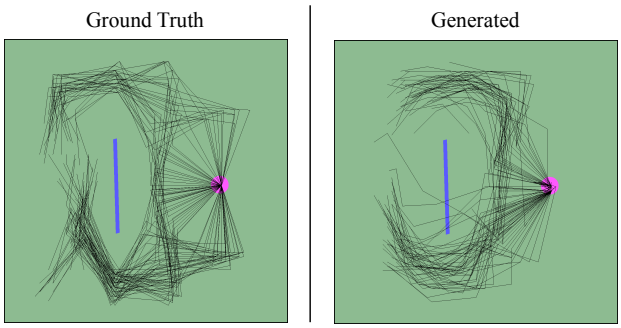

Figure 6. Sample of demonstration trajectories in the Pushing data (left) and samples from our model (right). Each black line denotes one of a 100 trajectories of the manipulated object. The obstacle is shown in blue and the initial position in pink. We see that our model covers both modes of the distribution, producing both trajectories that go to the right and to the left of the obstacle.

A keyframe model that finds the positions at which the ball changes directions will be able to learn to interpolate the motion between such keyframes. In contrast, if the model fails to select the direction change frames, interpolation will be impossible. If a sequence between two frames contains a direction change, it is impossible to infer the sequence just from the two boundary frames. Figure 4 shows a comparison of KEYIN and the *Jumpy* baseline, which is an ablation of our model that places keyframes at constant offset of 6. We see that the Jumpy model that is unable to flexibly select the correct set of keyframes produces intermediate predictions that do not correspond to the true sequence. KEYIN is able to correctly find the direction change points, as these are the keyframes that determine the structure of a sequence. Furthermore, KEYIN faithfully predicts both the appearance and motion of the intermediate frames from the found keyframes.

4.1.2. PUSHING DATA

We probe our method's performance on demonstration data on the frame sequences of the Pushing Dataset. In this dataset, there are six pushes in each sequence and each push takes four to six frames. We set $N = 30$, $M = 6$ and $S = 6$ to again allow our method some flexibility in how it places the keyframes, and condition on one initial image.

A natural set of keyframes for such sequences are the frames in between the pushes which determine the complete trajectory of the manipulated object. Figure 7 shows example frame sequences for the Pushing data and predictions by KEYIN. Our network is able to predict both the motion of objects in the scene and the interactions between them.

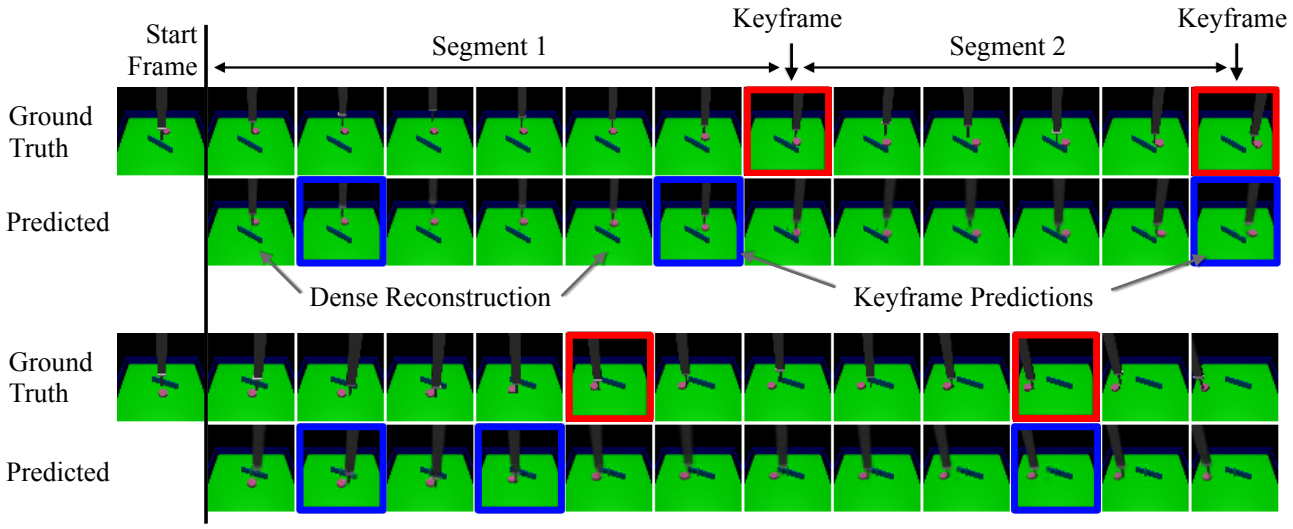

Figure 7. Example frame predictions on the Pushing dataset. In each sequence, the top row corresponds to the ground truth, and the bottom row to the predicted image sequence. The prediction is conditioned on the first five ground truth frames. Only 12 of the 30 predicted frames are shown for clarity. Movement direction changes are marked with red in the ground truth sequence. In this figure, and in general, we observe that for each direction change our network predicts a keyframe either exactly or at the timestep next to the direction change. These keyframes can effectively serve as subgoals for our planning method.

Moreover, we see that the network discovers semantically meaningful keyframes in this more complex dataset at transitions between pushes. These keyframes decompose the motion of the manipulated object into straight segments, providing for a subgoal structure that can be used for a planning task. In Fig. 6 (bottom), we show a sample from the distribution of different continuations our network predicts given the initial frame. Our network is successfully able to model the distribution of trajectories that push the object to the other side of the obstacle and are consistent with the demonstrations.[1]

On keyframe discovery, we compare to a frame-by-frame sequential prediction baseline that measures *surprise* of seeing the next frame given the previous frames and selects keyframes as top six frames t where the surprise peaks. We use a sequential stochastic predictor based on (Denton & Fergus, 2018) and measure the surprise via the KL-divergence $KL[q(z_t|I_{0:t})||p(z_t)]$ (more details in the appendix). This simple baseline determines the frames at which unexpected events happen, however, it is unable to globally reason about which frames will be the most helpful to reconstruct the entire trajectory.

In Table 1, we evaluate the frequency of keyframe discovery using the F1 score, i.e. the harmonic mean of precision and recall. We see that our method correctly recovers the keyframes more often than the baselines. We attribute low F1 scores on Pushing data to off-by-one errors in keyframe placement, and note that the baselines often make larger errors. The surprise-based baseline can correctly recover the keyframes on the simple Structured Brownian Motion dataset, but struggles on the more complex Pushing dataset.

[1]Videos, including reconstructions and samples from our model can be found at `https://sites.google.com/view/keyin`.

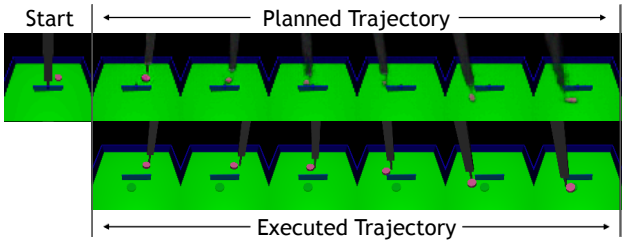

Figure 8. Planning on the pushing dataset. The top row shows the planned subgoals using KEYIN. The bottom sequence shows snapshots from a successful trajectory between the start state on the left and the goal state that is depicted transparently in each frame. The low-level execution closely follows the given subgoals and successfully reaches the goal.

4.2. Hierarchical planning

In this section, we evaluate whether our method, when applied for the task of finding subgoals in demonstration data, can successfully guide planning by predicting the planning subgoals. In the previous section, we validated that our model, when trained on such a dataset, can successfully discover keyframes in such data. We apply this model on the planning task of moving the object in the Pushing Dataset around an wall to a target position, which is specified with an image. To do this, we first find a sequence of keyframes that reaches the target and treat this as our subgoal plan. We then execute this sequence by employing a low-level planner that takes each subgoal and executes a sequence of

Algorithm 1 Planning in the subgoal space.

Input: Keyframe model KEYIN$(.,.)$, cost function c
Input: Start and target images I_0 and I_{target}
Set the sampling distribution to the prior:
$\mu_n = 0, \sigma_n = I$
for $n = 0 \dots H$ **do**
  Sample L sequences of latent variables:
  $z^{0:M} \sim \mathcal{N}(\mu_n, \sigma_n)$
  Produce subgoal plans: $\hat{K}^{0:M} = \text{KEYIN}(I_0, z^{0:M})$
  Compute cost between produced and true target:
  $c(\hat{K}^M)$
  Choose L' best plans, refit sampling distribution:
  $\mu_{n+1}, \sigma_{n+1} = \text{fit}(z'_n)$
end for
Return: Best subgoal plan $K^{0:M}$

Table 2. Planning performance on a pushing task.

METHOD	POSITION ERROR	SUCCESS RATE
INTITIAL	1.32 ± 0.06	-
RANDOM	1.32 ± 0.07	-
NO SUBGOALS	0.90 ± 0.14	15.0 %
TAP	0.80 ± 0.16	23.3 %
SURPRISE	0.64 ± 0.28	50.8 %
JUMPY	0.62 ± 0.33	58.8 %
KEYIN (OURS)	$\mathbf{0.50 \pm 0.26}$	**64.2 %**

actions that reaches the subgoal. This planning procedure is illustrated in Fig. 9.

Since our keyframe predictor network is trained as a conditional VAE, it can model the distribution of possible subgoals that are consistent with demonstration data. To construct a subgoal plan, we search this space to find the sequence of subgoals such that the final (sub)goal optimizes the planning cost. We base our planning on the Cross-Entropy Method (CEM, Rubinstein & Kroese (2004)). However, instead of planning in action space, we plan in the latent space of our model. Algorithm 1 details the process.

Algorithm 1 allows us to find a subgoal plan that is optimized for reaching the target image. To execute this plan, we need another, low-level planner which is able to reach each subgoal individually. We again employ CEM-based planning based on the ground truth dynamics model of the simulator. We compare our procedure against baselines that use the same low-level planner and thus measure only the quality of the predicted subgoals. We show an example successful planning execution using the KEYIN subgoals in Fig. 8.[2] Additional details and examples of planned trajectories are given in the appendix.

[2]Videos, including all test executions of our method and baselines, can be found at `https://sites.google.com/view/keyin`.

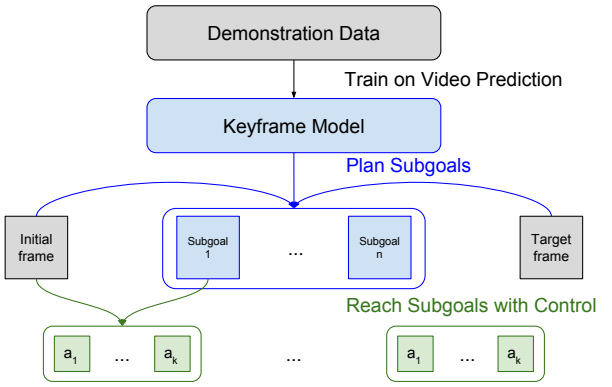

Figure 9. Hierarchical planning procedure. First, we train our KEYIN model on a dataset of demonstrations. At planning time, we can use the model to produce the keyframes between the current observation image and the goal. Finally, we use a low-level planner based on model predictive control (MPC) to reach each keyframe individually, until the final goal is reached.

We compare our method against a method that plans directly towards the final goal (No Subgoal), a method that stochastically predicts subgoals at a fixed time offset (Jumpy), and a bottleneck-based subgoal predictor (TAP, Jayaraman et al. (2019)). We evaluate all methods on the average shortest path distance, i.e. on the average of the shortest possible path between the goal position on a run and the final position of the object after the plan is executed. We choose this metric instead of the Euclidean distance to account for the presence of the wall, which can render the direct path unusable.

In Tab. 2, we see that our method outperforms previous work in terms of planning performance. All subgoal-based methods outperform the no subgoal baseline. TAP shows only a moderate increase in performance over the No Subgoal baseline, which we attribute to the fact that it fails to predict good subgoals and often simply predicts the final image as the bottleneck. This is likely due to the increased stochasticity of our dataset and the absence of the clear bottlenecks that TAP is designed to find. Our method outperforms the Jumpy and Surprise baseline as it is better able to find meaningful keyframes that make the low-level planning easier by partitioning the task down into simple subtasks.

5. Conclusion

We present a method for discovering informative keyframes in video sequences by variational video prediction. We do so using a hierarchical model, called KEYIN, that first predicts the keyframes of a sequence and their offsets in time using stochastic prediction and then interpolates the remaining intermediate frames deterministically. We show that our method discovers meaningful keyframe structure

on several datasets with stochastic dynamics, and that when used to produce planning subgoals, our method outperforms several other hierarchical prediction methods.

ACKNOWLEDGEMENTS

We thank Kenneth Chaney, Bernadette Bucher, and Nikolaos Kolotouros for computing support, and the members of the GRASP laboratory and CLVR laboratory for many fruitful discussions. We also thank Dinesh Jayaraman for help with TAP training and Frederik Ebert for help with the CEM planning. We are grateful for support through the following grants: NSF-DGE-0966142 (IGERT), NSF-IIP-1439681 (I/UCRC), NSF-IIS-1426840, NSF-IIS-1703319, NSF MRI 1626008, ARL RCTA W911NF-10-2-0016, ONR N00014-17-1-2093, and by Honda Research Institute. K.G.D. is supported by a Canadian NSERC Discovery grant.

References

Babaeizadeh, M., Finn, C., Erhan, D., Campbell, R. H., and Levine, S. Stochastic variational video prediction. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *Proceedings of International Conference on Learning Representations (ICLR)*, 2015.

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A., Jozefowicz, R., and Bengio, S. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 10–21, 2016.

Buesing, L., Weber, T., Racanière, S., Eslami, S. M. A., Rezende, D. J., Reichert, D. P., Viola, F., Besse, F., Gregor, K., Hassabis, D., and Wierstra, D. Learning and querying fast generative models for reinforcement learning. *arXiv:1802.03006*, 2018.

Byravan, A., Leeb, F., Meier, F., and Fox, D. Se3-posenets: Structured deep dynamics models for visuomotor planning and control. *Proceedings of IEEE International Conference on Robotics and Automation*, 2017.

Chiappa, S., Racanière, S., Wierstra, D., and Mohamed, S. Recurrent environment simulators. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., and Bengio, Y. A recurrent latent variable model for sequential data. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2015.

Denton, E. and Fergus, R. Stochastic video generation with a learned prior. In *Proceedings of International Conference on Machine Learning (ICML)*, 2018.

Ebert, F., Finn, C., Lee, A. X., and Levine, S. Self-supervised visual planning with temporal skip connections. In *Conference on Robotic Learning (CoRL)*, 2017.

Ebert, F., Finn, C., Dasari, S., Xie, A., Lee, A., and Levine, S. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv:1812.00568*, 2018.

Finn, C. and Levine, S. Deep visual foresight for planning robot motion. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2017.

Finn, C., Goodfellow, I., and Levine, S. Unsupervised learning for physical interaction through video prediction. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2016.

Ghosh, D., Singh, A., Rajeswaran, A., Kumar, V., and Levine, S. Divide-and-conquer reinforcement learning. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

Graves, A. Adaptive computation time for recurrent neural networks. *arXiv:1603.08983*, 2016.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

Hausman, K., Chebotar, Y., Schaal, S., Sukhatme, G., and Lim, J. J. Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2017.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukcuoglu, K. Spatial transformer networks. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2015.

Jayaraman, D., Ebert, F., Efros, A. A., and Levine, S. Time-agnostic prediction: Predicting predictable video frames. *Proceedings of International Conference on Learning Representations (ICLR)*, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2015.

Kingma, D. P. and Welling, M. Auto-encoding variational Bayes. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2014.

Kipf, T., Li, Y., Dai, H., Zambaldi, V., Grefenstette, E., Kohli, P., and Battaglia, P. Compositional imitation learning: Explaining and executing one task at a time. *arXiv:1812.01483*, 2018.

Konidaris, G., Kuindersma, S., Grupen, R., and Barto, A. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31 (3):360–375, 2012.

Kroemer, O., Daniel, C., Neumann, G., Van Hoof, H., and Peters, J. Towards learning hierarchical skills for multi-phase manipulation tasks. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2015.

Lee, A. X., Zhang, R., Ebert, F., Abbeel, P., Finn, C., and Levine, S. Stochastic adversarial video prediction. *arXiv:1804.01523*, abs/1804.01523, 2018.

Luong, T., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1412–1421, 2015.

Mathieu, M., Couprie, C., and LeCun, Y. Deep multi-scale video prediction beyond mean square error. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2016.

Neitz, A., Parascandolo, G., Bauer, S., and Schölkopf, B. Adaptive skip intervals: Temporal abstraction for recurrent dynamical models. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2018.

Niekum, S., Osentoski, S., Konidaris, G., Chitta, S., Marthi, B., and Barto, A. G. Learning grounded finite-state representations from unstructured demonstrations. *The International Journal of Robotics Research*, 34(2):131–157, 2015.

Oh, J., Guo, X., Lee, H., Lewis, R., and Singh, S. Action-conditional video prediction using deep networks in atari games. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2015.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of International Conference on Machine Learning (ICML)*, 2014.

Rubinstein, R. Y. and Kroese, D. P. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. Springer-Verlag New York, 2004.

Sohn, K., Lee, H., and Yan, X. Learning structured output representation using deep conditional generative models. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2015.

Srivastava, N., Mansimov, E., and Salakhutdinov, R. Unsupervised learning of video representations using LSTMs. In *Proceedings of International Conference on Machine Learning (ICML)*, pp. 843–852, 2015.

Sutton, R. S., Precup, D., and Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112: 181211, 1999.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. Distral: Robust multitask reinforcement learning. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2017.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Villegas, R., Yang, J., Hong, S., Lin, X., and Lee, H. Decomposing motion and content for natural video sequence prediction. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

Vondrick, C., Pirsiavash, H., and Torralba, A. Anticipating visual representations from unlabeled video. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

6. Experimental setup

Each network was trained on a single high-end NVIDIA GPU. We trained the interpolator for 100K iterations, and the keyframe predictor for 200K iterations, which took about a day in total.

7. Data collection in the MuJoCo environment

The data collection for our pushing dataset is completed in an environment simulated by MuJoCo (Todorov et al., 2012). In the environment, a robot arm initialized at the center of the table has to push an object to a goal position at the other side of a wall-shaped obstacle.

Demonstrations follow a rule-based algorithm that first samples subgoals between the initial position of the object and the goal and then runs a deterministic pushing procedure to the subgoals in order. Ground truth keyframes of the demonstrations are defined by frames at which subgoals are finished.

We subsample demonstration videos by a factor of 2 when saving them to the dataset, dropping every other frame in the trajectory and averaging actions of every two consecutive frames. For all datasets we generate for this environment following a rule-based algorithm, we only take successful demonstrations and drop the ones that fail to push the object to the goal position within a predefined horizon.

8. Details of the loss computation algorithm

We describe the details of the continuous relaxation loss computation in Algorithm 2.

Algorithm 2 Continuous relaxation loss computation

Input: Ground truth frames $I_{0:N}$, Produced frames $\hat{I}_i^t$, produced offset distributions δ^t
Compute the distribution of keyframe timesteps. For the first keyframe, $\tilde{\delta}^1 = \delta^1$.
for $t = 2 \dots M$ **do**
  Compute further $\tilde{\delta}^t$ with convolution: $\tilde{\delta}^t = \tilde{\delta}^{t-1} * \delta^t$, i.e. $\tilde{\delta}_i^t = \sum_j \tilde{\delta}_{i-j+1}^{t-1} \delta_j^t$.
end for
Compute probabilities of keyframes being within the predicted sequence: $c^t = \sum_{j \le N} \tilde{\delta}_j^t$.
Compute soft keyframe targets: $\tilde{K}^t = \sum_j \tilde{\delta}_j^t I_j$.
Compute the keyframe loss: $(\sum_t c^t ||\hat{K}^t - \tilde{K}^t||^2) / \sum_t c^t$.

Get probabilities of segments ending after particular frames: $e_i^t = \sum_{j>i} \delta_j^t$.
Get distributions of individual frames timesteps: $\tilde{\delta}_{i,j}^t \propto \tilde{\delta}_{j-i+1}^{t-1} e_i^t$.
Compute soft individual frames: $\tilde{I}_j = \sum_{t,i} \tilde{\delta}_{i,j}^t \hat{I}_i^t$
Compute the sequence loss: $\sum_{t,i} ||I_j - \tilde{I}_j||^2$.

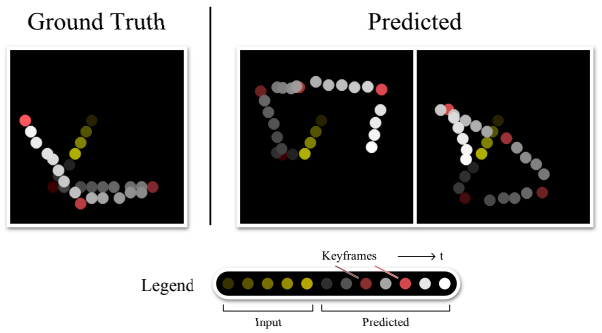

Figure 10. A training sequence and two samples from our model on the Structured Brownian motion dataset. Each image shows an entire trajectory. Our model first samples the keyframes (shown in red), and then deterministically predicts the rest of the sequence. The image resolution was enhanced for viewability.

9. Surprise baseline

Denton & Fergus (2018) observe that the variance of the learned prior of a stochastic video prediction model tends to spike before an uncertain event happens. We use a similar observation to find the points of high uncertainty for the Surprise baseline. We use the KL divergence between the prior and the approximate posterior $KL[q(z_t|I_{0:t})||p(z_t)]$ to measure the surprise. This quantity can be interpreted as the number of bits needed to encode the latent variable describing the next state, and will be larger if the next state is more stochastic.

We train a stochastic video prediction network SVG-FP (Denton & Fergus, 2018) with the same architecture of encoder, decoder and LSTM as our model. We found that selecting the peaks of suprise works the best for finding true keyframes. The procedure that we use to select the keyframes is described in Algorithm 3. In order to find the keyframes in a sequence sampled from the prior, we run the inference network on the produced sequence.

10. Planning Algorithm

To apply the KEYIN model for planning, we use an approach for visual servoing that is outlined in Algorithm 1. At the initial timestep, we use the cross-entropy method (CEM) (Rubinstein & Kroese, 2004) to select subgoals for the task. To do so, we sample $\tilde{M}$ latent sequences $\boldsymbol{z}_0$ from the prior $\mathcal{N}(0, I)$ and use the keyframe model to retrieve $\tilde{M}$ corresponding keyframe sequences $\boldsymbol{\tau}^{key}$, each with L frames. We define the cost of an image trajectory as the distance between the target image and the final image of each keyframe sequence defined under a domain-specific distance function (see below). In the update step of the Cross Entropy Method (CEM) algorithm, we rank the trajectories based on their cost and fit a diagonal Gaussian distribution to the latents $\boldsymbol{z}'$ that generated the $\tilde{M}' = r\tilde{M}$ best sequences. We repeat the procedure above for a total of N iterations.

We define the cost between two frames used during planning as the euclidean distance between the center pixels of the object in both frames. We recover the center pixel via color-based segmentation of the object. While this cost function is designed for the particular planning environment we are testing on, our algorithm can be easily extended to use alternative, more domain-agnostic cost formulations that are proposed in the literature (Finn & Levine, 2017; Ebert et al., 2017; 2018).

After subgoals are selected, we use a cross-entropy method (CEM) based planner to produce rollout trajectories. Similar to the subgoal generation procedure, at each time step, we initially sample M action sequences $\boldsymbol{u}_0$ from the prior $\mathcal{N}(0, I)$ and use the ground truth dynamics of the simulator to retrieve M corresponding image sequences $\boldsymbol{\tau}$, each with l frames[3]. We define the cost of an image trajectory as the

[3]In practice we clip the sampled actions to a maximal action range $[-a_{\max}, +a_{\max}]$ before passing them to the simulator.

Algorithm 3 Selecting keyframes via Surprise

Input: Input sequence $I_{0:N}$, Stochastic Video Prediction model $SVG(.)$
Run the inference network over the sequence: $q(z_{0:N}|I_{0:N}) = SVG(I_{0:N})$
Get the surprise measure: $s_t = KL[q(z_t|I_{0:t})||p(z_t)]$
Find the set of peak surprise points S where: $s_t > s_{t+1} \wedge s_t < s_{t-1}$
if $|S| < M$ **then**
  add $M - |S|$ maximum surprise points to S.
end if
Return: M maximum surprise peaks: $\arg\max \mathrm{M}_i \, S_i$

Table 3. Hyperparameters for the visual servoing experiments.

Servoing Parameters	
Max. servoing timesteps (T_{max})	60
Max. per subgoal timesteps ($T_{s,\text{max}}$)	10
Keyframe prediction horizon (L)	6
# keyframe sequences ($\tilde{M}$)	200
Servoing horizon (l)	8
# servoing sequences (M)	200
Elite fraction ($r = M'/M$)	0.05
# refit iterations (N)	3
max. action (a_{max})	1.0

distance between the target image and the final image of each trajectory. In the update step, we rank the trajectories based on their cost and fit a diagonal Gaussian distribution to the actions $\boldsymbol{u}'$ that generated the $M' = rM$ best sequences. After sampling a new set of actions $\boldsymbol{u}_{n+1}$ from the fitted Gaussian distributions we repeat the procedure above for a total of N iterations.

Finally, we execute the first action in the action sequence corresponding to the best rollout of the final CEM iteration. The action at the next time step is chosen using the same procedure with the next observation as input and reinitialized action distributions. The algorithm terminates when the specified maximal number of servoing steps T_{max} has been executed or the distance to the goal is below a set threshold.

We switch between planned subgoals if (i) the subgoal is reached, i.e. the distance to the subgoal is below a threshold or (ii) the current subgoal was not reached for $T_{s,\text{max}}$ execution steps. We use the true goal image as additional, final subgoal.

The parameters used for our visual servoing experiments are listed in Tab. 3.

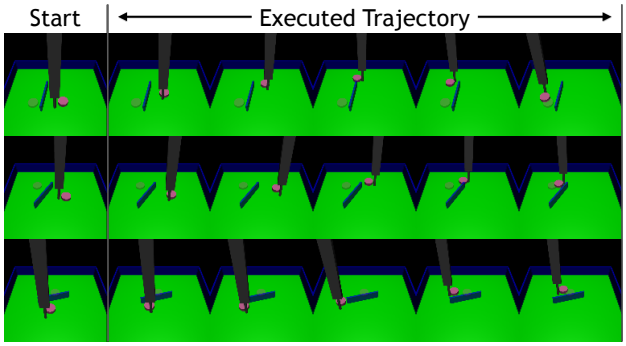

Figure 11. Sample planning task executions from the test set. From a start state depicted on the left, the robot arm successfully pushes the object into the goal position (semi-transparent object) guided by the KEYIN subgoals. The right side of the figure shows intermediate frames of the execution trajectories.

Algorithm 4 Visual Servoing with Video Prediction Model

Input: Keyframe model $\hat{K}_{1:L} = \text{LSTM}_{key}(I, z_{1:L})$
Input: Video prediction model $\hat{I}_{t:t+l} = \text{LSTM}_{inter}(I_{1:t-1}, u_{2:t+l})$
Input: Subgoal index update heuristics $\text{ix}_{t+1} = f(\text{ix}_t, I_t, K_{1:L})$
Input: Start and goal images I_0 and I_{goal}
Initialize latents from prior: $\boldsymbol{z}_0 \sim \mathcal{N}(0, I)$
for $n = 0 \dots N$ **do**
  Rollout keyframe model for L steps, obtain $\tilde{M}$ future keyframe sequences $\boldsymbol{\tau}^{key} = \hat{\boldsymbol{K}}_{1:L}$
  Compute distance between final and goal image: $c(\tau^{key}) = \text{dist}(\hat{K}_L, I_{\text{goal}})$
  Choose $\tilde{M}'$ best sequences, refit Gaussian distribution: $\boldsymbol{\mu}^{key}_{n+1}, \boldsymbol{\sigma}^{key}_{n+1} = \text{fit}(\boldsymbol{K}'_n)$
  Sample new latents from updated distribution: $\boldsymbol{z}_{n+1} \sim \mathcal{N}(\boldsymbol{\mu}^{key}_{n+1}, \boldsymbol{\sigma}^{key}_{n+1})$
end for
Feed best sequence of latents into keyframe model to obtain subgoals: $K^*_{1:L} = \text{LSTM}_{key}(I_0, z^*_{N,0})$
Set current subgoal to $\text{ix}_1 = 1$
for $t = 1 \dots T$ **do**
  Perform subgoal update $\text{ix}_t = f(\text{ix}_{t-1}, I_{t-1}, K^*_{1:L})$
  Initialize latents from prior: $\boldsymbol{u}_0 \sim \mathcal{N}(0, I)$
  for $n = 0 \dots N$ **do**
    Rollout prediction model for l steps, obtain M future sequences $\boldsymbol{\tau} = \hat{\boldsymbol{I}}_{t:t+l}$
    Compute distance between final and subgoal image: $c(\tau) = \text{dist}(\hat{I}_{t+l}, K_{\text{ix}_t})$
    Choose M' best sequences, refit Gaussian distribution: $\boldsymbol{\mu}_{n+1}, \boldsymbol{\sigma}_{n+1} = \text{fit}(\boldsymbol{u}'_n)$
    Sample new latents from updated distribution: $\boldsymbol{u}_{n+1} \sim \mathcal{N}(\boldsymbol{\mu}_{n+1}, \boldsymbol{\sigma}_{n+1})$
  end for
  Execute $u^*_{N,0}$ and observe next image I_t
end for