KEYIN: DISCOVERING SUBGOAL STRUCTURE WITH KEYFRAME-BASED VIDEO PREDICTION

Karl Pertsch[*,1], **Oleh Rybkin**[*,2], **Jingyun Yang**[1],
Konstantinos G. Derpanis[3,4], **Joseph Lim**[1], **Kostas Daniilidis**[2], **Andrew Jaegle**[2,†]
[1]University of Southern California
[2]University of Pennsylvania
[3]Ryerson University
[4]Samsung AI Centre Toronto
[†]Now at DeepMind

ABSTRACT

Real-world image sequences can often be naturally decomposed into a small number of frames depicting interesting, highly stochastic moments (its *keyframes*) and the low-variance frames in between them. 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 picked from such sequences (examples in Figure 1). 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 times 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

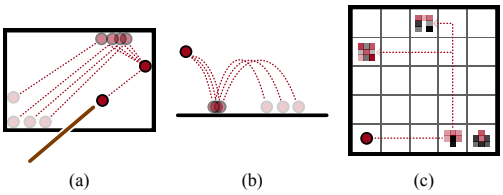

Figure 1: A variety of sequences exhibit an apparent keyframe structure. These sequences can be compactly described in terms of the frames containing bounce points, because the motion between these points is largely deterministic. Similarly, in many sequences depicting behavior, the frames depicting the subgoals of a sequence are difficult to predict (c). In the example shown, an agent must collect a subset of the objects in a grid-world environment. If the agent's subgoals are known the full trajectory can be easily recovered.

[*]Equal contribution. Ordering determined by a coin flip.

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.

In this work, we use keyframe-based video prediction to address the problem of discovering subgoal structure in image sequences depicting behavior. 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); Babaeizadeh et al. (2018); Denton & Fergus (2018); Lee et al. (2018); Chung et al. (2015)). However, most previous approaches predict trajectories one frame at a time without attempting to capture hierarchical temporal structure.

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 find 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. 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 break the planning task into shorter parts that correspond to single subtasks.

2 RELATED WORK

Jayaraman et al. (2019); Neitz et al. (2018) 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 contrast 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, 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 $I_{0:N}$ 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 LSTM (Hochreiter & Schmidhuber (1997)) modules described in this section. 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 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

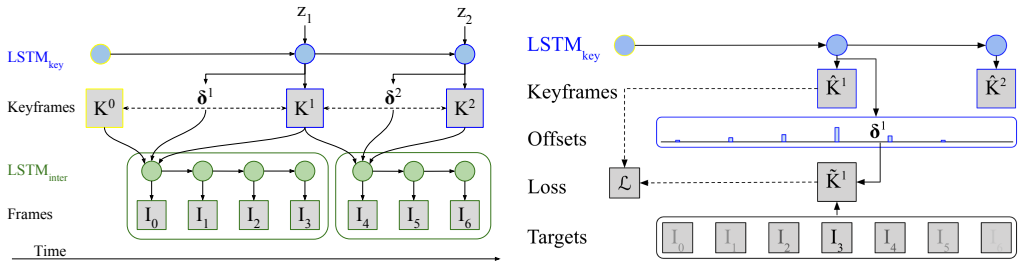

Figure 2: **Left**: Overview of our keyframe-based prediction approach. The keyframe network LSTM_{key} (shown in blue), predicts a set of keyframes $\hat{K}^t$ of the sequence and the length of the intervals between them δ^t. $\text{LSTM}_{\text{inter}}$ (shown in green) takes two consecutive keyframes and the interframe offset between them as input and predicts the intermediate frames I_i. **Right:** Soft loss shown for the first keyframe. The target image $\tilde{K}^t$ is composed of the ground truth images by linearly weighting them with the distribution δ^t. Finally, we use a reconstruction loss between the produced image $\hat{K}^t$ and the soft target $\tilde{K}^t$.

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. Each pair of keyframes produced by 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 $(\hat{I}^t_i)_{i \leq S}$ for each segment t. The initial state of LSTM_{inter} is computed as a feedforward function of the corresponding segment information: $\text{Init}^t_{inter} = \text{MLP}_{init\_inter}(\hat{K}^{e,t-1}, \hat{K}^{e,t}, \hat{\delta}^t_{0:S})$.

To produce a sequence given the output of the predictive model, we choose the interframe offset for keyframe t as $\hat{\delta}^t = \arg\max_i \delta^t_i$. 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. 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. The intuition behind the loss is depicted in Fig. 2.

Soft targets. We produce a keyframe target as: $\tilde{K}^t = \sum_j \tilde{\delta}^t_j I_j$, where $\tilde{\delta}^t$ is the distribution of possible timesteps computed from $\delta^{0:t}$. We further compute the probability that $\hat{K}^t$ is inside the predicted sequence $c^t = \sum_{j \leq N} \tilde{\delta}^t_j$. In a similar fashion, we define a soft loss for the generated interpolating frames $\hat{I}^t_i$. 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.

Soft Loss. We compute the losses both for keyframe embeddings $\hat{K}^{e,t}$, decoded keyframes $\hat{K}^t$, and intermediate frames $\hat{I}^t_j$. The keyframe loss is:

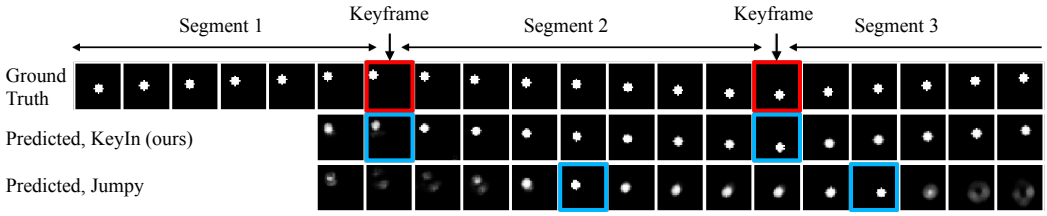

Figure 3: Comparison of reconstruction sequences by KEYIN and by the baseline with static inter-frame offset. The prediction is conditioned on the first five ground truth frames. Only half of the predicted sequence is shown for clarity. Movement direction changes are marked with red in the ground truth sequence, inferred keyframes with blue in the predicted sequences. We see that KEYIN can correctly reconstruct the motion as it selects an informative set of keyframes. The baseline fails to model 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 keyframes of the respective segments.

$$\mathcal{L}_{key} = (\sum_t c^t \beta_{ki} ||\hat{K}^t - \tilde{K}^t||^2 + c^t \beta_{ke} ||\hat{K}^{e,t} - \tilde{K}^{e,t}||^2 + c^t \beta \text{KL}[\mathcal{N}(\mu^t, \sigma^t)||\mathcal{N}(0, I)])/\sum_t c^t \tag{1}$$

The full loss for our model is:

$$\mathcal{L}_{total} = \mathcal{L}_{key} + \sum_{t,i} ||I_j - \tilde{I}_j||^2. \tag{2}$$

We found that pre-training the interpolator on random subsegments helps training stability of our method. We freeze the interpolator weights after pre-training and train the full model with the full loss.

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. 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 on the opposite side of a wall in a sequence of pushes (see Fig. 4, left). Both start and target position, and the placement of the wall are varying, and the individual demonstration pushes vary in length and direction.

4.1 KEYFRAME DISCOVERY

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.

Fig. 3 shows a comparison of KEYIN and the *static* ablation that does not adapt prediction of δ^t to the sequence at hand, but instead learns a *static* keyframe offset pattern. The *static* baseline produces intermediate predictions that do not correspond to the true sequence. KEYIN in contrast is able to correctly find the direction change points, as these are the keyframes that determine the structure of the sequence. Furthermore, KEYIN faithfully predicts both appearance and the motion of the intermediate frames from the found keyframes. The keyframe discovery examples on the Pushing dataset are in Fig. 6 in the appendix. They show that the network is able to model the distribution of length and direction of pushes as well as find correct keyframes in the more complex pushing dataset, too.

Table 1: F1 accuracy score for keyframe discovery, higher is better.

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

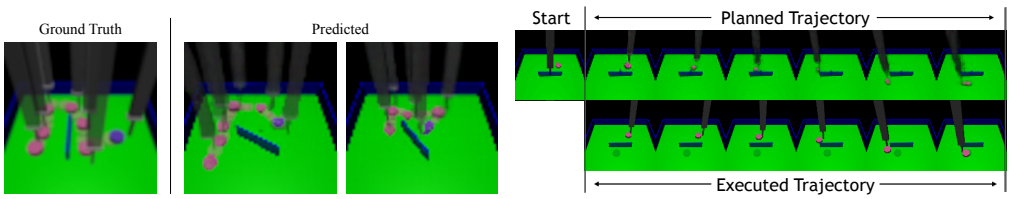

Figure 4: **Left**: A training sequence and two samples from our model on the Pushing dataset. Each image shows an entire trajectory. Our model first samples the keyframes (shown in red), and then deterministically predicts the rest of the sequence. The start object position is colored purple and the robot arm is displayed only for the keyframes. **Right**: 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.

In Table 1, we evaluate the frequency of keyframe discovery using the F1 score. Our method correctly recovers the keyframes more often than the baselines. We compare to a simple surprise-based baseline based on the stochastic video predictor of Denton & Fergus (2018) that measures surprise via KL-divergence with the prior. The Surprise baseline can correctly recover the keyframes on the simple Structured Brownian Motion dataset, but struggles on the more complex Pushing dataset.

4.2 HIERARCHICAL PLANNING

We evaluate whether our method can successfully guide planning by using discovered keyframes as subgoals during planning. We apply KEYIN on the planning task of the Pushing Dataset: pushing the object around the wall to a target position, which is specified with an image. To achieve 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 iteratively plans a sequence of actions that reaches the subgoal. This planning procedure is illustrated in Fig. 7 and described in more detail in Sec. F in the appendix.

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 %**

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, similar to Buesing et al. (2018)), and a bottleneck-based subgoal predictor (TAP, Jayaraman et al. (2019)) as well as the surprise-based model described in Sec. 4.1.[1]

We show that our method outperforms previous work (see Tab. 2). All subgoal-based methods outperform the no subgoal baseline. 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 breaking the task down into simple subtasks. Our method also outperforms TAP, a bottleneck-based method, as the environment doesn't have obvious bottlenecks.

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.

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

ACKNOWLEDGEMENTS

We thank Kenneth Chaney, Bernadette Bucher, and Nikolaos Kolotouros for computing support, and the members of the GRASP laboratory at Penn and CLVR laboratory at USC 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-IIP-1439681 (I/UCRC), NSF-IIS-1703319, NSF MRI 1626008, ARL RCTA W911NF-10-2-0016, ONR N00014-17-1-2093, ARL DCIST CRA W911NF-17-2-0181, the DARPA-SRC C-BRIC, and by Honda Research Institute. K.G.D. is supported by a Canadian NSERC Discovery grant.

REFERENCES

Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic variational video prediction. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

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

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 10–21, 2016.

Lars Buesing, Theophane Weber, Sébastien Racanière, S. M. Ali Eslami, Danilo Jimenez Rezende, David P. Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, and Daan Wierstra. Learning and querying fast generative models for reinforcement learning. *arXiv:1802.03006*, 2018.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2015.

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

Frederik Ebert, Chelsea Finn, Alex X Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. In *Conference on Robotic Learning (CoRL)*, 2017.

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv:1812.00568*, 2018.

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

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

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

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

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

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

Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. Compositional imitation learning: Explaining and executing one task at a time. *arXiv:1812.01483*, 2018.

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

Thang Luong, Hieu Pham, and Christopher D Manning. 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.

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

Alexander Neitz, Giambattista Parascandolo, Stefan Bauer, and Bernhard Schölkopf. Adaptive skip intervals: Temporal abstraction for recurrent dynamical models. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2018.

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

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

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

Emanuel Todorov, Tom Erez, and Yuval Tassa. 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.

Ruben Villegas, Jimei Yang, Seunghoon Hong, Xunyu Lin, and Honglak Lee. Decomposing motion and content for natural video sequence prediction. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

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

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

A METHOD DETAILS

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 a 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):

$$a_{t,j} = \exp(d(\hat{K}^{e,t-1}, K_j^{e,inf})) \tag{3}$$

$$\mu^t, \sigma^t = (\sum_j a_{t,j}\tau_j) / \sum_j a_{t,j}. \tag{4}$$

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

Two-stage training procedure. To train our model correctly, 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 whole 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 in 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 restof 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.

Continuous relaxation 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 one-hot. In practice, we did not observe this being a 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)).

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

B EXPERIMENTAL PARAMETERS

For training the interpolator network, we set $\beta_{ki} = 1$. We train the interpolator on segments of 2-8 frames for Structured Brownian motion data, and 2-6 frames for Pushing data. The KL-divergence term for the interpolator VAE is 10^{-3}. For training the keyframe predictor, we set $\beta_{ke} = 1, \beta_{ki} = 0, \beta = 5{*}10^{-2}$. 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

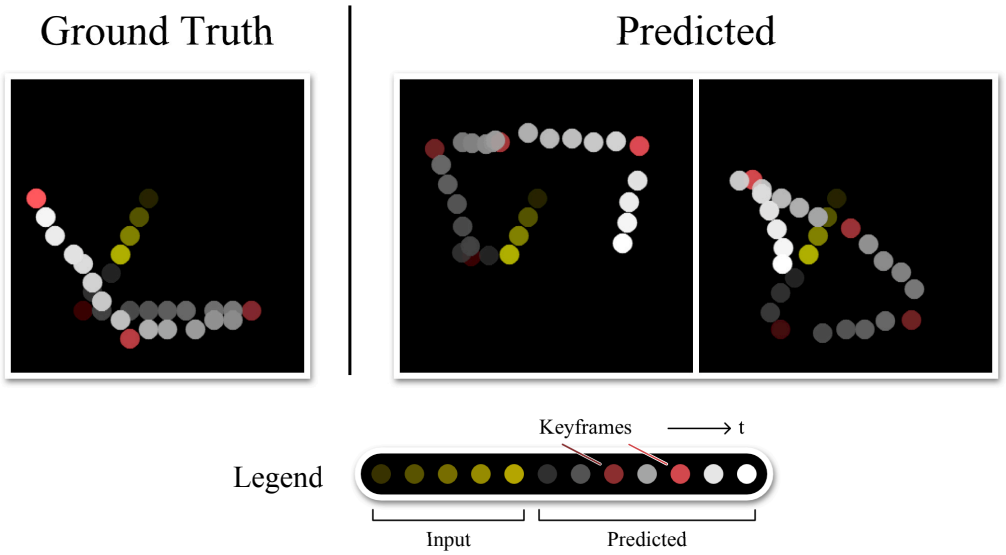

Figure 5: A training sequence and two samples from our model on the Structured Brownian motion dataset. Each image shows an entire trajectory. Our model first samples the keyframes (shown in red), and then deterministically predicts the rest of the sequence. The image resolution was enhanced for viewability.

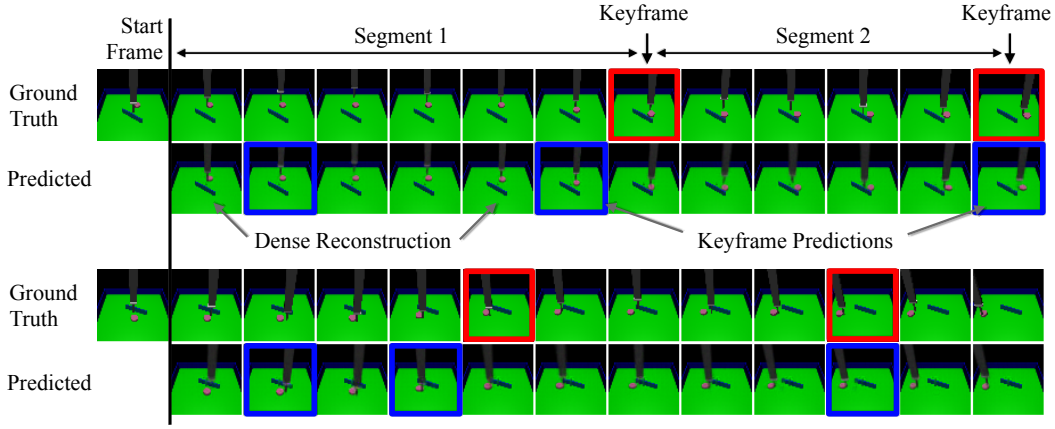

Figure 6: Example frame predictions on the Pushing dataset. In each sequence, the top row corresponds to the ground truth, and the bottom row to the predicted image sequence. The prediction is conditioned on the first five ground truth frames. Only 12 of the 30 predicted frames are shown for clarity. 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.

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 $2e - 4$.

C 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.

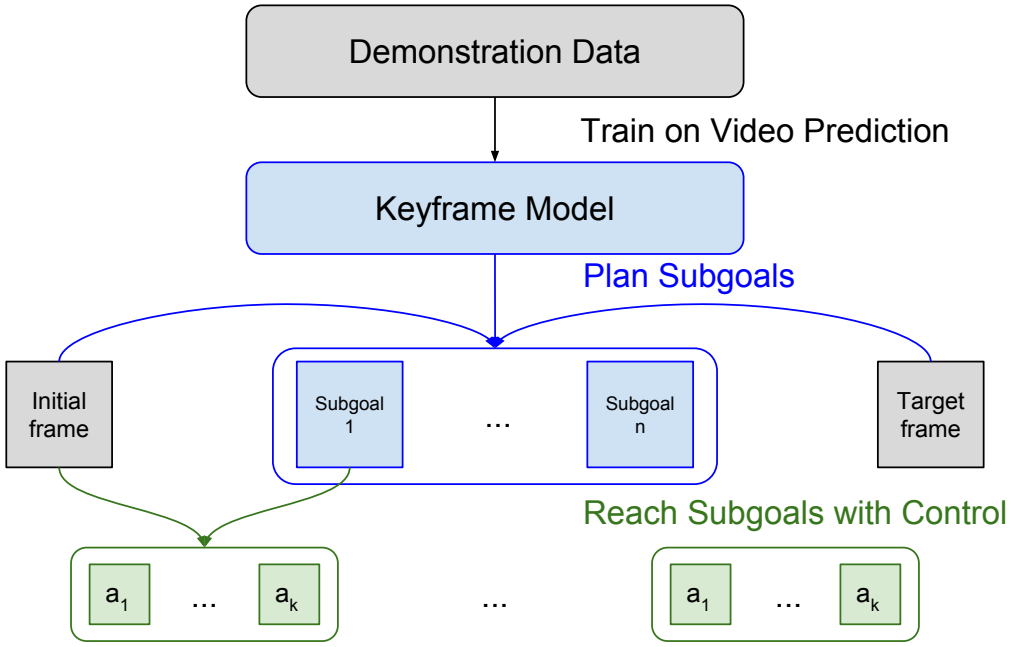

Figure 7: Hierarchical planning procedure. First, we train our KEYIN model on a dataset of demonstrations. At planning time, we can use the model to produce the keyframes between the current observation image and the goal. Finally, we use a low-level MPC planner to reach each keyframe individually, until the final goal is reached.

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 $K^{0:M}$

D 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

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: $c^t = \sum_{j \leq N} \tilde{\delta}_j^t$.
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: $\sum_{t,i} ||I_j - \tilde{I}_j||^2$.

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: $\arg\max \mathrm{M}_i \, S_i$

demonstrations and drop the ones that fail to push the object to the goal position within a predefined horizon.

E SURPRISE BASELINE

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 6 frames t where the surprise peaks. We use a sequential stochastic predictor based on Denton & Fergus (2018) measure the surprise via the KL-divergence $KL[q(z_t|I_{0:t})||p(z_t)]$ (more details in the supplement). 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 whole trajectory.

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.

F PLANNING ALGORITHM

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. 4.

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[2]. We define the cost of an image trajectory as the 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.

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

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

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

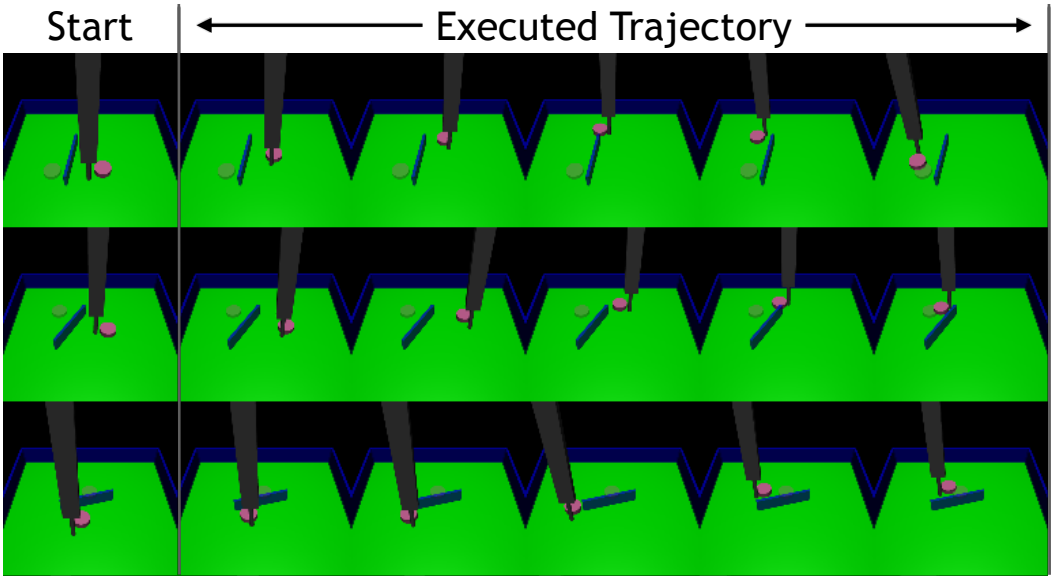

Figure 8: Sample planning task executions from the test set. From a start state depicted on the left, the robot arm successfully pushes the object into the goal position (semi-transparent object) guided by the KEYIN subgoals. The right side of the figure shows intermediate frames of the execution trajectories.