

Visual Dynamics Models for Robotic Planning and Control

Alex Lee



Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-121

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-121.html>

August 16, 2019

Copyright © 2019, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Visual Dynamics Models for Robotic Planning and Control

by

Alex Xavier Lee

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Pieter Abbeel, Co-chair

Assistant Professor Sergey Levine, Co-chair

Professor Ken Goldberg

Summer 2019

Visual Dynamics Models for Robotic Planning and Control

Copyright 2019
by
Alex Xavier Lee

Abstract

Visual Dynamics Models for Robotic Planning and Control

by

Alex Xavier Lee

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Pieter Abbeel, Co-chair

Assistant Professor Sergey Levine, Co-chair

For a robot to interact with its environment, it must perceive the world and understand how the world evolves as a consequence of its actions. This thesis studies a few methods that a robot can use to respond to its observations, with a focus on instances that can leverage *visual dynamic models*. In general, these are models of how the visual observations of a robot evolves as a consequence of its actions. This could be in the form of predictive models that directly predict the future in the space of image pixels, in the space of visual features extracted from these images, or in the space of compact learned latent representations. The three instances that this thesis studies are in the context of visual servoing, visual planning, and representation learning for reinforcement learning. In the first case, we combine learned visual features with learning single-step predictive dynamics models and reinforcement learning to learn visual servoing mechanisms. In the second case, we use a deterministic multi-step video prediction model to achieve various manipulation tasks through visual planning. In addition, we show that conventional video prediction models are unequipped to model uncertainty and multiple futures, which could limit the planning capabilities of the robot. To address this, we propose a stochastic video prediction model that is trained with a combination of variational losses, adversarial losses, and perceptual losses, and show that this model can predict futures that are more realistic, diverse, and accurate. Unlike the first two cases, in which the dynamics model is used to make predictions for decision-making, the third case learns the model solely for representation learning. We learn a stochastic sequential latent variable model to learn a latent representation, and then use it as an intermediate representation for reinforcement learning. We show that this approach improves final performance and sample efficiency.

To my parents—Mireya Gen-Kuong de Lee and John Lee

Contents

Contents	ii
1 Introduction	1
2 Learning Visual Servoing with Deep Features and FQI	4
2.1 Introduction	4
2.2 Related Work	5
2.3 Problem Statement	6
2.4 Visual Features Dynamics	8
2.5 Learning Visual Servoing with Reinforcement Learning	9
2.6 Experiments	11
2.7 Discussion	16
3 Self-Supervised Visual Planning with Temporal Skip Connections	17
3.1 Introduction	17
3.2 Related Work	18
3.3 Preliminaries	19
3.4 Skip Connection Neural Advection Model	21
3.5 Visual MPC with Pixel Distance Costs	22
3.6 Sampling-Based MPC with Continuous and Discrete Actions	24
3.7 Experiments	25
3.8 Discussion	26
4 Stochastic Adversarial Video Prediction	28
4.1 Introduction	28
4.2 Related Work	29
4.3 Video Prediction with Stochastic Adversarial Models	30
4.4 Experiments	34
4.5 Discussion	37
5 Stochastic Latent Actor-Critic	42
5.1 Introduction	42

5.2	Related Work	44
5.3	Reinforcement Learning and Modeling	45
5.4	Joint Modeling and Control as Inference	47
5.5	Stochastic Latent Actor Critic	48
5.6	Latent Variable Model	50
5.7	Experimental Evaluation	52
5.8	Discussion	57
6	Conclusion	58
	Bibliography	60
A	Learning Visual Servoing with Deep Features and FQI	71
A.1	Linearization of the Bilinear Dynamics	71
A.2	Servoing Cost Function for Reinforcement Learning	71
A.3	Experiment Details	72
B	Self-Supervised Visual Planning with Temporal Skip Connections	80
B.1	Hyperparameters	80
B.2	Prototyping in Simulation	80
B.3	Dependence of Model Performance on Amount of Training Data	80
B.4	Failure Cases and Limitations	82
B.5	Robustness of the Model	82
B.6	General skip connection neural advection model	82
C	Stochastic Adversarial Video Prediction	84
C.1	Networks and Training Details	84
D	Stochastic Latent Actor-Critic	87
D.1	Training and Evaluation Details	87
D.2	Reward Functions for the Manipulation Tasks	88
D.3	Additional samples from our model	88

Acknowledgments

I am immensely grateful to my research advisors Pieter Abbeel and Sergey Levine for their continuous support and advice throughout my PhD, and for the freedom that they gave me to explore a wide range of research directions. Pieter and Sergey are inspiring mentors who always found the time to talk whenever I needed advice about research, career choices, or life in general. I have had the great fortune to work with Pieter since my undergraduate years, when he first introduced me to robotic learning research. Since then and throughout my PhD, Pieter helped me learn how to approach research problems and improve the presentation of my research, all while always being endlessly positive and supportive. Throughout my collaboration with Sergey, first while he was a postdoc and later as my faculty advisor, he played an invaluable role in helping me to further refine my research goals and technical skills. Though he was often one (or multiple) steps ahead when it came to discussing research ideas, he never hesitated to find extra time to delve into details together. I am extremely lucky to have two advisors that genuinely cared about not only the research successes but also the growth and personal wellbeing of their students.

I am fortunate to have had the opportunity to work with many amazing students and researchers, from whom I have learned so much. I would like to thank my collaborators for the works in this thesis, including Anusha Nagabandi, Frederik Ebert, Richard Zhang, and Chelsea Finn. I would also like to thank the many other collaborators for the works beyond this thesis, including Abhishek Gupta, Henry Lu, Max Goldstein, Shane Barratt, Laura Smith, Marvin Zhang, Sandy Huang, Dylan Hadfield-Menell, Eric Tzeng, and Sudeep Ansari. I would also like to thank members of RLL, RAIL, and the Berkeley AI Research for the fruitful research discussions and fostering a collaborative environment. I would like to thank my mentors and students during my undergraduate research, including John Schulman, Sachin Patil, Sameep Tandon, Ziang Xie, and Jonathan Ho, from whom I learned research and coding practices that I still use today. Lastly, I would like to thank my collaborators at DeepMind, including Konstantinos Bousmalis, David Khosid, Jost Tobias Springenberg, and Raia Hadsell.

I am thankful for everyone who supported and inspired me in and outside of research. I would like to thank everyone with whom I exchanged ideas, discussed research topics, and found encouragement along the way, including Sandy Huang, Abhishek Gupta, Chelsea Finn, Cathy Wu, Eric Tzeng, Andreea Bobu, Richard Zhang, Coline Devin, Anusha Nagabandi, Gregory Khan, Marvin Zhang, Zoe McCarthy, Lisa Hendricks, Jonathan Ho, and many others. In particular, I would like to thank: Sandy, for being someone I could turn to share struggles and successes, and for always having a positive perspective; Abhishek, for being on my side (literally and figuratively!) through many years and for being harsh in regards to what constitutes novel ideas; Chelsea, for the insightful and sincere conversations about life and research; and Eric, for being a fun and supportive housemate. I would also like to thank Jennifer Sun for all the shared experiences and her continuous support, from close and far away. And I would like to thank my friends from Ecuador, including Isabel Alcivar, Michelle Balseca, Ana Rodriguez, and Belen Rodriguez, who often reminded me of where I come from.

Finally, this thesis is dedicated to my parents, Mireya and John, for all the years of their love, encouragement, and support.

Chapter 1

Introduction

Robots have the potential to help people and improve human lives. In robotic manipulation, robots can, for example, help perform tasks in the house, at a factory, or in surgical procedures. These robots can perform tedious and repetitive tasks, such as house cleaning and manufacturing, and thus free up people to do tasks that they find more enjoyable. Or, these robots can specialize in tasks that require a high level of accuracy and precision, such as surgeries, and thus enable surgical procedures to be safer and more accessible to patients. In mobile robotics, self-driving cars can reduce car accidents and traffic, delivery drones can reduce delivery times and costs, and ambulance drones can deliver critical medical equipment to a patient going through an emergency.

All of these robotic tasks share the same challenges: the robot needs to perceive and understand its complex environment in order to decide how to interact with it. Specifically, since we live in a world with a myriad of visual information, it is important for robots to be equipped with perception skills that can process visual observations, such as images from camera sensors.

This thesis studies a few methods that a robot can use to respond to its observations, with a focus on instances that leverage dynamic models of image observations. We consider the case where a robot continuously perceives its environment and adapts its execution based on those observations. The robot executes closed-loop trajectories, which can manifest itself in various forms. In this thesis we study three methods that use visual dynamics models:

- The single-step predictive model is used to choose actions based on predictions of the immediate future.
- The multi-step predictive model is used to plan a trajectory based on predictions from multiple steps into the future.
- The model is used for representation learning to improve and speed up the learning, but its predictions are not used for decision-making.

Although the techniques presented here are general and can be applied to any sensor modality, this thesis focuses on image observations due to the challenging nature of perceiving and modeling high-dimensional image observations.

In order for the robot to perceive and act in its environment, it should understand how the world evolves as a consequence of its actions, which can be accomplished using a visual dynamics model. A visual dynamics model is a model of how the visual observations of a robot evolve as a consequence of its actions. This could be in the form of a video prediction model, which predicts future video frames conditioned on past and current frames, as well as a sequence of future actions. These models directly predict the future in the space of image pixels. Alternatively, the model could predict the future in some visual feature space, where the features are extracted from the image observations. In general, these dynamics models operate directly on images or on features that can readily be extracted from the images, as opposed to some ground-truth state. These models allow the robot to predict the consequences of its actions, either one step or multiple steps into the future, and then act intelligently based on these predictions. Alternatively, the model could be used specifically for representation learning, without using its predictions for decision-making.

The contributions of this thesis are as follows:

- Chapter 2 considers the case where single-step dynamics models are used in the context of visual servoing. Visual servoing is a classical problem in robotics and, in fact, several robotic problems can be formulated as such. Examples of visual servoing problems include path following, space docking, and manipulation. Visual servoing involves choosing actions that move a robot in response to camera observations in order to reach a goal configuration. An ubiquitous approach to servoing involves using one-step dynamics models, and choosing actions that minimize errors between the one-step prediction and the goal configuration. This error is classically defined in the space of poses or feature points extracted from images, or more recently, directly in the space of pixel intensities. We propose a different approach that uses spatial feature maps, derived from a deep convolutional neural network pre-trained on image classification, and learned dynamics models in the space of these feature maps. However, using these feature maps introduces a new problem. The features have arbitrary units and their relative magnitudes can be scaled arbitrarily. This is problematic since the scale determines how much a particular feature is penalized in the error and, depending on the task, we may want to penalize some features more than others. To address this, we learn a relative scaling of the feature maps by formulating the visual servoing problem as a sample-efficient reinforcement learning problem. The overall approach combines learned visual features with learning predictive dynamics models and reinforcement learning to learn visual servoing mechanisms. This work was published previously as Lee et al. (2017).
- Chapter 3 considers the case in which multi-step dynamics models are used for visual planning, in order to solve manipulation tasks involving pushing objects to desired positions on a table. This work uses a multi-step video prediction model, which enables the robot to predict which sequence of actions results in the best outcome, and then use those actions to solve a task. The video prediction model predicts future frames by predicting spatial transformations of pixels over time, conditioned on the current and past frames as well as a sequence of actions. This allows us to define cost functions in terms of desired pixel motions that are provided as a user input, and then use these cost functions to determine which actions

lead to the best outcome. This work proposes a deterministic video prediction model and demonstrates that it can be used for visual planning. Using this method, we are able to accomplish tasks that involve pushing objects on a table by a robot manipulator. These tasks are intuitively specified by a human, who indicates the desired motions of objects that lie on a table. This work appeared previously as Ebert et al. (2017).

- The aforementioned visual planning work uses a deterministic video prediction model that is trained with standard pixel-wise regression losses. This leads to blurry and unrealistic predictions for complex environments with inherent uncertainty and ambiguity. However, it is imperative to have an accurate predictive model, especially for long horizons, in order to equip the robot with better planning capabilities. Chapter 4 presents a stochastic video prediction model that is trained with a combination of variational losses, adversarial losses, and perceptual losses. We show that the proposed approach leads to predictions that are more realistic, diverse, and accurate, even for several time steps into the future. This work was previously presented in Lee et al. (2018).
- The approach in Chapter 5 considers to explicitly learn representations that can accelerate reinforcement learning from images. This work learns a compact latent representation space using a stochastic sequential latent variable model, and then uses the representation in an actor-critic reinforcement learning framework, where the critic model is learned within this latent space. By learning a critic within a compact state space, we can learn much more efficiently than standard reinforcement learning methods. We show that the proposed method outperforms both model-free and model-based alternatives in terms of final performance and sample efficiency, on a range of difficult image-based control simulated benchmark tasks. This work is currently in preparation (Lee et al., 2019).
- Finally, we conclude and discuss future directions in Chapter 6.

Chapter 2

Learning Visual Servoing with Deep Features and Fitted Q-Iteration

2.1 Introduction

Visual servoing is a classic problem in robotics that requires moving a camera or robot to match a target configuration of visual features or image intensities. Many robot control tasks that combine perception and action can be posed as visual servoing, including navigation (DeSouza and Kak, 2002; Chen et al., 2006), where a robot must follow a desired path; manipulation, where the robot must servo an end-effector or a camera to a target object to grasp or manipulate it (Malis et al., 1999; Corke, 1993; Hashimoto, 1993; Hosoda and Asada, 1994; Kragic and Christensen, 2002); and various other problems, as surveyed in Hutchinson et al. (1996). Most visual servoing methods assume access to good geometric image features (Chaumette and Hutchinson, 2006; Collewet et al., 2008; Caron et al., 2013) and require knowledge of their dynamics, which are typically obtained from domain knowledge about the system. Using such hand-designed features and models prevents exploitation of statistical regularities in the world, and requires manual engineering for each new system.

In this work, we study how learned visual features, learned predictive dynamics models, and reinforcement learning can be combined to learn visual servoing mechanisms. We focus on target following, with the goal of designing algorithms that can learn a visual servo using low amounts of data of the target in question, so as to be easy and quick to adapt to new targets. Successful target following requires the visual servo to tolerate moderate variation in the appearance of the target, including changes in viewpoint and lighting, as well as occlusions. Learning invariances to all such distractors typically requires a considerable amount of data. However, since a visual servo is typically specific to a particular task, it is desirable to be able to learn the servoing mechanism very quickly, using a minimum amount of data. Prior work has shown that the features learned by large convolutional neural networks on large image datasets, such as ImageNet classification (Deng et al., 2009), tend to be useful for a wide range of other visual tasks (Donahue et al., 2014). We explore whether the usefulness of such features extends to visual servoing.

To answer this question, we propose a visual servoing method that uses pre-trained features, in our case obtained from the VGG network (Simonyan and Zisserman, 2015) trained for ImageNet classification. Besides the visual features, our method uses an estimate of the feature dynamics in visual space by means of a bilinear model. This allows the visual servo to predict how motion of the robot’s camera will affect the perceived feature values. Unfortunately, servoing directly on the high-dimensional features of a pre-trained network is insufficient by itself to impart robustness on the servo: the visual servo must not only be robust to moderate visual variation, but it must also be able to pick out the target of interest (such as a car that the robot is tasked with following) from irrelevant distractor objects. To that end, we propose a sample-efficient fitted Q-iteration procedure that automatically chooses weights for the most relevant visual features. Crucially, the actual servoing mechanism in our approach is extremely simple, and simply seeks to minimize the Euclidean distance between the weighted feature values at the next time step and the target. The form of the servoing policy in our approach leads to an analytic and tractable linear approximator for the Q-function, which leads to a computationally efficient fitted Q-iteration algorithm. We show that we can learn an effective visual servo on a complex synthetic car following benchmark using just 20 training trajectory samples for reinforcement learning. We demonstrate substantial improvement over a conventional approach based on image pixels or hand-designed keypoints, and we show an improvement in sample-efficiency of more than two orders of magnitude over standard model-free deep reinforcement learning algorithms.

The environment for the synthetic car following benchmark is available online as the package CitySim3D¹, and the code to reproduce our method and experiments is also available online². Supplementary videos of all the test executions are available on the project’s website³.

2.2 Related Work

Visual servoing is typically (but not always) performed with calibrated cameras and carefully designed visual features. Ideal features for servoing should be stable and discriminative, and much of the work on visual servoing focuses on designing stable and convergent controllers under the assumption that such features are available (Espiau et al., 2002; Mohta et al., 2014; Wilson et al., 1996). Some visual servoing methods do not require camera calibration (Jagersand et al., 1997; Yoshimi and Allen, 1994), and some recent methods operate directly on image intensities (Caron et al., 2013), but generally do not use learning to exploit statistical regularities in the world and improve robustness to distractors.

Learning is a relatively recent addition to the repertoire of visual servoing tools. Several methods have been proposed that apply ideas from reinforcement learning to directly acquire visual servoing controllers (Lampe and Riedmiller, 2013; Sadeghzadeh et al., 2015). However, such methods have not been demonstrated under extensive visual variation, and do not make use of state-of-the-art convolutional neural network visual features. Though more standard deep rein-

¹<https://github.com/alexlee-gk/citysim3d>

²https://github.com/alexlee-gk/visual_dynamics

³http://rll.berkeley.edu/visual_servoing

forcement learning methods (Lange et al., 2012; Mnih et al., 2013; Levine et al., 2016a; Lillicrap et al., 2016) could in principle be applied to directly learn visual servoing policies, such methods tend to require large numbers of samples to learn task-specific behaviors, making them poorly suited for a flexible visual servoing algorithm that can be quickly repurposed to new tasks (e.g. to following a different object).

Instead, we propose an approach that combines learning of predictive models with pre-trained visual features. We use visual features trained for ImageNet (Deng et al., 2009) classification, though any pre-trained features could in principle be applicable for our method, so long as they provide a suitable degree of invariance to visual distractors such as lighting, occlusion, and changes in viewpoint. Using pre-trained features allows us to avoid the need for large amounts of experience, but we must still learn the policy itself. To further accelerate this process, we first acquire a predictive model that allows the visual servo to determine how the visual features will change in response to an action. General video prediction is an active research area, with a number of complex but data-hungry models proposed in recent years (Oh et al., 2015; Watter et al., 2015; Mathieu et al., 2016; Xue et al., 2016; Lotter et al., 2017; Jia et al., 2016; Walker et al., 2016; Vondrick et al., 2016).

However, we observe that convolutional response maps can be interpreted as images and, under mild assumptions, the dynamics of image pixels during camera motion can be well approximated by means of a bilinear model (Censi and Murray, 2015). We therefore train a relatively simple bilinear model for short-term prediction of visual feature dynamics, which we can use inside a very simple visual servo that seeks to minimize the error between the next predicted feature values and a target image.

Unfortunately, simply training predictive models on top of pre-trained features is insufficient to produce an effective visual servo, since it weights the errors of distractor objects the same amount as the object of interest. We address this challenge by using an efficient Q-iteration algorithm to train the weights on the features to maximize the servo’s long-horizon reward. This method draws on ideas from regularized fitted Q-iteration (Gordon, 1995; Ernst et al., 2005; Farahmand et al., 2009) and neural fitted Q-iteration (Riedmiller, 2005) to develop a sample-efficient algorithm that can directly estimate the expected return of the visual servo without the use of any additional function approximator.

2.3 Problem Statement

Let \mathbf{y}_t be a featurization of the camera’s observations \mathbf{x}_t and let \mathbf{y}_* be some given goal feature map. For the purposes of this work, we define *visual servoing* as the problem of choosing controls \mathbf{u}_t for a fixed number of discrete time steps t as to minimize the error $\|\mathbf{y}_* - \mathbf{y}_t\|$.

We use a relatively simple gradient-based servoing policy that uses one-step feature dynamics, $f : \{\mathbf{y}_t, \mathbf{u}_t\} \rightarrow \mathbf{y}_{t+1}$. The policy chooses the control that minimizes the distance between the goal feature map and the one-step prediction:

$$\pi(\mathbf{x}_t, \mathbf{x}_*) = \arg \min_{\mathbf{u}} \|\mathbf{y}_* - f(\mathbf{y}_t, \mathbf{u})\|^2. \quad (2.1)$$

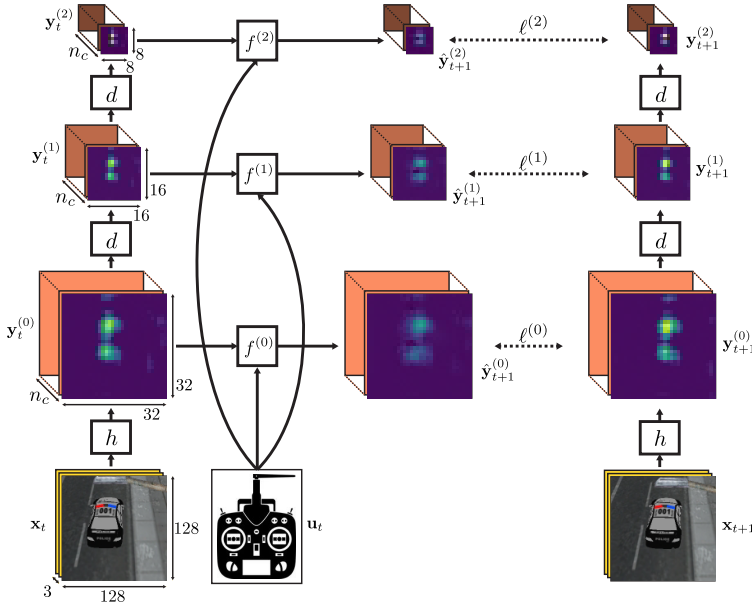


Figure 2.1: Multiscale bilinear model. The function h maps images \mathbf{x} to feature maps $\mathbf{y}^{(0)}$, the operator d downsamples the feature maps $\mathbf{y}^{(l-1)}$ to $\mathbf{y}^{(l)}$, and the bilinear function $f^{(l)}$ predicts the next feature $\hat{\mathbf{y}}^{(l)}$. The number of channels for each feature map is n_c , regardless of the scale l .

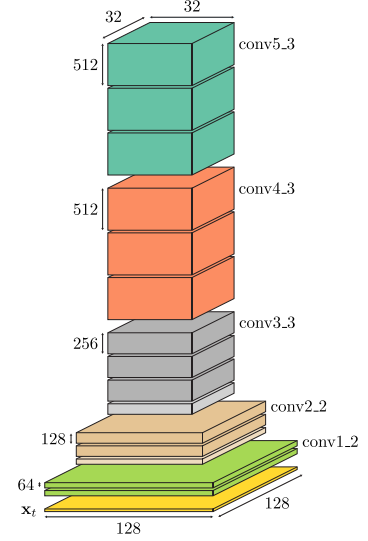


Figure 2.2: Dilated VGG-16 network. The intermediate feature maps drawn in a lighter shade are outputs of max-pooling layers. The features maps in the conv4 and conv5 blocks are outputs of dilated convolutions with dilation factors of 2 and 4, respectively.

Learning this policy amounts to learning the robot dynamics and the distance metric $\|\cdot\|$.

To learn the robot dynamics, we assume that we have access to a dataset of paired observations and controls $\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1}$. This data is relatively easy to obtain as it involves collecting a stream of the robot's observations and controls. We use this dataset to learn a general visual dynamics model that can be used for any task.

To learn the distance metric, we assume that the robot interacts with the world and collects tuples of the form $\mathbf{x}_t, \mathbf{u}_t, c_t, \mathbf{x}_{t+1}, \mathbf{x}_*$. At every time step during learning, the robot observes \mathbf{x}_t and takes action \mathbf{u}_t . After the transition, the robot observes \mathbf{x}_{t+1} and receives an immediate cost c_t . This cost is task-specific and it quantifies how good that transition was in order to achieve the goal. At the beginning of each trajectory, the robot is given a goal observation \mathbf{x}_* , and it is the same throughout the trajectory. We define the goal feature map to be the featurization of the goal observation. We learn the distance metric using reinforcement learning and we model the environment as a Markov Decision Process (MDP). The state of the MDP is the tuple of the current observation and the episode's target observation, $\mathbf{s}_t = (\mathbf{x}_t, \mathbf{x}_*)$, the action \mathbf{u}_t is the discrete-time continuous control of the robot, and the cost function maps the states and action $(\mathbf{s}_t, \mathbf{u}_t, \mathbf{s}_{t+1})$ to a scalar cost c_t .

2.4 Visual Features Dynamics

We learn a multiscale bilinear model to predict the visual features of the next frame given the current image from the robot’s camera and the action of the robot. An overview of the model is shown in Figure 2.1. The learned dynamics can then be used for visual servoing as described in Section 2.5.

Visual Features

We consider both pixels and semantic features for the visual representation. We define the function h to relate the image \mathbf{x} and its feature $\mathbf{y} = h(\mathbf{x})$. Our choice of semantic features are derived from the VGG-16 network (Simonyan and Zisserman, 2015), which is a convolutional neural network trained for large-scale image recognition on the ImageNet dataset (Deng et al., 2009). Since spatial invariance is undesirable for servoing, we remove some of the max-pooling layers and replace the convolutions that followed them with dilated convolutions, as done by Yu and Koltun (2016). The modified VGG network is shown in Figure 2.2. We use the model weights of the original VGG-16 network, which are publicly available as a Caffe model (Jia et al., 2014). The features that we use are the outputs of some of the intermediate convolutional layers, that have been downsampled to a 32×32 resolution (if necessary) and standardized with respect to our training set.

We use multiple resolutions of these features for servoing. The idea is that the high-resolution representations have detailed local information about the scene, while the low-resolution representations have more global information available through the image-space gradients. The features at level l of the multiscale pyramid are denoted as $\mathbf{y}^{(l)}$. The features at each level are obtained from the features below through a downsampling operator $d(\mathbf{y}^{(l-1)}) = \mathbf{y}^{(l)}$ that cuts the resolution in half.

Bilinear Dynamics

The features $\mathbf{y}_t^{(l)}$ are used to predict the corresponding level’s features $\mathbf{y}_{t+1}^{(l)}$ at the next time step, conditioned on the action \mathbf{u}_t , according to a prediction function $f^{(l)}(\mathbf{y}_t^{(l)}, \mathbf{u}_t) = \hat{\mathbf{y}}_{t+1}^{(l)}$. We use a bilinear model to represent these dynamics, motivated by prior work (Censi and Murray, 2015). In order to servo at different scales, we learn a bilinear dynamics model at *each* scale. We consider two variants of the bilinear model in previous work in order to reduce the number of model parameters.

The first variant uses *fully connected* dynamics as in previous work but models the dynamics of each channel independently. When semantic features are used, this model interprets the feature maps as being abstract images with spatial information within a channel and different entities or factors of variation across different channels. This could potentially allow the model to handle moving objects, occlusions, and other complex phenomena.

The fully connected bilinear model is quite large, so we propose a bilinear dynamics that enforces sparsity in the parameters. In particular, we constrain the prediction to depend only on the features that are in its local spatial neighborhood, leading to the following *locally connected*

bilinear model:

$$\hat{\mathbf{y}}_{t+1,c}^{(l)} = \mathbf{y}_{t,c}^{(l)} + \sum_j \left(\mathbf{W}_{c,j}^{(l)} * \mathbf{y}_{t,c}^{(l)} + \mathbf{B}_{c,j}^{(l)} \right) \mathbf{u}_{t,j} + \left(\mathbf{W}_{c,0}^{(l)} * \mathbf{y}_{t,c}^{(l)} + \mathbf{B}_{c,0}^{(l)} \right). \quad (2.2)$$

The parameters are the 4-dimensional tensor $\mathbf{W}_{c,j}^{(l)}$ and the matrix $\mathbf{B}_{c,j}^{(l)}$ for each channel c , scale l , and control coordinate j . The last two terms are biases that allow to model action-independent visual changes, such as moving objects. The $*$ is the locally connected operator, which is like a convolution but with untied filter weights⁴.

Training Visual Feature Dynamics Models

The loss that we use for training the bilinear dynamics is the sum of the losses of the predicted features at each level, $\sum_{l=0}^L \ell^{(l)}$, where the loss for each level l is the squared ℓ_2 norm between the predicted features and the actual features of that level, $\ell^{(l)} = \|\mathbf{y}_{t+1}^{(l)} - \hat{\mathbf{y}}_{t+1}^{(l)}\|^2$.

We optimize for the dynamics while keeping the feature representation fixed. This is a supervised learning problem, which we solve with Adam (Kingma and Ba, 2015). The training set, consisting of triplets $\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1}$, was obtained by executing a hand-coded policy that moves the robot around the target with some Gaussian noise.

2.5 Learning Visual Servoing with Reinforcement Learning

We propose to use a multiscale representation of semantic features for servoing. The challenge when introducing multiple scales and multi-channel feature maps for servoing is that the features do not necessarily agree on the optimal action when the goal is unattainable or the robot is far away from the goal. To do well, it's important to use a good weighing of each of the terms in the objective. Since there are many weights, it would be impractically time-consuming to set them by hand, so we resort to learning. We want the weighted one-step lookahead objective to encourage good long-term behavior, so we want this objective to correspond to the state-action value function Q . So we propose a method for learning the weights based on fitted Q-iteration.

Servoing with Weighted Multiscale Features

Instead of attempting to build an accurate predictive model for multi-step planning, we use the simple greedy servoing method in Equation (2.1), where we minimize the error between the target and predicted features for all the scales. Typically, only a few objects in the scene are relevant, so

⁴ The locally connected operator, with a local neighborhood of $n_f \times n_f$ (analogous to the filter size in convolutions), is defined as:

$$(\mathbf{W} * \mathbf{y})_{k_h, k_w} = \sum_{i_h = k_h - \lfloor n_f/2 \rfloor}^{k_h + \lfloor n_f/2 \rfloor} \sum_{i_w = k_w - \lfloor n_f/2 \rfloor}^{k_w + \lfloor n_f/2 \rfloor} \mathbf{W}_{k_h, k_w, i_h - k_h, i_w - k_w} \mathbf{y}_{i_h, i_w}.$$

the errors of some channels should be penalized more than others. Similarly, features at different scales might need to be weighted differently. Thus, we use a weighting $\mathbf{w}_c^{(l)} \geq 0$ per channel c and scale l :

$$\pi(\mathbf{x}_t, \mathbf{x}_*) = \arg \min_{\mathbf{u}} \sum_c \sum_{l=0}^L \frac{\mathbf{w}_c^{(l)}}{|\mathbf{y}_{*,c}^{(l)}|} \left\| \mathbf{y}_{*,c}^{(l)} - f_c^{(l)}(\mathbf{y}_{t,c}^{(l)}, \mathbf{u}) \right\|_2^2 + \sum_j \lambda_j \mathbf{u}_j^2, \quad (2.3)$$

where $|\cdot|$ denotes the cardinality operator and the constant $1/|\mathbf{y}_{*,c}^{(l)}|$ normalizes the feature errors by its spatial resolution. We also use a separate weight λ_j for each control coordinate j . This optimization can be solved efficiently since the dynamics is linear in the controls (see Appendix A.1).

Q-Function Approximation for the Weighted Servoing Policy

We choose a Q-value function approximator that can represent the servoing objective such that the greedy policy with respect to the Q-values results in the policy of Equation (2.3). In particular, we use a function approximator that is linear in the weight parameters $\boldsymbol{\theta}^\top = [\mathbf{w}^\top \quad \boldsymbol{\lambda}^\top]$:

$$Q_{\boldsymbol{\theta},b}(\mathbf{s}_t, \mathbf{u}) = \phi(\mathbf{s}_t, \mathbf{u})^\top \boldsymbol{\theta} + b, \quad \phi(\mathbf{s}_t, \mathbf{u})^\top = \left[\left[\frac{1}{|\mathbf{y}_{*,c}^{(l)}|} \left\| \mathbf{y}_{*,c}^{(l)} - f_c^{(l)}(\mathbf{y}_{t,c}^{(l)}, \mathbf{u}) \right\|_2^2 \right]_{c,l}^\top \quad [\mathbf{u}_j^2]_j^\top \right].$$

We denote the state of the MDP as $\mathbf{s}_t = (\mathbf{x}_t, \mathbf{x}_*)$ and add a bias b to the Q-function. The servoing policy is then simply $\pi_{\boldsymbol{\theta}}(\mathbf{s}_t) = \arg \min_{\mathbf{u}} Q_{\boldsymbol{\theta},b}(\mathbf{s}_t, \mathbf{u})$. For reinforcement learning, we optimized for the weights $\boldsymbol{\theta}$ but kept the feature representation and its dynamics fixed.

Learning the Q-Function with Fitted Q-Iteration

Reinforcement learning methods that learn a Q-function do so by minimizing the Bellman error:

$$\left\| Q(\mathbf{s}_t, \mathbf{u}_t) - \left(c_t + \gamma \min_{\mathbf{u}} Q(\mathbf{s}_{t+1}, \mathbf{u}) \right) \right\|_2^2. \quad (2.4)$$

In fitted Q-iteration, the agent iteratively gathers a dataset $\{\mathbf{s}_t^{(i)}, \mathbf{u}_t^{(i)}, c_t^{(i)}, \mathbf{s}_{t+1}^{(i)}\}_i^N$ of N samples according to an exploration policy, and then minimizes the Bellman error using this dataset. We use the term *sampling iteration* to refer to each iteration j of this procedure. At the beginning of each sampling iteration, the current policy with added Gaussian noise is used as the exploration policy.

It is typically hard or unstable to optimize for both Q-functions that appear in the Bellman error of Equation (2.4), so it is usually optimized by iteratively optimizing the current Q-function while keeping the target Q-function constant. However, we notice that for a given state, the action that minimizes its Q-values is the same for any non-negative scaling α of $\boldsymbol{\theta}$ and for any bias b . Thus, to speed up the optimization of the Q-function, we first set $\alpha^{(k-\frac{1}{2})}$ and $b^{(k-\frac{1}{2})}$ by jointly solving for α and b of *both* the current and target Q-function:

$$\min_{\alpha \geq 0, b} \frac{1}{N} \sum_{i=1}^N \left\| Q_{\alpha \boldsymbol{\theta}^{(k-1)}, b}(\mathbf{s}_t^{(i)}, \mathbf{u}_t^{(i)}) - \left(c_t^{(i)} + \gamma \min_{\mathbf{u}} Q_{\alpha \boldsymbol{\theta}^{(k-1)}, b}(\mathbf{s}_{t+1}^{(i)}, \mathbf{u}) \right) \right\|_2^2 + \nu \|\boldsymbol{\theta}\|_2^2. \quad (2.5)$$

This is similar to how, in policy evaluation, state values can be computed by solving a linear system. We regularize the parameters with an ℓ_2 penalty, weighted by $\nu \geq 0$. We use the term *FQI iteration* to refer to each iteration k of optimizing the Bellman error, and we use the notation $(k-\frac{1}{2})$ to denote an intermediate step between iterations $(k-1)$ and (k) . The parameters θ can then be updated with $\theta^{(k-\frac{1}{2})} = \alpha^{(k-\frac{1}{2})}\theta^{(k-1)}$. Then, we update $\theta^{(k)}$ and $b^{(k)}$ by optimizing for θ and b of the current Q-function while keeping the parameters of the target Q-function fixed:

$$\min_{\theta \geq 0, b} \frac{1}{N} \sum_{i=1}^N \left\| Q_{\theta, b}(\mathbf{s}_t^{(i)}, \mathbf{u}_t^{(i)}) - \left(c_t^{(i)} + \gamma \min_{\mathbf{u}} Q_{\theta^{(k-\frac{1}{2})}, b^{(k-\frac{1}{2})}}(\mathbf{s}_{t+1}^{(i)}, \mathbf{u}) \right) \right\|_2^2 + \nu \|\theta\|_2^2. \quad (2.6)$$

A summary of the algorithm used to learn the feature weights is shown in Algorithm 1.

Algorithm 1 FQI with initialization of policy-independent parameters

```

1: procedure FQI( $\theta^{(0)}, \sigma_{\text{exploration}}^2, \nu$ )
2:   for  $s = 1, \dots, S$  do ▷ sampling iterations
3:     Gather dataset  $\{\mathbf{s}_t^{(i)}, \mathbf{u}_t^{(i)}, c_t^{(i)}, \mathbf{s}_{t+1}^{(i)}\}_i^N$  using exploration policy  $\mathcal{N}(\pi_{\theta^{(0)}}, \sigma_{\text{exploration}}^2)$ 
4:     for  $k = 1, \dots, K$  do ▷ FQI iterations
5:       Fit  $\alpha^{(k-\frac{1}{2})}$  and  $b^{(k-\frac{1}{2})}$  using (2.5)
6:        $\theta^{(k-\frac{1}{2})} \leftarrow \alpha^{(k-\frac{1}{2})}\theta^{(k-1)}$ 
7:       Fit  $\theta^{(k)}$  and  $b^{(k)}$  using (2.6)
8:     end for
9:      $\theta^{(0)} \leftarrow \theta^{(K)}$ 
10:  end for
11: end procedure

```

2.6 Experiments

We evaluate the performance of the model for visual servoing in a simulated environment. The simulated quadcopter is governed by rigid body dynamics. The robot has 4 degrees of freedom, corresponding to translation along three axis and yaw angle. This simulation is inspired by tasks in which an autonomous quadcopter flies above a city, with the goal of following some target object (e.g., a car).

Learning Feature Dynamics and Weights with FQI

The dynamics for each of the features were trained using a dataset of 10000 samples (corresponding to 100 trajectories) with Adam (Kingma and Ba, 2015). A single dynamics model was learned for each feature representation for all the training cars (Figure 2.3). This training set was generated by executing a hand-coded policy that navigates the quadcopter around a car for 100 time steps per trajectory, while the car moves around the city.



Figure 2.3: Cars used to learn the dynamics and the feature weights. They were also used in some of the test experiments.



Figure 2.4: Novel cars used only in the test experiments. They were never seen during training or validation.

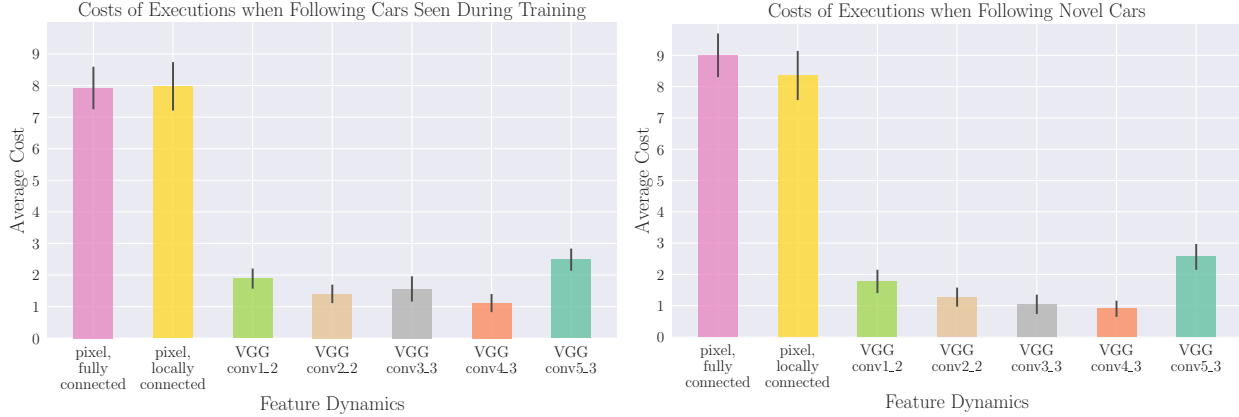


Figure 2.5: Costs of test executions using various feature dynamics models, where the feature weights are optimized with FQI. We test on cars that were used during learning (left plot) and on novel cars that were only used at test time (right plot). The reported values are the mean and standard error across 100 trajectories, of up to 100 time steps each. The policies based on pixel intensities use either fully connected or locally connected dynamics, whereas all the policies based on VGG features use locally connected dynamics. The policies based on deeper VGG features generally achieve better performance, except for the deepest feature representation, VGG conv5_3, which is not as suitable for approximating Q-values. The policies based on pixel intensities and VGG conv5_3 features perform worse on the novel cars. However, VGG features conv1_2 through conv4_3 achieve some degree of generalization on the novel cars.

We used the proposed FQI algorithm to learn the weightings of the features and control regularizer. At every sampling iteration, the current policy was executed with Gaussian noise to gather data from 10 trajectories. All the trajectories in our experiments were up to 100 time steps long. The immediate cost received by the agent encodes the error of the target in image coordinates (details in Appendix A.2). Then, the parameters were iteratively updated by running $K = 10$ iterations of FQI. We ran the overall algorithm for only $S = 2$ sampling iterations and chose the parameters that achieved the best performance on 10 validation trajectories. These validation trajectories were obtained by randomly choosing 10 cars from the set of training cars and randomly sampling initial states, and executing the policy with the parameters of the current iteration. All the experiments share the same set of validation trajectories.

Comparison of Feature Representations for Servoing

We compare the servoing performance for various feature dynamics models, where the weights are optimized with FQI. We execute the learned policies on 100 test trajectories and report the average cost of the trajectory rollouts on Figure 2.5. The cost of a single trajectory is the (undiscounted) sum of costs c_t . We test the policies with cars that were seen during training as well as with a set of novel cars (Figure 2.4), to evaluate the generalization of the learned dynamics and optimized policies.

The test trajectories were obtained by randomly sampling 100 cars (with replacement) from one of the two sets of cars, and randomly sampling initial states (which are different from the ones used for validation). For consistency and reproducibility, the same sampled cars and initial states were used across all the test experiments, and the same initial states were used for both sets of cars. These test trajectories were never used during the development of the algorithm or for choosing hyperparameters.

From these results, we notice that policies based on deeper VGG features, up to VGG conv4_3, generally achieve better performance. However, the deepest feature representation, VGG conv5_3, is not as suitable for approximating Q-values. We hypothesize that this feature might be too spatially invariant and it might lack the necessary spatial information to differentiate among different car positions. The policies based on pixel intensities and VGG conv5_3 features perform worse on the novel cars. However, VGG features conv1_2 through conv4_3 achieve some degree of generalization on the novel cars.

We show sample trajectories in Table 2.1. The policy based on pixel-intensities is susceptible to occlusions and distractor objects that appear in the target image or during executions. This is because distinguishing these occlusions and distractors from the cars cannot be done using just RGB features.

Comparison of Weightings from Other Optimization Methods

We compare our policy using conv4_3 feature dynamics, with weights optimized by FQI, against policies that use these dynamics but with either no feature weighting or weights optimized by other algorithms.

For the case of no weighting, we use a single feature weight w but optimize the relative weighting of the controls λ with the cross entropy method (CEM) (De Boer et al., 2005). For the other cases, we learn the weights with Trust Region Policy Optimization (TRPO) (Schulman et al., 2015). Since the servoing policy is the minimizer of a quadratic objective (Equation (2.3)), we represent the policy as a neural network that has a matrix inverse operation at the output. We train this network for 2 and 50 sampling iterations, and use a batch size of 4000 samples per iteration. All of these methods use the same feature representation as ours, the only difference being how the weights w and λ are chosen.

We report the average costs of these methods on the right of Figure 2.6. In 2 sampling iterations, the policy learned with TRPO does not improve by much, whereas our policy learned with FQI significantly outperforms the other policies. The policy learned with TRPO improves further in 50





























































Feature Dynamics	Observations from Test Executions										Cost
pixel, locally connected											0.95
											6.26
											14.49
VGG conv4_3											0.38
											0.48
											1.02

Table 2.1: Sample observations from test executions in our experiments with the novel cars, and the costs for each trajectory, for different feature dynamics. We use the weights learned by our FQI algorithm. In each row, we show the observations of every 10 steps and the last one. The first observation of each trajectory is used as the target observation. The trajectories shown here were chosen to reflect different types of behaviors. The servoing policy based on pixel feature dynamics can generally follow cars that can be discriminated based on RGB pixel intensities (e.g., a yellow car with a relatively uniform background). However, it performs poorly when distractor objects appear throughout the execution (e.g., a lamp) or when they appear in the target image (e.g., the crosswalk markings on the road). On the other hand, VGG conv4_3 features are able to discriminate the car from distractor objects and the background, and the feature weights learned by the FQI algorithm are able to leverage this. Additional sample executions with other feature dynamics can be found in Table A.2 in the Appendix.

iterations; however, the cost incurred by this policy is still about one and a half times the cost of our policy, despite using more than 100 times as many trajectories.

Comparison to Prior Methods

We also consider other methods that do not use the dynamics-based servoing policy that we propose. We report their average performance on the left of Figure 2.6.

For one of the prior methods, we train a convolutional neural network (CNN) policy end-to-end with TRPO. The policy is parametrized as a 5-layer CNN, consisting of 2 convolutional and 3 fully-connected layers, with ReLU activations except for the output layer; the convolutional layers use 16 filters (4×4 , stride 2) each and the first 2 fully-connected layers use 32 hidden units each. The policy takes in raw pixel-intensities and outputs controls.

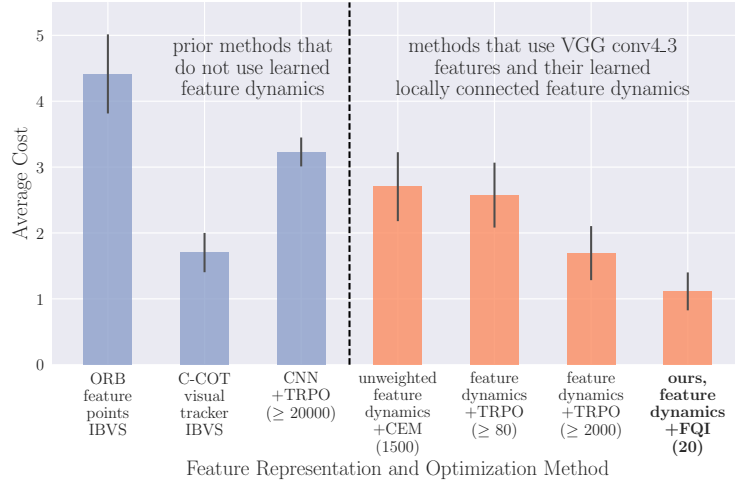


Figure 2.6: Comparison of costs on test executions of prior methods against our method based on VGG conv4_3 feature dynamics. These costs are from executions with the training cars; the costs are comparable when testing with the novel cars (Table A.1). The first two methods use classical image-based visual servoing (IBVS) with feature points from an off-the-shelf keypoint detector and descriptor extractor (ORB features), and with feature points extracted from bounding boxes predicted by a state-of-the-art visual tracker (C-COT tracker), respectively. The third method trains a convolutional neural network (CNN) policy end-to-end with Trust Region Policy Optimization (TRPO). The other methods use the servoing policy based on VGG conv4_3 feature dynamics, either with unweighted features or weights trained with TRPO for either 2 or 50 iterations. In the case of unweighted features, we learned the weights λ and a single weight w with the cross entropy method (CEM). We report the number of training trajectories in parenthesis for the methods that require learning. For TRPO, we use a fixed number of training *samples* per iteration, whereas for CEM and FQI, we use a fixed number of training *trajectories* per iteration. We use a batch size of 4000 samples for TRPO, which means that at least 40 trajectories were used per iteration (since trajectories can terminate early, i.e. in less than 100 time steps).

This policy achieves a modest performance (although still worse than the policies based on conv4_3 feature dynamics) but it requires significantly more training samples than any of the other learning-based methods. We also trained CNN policies that take in extracted VGG features (without any dynamics) as inputs, but they perform worse (see Table A.3 in the Appendix). This suggests that given a policy parametrization that is expressive enough and given a large number of training samples, it is better to directly provide the raw pixel-intensity images to the policy instead of extracted VGG features. This is because VGG features are not optimized for this task and their representation loses some information that is useful for servoing.

The other two prior methods use classical image-based visual servoing (IBVS) (Chaumette and Hutchinson, 2006) with respect to Oriented FAST and Rotated BRIEF (ORB) feature points (Ruble et al., 2011), or feature points extracted from a visual tracker. For the former, the target features consist of only the ORB feature points that belong to the car, and this specifies that the car is relevant for the task. For the tracker-based method, we use the Continuous Convolution Operator

Tracker (C-COT) (Danelljan et al., 2016) (the current state-of-the-art visual tracker) to get bounding boxes around the car and use the four corners of the box as the feature points for servoing. We provide the ground truth car’s bounding box of the first frame as an input to the C-COT tracker. For all of the IBVS methods, we provide the ground truth depth values of the feature points, which are used in the algorithm’s interaction matrix⁵.

The first method performs poorly, in part because ORB features are not discriminative enough for some of the cars, and the target feature points are sometimes matched to feature points that are not on the car. The tracker-based method achieves a relatively good performance. The gap in performance with respect to our method is in part due to the lack of car dynamics information in the IBVS model, whereas our method implicitly incorporates that in the learned feature dynamics. It is also worth noting that the tracker-based policy runs significantly slower than our method. The open-source implementation of the C-COT tracker⁶ runs at about 1 Hz whereas our policy based on conv4_3 features runs at about 16 Hz. Most of the computation time of our method is spent computing features from the VGG network, so there is room for speedups if we use a network that is less computationally demanding.

2.7 Discussion

Manual design of visual features and dynamics models can limit the applicability of visual servoing approaches. We described an approach that combines learned visual features with learning predictive dynamics models and reinforcement learning to learn visual servoing mechanisms. Our experiments demonstrate that standard deep features, in our case taken from a model trained for object classification, can be used together with a bilinear predictive model to learn an effective visual servo that is robust to visual variation, changes in viewing angle and appearance, and occlusions. For control we propose to learn Q-values, building on fitted Q-iteration, which at execution time allows for one-step lookahead calculations that optimize long term objectives. Our method can learn an effective visual servo on a complex synthetic car following benchmark using just 20 training trajectory samples for reinforcement learning. We demonstrate substantial improvement over a conventional approach based on image pixels or hand-designed keypoints, and we show an improvement in sample-efficiency of more than two orders of magnitude over standard model-free deep reinforcement learning algorithms.

⁵The term interaction matrix, or feature Jacobian, is used in the visual servo literature to denote the Jacobian of the features with respect to the control.

⁶<https://github.com/martin-danelljan/Continuous-ConvOp>

Chapter 3

Self-Supervised Visual Planning with Temporal Skip Connections

3.1 Introduction

Chapter 2 exemplifies how single-step dynamics models can be used in the context of visual servoing. Although these models are sufficient to accomplish tasks that can be solved greedily, multi-step dynamics models are necessary to accomplish more complex tasks that require planning. In this chapter, we use multi-step dynamics models for this purpose.

In order for a robot to be able to predict what will happen in response to its actions, it needs a representation of the environment that is suitable for prediction. The last chapter uses visual features for its representation, in which the features were extracted using a pre-trained network. However, in complex and open-world environments, it is difficult to construct a concise and sufficient representation for prediction. At a high level, the number and types of objects in the scene might change substantially trial to trial, making it impossible to provide a single fixed-size representation. At a low level, the type of information that is important about each object might change from object to object: the motion of a rigid box might depend only on its shape and friction coefficient, while the motion of a deformable stuffed animal might depend on a variety of other factors which are hard to determine by hand. Instead of engineering representations for different object types and object scenes, we can instead directly predict the robot's sensory observations. The rationale behind this is that, if the robot is able to predict future observations, it has acquired a sufficient understanding of the environment that it can leverage for planning actions.

A major challenge in directly predicting visual observations for robotic control lies in deducing

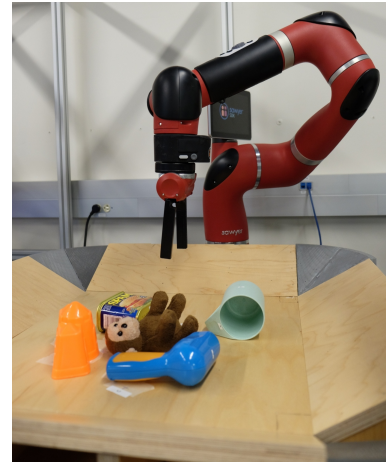


Figure 3.1: The robot learns to move new objects from self-supervised experience.

the spatial arrangement of objects from ambiguous observations. For example, when one object passes in front of another, the robot must remember that the occluded object still persists. In human visual perception, this is referred to as object permanence, and is known to take several months to emerge in infants (Bogartz et al., 2000). When the robot is commanded to manipulate an object which can become occluded during a manipulation, it must be able to accurately predict how that object will respond to an occlusion. In this work, we propose a visual predictive model for robotic control that can reason about spatial arrangements of objects in 3D, while using only monocular images and without providing any form of camera calibration. Previous learning methods have proposed to explicitly model object motion in image-space (Finn and Levine, 2017; De Brabandere et al., 2016) or to explicitly model 3D motion using point cloud measurements from depth cameras (Byravan and Fox, 2017), and lacked the capability to maintain information about objects which are occluded during the predicted motion. We propose a simple model that does not require explicitly deducing full 3D structure in the scene, but does provide effective handling of occlusions by storing the appearance of occluded objects in memory. Furthermore, unlike prior prediction methods (Byravan and Fox, 2017; Walker et al., 2015) our model does not require any additional, external systems for providing point-to-point correspondences between frames of different time-steps.

The technical contribution of this work is three-fold. First, we present a video prediction method that can more accurately maintain object permanence through occlusions, by incorporating temporal skip connections. Second, we propose a planning objective for control through video prediction that leads to significantly improved long-term planning performance, including planning through occlusions, when compared to prior work (Finn and Levine, 2017). Finally, we propose a mechanism for planning with both discrete and continuous actions with video prediction models. Our evaluation demonstrates that these components can be combined to enable a learned video prediction model to perform a range of real-world pushing tasks. Our experiments include manipulation of previously unseen objects, handling multiple objects, pushing objects around obstructions, and moving the arm around and over other obstacle-objects, representing a significant advance in the range and complexity of skills that can be acquired through entirely self-supervised learning.

3.2 Related Work

Large-Scale, Self-Supervised Robotic Learning. Large-scale robotic data collection has been explored in a number of recent works. John Oberlin (2015) proposed to collect object scans using multiple robots. Several prior works have focused on autonomous data collection for individual skills, such as grasping (Pinto and Gupta, 2016; Levine et al., 2016b) or obstacle avoidance (Kahn et al., 2017; Gandhi et al., 2017). In contrast to these methods, our approach learns predictive models that can be used to perform a variety of manipulations, and does not require a success measure or reward function during data collection. Several prior approaches have also sought to learn inverse or forward models from raw sensory data without any supervision (Agrawal et al., 2016; Finn and Levine, 2017). While these methods demonstrated effective generalization to new

objects, they were limited in the complexity of tasks and time-scale at which these tasks could be performed. The method proposed by Agrawal et al. (2016) was able to plan single pokes, and then greedily execute multiple pokes in sequence. The method of Finn and Levine (2017) performed long-horizon planning, but was only effective for short motions. In our comparisons to this method, we demonstrate a substantial improvement in the length and complexity of manipulations that can be performed with our models.

Sensory Prediction Models. Action-conditioned video prediction has been explored in the context of synthetic video game images (Oh et al., 2015; Chiappa et al., 2017) and robotic manipulation (Boots et al., 2014; Finn et al., 2016a; Kalchbrenner et al., 2017), and video prediction without actions has been studied for unstructured videos (Mathieu et al., 2016; Xingjian et al., 2015; Vondrick et al., 2016) and driving (Lotter et al., 2017; De Brabandere et al., 2016).

Several works have sought to use more complex distributions for future images, for example by using autoregressive models (Kalchbrenner et al., 2017; Reed et al., 2017). While this often produces sharp predictions, the resulting models are extremely demanding computationally, and have not been applied to real-world robotic control. In this work, we extend video prediction methods that are based on predicting a transformation from the previous image (Finn et al., 2016a; De Brabandere et al., 2016). Prior work has also sought to predict motion directly in 3D, using 3D point clouds obtained from a depth camera (Byravan and Fox, 2017), requiring point-to-point correspondences over time, which makes it hard to apply to previously unseen objects. Our predictive model is effective for a wide range of real-world object manipulations and does not require 3D depth sensing or point-to-point correspondences between frames. Prior work has also proposed to plan through learned models via differentiation, though not with visual inputs (Lenz and Saxena, 2015). We instead use a stochastic, sampling-based planning method (Rubinstein and Kroese, 2013; Finn and Levine, 2017), which we extend to handle a mixture of continuous and discrete actions.

3.3 Preliminaries

In this section, we define our image-based robotic control problem, present a formulation of visual model predictive control (visual MPC) over pixel motion, and summarize prior video prediction models based on image transformation.

Visual Model Predictive Control

Our visual MPC problem formulation follows the problem statement outlined in prior work (Finn and Levine, 2017). We assume that the user defines a goal for the robot in terms of pixel motion: given an image from the robot’s camera, the user can choose one or more pixels in the image, and choose a destination where each pixel should be moved. For example, the user might select a pixel on an object and ask the robot to move it 10 cm to the left. Formally, the user specifies P source pixel locations $d_0^{(1)}, \dots, d_0^{(P)}$ in the initial image I_0 , and P goal locations $g^{(1)}, \dots, g^{(P)}$. The source and goal pixel locations are denoted by the coordinates $(x_d^{(i)}, y_d^{(i)})$ and $(x_g^{(i)}, y_g^{(i)})$. Given a

goal, the robot plans for a sequence of actions $\mathbf{a}_{1:T}$ over T time steps, where T is the planning horizon. The problem is formulated as the minimization of a cost function c which depends on the predicted pixel positions $d_t^{(j)}$. The planner makes use of a learned model that predicts pixel motion. Given a distribution over pixel positions $P_{t_0, d^{(i)}} \in \mathbb{R}^{H \times W}$, $\sum_{H, W} P_{t_0, d^{(i)}} = 1$ at time $t = 0$, the model predicts distributions over its positions $P_{t, d^{(i)}}$ at time $t \in \{1, \dots, T\}$. To achieve the best results with imperfect models, the actions can be iteratively replanned at each real-world time step $\tau \in \{0, \dots, \tau_{max}\}$ following the framework of model-predictive control (MPC): at each real-world step τ , the model is used to plan T steps into the future, and the first action of the plan is executed. At the first real-world time step $\tau = 0$, the distribution $P_{t=0, d^{(i)}}$ is initialized as 1 at the location of the designated pixel and zero elsewhere. In subsequent steps ($\tau > 0$), the 1-step ahead prediction of the previous step is used to initialize $P_{t=0, d^{(i)}}$. The cost c is a function of $P_{t, d^{(i)}}$ at all predicted time steps $0 \dots T$. Prior work proposed to use the negative log-probability of placing each pixel at its goal location as the cost (Finn and Levine, 2017). We will discuss in Section 3.5 how performance can be improved substantially by instead using expected distances. Planning is performed using the cross-entropy method (CEM), a gradient-free optimization procedure that consists of iteratively resampling action sequences and refitting Gaussian distributions to the actions with the best predicted cost. Further details can be found in prior work (Finn and Levine, 2017).

Video Prediction via Pixel Transformations

Visual MPC requires a model that can effectively predict the motion of the selected pixels $d_0^{(1)}, \dots, d_0^{(P)}$ up to T steps into the future. In this work, we extend the model proposed in (Finn et al., 2016a), where this flow prediction capability emerges implicitly, and therefore no external pixel motion supervision is required. Future images are generated by transforming past observations. The model uses stacked convolutional LSTMs that predict a collection of pixel transformations at each time step, with corresponding composition masks. In this model, the previous image I_t is transformed by N separate transformations, and all of the transformed images $\tilde{I}_t^{(i)}$ are composited together according to weights obtained from the predicted masks. Intuitively, each transformation corresponds to the motion of a different object in the scene, and each mask corresponds to the spatial extents of that object. Let us denote the N transformed images as $\tilde{I}_t^{(1)}, \dots, \tilde{I}_t^{(N)}$, and the predicted masks as $\mathbf{M}_1, \dots, \mathbf{M}_N$, where each 1-channel mask is the same size as the image. The next image prediction is then computed by compositing the images together using the masks: $\hat{I}_{t+1} = \sum_{i=1}^N \tilde{I}_t^{(i)} \mathbf{M}_i$. To predict multiple time steps into the future, the model is applied recursively. These transformations can be represented as convolutions, where each pixel in the transformed image is formed by applying a convolution kernel to the previous one. This method can represent a wide range of local transformations. When these convolution kernels are normalized, they can be interpreted as transition probabilities, allowing us to make probabilistic predictions about future locations of individual pixels. To predict the future positions of the designated pixels $d^{(i)}$, the same transformations which are used for the images are applied to $P_{t, d^{(i)}}$ such that $P_{t+1, d^{(i)}} = \frac{1}{P_s} \sum_{i=1}^N \tilde{P}_{t, d^{(i)}}^{(i)} \mathbf{M}_i$, where $\frac{1}{P_s}$ is a normalizing constant to ensure that $P_{t+1, d^{(i)}}$ adds up to 1 over the spatial dimension of the image. Since this prior predictive model outputs a single image at

each time step, it is unable to track pixel positions through occlusions. Therefore, this model is only suitable for planning motions where the user-selected pixels are not occluded during the manipulation, which restricts its use in cluttered environments or with multiple selected pixels. In the next section, we discuss our proposed occlusion-aware model, which lifts this limitation by employing temporal skip connections.

3.4 Skip Connection Neural Advection Model

To enable effective tracking of objects through occlusions, we propose an extension to the dynamic neural advection (DNA) model (Finn et al., 2016a) that incorporates temporal skip connections. This model uses a similar multilayer convolutional LSTM structure; however, the transformations are now conditioned on a context of previous images. In the most generic version, this involves conditioning the transformation at time t on all of the previous images I_0, \dots, I_t , though in practice we found that a greatly simplified version of this model performed just as well in practice. We will therefore first present the generic model, and then describe the practical simplifications. We refer to this model as the skip connection neural advection model (SNA), since it handles occlusions by copying pixels from prior images in the history such that when a pixel is occluded (e.g., by the robot arm or by another object) it can still reappear later in the sequence. When predicting the next image \hat{I}_{t+1} , the generic SNA model transforms each image in the history according to a different transformation and with different masks to produce \hat{I}_{t+1} (see Figure B.2 in the appendix):

$$\hat{I}_{t+1} = \sum_{j=t-T}^t \sum_{i=1}^N \mathbf{M}_{i,j} \tilde{I}_j^{(i)} \quad (3.1)$$

In the case where $t < T$, negative values of j simply reuse the first image in the sequence. This generic formulation can be computationally expensive, since the number of masks and transformations scales with $T \times N$. A more tractable model, which we found works comparably well in practice in our robotic manipulation setting, assumes that occluded objects are typically static throughout the prediction horizon. This assumption allows us to dispense the intermediate transformations and only provide a skip connection from the very first image in the sequence, which is also the only real image, since all of the subsequent images are predicted by the model:

$$\hat{I}_{t+1} = I_0 \mathbf{M}_{N+1} + \sum_{i=1}^N \tilde{I}_t^{(i)} \mathbf{M}_i \quad (3.2)$$

This model only needs to output $N + 1$ masks. We observed similar prediction performance when using a transformed initial image \tilde{I}_0 in place of I_0 , and therefore used the simplified model in Equation (3.2) in all of our experiments. We provide an example of the model recovering from occlusion in Figure 3.5. In the figure, the arm moves in front of the designated pixel, marked in blue in Figure 3.3. The graphs in Figure 3.4 show the predicted probability of the designated pixel, which is stationary during the entire motion, being at its original position at each step. Precisely when the arm occludes the designated pixel, the pixel’s probability of being at this point

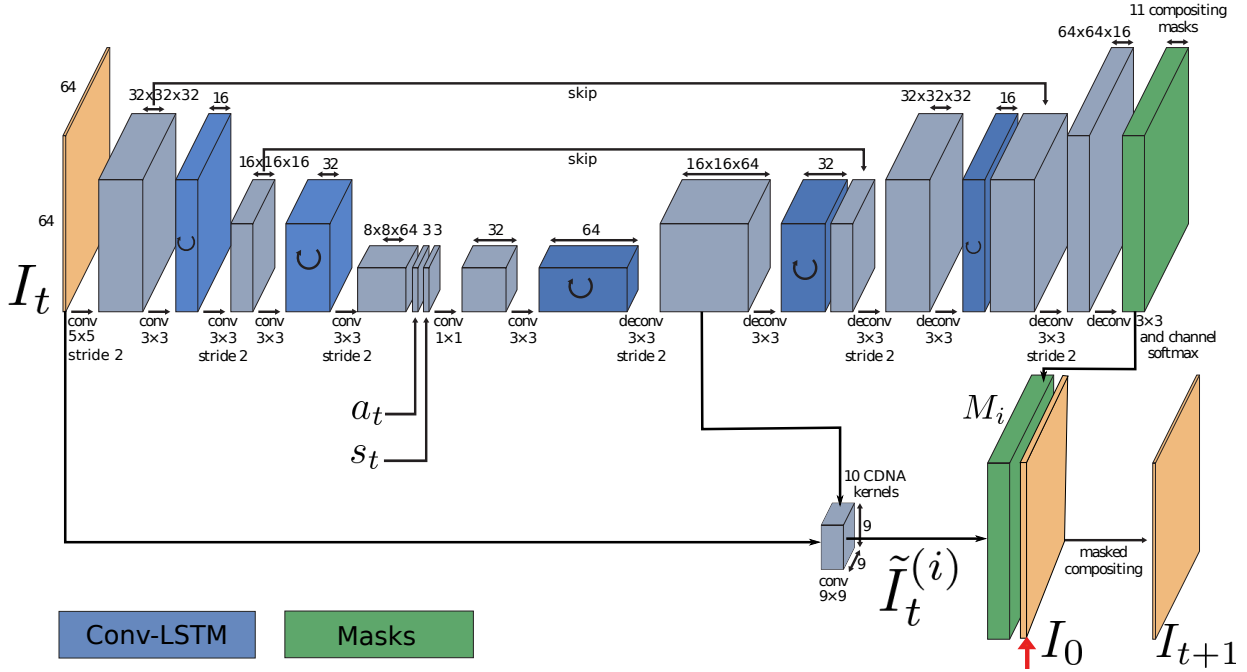


Figure 3.2: Simplified SNA model based on Equation (3.2). The red arrow indicates where the image from the first time step I_0 is concatenated with the transformed images $\tilde{I}_t^{(i)}$ multiplying each channel with a separate mask to produce the predicted frame for step $t + 1$.

decreases. This indicates that the model is ‘unsure’ where this pixel is. When the arm unoccludes the designated pixel, it should become visible again, and the probability of the designated pixel being at its original position should go up. In the case of the DNA model and its variants (Finn et al., 2016a), the probability mass does not increase after the object reappears. This is because the DNA model cannot recover information about object that it has ‘overwritten’ during its predictions. Consequently, for these models, the probability stays low and *moves with the arm*, which causes the model to believe that the occluded pixel also moves with the arm. We identified this as one of the major causes of planning failure when using the prior models. By contrast, in the case of our SNA model, the probability of the correct pixel position increases again rapidly right after the arm unoccludes the object. Furthermore, the probability of the unoccluded object’s position becomes increasingly sharp at its original position as the arm moves further away.

3.5 Visual MPC with Pixel Distance Costs

The choice of objective function for visual MPC has a large impact on the performance of the method. Intuitively, as the model’s predictions get more uncertain further into the future, the planner relies more and more on the cost function to provide a reasonable estimate of the overall distance to the goal. Prior work used the probability of the chosen pixel(s) reaching their goal position(s) after T time steps, as discussed in Section 3.3. When the trajectories needed to reach

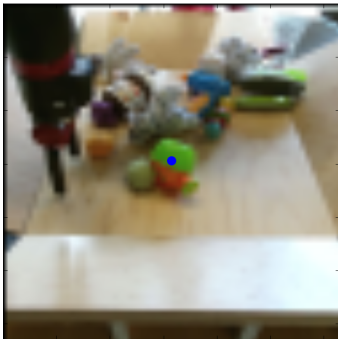


Figure 3.3: The blue dot indicates the designated pixel

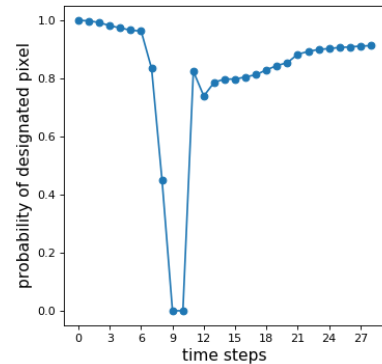
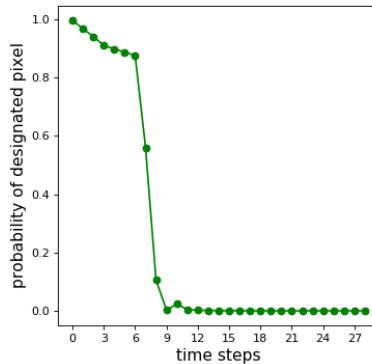
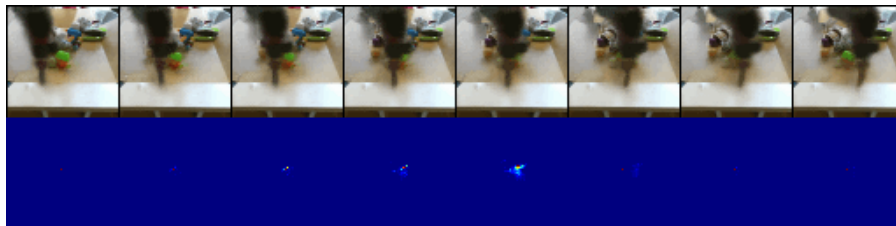


Figure 3.4: Predicted probability $P_{d^{(0)}}(t)$ of the designated pixel being at the location of the blue dot indicated in Figure 3.3 for the DNA model (left) and the SNA model (right).



(a) Skip connection neural advection (SNA) does not erase or move objects in the background



(b) Standard DNA (Finn and Levine, 2017) exhibits undesirable movement of the distribution $P_{d^{(0)}}(t)$ and erases the background

Figure 3.5: Top rows: Predicted images of arm moving *in front of* green object with designated pixel (as indicated in Figure 3.3). Bottom rows: Predicted probability distributions $P_{d^{(0)}}(t)$ of designated pixel obtained by repeatedly applying transformations.

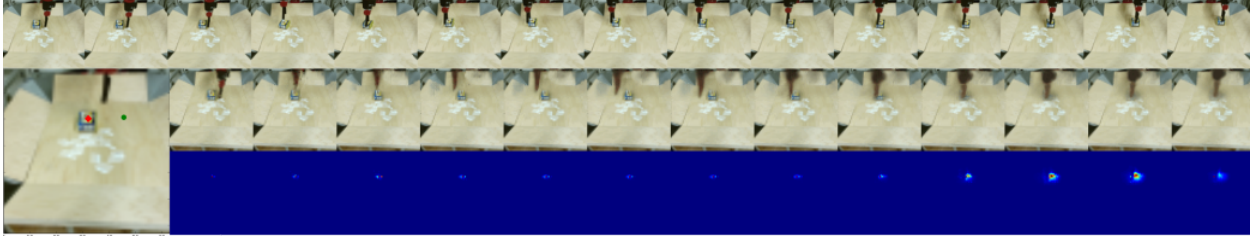


Figure 3.6: Integrating actions in the vertical direction enables moving over obstacles. This results in more natural and shorter paths. Top row: executed trajectory, left: red diamond indicates designated pixel, green dot indicates goal, 2nd and 3rd row: predictions and probability of designated pixel at time step 3.

the goal are long, the objective provides relatively little information about the progress towards the goal, since most of the predicted probabilities will have a value close to zero at the goal-pixel. In this work, we propose a cost function that still makes use of the uncertainty estimates about the pixel position, but provides a substantially smoother planning objective, resulting in improved performance for more complex, longer-horizon tasks. A straightforward choice of smooth cost function in deterministic settings is the Euclidean distance between the current and desired location of the desired pixel(s), given by $\|d_{t'} - g\|_2$. Since our video prediction model produces a distribution over the pixel location at each time step, we can use the expected value of this distance as a cost function, summed over the entire horizon T :

$$\begin{aligned} c_{t+1:t+T}(g) &= c(I_{t-1:t}, \mathbf{x}_{t-1:t}, \mathbf{a}_{t:t+T}, d_t, g) \\ &= \sum_{t'=t+1, \dots, t+T} \mathbb{E}_{d_{t'} \sim P_{t', d^{(i)}}} [\|d_{t'} - g\|_2] \end{aligned} \quad (3.3)$$

where the expectation can be computed by summing over all of the positions in each predicted image, and the cost corresponds to a (element-wise) Hadamard product of the pixel location probability map and the distances between each pixel position and the goal. This cost function encourages the movement of the designated objects in the right direction for each step of the execution, regardless of whether the g position can be reached within T time steps. For multi-objective tasks with multiple designated pixels $d^{(i)}$ the costs are summed together weighting them equally. Although the use of well-shaped cost functions in MPC has been explored extensively in prior work (Li et al., 2005), the combination of cost shaping and visual MPC has not been studied extensively.

3.6 Sampling-Based MPC with Continuous and Discrete Actions

The choice of action representation for visual MPC has a significant effect on the performance of the resulting controller. The action representation must allow the planner sufficient freedom to maneuver the arm to perform a wide variety of tasks, while constraining sufficiently so as to create a tractable search space. We found that a particularly well-suited action representation for tabletop

manipulation can be constructed by combining continuous and discrete actions, in contrast to prior work that used only continuous end-effector motion vectors as actions (Finn and Levine, 2017). The actions consist of the horizontal motion of the end-effector, as well as a discrete “lift” action that allows the robot to command the end-effector to lift off the table vertically. The discrete lifting action can take on N values (4 in our implementation) that specify for how many time steps the robot should lift the end-effector off the table. Unlike with continuous vertical motion commands, these discrete commands result in the end-effector staying off the table for multiple time steps even during random data collection. In order to incorporate this hybrid action space into the stochastic CEM-based optimization in visual MPC, we sample real-valued parameters for the discrete action, and then round them to the nearest valid integer to obtain discrete actions. As usual with CEM, we iteratively refit a multivariate Gaussian distribution to the best performing samples (which in our case are chosen to be in the 90th percentile of samples), treating the entire action vector as if it was continuous. Figure 3.6 shows an example of a situation where the discrete vertical motion component of the action is used by our model to lift the gripper over the object in order to push it from the opposite side.

3.7 Experiments

Our experimental evaluation compares the proposed occlusion-aware SNA video prediction model, as well as the improved cost function for planning, with a previously proposed model based on dynamic neural advection (DNA) (Finn and Levine, 2017). We use a Sawyer robot, shown in Figure 3.1, to push a variety of objects in a tabletop setting. In the appendix, we include a details on hyperparameters, analysis of sample complexity, and a discussion of robustness and limitations. We evaluate long pushes and multi-objective tasks where one object must be pushed without disturbing another. The supplementary video and links to the code and data are available on the project’s website¹

Figure 3.7 shows an example task for the pushing benchmark. We collected 20 trajectories with 3 novel objects and 1 training object. Table 3.1 shows the results for the pushing benchmark. The column *distance* refers to the mean distance between the goal pixel and the designated pixel at the final time-step. The column *improvement* indicates how much the designated pixel of the objects could be moved closer to their goal (or further away for negative values) compared to the starting location. The true locations of the designated pixels after pushing were annotated by a human.

The results in Table 3.1 show that our proposed planning cost in Equation (3.3) substantially outperforms the planning cost used in prior work (Finn and Levine, 2017). The performance of the SNA model in these experiments is comparable to the prior DNA model (Finn

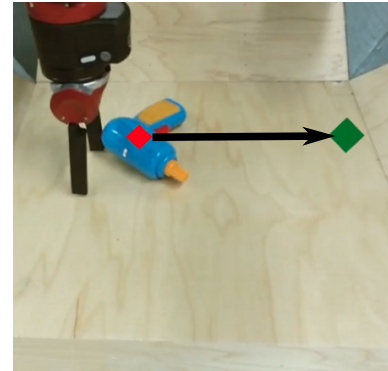


Figure 3.7: Pushing task. The designated pixel (red diamond) needs to be pushed to the green circle.

¹<https://sites.google.com/view/sna-visual-mpc>

	seen		unseen	
	distance mean and standard deviation	improvement mean and standard deviation	distance mean and standard deviation	improvement mean and standard deviation
random actions	29.4 ± 3.2	-0.9 ± 2.3	N/A	N/A
DNA with the distance metric of Finn and Levine (2017)	25.9 ± 12.2	5.2 ± 13.7	24.6 ± 10.6	2.1 ± 12.4
DNA with our distance metric	9.2 ± 7.4	15.0 ± 11.5	17.5 ± 10.2	8.3 ± 11.8
SNA with our distance metric	13.0 ± 5.6	12.4 ± 8.8	18.2 ± 9.5	7.7 ± 10.5

Table 3.1: Results of the pushing benchmark on 20 different object/goal configurations. Units are pixels in the 64×64 images.

and Levine, 2017) when both use the new planning cost, since this task does not involve any occlusions. Although the DNA model has a slightly better mean distance compared to SNA, it is well within the standard deviation, suggesting that the difference is not significant.

To examine how well each approach can handle occlusions, we devised a second task that requires the robot to push one object, while keeping another object stationary. When the stationary object is in the way, the robot must move the goal object around it, as shown in Figure 3.8 on the left. While doing this, the gripper may occlude the stationary object, and the task can only be performed successfully if the model can make accurate predictions through this occlusion. These tasks are specified by picking one pixel on the target object, and one on the obstacle. The obstacle is commanded to remain stationary, while the target object destination location is chosen on the other side of the obstacle.

We used four different object arrangements, with two training objects and two objects that were unseen during training. We found that, in most of the cases, the SNA model was able to find a valid trajectory, while the prior DNA model was mostly unable to find a solution. Figure 3.8 shows an example of the SNA model successfully predicting the position of the obstacle through an occlusion and finding a trajectory that avoids the obstacle. These findings are reflected by our quantitative results shown in Table 3.2, indicating the importance of temporal skip connections.

3.8 Discussion

We showed that visual predictive models trained entirely with videos from random pushing motions can be leveraged to build a model-predictive control scheme that is able to solve a wide range multi-objective pushing tasks in spite of occlusions. We also demonstrated that we can combine both discrete and continuous actions in an action-conditioned video prediction framework to perform more complex behaviors, such as lifting the gripper to move over objects.

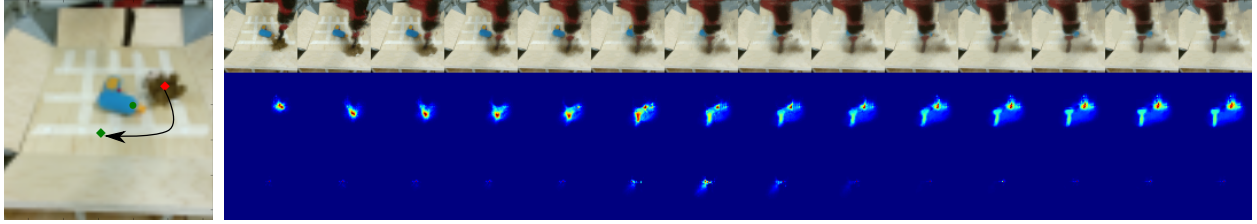


Figure 3.8: Left: Task setup with green dot marking the obstacle. Right, first row: the predicted frames generated by SNA. Second row: the probability distribution of the designated pixel on the *moving* object (brown stuffed animal). Note that part of the distribution shifts down and left, which is the indicated goal. Third row: the probability distribution of the designated pixel on the obstacle-object (blue power drill). Although the distribution increases in entropy during the occlusion (in the middle), it then recovers and remains on its original position.

	seen		unseen	
	moved improvement mean and standard deviation	stationary improvement mean and standard deviation	moved improvement mean and standard deviation	stationary improvement mean and standard deviation
DNA (Finn and Levine, 2017)	3.5 ± 5.0	-1.4 ± 0.4	0.8 ± 1.1	-1.1 ± 0.9
SNA (ours)	8.2 ± 2.9	-0.9 ± 0.7	10.6 ± 3.7	-1.5 ± 0.9

Table 3.2: Results for multi-objective pushing on 8 object and goal configurations with 2 seen and 2 novel objects. Values indicate improvement in distance from starting position, higher is better. Units are pixels in the 64×64 images.

Although our method achieves significant improvement over prior work, it does have a number of limitations. The behaviors in our experiments are relatively short. In principle, the visual MPC approach can allow the robot to repeatedly retry the task until it succeeds, but the ability to retry is limited by the model’s ability to track the target pixel: the tracking deteriorates over time, and although our model achieves substantially better tracking through occlusions than prior work, repeated occlusions still cause it to lose track. Improving the quality of visual tracking of the designated pixels may allow the system to retry the task until it succeeds. More complex behaviors, such as picking and placing (e.g., to arrange a table setting), may also be difficult to learn with only randomly collected data. We expect that more goal-directed data collection would substantially improve the model’s ability to perform complex tasks. Furthermore, better predictive models that incorporate hierarchical structure or reason at variable time scales would further improve the capabilities of visual MPC to carry out temporally extended tasks. Fortunately, as video prediction methods continue to improve, we expect methods such as ours to further improve in their capability.

Chapter 4

Stochastic Adversarial Video Prediction

4.1 Introduction

Chapter 3 uses a deterministic video prediction model for planning and determining which actions can bring about desired outcomes. Unfortunately, accurate and naturalistic video prediction remains an open problem.

Ambiguity is a major challenge in video prediction. While frames in the immediate future can be extrapolated with high precision, the space of possibilities diverges beyond a few frames, and the problem becomes multimodal. Methods that use deterministic models and loss functions unequipped to handle this inherent uncertainty, such as mean-squared error (MSE), will average together possible futures, producing blurry predictions. Prior works have explored stochastic models for video prediction (Babaeizadeh et al., 2018; Denton and Fergus, 2018), using the framework of variational autoencoders (VAEs) (Kingma and Welling, 2014). These models predict possible futures by sampling latent variables. During training, they optimize a variational lower bound on the likelihood of the data in a latent variable model. However, the posterior is still a pixel-wise MSE loss, corresponding to the log-likelihood under a fully factorized Gaussian distribution. This causes them to make blurry and unrealistic predictions when the latent variables alone do not adequately capture the uncertainty.

Another relevant branch of recent work has been generative adversarial networks (GANs) (Goodfellow et al., 2014) for image generation. Here, a generator network is trained to produce images that are indistinguishable from real images, using a learned discriminator network trained to classify images as real or generated. The discriminator looks at the whole image, and is capable of modeling the joint distribution of pixels. Although this overcomes the limitations of pixel-wise losses, GANs are notoriously susceptible to mode collapse, especially in the conditional setting. This makes them difficult to apply to generation of diverse and plausible futures, conditioned on context frames.

To address these challenges, we propose a model that combines adversarial losses and latent variables to enable realistic stochastic video prediction. Our model consists of a video prediction network that can produce multiple futures by sampling time-varying stochastic latent variables.

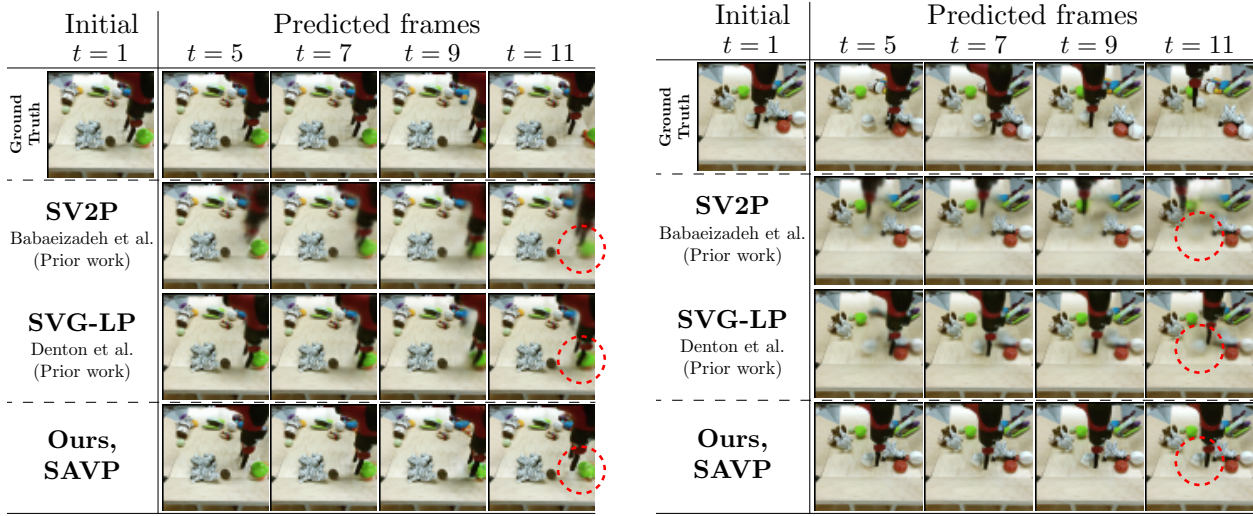


Figure 4.1: **Example results.** While the SV2P method (Babaeizadeh et al., 2018) produces blurry images, our method maintains sharpness and realism through time. The prior SVG-LP method (Denton and Fergus, 2018) produces sharper predictions of the arm, but still blurs out objects in the background or objects that interact with the robot.

At training time, an inference network estimates the distribution of these latent variables, and video discriminator networks classify generated videos from real. The full training objective is the variational lower bound used in VAEs combined with the adversarial loss used in GANs, as well as a perceptual loss that we introduce to further improve realism. This enables us to capture stochastic posterior distributions of videos while also modeling the spatiotemporal joint distribution of pixels.

The primary contribution of our work is a stochastic video prediction model based on VAE-GANs. Our experiments show that the VAE component greatly improves the diversity of the generated images, while the adversarial loss attains prediction results that are substantially more realistic than state-of-the-art methods, as shown in Figure 4.1. We further introduce a perceptual loss that incorporates features from the GAN discriminator into the reconstruction objective for each frame, and show that this substantially improves the accuracy of the predicted videos over our standard VAE-GAN video model. We present a comparison of various of prediction models and losses, including VAE, GAN, and VAE-GAN models, and evaluate them on prediction realism, diversity, and accuracy.

4.2 Related Work

Recent developments have led to impressive results in video generation and prediction. Earlier approaches to prediction focused on models that generate pixels directly using both feed-forward (Ranzato et al., 2014; Mathieu et al., 2016) and recurrent (Oh et al., 2015; Xingjian et al., 2015) architectures. An alternative to generating pixels is transforming them by applying a constrained geometric distortion to a previous frame (Finn et al., 2016a; De Brabandere et al., 2016; Xue et al.,

2016; Byravan and Fox, 2017; Vondrick and Torralba, 2017; van Amersfoort et al., 2017; Liu et al., 2017; Chen et al., 2017; Lu et al., 2017; Walker et al., 2015; 2016; Liang et al., 2017). Predictions are also affected by the training objective. Minimizing MSE loss can lead to strong results for deterministic synthetic videos (Oh et al., 2015; Chiappa et al., 2017), but on stochastic real-world videos, this loss can result in blurry predictions, as the model averages different futures.

Handling uncertainty is critical for addressing this issue. One approach is to model the full joint distribution using pixel-autoregressive models (van den Oord et al., 2016; Kalchbrenner et al., 2017; Reed et al., 2017), though training and inference are impractically slow. Another approach is to train a latent variable model, such as in variational autoencoders (VAEs) (Kingma and Welling, 2014). Conditional VAEs have been used for prediction of optical flow (Walker et al., 2016), single-frame prediction (Xue et al., 2016), and stochastic multi-frame video prediction (Babaeizadeh et al., 2018; Denton and Fergus, 2018). While these models can model distributions over possible futures, the prediction distribution is still fully factorized over pixels, which still tends to produce blurry predictions.

Adversarial losses (Goodfellow et al., 2014) for image generation can produce substantially improved realism. However, these networks tend to be difficult to train and are susceptible to mode collapse. A number of prior works have used adversarial losses for deterministic video prediction (Mathieu et al., 2016; Vondrick and Torralba, 2017; Villegas et al., 2017; Lu et al., 2017; Zhou and Berg, 2016; Bhattacharjee and Das, 2017). Several prior works have also sought to produce unconditioned video generations (Vondrick et al., 2016; Saito et al., 2017; Tulyakov et al., 2018) and conditional generation with input noise (Chen et al., 2017; Tulyakov et al., 2018; Wang et al., 2018). We show that a GAN model with input noise can indeed generate realistic videos, but fails to adequately cover the space of possible futures.

Prior works have combined VAEs and GANs to produce stochastic and realistic predictions. Walker et al. (2017) predicts videos of humans by decomposing the problem into a VAE that predicts stochastic future poses and a GAN that generates videos conditioned on those poses and an image. VAE-GANs, which jointly optimize the VAE and GAN losses, have shown promising results for unconditional and conditional image generation (Larsen et al., 2016; Bao et al., 2017; Zhu et al., 2017), but have not been applied to video prediction. The video prediction setting presents two important challenges. First, conditional image generation can handle large appearance changes between the input and output, but suffer when attempting to produce large spatial changes. The video prediction setting is precisely the opposite—the appearance remains largely the same across frames, but the most important changes are spatial. Secondly, video prediction involves sequential prediction, where it’s increasingly difficult to predict farther into the future.

4.3 Video Prediction with Stochastic Adversarial Models

Our goal is to learn a stochastic video prediction model that can predict videos that are diverse and perceptually realistic, and where all predictions are plausible futures for a given initial image. In practice, we use a short initial sequence of images (typically two frames), though we will omit this in our derivation for ease of notation. Our model consists of a recurrent generator network

G , which is a deterministic video prediction model that maps an initial image \mathbf{x}_0 and a sequence of latent random codes $\mathbf{z}_{0:T-1}$, to the predicted sequence of future images $\hat{\mathbf{x}}_{1:T}$. Intuitively, the latent codes encapsulate any ambiguous or stochastic events that might affect the future. At test time, we sample videos by first sampling the latent codes from a prior distribution $p(\mathbf{z}_t)$, and then passing them to the generator. We use a fixed unit Gaussian prior, $\mathcal{N}(0, 1)$. The training procedure includes elements of variational inference and generative adversarial networks. Before describing the training procedure, we formulate the problem in the context of VAEs and GANs.

Variational Autoencoders

Our recurrent generator predicts each frame given the previous frame and a random latent code. The previous frame passed to the generator is denoted as $\tilde{\mathbf{x}}_{t-1}$ to indicate that it could be a ground truth frame \mathbf{x}_{t-1} (for the initial frames) or the last prediction $\hat{\mathbf{x}}_{t-1}$. The generator specifies a distribution $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{z}_{t-1})$, parametrized as a fixed-variance Laplacian distribution with mean $\hat{\mathbf{x}}_t = G(\mathbf{x}_{t-1}, \mathbf{z}_{t-1})$. The likelihood of the data $p(\mathbf{x}_{1:T}|\mathbf{x}_0)$ cannot be directly maximized, since it involves marginalizing over the latent variables, which is intractable in general. Thus, we instead maximize the variational lower bound of the log-likelihood. We approximate the posterior with a recognition model $q(\mathbf{z}_t|\mathbf{x}_{t:t+1})$, which is parametrized as a conditionally Gaussian distribution $\mathcal{N}(\mu_{\mathbf{z}_t}, \sigma_{\mathbf{z}_t}^2)$, represented by a network $E(\mathbf{x}_{t:t+1})$. The encoder E is conditioned on adjacent frames in order to have time-varying latent variables, where each latent \mathbf{z}_t is temporally local and capture the ambiguity of the transition from frame \mathbf{x}_t to \mathbf{x}_{t+1} . This is a sensible choice for latent variables that are independent and identically distributed, which is the case for the per-time-step unit Gaussian prior. Another choice is to use temporally correlated latent variables, which would require a stronger prior (e.g. as in (Denton and Fergus, 2018)). For simplicity, we opted for the former.

During training, the latent code is sampled from $q(\mathbf{z}_t|\mathbf{x}_{t:t+1})$. The generation of each frame can be thought of as the reconstruction of frame $\hat{\mathbf{x}}_{t+1}$, where the ground truth frame \mathbf{x}_{t+1} (along with \mathbf{x}_t) is encoded into a latent code \mathbf{z}_t , and then it (along with the last frame) is mapped back to $\hat{\mathbf{x}}_{t+1}$. Since the latent code has ground truth information about the frame being reconstructed, the model is encouraged to use it during training. To allow back-propagation through the encoder, the reconstruction term is rewritten using the re-parametrization trick,

$$\mathcal{L}_1(G, E) = \mathbb{E}_{\mathbf{x}_{0:T}, \mathbf{z}_{0:T-1} \sim E(\mathbf{x}_{0:T})} \left[\sum_{t=0}^{T-1} \|\mathbf{x}_{t+1} - G(\tilde{\mathbf{x}}_t, \mathbf{z}_t)\|_1 \right]. \quad (4.1)$$

To enable sampling from the prior at test time, a regularization term encourages the approximate posterior to be close to the prior distribution,

$$\mathcal{L}_{\text{KL}}(E) = \mathbb{E}_{\mathbf{x}_{0:T}} \left[\sum_{t=0}^{T-1} \mathcal{D}_{\text{KL}}(E(\mathbf{x}_{t:t+1}) \| p(\mathbf{z}_t)) \right]. \quad (4.2)$$

Given some weight hyperparameters λ_1 and λ_{KL} , a VAE is trained to optimize

$$\min_{G, E} \lambda_1 \mathcal{L}_1(G, E) + \lambda_{\text{KL}} \mathcal{L}_{\text{KL}}(E). \quad (4.3)$$

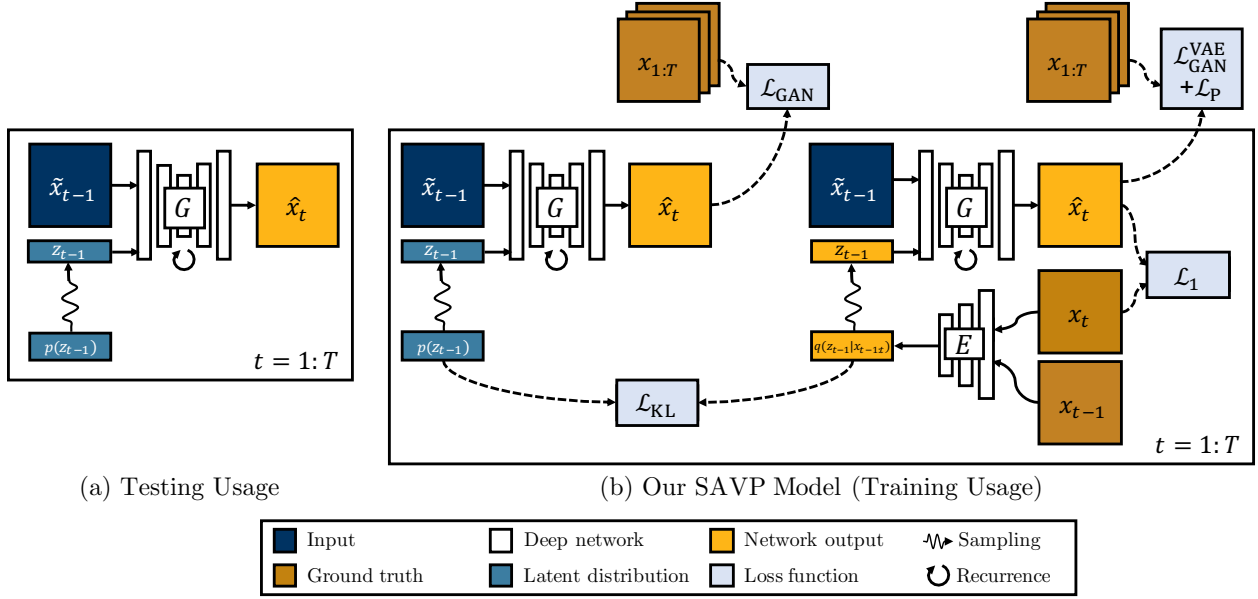


Figure 4.2: **Our proposed video prediction model.** (a) During testing, we synthesize new frames by sampling random latent codes \mathbf{z} from a prior distribution $p(\mathbf{z})$ independently at each time step. The generator G takes a previous frame and a latent code to synthesize a new frame. (b) During training, the generator is optimized to predict videos that match the distribution of real videos, using learned discriminators. The discriminators operate on entire sequences. We sample latent codes from two distributions: (1) the prior distribution, and (2) a posterior distribution approximated by a learned encoder E . For the latter, the regression \mathcal{L}_1 and feature matching \mathcal{L}_P losses are used. Separate discriminators D and D^{VAE} are used depending on the distribution used to sample the latent code.

Generative Adversarial Networks

Without overcoming the problem of modeling pixel covariances, it is likely not possible to produce sharp predictions. Indeed, as shown in our experiments, the pure VAE model tends to produce blurry futures. We can force the predictions to stay on the video manifold by matching the distributions of predicted and real videos. Given a classifier D that is capable of distinguishing generated videos $\hat{\mathbf{x}}_{1:T}$ from real videos $\mathbf{x}_{1:T}$, the generator can be trained to match the statistics of the real data distribution using this objective,

$$\mathcal{L}_{\text{GAN}}(G, D) = \mathbb{E}_{\mathbf{x}_{1:T}}[\log D(\mathbf{x}_{1:T})] + \mathbb{E}_{\mathbf{x}_{1:T}, \mathbf{z}_t \sim p(\mathbf{z}_t)}^{\mathbf{z}_t}_{t=0}^{T-1}[\log(1 - D(G(\mathbf{x}_0, \mathbf{z}_{0:T-1})))] \quad (4.4)$$

The overloaded notation $G(\mathbf{x}_0, \mathbf{z}_{0:T-1})$ indicates the generated sequence $\hat{\mathbf{x}}_{1:T}$. The classifier, which is not known a priori and is problem-specific, is realized as a deep discriminator network that is adversarially learned. A GAN is trained to optimize

$$\min_G \max_D \mathcal{L}_{\text{GAN}}(G, D). \quad (4.5)$$

Stochastic Adversarial Video Prediction

The VAE and GAN models provide complementary strengths. GANs use a learned loss function through the discriminator, which learns the statistics of natural videos. However, GANs can suffer from the problem of mode collapse, especially in the conditional setting (Pathak et al., 2016; Isola et al., 2017; Zhu et al., 2017). VAEs explicitly encourage the latent code to be more expressive and meaningful, since the learned encoder produces codes that are useful for making accurate predictions at training time. Our stochastic adversarial video prediction (SAVP) model combines both approaches, shown in Figure 4.2. Another term $\mathcal{L}_{\text{GAN}}^{\text{VAE}}$ is introduced, which is analogous to \mathcal{L}_{GAN} except that it uses latent codes sampled from $q(\mathbf{z}_t | \mathbf{x}_{t:t+1})$ and a video discriminator D^{VAE} .

In the video prediction setting, predicted videos should not only match the statistics of real videos, but the individual samples should also be accurate. That is the role of the \mathcal{L}_1 reconstruction loss, which brings samples from the posterior closer to the ground truth. However, this loss has the limitations of pixel-wise losses. We attempt to overcome this by using a perceptual loss,

$$\mathcal{L}_P(G, E, D^{\text{VAE}}) = \mathbb{E}_{\mathbf{x}_{0:T}, \mathbf{z}_{0:T-1} \sim E(\mathbf{x}_{0:T})} [d_{D^{\text{VAE}}}(\hat{\mathbf{x}}_{1:T}, \mathbf{x}_{1:T})]. \quad (4.6)$$

The function $d_{D^{\text{VAE}}}$ computes the sum of cosine distances of all the intermediate features given by D^{VAE} . This loss serves a similar role as the reconstruction loss, except that in the feature space of the learned discriminator. Our SAVP model is trained by alternating optimization of the following objectives,

$$\max_{D, D^{\text{VAE}}} \mathcal{L}_{\text{GAN}}(G, D) + \mathcal{L}_{\text{GAN}}^{\text{VAE}}(G, E, D^{\text{VAE}}), \quad (4.7)$$

$$\min_{G, E} \lambda_1 \mathcal{L}_1(G, E) + \lambda_{\text{KL}} \mathcal{L}_{\text{KL}}(E) + \lambda \mathcal{L}_{\text{GAN}}(G, D) + \lambda \mathcal{L}_{\text{GAN}}^{\text{VAE}}(G, E, D^{\text{VAE}}) + \lambda_P \mathcal{L}_P(G, E, D^{\text{VAE}}). \quad (4.8)$$

Network Architectures

The generator is a convolutional LSTM (Xingjian et al., 2015) that predicts pixel-space transformations between the current and next frame, with additional skip connections with the first frame as done in SNA (Ebert et al., 2017). At every time step, the network is conditioned on the current frame and latent code. After the initial frames, the network is conditioned on its own predictions. The conditioning on the latent codes is realized by concatenating them along the channel dimension to the inputs of all the convolutional layers of the convolutional LSTM.

The video discriminator is a feed-forward convolutional network with 3D filters, based on SNGAN (Miyato et al., 2018) but with the filters “inflated” from 2D to 3D. The network takes in a spatiotemporal cube of all the predicted pixels and outputs a single logit. The ground-truth context frames are not provided to the network. We found that spectral normalization in the discriminator and conditioning only on the predicted frames were important for a stable training. We also found that image discriminators that operate on single frames were not necessary. See the supplemental material for additional details.

4.4 Experiments

Our experimental evaluation studies the realism, diversity, and accuracy of the videos generated by our approach and prior methods, and evaluates the importance of various design decisions, such as the presence of the variational and adversarial objectives. Evaluating the performance of stochastic video prediction models is exceedingly challenging: not only should the samples from the model be physically realistic and visually plausible given the context frames, but the model should also be able to produce diverse samples that match the conditional distribution in the data. This is difficult to evaluate precisely: realism is not accurately reflected with simple metrics of reconstruction accuracy, and the true conditional distribution in the data is unknown, since real-world datasets only have a single future for each initial sequence. Below, we discuss the metrics that we use for evaluation. No single metric alone provides a clear answer as to which model is better, but considering multiple metrics can provide us with a more complete understanding of the performance and trade-offs of each approach.

Evaluation Metrics

Realism: comparisons to real videos using human judges. The realism of the predicted videos is evaluated based on a *real vs. fake* two-alternative forced choice (2AFC) test. Human judges on Amazon Mechanical Turk (AMT) are presented with a pair of videos—one generated and one real—and asked to identify the generated “fake” video. We use the implementation from Zhang et al. (2016), modified for videos. Each video is 12 frames long (2 context frames and 10 predicted frames) and shown over 3 seconds. For each method, we gather 1000 judgments from 25 human judges. Each human evaluator is provided with 10 training trials followed by 40 test trials. A method that produces perfectly realistic videos would achieve a 50% fooling rate.

Diversity: distance between samples. Realism is not the only factor in determining the performance of a video prediction model: aside from generating predictions that look physically plausible and realistic, a successful model must also adequately cover the range of possible futures. We compute diversity as the average distance between randomly sampled video predictions, similar to Zhu et al. (2017). Distance is measured using the Learned Perceptual Image Patch Similarity (LPIPS) metric (Zhang et al., 2018).

Accuracy: similarity of the best sample. One weakness of the above metric is that the samples may be diverse but still not cover the feasible output space. Though we do not have the true output distribution, we can still leverage the single ground truth instance. This can be done by sampling the model a finite number of times, and evaluating the similarity between the best sample and the ground truth. This has been explored in prior work on stochastic video prediction (Babaeizadeh et al., 2018; Denton and Fergus, 2018), using peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) as the evaluation metric.

While PSNR and SSIM are commonly used for video prediction, these metrics are not necessarily indicative of prediction quality. In video prediction, structural ambiguities and geometric

deformations are a dominant factor, and SSIM is not an appropriate metric in such situations (Sampat et al., 2009; Zhang et al., 2018). To partially overcome the limitations of these metrics, we also evaluate using the LPIPS metric, which has been shown to correlate well with human perception (Zhang et al., 2018). This distance is measured in the AlexNet feature space (pretrained on ImageNet classification) with linear weights calibrated to match human judgements.

Datasets

We evaluate on two real-world datasets: the BAIR action-free robot pushing dataset (Ebert et al., 2017) and the KTH human actions dataset (Schuldt et al., 2004). The BAIR dataset consists of a randomly moving robotic arm that pushes objects on a table. The KTH dataset consists of a human subject doing one of six activities: walking, jogging, running, boxing, hand waving, and hand clapping. In both cases, we condition on 2 frames and train to predict the next 10 frames. The frame resolution is 64×64 .

Methods: Ablations and Comparisons

We compare the following variants of our method, in our effort to evaluate the effect of each loss term. Videos, code, and models are available at our website¹.

- **Ours, SAVP.** Our stochastic adversarial video prediction model, with the VAE and GAN losses. The GAN loss includes the perceptual loss.
- **Ours, VAE-only.** An ablation of our model with only a conditional VAE, with the reconstruction \mathcal{L}_1 loss but without the adversarial nor perceptual loss.
- **Ours, GAN-only.** An ablation of our model with only a conditional GAN, without the variational autoencoder. This model still takes a noise sample as input, but the noise is sampled from the prior during training. This model uses reconstruction and perceptual losses similar to \mathcal{L}_1 and \mathcal{L}_P , but using predictions sampled from the prior.

We also compare to prior stochastic VAE-based methods Stochastic Variational Video Prediction (SV2P) (Babaeizadeh et al., 2018) and Stochastic Video Generation (SVG) (Denton and Fergus, 2018), both of which use the MSE reconstruction loss and no adversarial loss.

Experimental Results

We show qualitative results on the BAIR and KTH datasets in Figure 4.3 and Figure 4.4, respectively. For the quantitative results, we evaluate the realism, diversity, and accuracy of the predicted videos.

¹https://alexlee-gk.github.io/video_prediction

Method	BAIR Dataset	KTH Dataset
SAVP (no \mathcal{L}_p)	0.0411 ± 0.0019	0.0564 ± 0.0015
SAVP	0.0386 ± 0.0018	0.0500 ± 0.0013

Table 4.1: **LPIPS distance of the best sample, perceptual loss ablation.** The distance (lower is better) is computed as the average over 10 prediction steps. The accuracy of our model is better when the perceptual loss is used.

Realism. In Figure 4.5, variants of our method are compared to prior work. Our SAVP model achieves the highest fooling rate, significantly outperforming prior methods SVG-LP and SV2P. Among the VAE-based methods without adversarial losses, our VAE-only model also achieves the highest realism. The SV2P method produces blurry and unrealistic videos. The SV2P method is unable to predict humans when sampling from the prior on the KTH dataset, and thus we do not include it in the KTH experiments. The pure GAN produces realistic results, but less realistic than our SAVP model.

Diversity. Diversity results are also shown in Figure 4.5. While the GAN-only approach achieves realistic results, it shows significantly lower diversity than the VAE-based methods. This is an example of the commonly known phenomenon of mode-collapse, where multiple latent codes produce the same or similar images on the output (Goodfellow, 2016). Intuitively, the VAE-based methods explicitly encourage the latent code to be more expressive by using an encoder from the output space into the latent space during training. This is verified in our experiments, as the VAE-based variants, including our SAVP model, achieve higher diversity than our GAN-only models on both datasets.

Accuracy. In Figure 4.6, we show similarity results as a function of time. Our SAVP method outperforms all the other methods on the LPIPS metric. On the BAIR dataset, our VAE-only variant and SVG-LP achieve similar accuracies, except for the first two time steps. We found that a learned prior, such as the one used by SVG-LP, contributes to higher accuracy at the beginning of the sequence. For simplicity, our models use the standard univariate Gaussian prior. On the KTH dataset, SVG-LP is significantly worse than any of our methods. We noticed that this is because SVG struggles to model different backgrounds, which is the case for the KTH dataset.

In stochastic domains, such as BAIR, there is correlation between diversity and accuracy of the best sample: a model with diverse predictions is more likely to sample a video that is close to the ground truth. For this reason, the GAN-only variant achieves significantly lower similarities than our other variants. The effect is less pronounced in domains that are less stochastic, such as KTH.

Finally, we evaluate the importance of the perceptual loss on our model in Table 4.1, which shows that the perceptual loss substantially improves the LPIPS accuracy on BAIR.

4.5 Discussion

We develop a video prediction model that combines latent variables trained via a variational lower bound with an adversarial loss to produce a high degree of visual and physical realism. VAE-style training enables our method to make diverse stochastic predictions, and our experiments show that the adversarial and perceptual losses are effective at producing predictions that are more visually realistic according to human raters and more accurate. Evaluation of video prediction models is a major challenge, and we evaluate our method, as well as ablated variants that consist of only the VAE or only the GAN loss, in terms of a variety of quantitative and qualitative measures, including human ratings, diversity, and accuracy of the predicted samples. Our results demonstrate that our approach produces more realistic and accurate predictions than prior methods, while preserving the sample diversity of VAE-based methods.

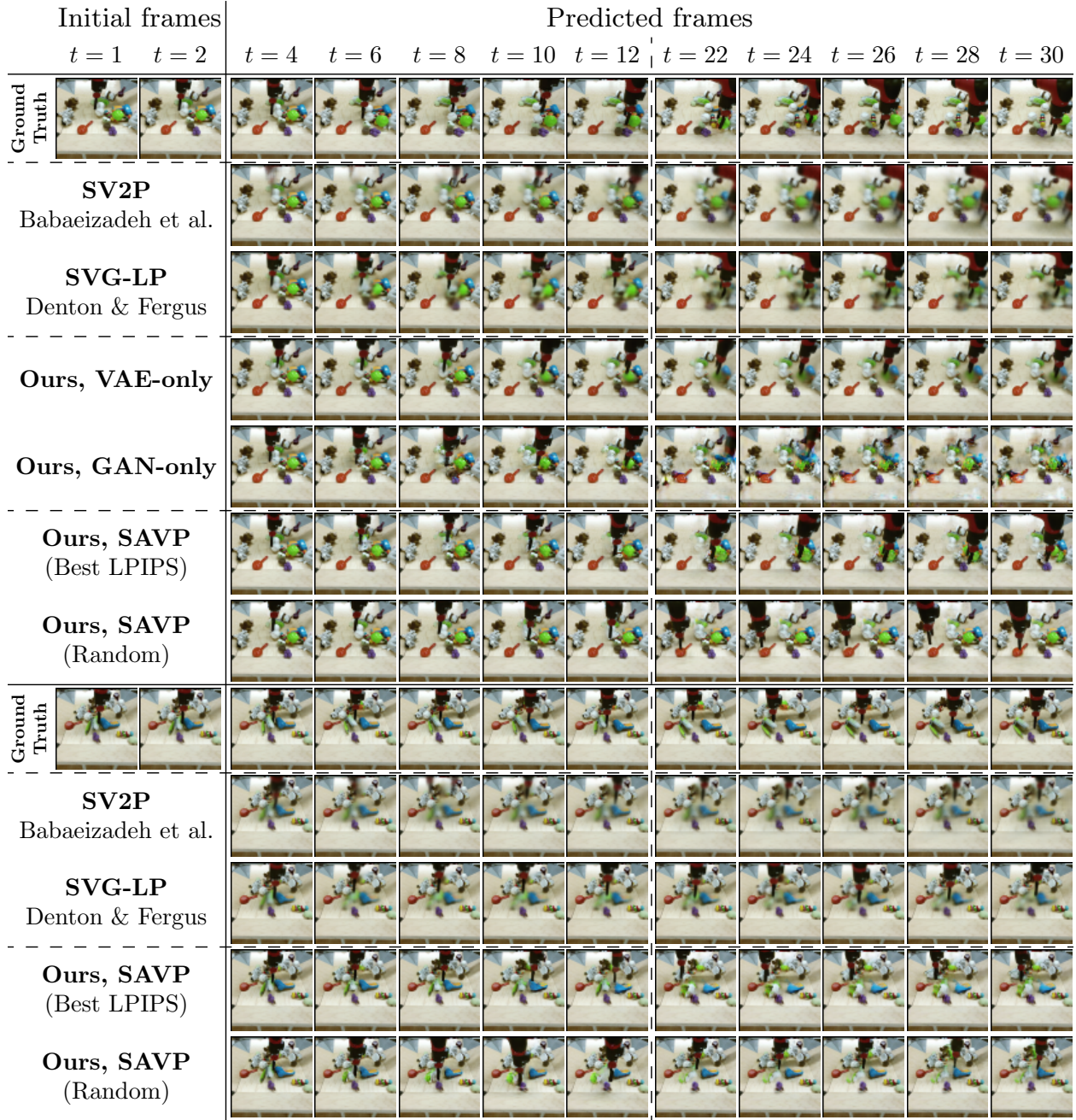


Figure 4.3: **Qualitative results (BAIR action-free dataset).** Unless labeled otherwise, we show the closest generated sample to ground truth (out of a 100) using LPIPS distance. SV2P produces blurry results. The VAE-based methods SVG-LP and our VAE-only ablation predict sharper robot arms, but still blur the objects that are being touched by the robot. The GAN-only method produce sharper predictions, but it is susceptible to visual artifacts due to training difficulties. We show three results for our SAVP model: using the closest and random samples. Note that there is large variation between the samples in the arm motion. Videos of the test set can be found at: https://alexlee-gk.github.io/video_prediction.

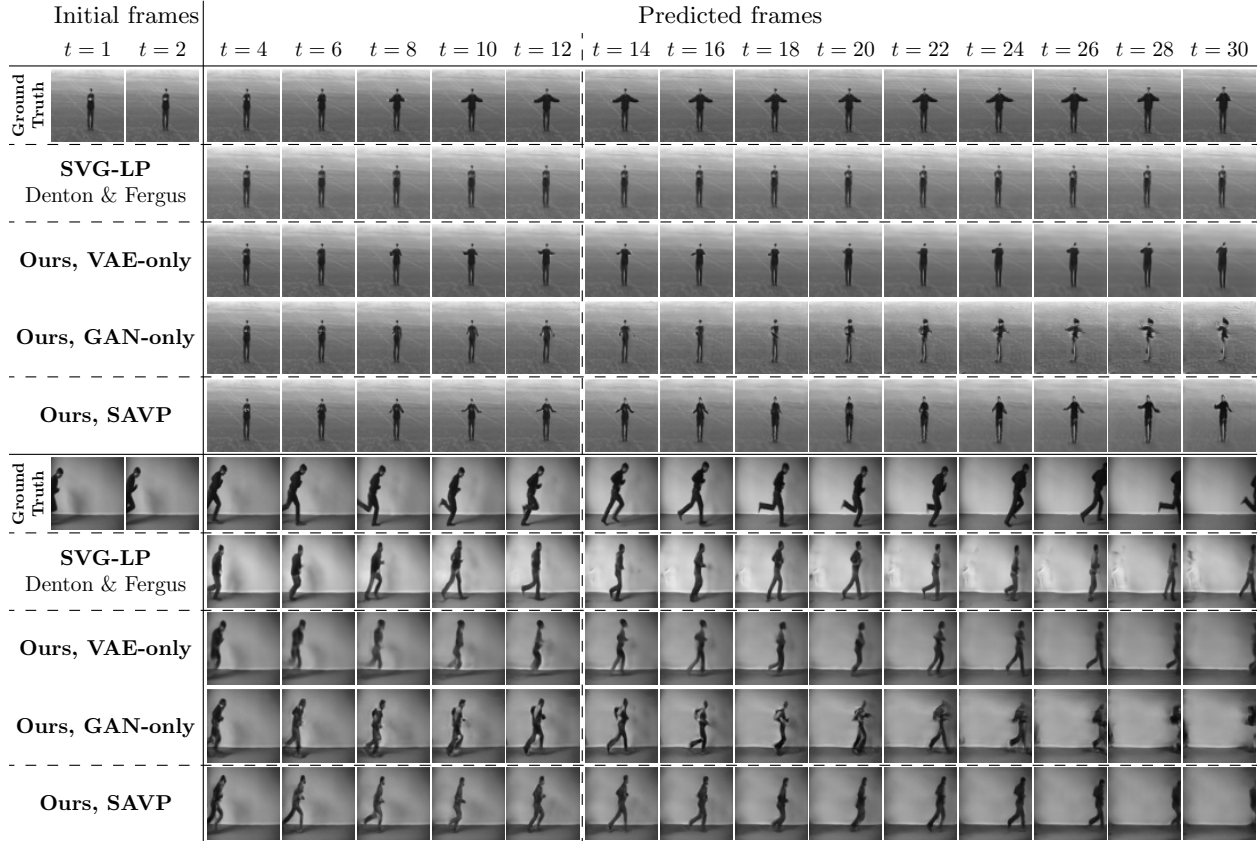


Figure 4.4: **Qualitative results (KTH dataset).** We show the closest generated sample to ground truth (out of a 100) using LPIPS distance. The VAE-based methods SVG-LP and our VAE-only ablation generate images where small limbs disappear further into the future, whereas our SAVP method preserves them. In some cases, the SVG-LP method struggles to predict the background, such as in the sequence of the running person. The GAN-only method produces images with visual artifacts, especially further into the future. Our SAVP model produces sharper and more realistic predictions. Videos of the test set can be found at: https://alexlee-gk.github.io/video_prediction.

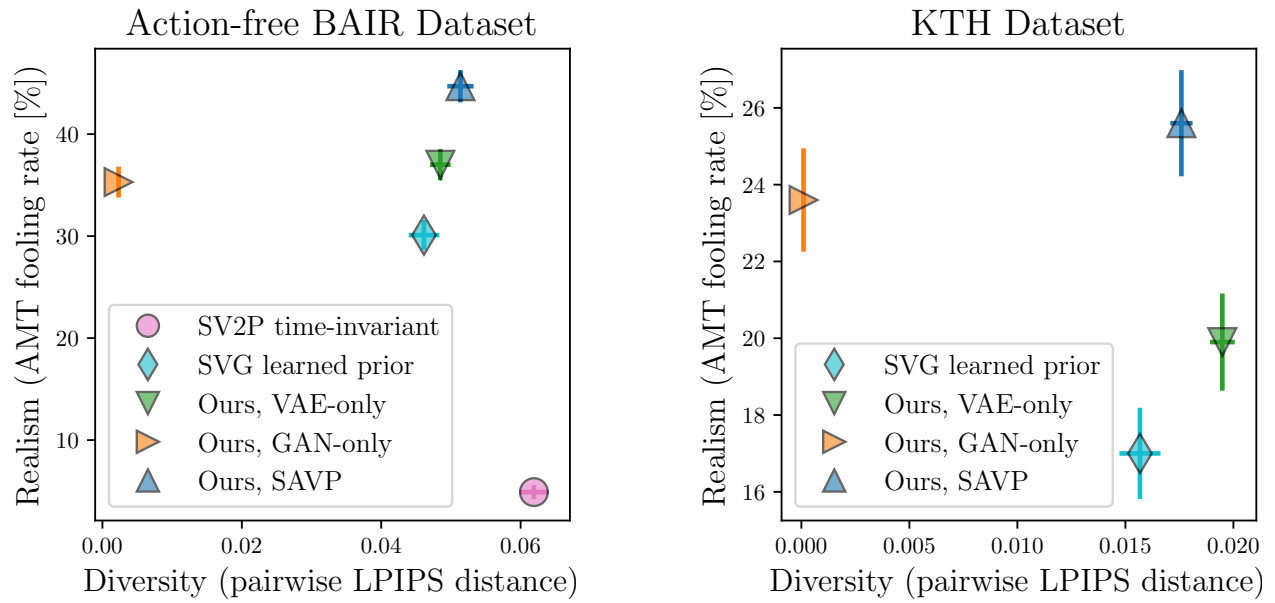


Figure 4.5: Realism vs diversity. We measure realism using a real vs fake Amazon Mechanical Turk (AMT) test, and diversity using average LPIPS distance between samples. Higher is better on both metrics. Our SAVP method achieves the highest realism compared to our ablations and VAE methods from prior work. All the methods are diverse, except for the pure GAN due to mode collapse. Our SAVP method, improves along the realism axis compared to a pure VAE method, and improves along the diversity axis compared to a pure GAN method. These metrics are computed using predicted videos that are 10 time steps long. The error bars show the standard error.

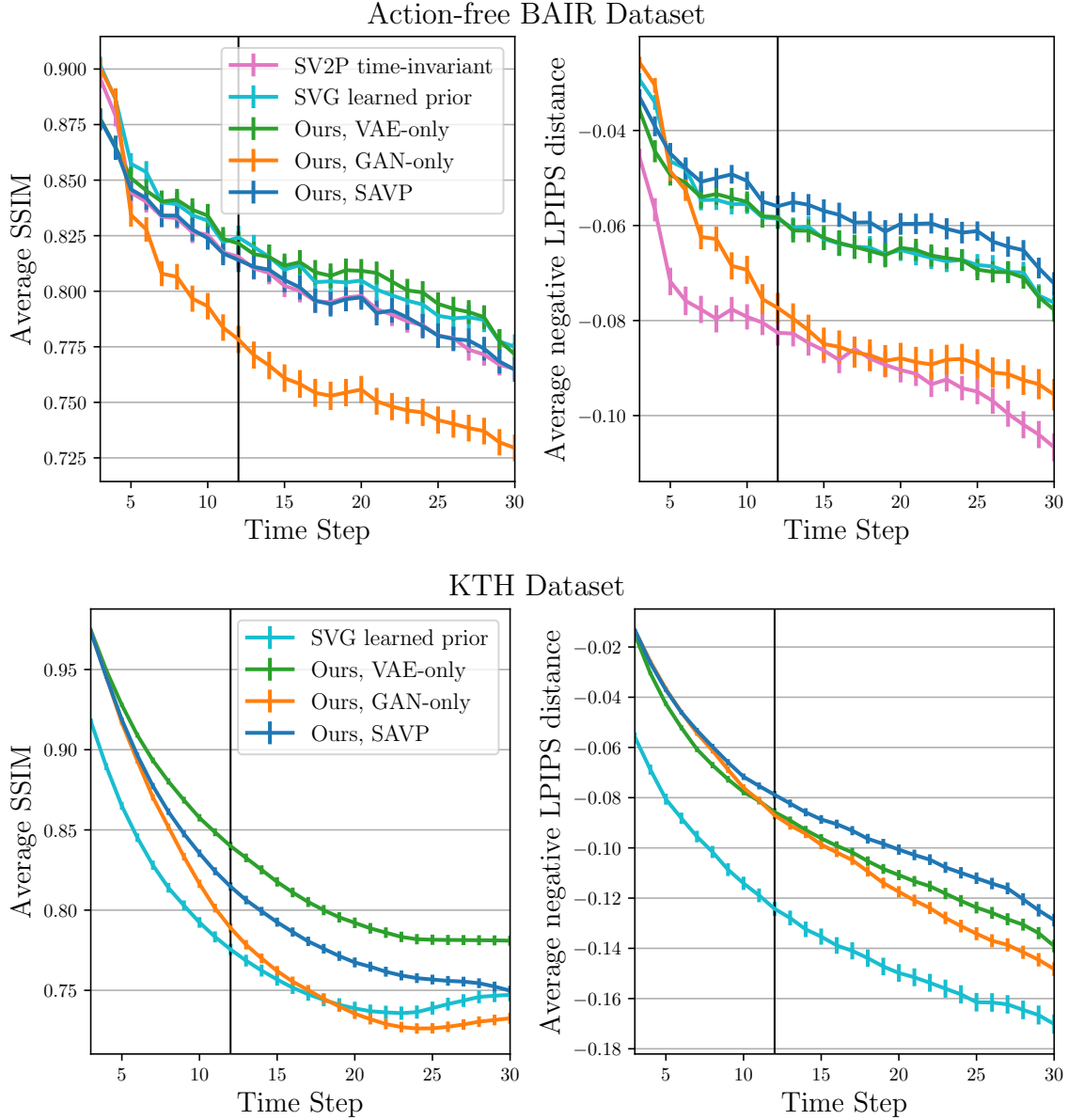


Figure 4.6: **Similarity of the best sample.** We show the similarity (higher is better) between the best sample (of 100) as a function of prediction time step. The VAE methods tend to achieve the highest SSIM while our SAVP method consistently achieves the best LPIPS. Among our ablations, the GAN-only method achieves the worst SSIM and LPIPS. On the KTH dataset, all of our variants perform significantly better than the SVG method from prior works. Note that the models were only trained to predict 10 future frames (indicated by the vertical line), but are being tested on longer sequences. The error bars show the standard error.

Chapter 5

Stochastic Latent Actor-Critic: Deep Reinforcement Learning with a Latent Variable Model

5.1 Introduction

The previous chapters use dynamics models for model-based control, i.e. the predictions from the models are used decision-making. The drawback of this is that the performance of the agent depends on the correctness of these predictions. In this chapter, we show that dynamics models can be used to improve sample efficiency and asymptotic performance in the context of reinforcement learning (RL), but without using the predictions from the models for decision-making.

Deep RL algorithms can automatically learn to solve certain tasks from raw, low-level observations such as images. However, these kinds of observation spaces present a number of challenges in practice: on one hand, it is difficult to directly learn from these high-dimensional inputs, but on the other hand, it is also difficult to tease out a compact representation of the underlying task-relevant information from which to learn instead. For these reasons, deep RL directly from low-level observations such as images remains a challenging problem. Particularly in continuous domains governed by complex dynamics, such as robotic control (Tassa et al., 2018; Brockman et al., 2016), standard approaches still require separate sensor setups to monitor details of interest in the environment, such as the joint positions of a robot or specific pose information of objects of interest. To instead be able to learn directly from the more general and rich modality of vision would greatly advance the current state of our learning systems, so we aim to study precisely this. Standard model-free deep RL aims to use direct end-to-end training to explicitly unify these tasks of representation learning and task learning. However, solving both problems together is difficult, since an effective policy requires an effective representation, but in order for an effective representation to emerge, the policy or value function must provide meaningful gradient information using only the model-free supervision signal (i.e., the reward function). In practice, learning directly from images with standard RL algorithms can be slow, sensitive to hyperparameters, and ineffi-

cient. In contrast to end-to-end learning with RL, predictive learning can benefit from a rich and informative supervision signal before the agent has even made progress on the task or received any rewards. This leads us to ask: can we explicitly *learn* a latent representation from raw low-level observations that makes deep RL easier, through learning a predictive latent variable model?

Predictive models are commonly used in model-based RL for the purpose of planning (Deisenroth and Rasmussen, 2011; Finn and Levine, 2017; Nagabandi et al., 2018; Chua et al., 2018; Zhang et al., 2019) or generating cheap synthetic experience for RL to reduce the required amount of interaction with the real environment (Sutton, 1991; Gu et al., 2016). However, in this work, we are primarily concerned with their potential to alleviate the *representation learning* challenge in RL. We devise a stochastic predictive model by modeling the high-dimensional observations as the consequence of a latent process, with a Gaussian prior and latent dynamics, as illustrated in Figure 5.1. A model with an entirely stochastic latent state has the appealing interpretation of being able to properly represent uncertainty about any of the state variables, given its past observations. We demonstrate in our work that fully stochastic state space models can in fact be learned effectively: With a well-designed stochastic network, such models outperform fully deterministic models, and contrary to the observations in prior work (Hafner et al., 2019; Buesing et al., 2018), are actually comparable to (if not better than) mixed deterministic/stochastic models. Finally, we note that this explicit representation learning, even on low-reward data, allows an agent with such a model to make progress on representation learning even before it makes progress on task learning.

Equipped with this model, we can then perform RL *in the learned latent space* of the predictive model. We posit—and confirm experimentally—that our latent variable model provides a useful representation for RL. Our model represents a partially observed Markov decision process (POMDP), and solving such a POMDP exactly would be computationally intractable (Astrom, 1965; Kaelbling et al., 1998; Igl et al., 2018). We instead propose a simple approximation that trains a Markovian critic on the (stochastic) latent state and trains an actor on a history of observations and actions. The resulting stochastic latent actor-critic (SLAC) algorithm loses some of the benefits of full POMDP solvers, but it is easy and stable to train. It also produces good results, in practice, on a range of challenging problems, making it an appealing alternative to more complex POMDP solution methods.

The main contributions of our SLAC algorithm are useful representations learned from our stochastic sequential latent variable model, as well as effective RL in this learned latent space. We show experimentally that our approach substantially improves on *both* model-free and model-based RL algorithms on a range of image-based continuous control benchmark tasks, attaining better final performance and learning more quickly than algorithms based on (a) end-to-end deep RL from images, (b) learning in a latent space produced by various alternative latent variable models, such as a variational autoencoder (VAE) (Kingma and Welling, 2014), and (c) model-based RL based on latent state-space models with mixed deterministic/stochastic variables (Hafner et al., 2019).

5.2 Related Work

Representation learning in RL. End-to-end deep RL can in principle learn representations directly as part of the RL process (Mnih et al., 2013). However, prior work has observed that RL has a “representation learning bottleneck”: a considerable portion of the learning period must be spent acquiring good representations of the observation space (Shelhamer et al., 2016). This motivates the use of a distinct representation learning procedure to acquire these representations before the agent has even learned to solve the task. The use of unsupervised learning to learn such representations has been explored in a number of prior works (Lange and Riedmiller, 2010; Finn et al., 2016b). In contrast to this class of representation learning algorithms where temporal connections between data are not explicitly considered, we utilize a latent-space dynamics model, which we find in empirical comparisons to result in substantially better representations for RL. By modeling covariances between consecutive latent states, we make it feasible for our proposed stochastic latent actor-critic (SLAC) algorithm to perform Bellman backups directly in the latent space of the learned model. In contrast to prior work that also uses latent-space dynamical system models (Watter et al., 2015; Karl et al., 2017; Zhang et al., 2019; Hafner et al., 2019), our approach benefits from the good asymptotic performance of model-free RL, while at the same time leveraging the improved latent space representation for sample efficiency, despite not using any model-based rollouts for data augmentation.

Partial observability. Our work is also related to prior research on RL under partial observability. Prior work has studied end-to-end RL with recurrent models (Hausknecht and Stone, 2015; Zhu et al., 2018), as well as explicit modeling of the POMDP (Astrom, 1965; Kaelbling et al., 1998) to incorporate belief state into the policy. Prior work has also proposed to train a mixed deterministic/stochastic hidden state model jointly with a policy for the purpose of solving POMDPs (Igl et al., 2018). Our approach is closely related to this prior method, in that we also use model-free RL with a hidden state representation that is learned via prediction. However, our model learns a fully stochastic latent state, and we focus on tasks with high-dimensional image observations for complex underlying continuous control tasks, in contrast to (Igl et al., 2018), which evaluates on Atari tasks and low-dimensional continuous control tasks that emphasize partial observability and knowledge-gathering actions. Our method learns a critic directly on the latent state. This makes our method simpler and more scalable, but at the cost of not being able to reason about the full belief state. In this sense, the strengths of our approach and (Igl et al., 2018) are complementary, and combining their strengths would be an interesting direction for future work.

Model stochasticity. Previous works have typically observed that mixed deterministic-stochastic state space models are more effective (Hafner et al., 2019; Igl et al., 2018; Buesing et al., 2018) than fully stochastic ones. In these models, the state of the underlying MDP is modeled with the deterministic state of a recurrent network (e.g., LSTM (Hochreiter and Schmidhuber, 1997) or GRU (Cho et al., 2014)), and optionally with some stochastic random variables. As mentioned earlier, a model with a latent state that is entirely stochastic has the appealing interpretation of learning a representation that can properly represent uncertainty about any of the state variables, given past observations. We demonstrate in our work that fully stochastic state space models can in

fact be learned effectively and, with a well-designed stochastic network, such models substantially outperform both fully deterministic and mixed deterministic/stochastic models.

5.3 Reinforcement Learning and Modeling

This work addresses the problem of learning maximum entropy policies from high-dimensional observations in POMDPs, by simultaneously learning a latent representation of the underlying MDP state using variational inference and learning the policy in a maximum entropy RL framework. In this section, we describe maximum entropy RL (Ziebart, 2010; Haarnoja et al., 2018a; Levine, 2018) in fully observable MDPs, as well as variational methods for training latent state space models for POMDPs.

Maximum Entropy RL in Fully Observable MDPs

In a Markov decision process (MDP), an agent at time t takes an action $\mathbf{a}_t \in \mathcal{A}$ from state $\mathbf{s}_t \in \mathcal{S}$ and reaches the next state $\mathbf{s}_{t+1} \in \mathcal{S}$ according to some stochastic transition dynamics $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. The initial state \mathbf{s}_1 comes from a distribution $p(\mathbf{s}_1)$, and the agent receives a reward r_t on each of the transitions. Standard RL aims to learn the parameters ϕ of some policy $\pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$ such that the expected sum of rewards is maximized under the induced trajectory distribution ρ_π . This objective can be modified to incorporate an *entropy* term, such that the policy also aims to maximize the expected entropy $\mathcal{H}(\pi_\phi(\cdot|\mathbf{s}_t))$ under the induced trajectory distribution ρ_π . This formulation has a close connection to variational inference (Ziebart, 2010; Haarnoja et al., 2018a; Levine, 2018), and we build on this in our work. The resulting maximum entropy objective is

$$\phi^* = \arg \max_{\phi} \sum_{t=1}^T \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} [r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi_\phi(\cdot|\mathbf{s}_t))], \quad (5.1)$$

where r is the reward function, and α is a temperature parameter that controls the trade-off between optimizing for the reward and for the entropy (i.e., stochasticity) of the policy. Soft actor-critic (SAC) (Haarnoja et al., 2018a) uses this maximum entropy RL framework to derive soft policy iteration, which alternates between policy evaluation and policy improvement within the described maximum entropy framework. SAC then extends this soft policy iteration to handle continuous action spaces by using parameterized function approximators to represent both the Q-function Q_θ (critic) and the policy π_ϕ (actor). The soft Q-function parameters θ are optimized to minimize the soft Bellman residual,

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1}) \sim \mathcal{D}} \left[\frac{1}{2} (Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - (r_t + \gamma V_{\bar{\theta}}(\mathbf{s}_{t+1})))^2 \right], \quad (5.2)$$

$$V_{\bar{\theta}}(\mathbf{s}_{t+1}) = \mathbb{E}_{\mathbf{a}_{t+1} \sim \pi_{\bar{\phi}}} [Q_{\bar{\theta}}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - \alpha \log \pi_{\bar{\phi}}(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})], \quad (5.3)$$

where \mathcal{D} is the replay buffer, γ is the discount factor, and $\bar{\theta}$ are delayed parameters. The policy parameters ϕ are optimized to update the policy towards the exponential of the soft Q-function,

$$J_{\pi}(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[\mathbb{E}_{\mathbf{a}_t \sim \pi_{\phi}} [\alpha \log(\pi_{\phi}(\mathbf{a}_t | \mathbf{s}_t)) - Q_{\bar{\theta}}(\mathbf{s}_t, \mathbf{a}_t)] \right]. \quad (5.4)$$

Results of this stochastic, entropy maximizing RL framework demonstrate improved robustness and stability. SAC also shows the sample efficiency benefits of an off-policy learning algorithm, in conjunction with the high performance benefits of a long-horizon planning algorithm. Precisely for these reasons, we choose to extend the SAC algorithm in this work to formulate our SLAC algorithm.

Sequential Latent Variable Models and Amortized Variational Inference in POMDPs

To learn representations for RL, we use latent variable models trained with amortized variational inference. The learned model must be able to process a large number of pixels that are present in the entangled image \mathbf{x} , and it must tease out the relevant information into a compact and disentangled representation \mathbf{z} . To learn such a model, we can consider maximizing the probability of each observed datapoint \mathbf{x} from some training set \mathcal{D} under the entire generative process $p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z}) d\mathbf{z}$. This objective is intractable to compute in general due to the marginalization of the latent variables \mathbf{z} . In amortized variational inference, we utilize the following bound on the log-likelihood (Kingma and Welling, 2014),

$$E_{\mathbf{x} \sim \mathcal{D}} [\log p(\mathbf{x})] \geq E_{\mathbf{x} \sim \mathcal{D}} [E_{\mathbf{z} \sim q} [\log p(\mathbf{x}|\mathbf{z})] - D_{\text{KL}}(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}))]. \quad (5.5)$$

We can maximize the probability of the observed datapoints (i.e., the left hand side of Equation (5.5)) by learning an encoder $q(\mathbf{z}|\mathbf{x})$ and a decoder $p(\mathbf{x}|\mathbf{z})$, and then directly performing gradient ascent on the right hand side of the equation. In this setup, the distributions of interest are the prior $p(\mathbf{z})$, the observation model $p(\mathbf{x}|\mathbf{z})$, and the posterior $q(\mathbf{z}|\mathbf{x})$.

Although such generative models have been shown to successfully model various types of complex distributions (Kingma and Welling, 2014) by embedding knowledge of the distribution into an informative latent space, they do not have a built-in mechanism for the use of temporal information when performing inference. In the case of partially observable environments, as we discuss below, the representative latent state \mathbf{z}_t corresponding to a given non-Markovian observation \mathbf{x}_t needs to be informed by past observations.

Consider a partially observable MDP (POMDP), where an action $\mathbf{a}_t \in \mathcal{A}$ from latent state $\mathbf{z}_t \in \mathcal{Z}$ results in latent state $\mathbf{z}_{t+1} \in \mathcal{Z}$ and emits a corresponding observation $\mathbf{x}_{t+1} \in \mathcal{X}$. We make an explicit distinction between an observation \mathbf{x}_t and the underlying latent state \mathbf{z}_t , to emphasize that the latter is unobserved and the distribution is not known a priori. Analogous to the fully observable MDP, the initial state distribution is $p(\mathbf{z}_1)$, the transition probability distribution is $p(\mathbf{z}_{t+1}|\mathbf{z}_t, \mathbf{a}_t)$, and the reward is r_t . In addition, the observation model is given by $p(\mathbf{x}_t|\mathbf{z}_t)$.

As in the case for VAEs, a generative model of these observations \mathbf{x}_t can be learned by maximizing the log-likelihood. In the POMDP setting, however, we note that \mathbf{x}_t alone does not provide

all necessary information to infer \mathbf{z}_t , and thus, prior temporal information must be taken into account. This brings us to the discussion of sequential latent variable models. The distributions of interest are the priors $p(\mathbf{z}_1)$ and $p(\mathbf{z}_{t+1}|\mathbf{z}_t, \mathbf{a}_t)$, the observation model $p(\mathbf{x}_t|\mathbf{z}_t)$, and the approximate posteriors $q(\mathbf{z}_1|\mathbf{x}_1)$ and $q(\mathbf{z}_{t+1}|\mathbf{x}_{t+1}, \mathbf{z}_t, \mathbf{a}_t)$. The log-likelihood of the observations can then be bounded, similarly to the VAE bound in Equation (5.5), as

$$\log p(\mathbf{x}_{1:\tau+1}|\mathbf{a}_{1:\tau}) \geq \mathbb{E}_{\mathbf{z}_{1:\tau+1} \sim q} \left[\sum_{t=1}^{\tau+1} \log p(\mathbf{x}_t|\mathbf{z}_t) - D_{\text{KL}}(q(\mathbf{z}_1|\mathbf{x}_1) \parallel p(\mathbf{z}_1)) + \sum_{t=1}^{\tau} -D_{\text{KL}}(q(\mathbf{z}_{t+1}|\mathbf{x}_{t+1}, \mathbf{z}_t, \mathbf{a}_t) \parallel p(\mathbf{z}_{t+1}|\mathbf{z}_t, \mathbf{a}_t)) \right]. \quad (5.6)$$

Prior work (Hafner et al., 2019; Buesing et al., 2018; Doerr et al., 2018) has explored modeling such non-Markovian observation sequences, using methods such as recurrent neural networks with deterministic hidden state, as well as probabilistic state-space models. In this work, we enable the effective training of a fully stochastic sequential latent variable model, and bring it together with a maximum entropy actor-critic RL algorithm to create SLAC: a sample-efficient and high-performing RL algorithm for learning policies for complex continuous control tasks directly from high-dimensional image inputs.

5.4 Joint Modeling and Control as Inference

Our method aims to learn maximum entropy policies from high-dimensional, non-Markovian observations in a POMDP, while also learning a model of that POMDP. The model alleviates the representation learning problem, which in turn helps with the policy learning problem. We formulate the control problem as inference in a probabilistic graphical model with latent variables, as shown in Figure 5.1.

For a fully observable MDP, the control problem can be embedded into a graphical model by introducing a binary random variable \mathcal{O}_t , which indicates if time step t is optimal. When its distribution is chosen to be $p(\mathcal{O}_t = 1|\mathbf{s}_t, \mathbf{a}_t) \propto \exp(r(\mathbf{s}_t, \mathbf{a}_t))$, then maximization of $p(\mathcal{O}_{1:T})$ via approximate inference in that model yields the optimal policy for the maximum entropy objective (Levine, 2018).

In a POMDP, the distribution can analogously be given by $p(\mathcal{O}_t = 1|\mathbf{z}_t, \mathbf{a}_t) \propto \exp(r(\mathbf{z}_t, \mathbf{a}_t))$. Instead of maximizing the likelihood of the optimality variables alone, we jointly model the observations (including the observed rewards of the past time steps) and learn maximum entropy policies by maximizing the likelihood $p(\mathbf{x}_{1:\tau+1}, \mathcal{O}_{\tau+1:T}|\mathbf{a}_{1:\tau})$. This objective represents both the likelihood of the observed data from the past τ steps, as well as the optimality of the agent’s actions for future

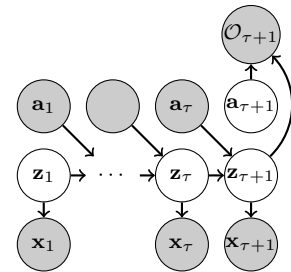


Figure 5.1: Graphical model of POMDP with optimality variables.

steps. We factorize our variational distribution into a product of *recognition* terms $q(\mathbf{z}_1|\mathbf{x}_1)$ and $q(\mathbf{z}_{t+1}|\mathbf{x}_{t+1}, \mathbf{z}_t, \mathbf{a}_t)$, and a *policy* term $\pi(\mathbf{a}_{\tau+1}|\mathbf{x}_{1:\tau+1}, \mathbf{a}_{1:\tau})$ that is independent of \mathbf{z} :

$$q(\mathbf{z}_{1:\tau+1}, \mathbf{a}_{\tau+1}|\mathbf{x}_{1:\tau+1}, \mathbf{a}_{1:\tau}) = q(\mathbf{z}_1|\mathbf{x}_1) \prod_{t=1}^{\tau} q(\mathbf{z}_{t+1}|\mathbf{x}_{t+1}, \mathbf{z}_t, \mathbf{a}_t) \pi(\mathbf{a}_{\tau+1}|\mathbf{x}_{1:\tau+1}, \mathbf{a}_{1:\tau}). \quad (5.7)$$

The posterior over the actions represents the agent’s policy π . We make a design choice in conditioning this policy directly on observations and actions, instead of the latent state, because this approximation allows us to directly execute the policy without having to perform inference on the latent belief state at run time. We use the posterior from Equation (5.7) and maximize the evidence lower bound (ELBO) of the marginal likelihood,

$$\begin{aligned} \log p(\mathbf{x}_{1:\tau+1}, \mathcal{O}_{\tau+1:T}|\mathbf{a}_{1:\tau}) \geq & \mathbb{E}_{(\mathbf{z}_{1:\tau+1}, \mathbf{a}_{1:\tau}) \sim q} \left[\sum_{t=1}^{\tau+1} \log p(\mathbf{x}_t|\mathbf{z}_t) \right. \\ & - \text{D}_{\text{KL}}(q(\mathbf{z}_1|\mathbf{x}_1) \parallel p(\mathbf{z}_1)) - \sum_{t=1}^{\tau} \text{D}_{\text{KL}}(q(\mathbf{z}_{t+1}|\mathbf{x}_{t+1}, \mathbf{z}_t, \mathbf{a}_t) \parallel p(\mathbf{z}_{t+1}|\mathbf{z}_t, \mathbf{a}_t)) \\ & \left. + r(\mathbf{z}_{\tau+1}, \mathbf{a}_{\tau+1}) - \log \pi(\mathbf{a}_{\tau+1}|\mathbf{x}_{1:\tau+1}, \mathbf{a}_{1:\tau}) + V(\mathbf{z}_{\tau+1}) \right]. \quad (5.8) \end{aligned}$$

This derivation assumes that the reward function, which determines $p(\mathcal{O}|\mathbf{x}, \mathbf{a})$, is known. However, in many RL problems, this is not the case. In that situation, we can simply append the reward to the observation, and learn the reward along with $p(\mathbf{x}|\mathbf{z})$. This requires no modification to our method other than changing the observation space, and we use this approach in all of our experiments.

5.5 Stochastic Latent Actor Critic

We now describe our stochastic latent actor critic (SLAC) algorithm, which approximately maximizes the ELBO of Equation (5.8) using function approximators to model the prior and posterior distributions.

Latent Variable Model: The first part of the ELBO corresponds to training the latent variable model to maximize the likelihood of the observations, analogous to the ELBO in Equation (5.6) for the sequential latent variable model. The distributions of the latent variable model are diagonal Gaussian distributions, where the means and variances are outputs of neural networks. The distribution parameters ψ of this model are optimized to maximize the first part of the ELBO,

$$\begin{aligned} J_M(\psi) = & \mathbb{E}_{(\mathbf{x}_{1:\tau+1}, \mathbf{a}_{1:\tau}, r_{1:\tau}) \sim \mathcal{D}} \left[\mathbb{E}_{\mathbf{z}_{1:\tau+1} \sim q_\psi} \left[\sum_{t=1}^{\tau+1} \log p_\psi(\mathbf{x}_t|\mathbf{z}_t) \right. \right. \\ & \left. \left. - \text{D}_{\text{KL}}(q_\psi(\mathbf{z}_1|\mathbf{x}_1) \parallel p_\psi(\mathbf{z}_1)) - \sum_{t=1}^{\tau} \text{D}_{\text{KL}}(q_\psi(\mathbf{z}_{t+1}|\mathbf{x}_{t+1}, \mathbf{z}_t, \mathbf{a}_t) \parallel p_\psi(\mathbf{z}_{t+1}|\mathbf{z}_t, \mathbf{a}_t)) \right] \right]. \quad (5.9) \end{aligned}$$

We sample from the filtering distribution $q_\psi(\mathbf{z}_{1:\tau+1}|\mathbf{x}_{1:\tau+1}, \mathbf{a}_{1:\tau})$ by using the reparameterization trick.

Critic and Actor: The second part of the ELBO corresponds to the maximum entropy RL objective. As in the fully observable case from Section 5.3 and as described by Levine (Levine, 2018), this optimization can be solved via message passing of soft Q-values, except that we use the latent states \mathbf{z} rather than the standard states \mathbf{s} . For continuous state and action spaces, this message passing is approximated by minimizing the soft Bellman residual, which we use to train our soft Q-function parameters θ ,

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{x}_{1:\tau+1}, \mathbf{a}_{1:\tau}, r_\tau) \sim \mathcal{D}} \left[\mathbb{E}_{\mathbf{z}_{1:\tau+1} \sim q_\psi} \left[\frac{1}{2} \left(Q_\theta(\mathbf{z}_\tau, \mathbf{a}_\tau) - \left(r_\tau + \gamma \mathbb{E}_{\mathbf{a}_{\tau+1} \sim \pi_\phi} [Q_{\bar{\theta}}(\mathbf{z}_{\tau+1}, \mathbf{a}_{\tau+1}) - \alpha \log \pi_\phi(\mathbf{a}_{\tau+1}|\mathbf{x}_{1:\tau+1}, \mathbf{a}_{1:\tau})] \right)^2 \right) \right] \right], \quad (5.10)$$

where $\bar{\theta}$ are delayed parameters, obtained as exponential moving averages of θ . Notice that the latents \mathbf{z}_τ and $\mathbf{z}_{\tau+1}$, which are used in the Bellman backup, are sampled from the same joint, i.e. $\mathbf{z}_{\tau+1} \sim q_\psi(\mathbf{z}_{\tau+1}|\mathbf{x}_{\tau+1}, \mathbf{z}_\tau, \mathbf{a}_\tau)$. Next, the policy parameters ϕ are optimized to update the policy towards the exponential of that soft Q-function, analogously to soft actor-critic (Haarnoja et al., 2018a) as shown in Equation (5.4):

$$J_\pi(\phi) = \mathbb{E}_{(\mathbf{x}_{1:\tau+1}, \mathbf{a}_{1:\tau}) \sim \mathcal{D}} \left[\mathbb{E}_{\mathbf{z}_{1:\tau+1} \sim q_\psi} \left[\mathbb{E}_{\mathbf{a}_{\tau+1} \sim \pi_\phi} [\alpha \log \pi_\phi(\mathbf{a}_{\tau+1}|\mathbf{x}_{1:\tau+1}, \mathbf{a}_{1:\tau}) - Q_\theta(\mathbf{z}_{\tau+1}, \mathbf{a}_{\tau+1})] \right] \right]. \quad (5.11)$$

We again use the reparameterization trick to sample from the policy, and the policy loss only uses the last sample $\mathbf{z}_{\tau+1}$ of the sequence. Finally, we note that for the expectation over latent states in the Bellman residual in Equation (5.10), rather than sampling latent states $\mathbf{z} \sim \mathcal{Z}$, we sample latent states from the filtering distribution $q_\psi(\mathbf{z}_{1:\tau+1}|\mathbf{x}_{1:\tau+1}, \mathbf{a}_{1:\tau})$. This design choice allows us to minimize the critic loss for samples that are most relevant for Q , while also allowing the critic loss to use the Q-function in the same way as implied by the policy loss in Equation (5.11).

SLAC is outlined in Algorithm 2. The actor-critic component follows prior work, with automatic tuning of the temperature α and two Q-functions to mitigate underestimation (Fujimoto et al., 2018; Haarnoja et al., 2018a;b). SLAC can be viewed as a variant of SAC (Haarnoja et al., 2018a) where the critic is trained on the stochastic latent state of our sequential latent variable model. The backup for the critic is performed on a tuple $(\mathbf{z}_\tau, \mathbf{a}_\tau, r_\tau, \mathbf{z}_{\tau+1})$, sampled from the posterior $q(\mathbf{z}_{\tau+1}, \mathbf{z}_\tau|\mathbf{x}_{1:\tau+1}, \mathbf{a}_{1:\tau})$. The critic can, in principle, take advantage of the perfect knowledge of the state \mathbf{z}_t , which makes learning easier. However, the actor does not have access to \mathbf{z}_t , and must make decisions based on a history of observations and actions. SLAC is not a model-based algorithm, in that it does not use the model for prediction, but we see in our experiments that SLAC has better sample efficiency than prior model-based algorithms.

Algorithm 2 Stochastic Latent Actor-Critic (SLAC)

Require: $E, \psi, \theta_1, \theta_2, \phi$ \triangleright Environment and initial parameters for model, actor, and critic
 $\mathbf{x}_1 \sim E_{\text{reset}}()$ \triangleright Sample initial observation from the environment
 $\mathcal{D} \leftarrow (\mathbf{x}_1)$ \triangleright Initialize replay buffer with initial observation

for each iteration **do**

for each environment step **do**

$\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t | \mathbf{x}_{1:t}, \mathbf{a}_{1:t-1})$ \triangleright Sample action from the policy

$r_t, \mathbf{x}_{t+1} \sim E_{\text{step}}(\mathbf{a}_t)$ \triangleright Sample transition from the environment

$\mathcal{D} \leftarrow \mathcal{D} \cup (\mathbf{a}_t, r_t, \mathbf{x}_{t+1})$ \triangleright Store the transition in the replay buffer

end for

for each gradient step **do**

$\psi \leftarrow \psi - \lambda_M \nabla_\psi J_M(\psi)$ \triangleright Update model weights

$\theta_i \leftarrow \theta_i - \lambda_Q \nabla_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ \triangleright Update the Q-function weights

$\phi \leftarrow \phi - \lambda_\pi \nabla_\phi J_\pi(\phi)$ \triangleright Update policy weights

$\bar{\theta}_i \leftarrow \nu \theta_i + (1 - \nu) \bar{\theta}_i$ for $i \in \{1, 2\}$ \triangleright Update target critic network weights

end for

end for

5.6 Latent Variable Model

The architecture in our implementation factorizes the latent variable \mathbf{z}_t into two stochastic layers, \mathbf{z}_t^1 and \mathbf{z}_t^2 , as shown in Figure 5.2. We found this design to provide a good balance between ease of training and expressivity, producing good reconstructions and generations and, crucially, providing good representations for reinforcement learning. Unlike models from prior work (Hafner et al., 2019; Buesing et al., 2018; Doerr et al., 2018), which have deterministic and stochastic paths and use recurrent neural networks, ours is fully stochastic, i.e. our latent state is a Markovian latent random variable formed by the concatenation of \mathbf{z}_t^1 and \mathbf{z}_t^2 . Inference over the latent variables is performed using amortized variational inference, with all training done via reparameterization. Hence, the computation graph can be deduced from the diagram by treating all solid arrows as part of the generative model and all dashed arrows as part of approximate posterior.

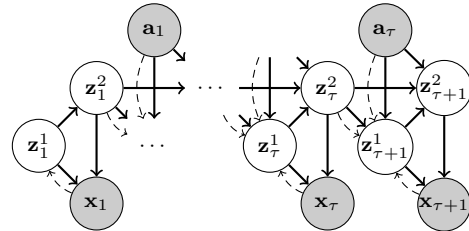


Figure 5.2: Diagram of our full model. Solid arrows show the generative model, dashed arrows show the inference model. Rewards are not shown for clarity.

The generative model consists of the following probability distributions:

$$\begin{aligned}
\mathbf{z}_1^1 &\sim p(\mathbf{z}_1^1) \\
\mathbf{z}_1^2 &\sim p_\psi(\mathbf{z}_1^2|\mathbf{z}_1^1) \\
\mathbf{z}_{t+1}^1 &\sim p_\psi(\mathbf{z}_{t+1}^1|\mathbf{z}_t^2, \mathbf{a}_t) \\
\mathbf{z}_{t+1}^2 &\sim p_\psi(\mathbf{z}_{t+1}^2|\mathbf{z}_{t+1}^1, \mathbf{z}_t^2, \mathbf{a}_t) \\
\mathbf{x}_t &\sim p_\psi(\mathbf{x}_t|\mathbf{z}_t^1, \mathbf{z}_t^2) \\
r_t &\sim p_\psi(r_t|\mathbf{z}_t^1, \mathbf{z}_t^2, \mathbf{a}_t, \mathbf{z}_{t+1}^1, \mathbf{z}_{t+1}^2).
\end{aligned}$$

The initial distribution $p(\mathbf{z}_1^1)$ is a multivariate standard normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. All of the other distributions are conditional and parametrized by neural networks with parameters ψ . The networks for $p_\psi(\mathbf{z}_1^2|\mathbf{z}_1^1)$, $p_\psi(\mathbf{z}_{t+1}^1|\mathbf{z}_t^2, \mathbf{a}_t)$, $p_\psi(\mathbf{z}_{t+1}^2|\mathbf{z}_{t+1}^1, \mathbf{z}_t^2, \mathbf{a}_t)$, and $p_\psi(r_t|\mathbf{z}_t^1, \mathbf{z}_t^2, \mathbf{a}_t, \mathbf{z}_{t+1}^1, \mathbf{z}_{t+1}^2)$ consist of two fully connected layers, each with 256 hidden units, and a Gaussian output layer. The Gaussian layer is defined such that it outputs a multivariate normal distribution with diagonal variance, where the mean is the output of a linear layer and the diagonal standard deviation is the output of a fully connected layer with softplus non-linearity. The observation model $p_\psi(\mathbf{x}_t|\mathbf{z}_t^1, \mathbf{z}_t^2)$ consists of 5 transposed convolutional layers (256 4×4 , 128 3×3 , 64 3×3 , 32 3×3 , and 3 5×5 filters, respectively, stride 2 each, except for the first layer). The output variance for each image pixel is fixed to 0.1.

The variational distribution q , also referred to as the inference model or the posterior, is represented by the following factorization:

$$\begin{aligned}
\mathbf{z}_1^1 &\sim q_\psi(\mathbf{z}_1^1|\mathbf{x}_1) \\
\mathbf{z}_1^2 &\sim p_\psi(\mathbf{z}_1^2|\mathbf{z}_1^1) \\
\mathbf{z}_{t+1}^1 &\sim q_\psi(\mathbf{z}_{t+1}^1|\mathbf{x}_{t+1}, \mathbf{z}_t^2, \mathbf{a}_t) \\
\mathbf{z}_{t+1}^2 &\sim p_\psi(\mathbf{z}_{t+1}^2|\mathbf{z}_{t+1}^1, \mathbf{z}_t^2, \mathbf{a}_t).
\end{aligned}$$

Note that the variational distribution over \mathbf{z}_1^2 and \mathbf{z}_{t+1}^2 is intentionally chosen to exactly match the generative model p , such that this term does not appear in the KL-divergence within the ELBO, and a separate variational distribution is only learned over \mathbf{z}_1^1 and \mathbf{z}_{t+1}^1 . This intentional design decision simplifies the inference process. The networks representing the distributions $q_\psi(\mathbf{z}_1^1|\mathbf{x}_1)$ and $q_\psi(\mathbf{z}_{t+1}^1|\mathbf{x}_{t+1}, \mathbf{z}_t^2, \mathbf{a}_t)$ both consist of 5 convolutional layers (32 5×5 , 64 3×3 , 128 3×3 , 256 3×3 , and 256 4×4 filters, respectively, stride 2 each, except for the last layer), 2 fully connected layers (256 units each), and a Gaussian output layer. The parameters of the convolution layers are shared among both distributions.

The latent variables have 32 and 256 dimensions, respectively, i.e. $\mathbf{z}_t^1 \in \mathbb{R}^{32}$ and $\mathbf{z}_t^2 \in \mathbb{R}^{256}$. For the image observations, $\mathbf{x}_t \in [0, 1]^{64 \times 64 \times 3}$. All the layers, except for the output layers, use leaky ReLU non-linearities. Note that there are no deterministic recurrent connections in the network—all networks are feedforward, and the temporal dependencies all flow through the stochastic units \mathbf{z}_t^1 and \mathbf{z}_t^2 .

For the reinforcement learning process, we use a critic network Q_θ consisting of 2 fully connected layers (256 units each) and a linear output layer. The actor network π_ϕ consists of 5 convolutional layers, 2 fully connected layers (256 units each), a Gaussian layer, and a tanh bijector,

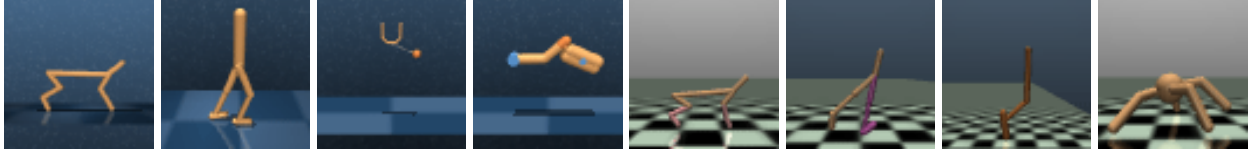


Figure 5.3: Example image observations for our continuous control benchmark tasks: DeepMind Control’s cheetah run, walker walk, ball-in-cup catch, and finger spin, and OpenAI Gym’s half cheetah, walker, hopper, and ant (left to right). Images are rendered at a resolution of 64×64 pixels.

which constrains the actions to be in the bounded action space of $[-1, 1]$. The convolutional layers are the same as the ones from the latent variable model, but the parameters of these layers are not updated by the actor objective. The same exact network architecture is used for every one of the experiments in the paper.

5.7 Experimental Evaluation

We evaluate SLAC on numerous image-based continuous control tasks from both the DeepMind Control Suite (Tassa et al., 2018) and OpenAI Gym (Brockman et al., 2016), as illustrated in Figure 5.3. We also evaluate on multiple simulated manipulation tasks with two different robotic manipulators. Aside from the value of action repeats (i.e. control frequency) for the tasks, we kept all of SLAC’s hyperparameters constant across all tasks in all domains. Training and evaluation details are given in Appendix D.1. Additionally, visualizations of our results and code are available on the project website.¹

Comparative Evaluation on Continuous Control Benchmark Tasks

To provide a comparative evaluation against prior methods, we evaluate SLAC on four tasks (cheetah run, walker walk, ball-in-cup catch, finger spin) from the DeepMind Control Suite (Tassa et al., 2018), and four tasks (cheetah, walker, ant, hopper) from OpenAI Gym (Brockman et al., 2016). Note that the Gym tasks are typically used with low-dimensional state observations, while we evaluate on them with raw image observations. We compare our method to the following state-of-the-art model-based and model-free algorithms:

SAC (Haarnoja et al., 2018a): This is an off-policy actor-critic algorithm, which represents a comparison to state-of-the-art model-free learning. We include experiments showing the performance of SAC based on true state (as an upper bound on performance) as well as directly from raw images.

D4PG (Barth-Maron et al., 2018): This is also an off-policy actor-critic algorithm, learning directly from raw images. The results reported in the plots below are the performance after 10^8 training steps, as stated in the benchmarks from (Tassa et al., 2018).

¹<https://alexlee-gk.github.io/slac/>

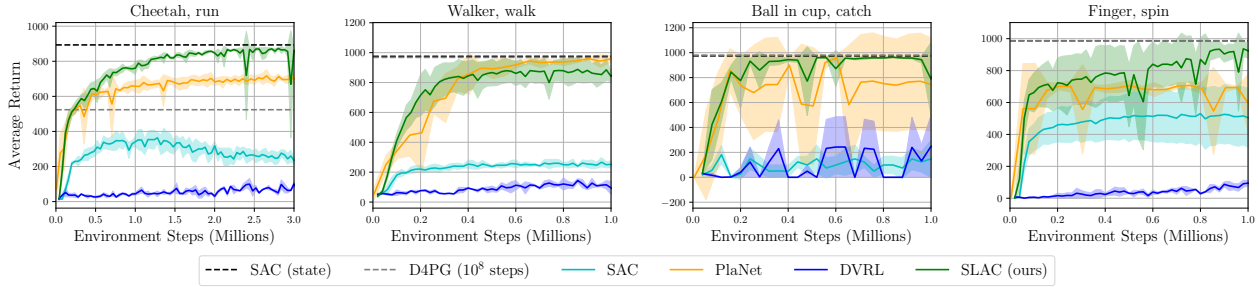


Figure 5.4: Experiments on the DeepMind Control Suite. SLAC (ours) converges to similar or better final performance than the other methods, while almost always achieving reward as high as the upper bound SAC baseline that learns from true state. Note that for these experiments, 1000 environments steps corresponds to 1 episode.

PlaNet (Hafner et al., 2019): This is a model-based RL method for learning from images, which uses a mixed deterministic/stochastic sequential latent variable model, but without explicit policy learning. Instead, the model is used for planning with model predictive control (MPC), where each plan is optimized with the cross entropy method (CEM).

DVRL (Igl et al., 2018): This is an on-policy model-free RL algorithm that also trains a mixed deterministic/stochastic latent-variable POMDP model. DVRL uses the *full belief* over the latent state as input into both the actor and critic, as opposed to our method, which trains the critic with the latent state and the actor with a history of actions and observations.

Our experiments on the DeepMind Control Suite in Figure 5.4 show that the sample efficiency of SLAC is comparable or better than *both* model-based and model-free alternatives. This indicates that overcoming the representation learning bottleneck, coupled with efficient off-policy RL, provides for fast learning similar to model-based methods, while attaining final performance comparable to fully model-free techniques that learn from state. SLAC also substantially outperforms DVRL. This difference can be explained in part by the use of an efficient off-policy RL algorithm, which can better take advantage of the learned representation.

We also evaluate SLAC on continuous control benchmark tasks from OpenAI Gym in Figure 5.5. We notice that these tasks are much more challenging than the DeepMind Control Suite tasks, because the rewards are not as shaped and not bounded between 0 and 1, the dynamics are different, and the episodes terminate on failure (e.g., when the hopper or walker falls over). PlaNet is unable to solve the last three tasks, while for the cheetah task, it learns a suboptimal policy that involves flipping the cheetah over and pushing forward while on its back. To better understand the performance of fixed-horizon MPC on these tasks, we also evaluated with the ground truth dynamics (i.e., the true simulator), and found that even in this case, MPC did not achieve good final performance, suggesting that infinite horizon policy optimization, of the sort performed by SLAC and model-free algorithms, is important to attain good results on these tasks.

For the plots in Figure 5.4 and Figure 5.5, we show the mean and standard deviation over 4 random seeds with 10 evaluation trajectories per seed. We compare all methods using a fixed action repeat per task, except for the reported D4PG numbers from (Tassa et al., 2018), which we could not rerun ourselves due to lack of open-source code. Due to action repeats, the number of samples

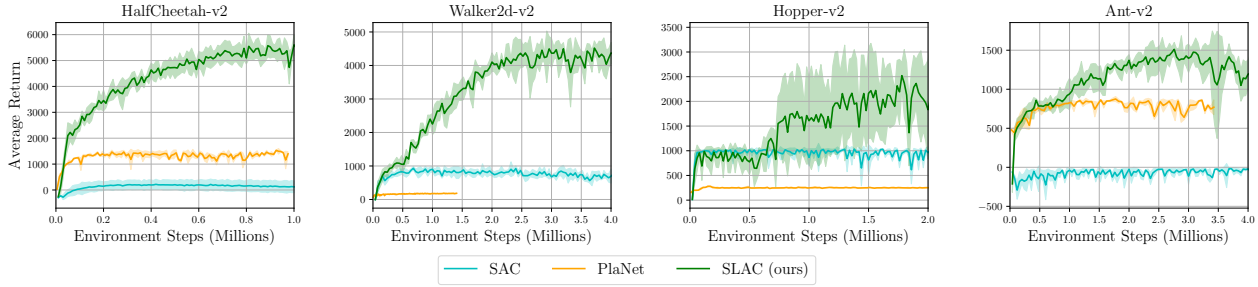


Figure 5.5: Experiments on the OpenAI Gym benchmark tasks from iamges. SLAC (ours) converges to higher performance than both PlaNet and SAC on all four of these tasks. The number of environments steps in each episode is variable, depending on the termination.

used for training is only a fraction of the environment steps reported in our plots. For example, an episode with 1000 environment steps at an action repeat of 4 only receives 250 observations. See Appendix D.1 for details.

Our experiments show that SLAC successfully learns complex continuous control benchmark tasks from raw image inputs, while demonstrating both high sample efficiency as well as high task performance through the use of explicit representation learning in conjunction with a robust maximum-entropy actor-critic RL algorithm.

Simulated Robotic Manipulation Tasks

Beyond standard benchmark tasks, we also aim to illustrate the flexibility of our method by demonstrating it on a variety of image-based robotic manipulation skills. The reward functions for these tasks can be found in Appendix D.2, and videos of these behaviors can be found on the supplementary website². In Figure 5.6, we show illustrations of SLAC’s execution of these manipulation tasks of using a simulated Sawyer robotic arm to push open a door, close a drawer, and reach out and pick up an object. Our method is able to learn these contact-rich manipulation tasks from raw images, succeeding even when the object of interest actually occupies only a small portion of the image.

In our next set of manipulation experiments, we use the 9-DoF 3-fingered DCIaw robot to rotate a valve (Zhu et al., 2019) from various starting positions to various desired goal locations, where the goal is illustrated as a green dot in the image. In all of our experiments, we allow the starting position of the valve to be selected randomly between $[-\pi, \pi]$, and we test three different settings for the goal location (see Figure 5.7). First, we prescribe the goal position to always be fixed. In this task setting, we see that SLAC, SAC from images, and SAC from state all perform similarly in terms of both sample efficiency as well as final performance. However, when we allow the goal location to be selected randomly from a set of 3 options $\{-\frac{\pi}{2}, 0, \frac{\pi}{2}\}$, we see that SLAC and SAC from images actually outperform SAC from state. This interesting result can perhaps be explained by the fact that, when learning from state, the goal is specified with just a single number within

²<https://alexlee-gk.github.io/slac/>

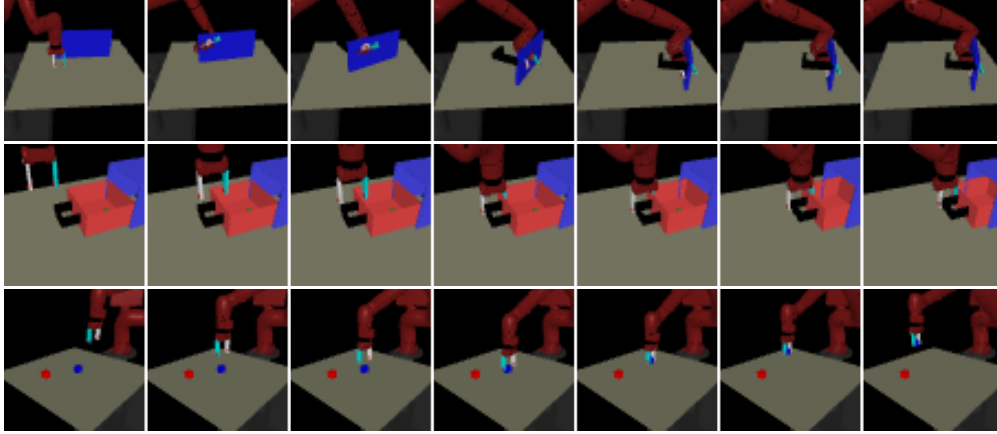


Figure 5.6: Qualitative results of SLAC learning to perform manipulation tasks such as opening a door, closing a drawer, and picking up a block with the Sawyer robot. SLAC is able to learn these contact-rich manipulation tasks from raw images, succeeding even when the object of interest occupies only a small portion of the image.

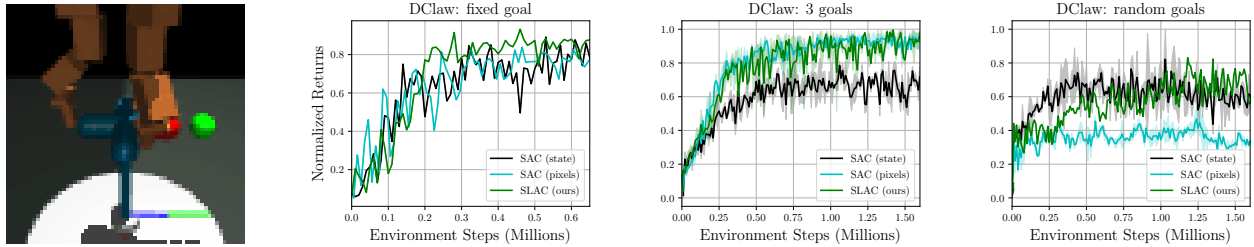


Figure 5.7: Experiments on the DClaw task (a) of turning a valve to a desired location, as shown by a green dot, including comparisons for achieving a (a) fixed goal, (b) three possible goals, and (c) random goals.

the state vector, rather than with the redundancy of numerous green pixels in the image. Finally, when we allow the goal position of the valve to be selected randomly from $[-\frac{\pi}{2}, \frac{\pi}{2}]$, we see that SLAC’s explicit representation learning improves substantially over image-based SAC, performing comparable to the oracle baseline that receives the true state observation.

Evaluating the Latent Variable Model

We next study the tradeoffs between different design choices for the latent variable model. We compare our **fully stochastic** model, as described in Section 5.6, to a standard non-sequential **VAE** model (Kingma and Welling, 2014), which has been used in multiple prior works for representation learning in RL (Higgins et al., 2017; Ha and Schmidhuber, 2018; Nair et al., 2018), the mixed deterministic/stochastic model used by **PlaNet** (Hafner et al., 2019), as well as three variants of our model: a **simple filtering** model that does not factorize the latent variable into two layers of stochastic units, a **fully deterministic** model that removes all stochasticity from the hidden state dynamics, and a **mixed** model that has both deterministic and stochastic transitions, similar to the

PlaNet model, but with our architecture. In all cases, we use the RL framework of SLAC and only vary the choice of model for representation learning. As shown in the comparison in Figure 5.8, our fully stochastic model substantially outperforms prior models as well as the deterministic and simple variants of our own model. Although even the naïve VAE model provides for fast learning in the beginning, the performance of these models quickly saturates. The mixed deterministic/stochastic variant of our model nearly matches the performance of the final fully stochastic variant but, contrary to the conclusions in prior work (Hafner et al., 2019; Buesing et al., 2018), the fully stochastic model performs on par or better, while retaining the appealing interpretation of a stochastic state space belief model.

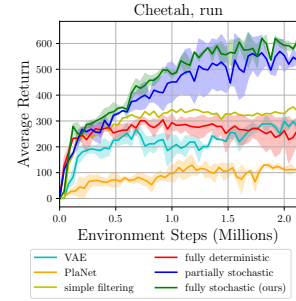


Figure 5.8: Comparison of different design choices for the latent variable model.

Qualitative Predictions from the Latent Variable Model

We show example image samples from our learned sequential latent variable model for the cheetah task in Figure 5.9, and we include the other tasks in Appendix D.3. Samples from the posterior show the images \mathbf{x}_t as constructed by the decoder $p_\psi(\mathbf{x}_t|\mathbf{z}_t)$, using a sequence of latents \mathbf{z}_t that are encoded and sampled from the posteriors, $q_\psi(\mathbf{z}_1|\mathbf{x}_1)$ and $q_\psi(\mathbf{z}_{t+1}|\mathbf{x}_{t+1}, \mathbf{z}_t, \mathbf{a}_t)$. Samples from the prior, on the other hand, use a sequence of latents where \mathbf{z}_1 is sampled from $p(\mathbf{z}_1)$ and all remaining latents \mathbf{z}_t are from the propagation of the previous latent state through the latent dynamics $p_\psi(\mathbf{z}_{t+1}|\mathbf{z}_t, \mathbf{a}_t)$. Note that these prior samples do not use any image frames as inputs, and thus they do not correspond to any ground truth sequence. We also show samples from the conditional prior, which is conditioned on the first image from the true sequence: for this, the sampling procedure is the same as the prior, except that \mathbf{z}_1 is encoded and sampled from the posterior $q_\psi(\mathbf{z}_1|\mathbf{x}_1)$, rather than being sampled from $p(\mathbf{z}_1)$. We notice that the generated images samples can be sharper and more realistic by using a smaller variance for $p_\psi(\mathbf{x}_t|\mathbf{z}_t)$ when training the model, but at the expense of a representation that leads to lower returns. Finally, note that we do not actually use the samples from the prior for training.



Figure 5.9: Example image sequence seen for the cheetah task (first row), corresponding posterior sample (reconstruction) from our model (second row), and generated prediction from the generative model (last two rows). The second to last row is conditioned on the first frame (i.e., the posterior model is used for the first time step while the prior model is used for all subsequent steps), whereas the last row is not conditioned on any ground truth images. Note that all of these sampled sequences are conditioned on the same action sequence, and that our model produces highly realistic samples, even when predicting via the generative model.

5.8 Discussion

We presented SLAC, an efficient RL algorithm for learning from high-dimensional image inputs that combines efficient off-policy model-free RL with representation learning. SLAC learns a compact latent representation space using a stochastic sequential latent variable model, and then learns a critic model within this latent space. By learning a critic within a compact state space, SLAC can learn much more efficiently than standard RL methods. Through representation learning in conjunction with effective task learning in the learned latent space, our method outperforms *both* model-free and model-based alternatives in terms of final performance and sample efficiency, on a range of difficult image-based control tasks.

While our current SLAC algorithm is fully model-free, in that predictions from the model are not utilized to speed up training, a natural extension of our approach would be to use the model predictions themselves to generate synthetic samples. Incorporating this additional synthetic model-based data into a mixed model-based/model-free method could further improve sample efficiency and performance. More broadly, the use of explicit representation learning with RL has the potential to not only accelerate training time and increase the complexity of achievable tasks, but also enable reuse and transfer of our learned representation across tasks.

Chapter 6

Conclusion

This thesis studies how visual dynamics models can be incorporated for robotic planning and control. These studies are conducted in the context of visual servoing, visual planning, and representation learning for reinforcement learning.

For visual servoing, we describe an approach that combines learned visual features with learning predictive dynamics models and reinforcement learning to learn visual servoing mechanisms. The visual dynamics model consists of a one-step predictive model in the space of deep spatial feature maps, which allows the robot to reactively choose an action so as to minimize the distance between the predicted feature of the next time step and the feature of the desired goal. We illustrate this on a complex synthetic car following benchmark. We demonstrate substantial improvement over a conventional approach based on image pixels or hand-designed keypoints, and we show an improvement in sample efficiency of more than two orders of magnitude over standard model-free deep reinforcement learning algorithms.

For visual planning, we propose a deterministic video prediction model and use it for planning. The model predicts future frames by predicting pixel transformations, conditioned on the current and past frames and a sequence of actions. This choice allows to define cost functions in terms of desired pixel motions specified by a human, and then find actions that the model predicts would lead to desired outcomes. We demonstrate this in a range of pushing tasks involving the motion of one or more objects, which is achieved by specifying the desired motion of these objects. We also demonstrate that this framework allows for more complex behaviors to arise, such as lifting the gripper to move over objects that should not be moved, as well as sequentially moving objects to achieve the desired motions of multiple objects.

Then, we present SAVP, a stochastic adversarial video prediction model that overcomes the limitations of deterministic models and the losses used in conventional video prediction models. We develop a video prediction model that combines latent variables trained via a variational lower bound with an adversarial loss to produce a high degree of visual and physical realism. VAE-style training enables our method to make diverse stochastic predictions, and the adversarial and perceptual losses are effective at producing predictions that are more visually realistic and accurate. We demonstrate that our approach produces more realistic and accurate predictions than prior methods, while preserving the sample diversity of VAE-based methods.

Lastly, we present SLAC, an efficient RL algorithm for learning from high-dimensional image inputs that combines efficient off-policy model-free RL with representation learning. Through representation learning in conjunction with effective task learning in the learned latent space, our method achieves improved sample efficiency and final task performance as compared to both prior model-based and model-free RL methods.

In control problems, models are conventionally used in model-based algorithms where predictions coming from a predictive model are directly used to choose actions, either reactively or by planning. This is the case for the first two problems studied in this thesis and also for many model-based reinforcement learning algorithms. Although model-based methods are appealing due to their sample efficiency, their asymptotic performance typically cannot match the returns of model-free methods, often due to inaccuracies of the model. For this reason, an interesting and active area of research is the combination of model-based and model-free techniques to achieve the benefits of sample-efficient learning without sacrificing final performance. Ideally, a model should help the agent learn faster, but it should not hinder the final performance of the agent. Our proposed algorithm SLAC is a step in that direction.

Although our SLAC algorithm does not use predictions from the model to speed up learning, a natural extension would be to use the model to generate synthetic data, which can then be incorporated into a model-free method to further improve sample efficiency and performance. More broadly, the use of dedicated representation learning with control has the potential to not only accelerate training, but also make it feasible to transfer representations across tasks, so as to leverage all experience from all tasks seen so far.

Bibliography

- Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Neural Information Processing Systems (NeurIPS)*, 2016.
- Karl J Astrom. Optimal control of markov processes with incomplete state information. *Journal of mathematical analysis and applications*, 1965.
- Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic variational video prediction. In *International Conference on Learning Representations (ICLR)*, 2018.
- Jianmin Bao, Dong Chen, Fang Wen, Houqiang Li, and Gang Hua. CVAE-GAN: fine-grained image generation through asymmetric training. In *International Conference on Computer Vision (ICCV)*, 2017.
- Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. In *International Conference on Learning Representations (ICLR)*, 2018.
- Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*, 2006.
- Prateep Bhattacharjee and Sukhendu Das. Temporal coherency based criteria for predicting video frames using deep multi-stage generative adversarial networks. In *Neural Information Processing Systems (NeurIPS)*, 2017.
- Richard S. Bogartz, Jeanne L. Shinskey, and Thomas Schilling. Object permanence in five-and-a-half month old infants? *Infancy*, 2000.
- Byron Boots, Arunkumar Byravan, and Dieter Fox. Learning predictive models of a depth camera & manipulator from raw execution traces. In *International Conference on Robotics and Automation (ICRA)*, 2014.
- Samuel Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Conference on Computational Natural Language Learning (CoNLL)*, 2016.

- Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 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 preprint arXiv:1802.03006*, 2018.
- Arunkumar Byravan and Dieter Fox. SE3-nets: Learning rigid body motion using deep neural networks. In *International Conference on Robotics and Automation (ICRA)*, 2017.
- Guillaume Caron, Eric Marchand, and El Mustapha Mouaddib. Photometric visual servoing for omnidirectional cameras. *Autonomous Robots*, 2013.
- Andrea Censi and Richard M. Murray. Bootstrapping bilinear models of simple vehicles. *International Journal of Robotics Research (IJRR)*, 2015.
- Francois Chaumette and Seth Hutchinson. Visual servo control. I. Basic approaches. *Robotics & Automation Magazine*, 2006.
- Baoyang Chen, Wenmin Wang, Jinzhuo Wang, Xiongtao Chen, and Weimian Li. Video imagination from a single image with transformation generation. In *Thematic Workshops of ACM Multimedia*, 2017.
- Jian Chen, Warren E. Dixon, M. Dawson, and Michael McIntyre. Homography-based visual servo tracking control of a wheeled mobile robot. *Transactions on Robotics*, 2006.
- Silvia Chiappa, Sébastien Racanière, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. In *International Conference on Learning Representations (ICLR)*, 2017.
- Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.
- Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Neural Information Processing Systems (NeurIPS)*, 2018.
- Christophe Collewet and Eric Marchand. Photometric visual servoing. *Transactions on Robotics*, 2011.
- Christophe Collewet, Eric Marchand, and Francois Chaumette. Visual servoing set free from image processing. In *International Conference on Robotics and Automation (ICRA)*, 2008.
- Peter I. Corke. Visual control of robot manipulators – A review. *Visual servoing*, 1993.

- Martin Danelljan, Andreas Robinson, Fahad Shahbaz Khan, and Michael Felsberg. Beyond correlation filters: Learning continuous convolution operators for visual tracking. In *European Conference on Computer Vision (ECCV)*, 2016.
- Pieter-Tjerk De Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 2005.
- Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic filter networks. In *Neural Information Processing Systems (NeurIPS)*, 2016.
- Marc Deisenroth and Carl E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, 2011.
- Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition (CVPR)*, 2009.
- Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. In *International Conference on Machine Learning (ICML)*, 2018.
- Guilherme N. DeSouza and Avinash C. Kak. Vision for mobile robot navigation: A survey. *Transactions on Pattern Analysis and Machine Intelligence*, 2002.
- Andreas Doerr, Christian Daniel, Martin Schiegg, Duy Nguyen-Tuong, Stefan Schaal, Marc Toussaint, and Sebastian Trimpe. Probabilistic recurrent state-space models. In *International Conference on Machine Learning (ICML)*, 2018.
- Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning (ICML)*, 2014.
- Frederik Ebert, Chelsea Finn, Alex Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. In *Conference on Robot Learning (CoRL)*, 2017.
- Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 2005.
- Bernard Espiau, Francois Chaumette, and Patrick Rives. A new approach to visual servoing in robotics. *Transactions on Robotics and Automation*, 2002.
- Amir Massoud Farahmand, Mohammad Ghavamzadeh, Csaba Szepesvári, and Shie Mannor. Regularized fitted Q-iteration for planning in continuous-space Markovian decision problems. In *American Control Conference (ACC)*, 2009.
- John T. Feddema and Owen Robert Mitchell. Vision-guided servoing with feature-based trajectory generation (for robots). *Transactions on Robotics and Automation*, 1989.

- Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *International Conference on Robotics and Automation (ICRA)*, 2017.
- Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Neural Information Processing Systems (NeurIPS)*, 2016a.
- Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *International Conference on Robotics and Automation (ICRA)*, 2016b.
- Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, 2018.
- Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. In *International Conference on Intelligent Robots and Systems (IROS)*, 2017.
- Ian Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.
- Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Neural Information Processing Systems (NeurIPS)*, 2014.
- Geoffrey J. Gordon. Stable function approximation in dynamic programming. In *International Conference on Machine Learning (ICML)*, 1995.
- Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning (ICML)*, 2016.
- David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.
- Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, 2018a.
- Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.
- Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning (ICML)*, 2019.
- Koichi Hashimoto. *Visual servoing*, volume 7. World scientific, 1993.

- Matthew Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable MDPs. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents*, 2015.
- Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. DARLA: Improving zero-shot transfer in reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2017.
- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.
- Koh Hosoda and Minoru Asada. Versatile visual servoing without knowledge of true Jacobian. In *International Conference on Intelligent Robots and Systems (IROS)*, 1994.
- Seth Hutchinson, Gregory D. Hager, and Peter I. Corke. A tutorial on visual servo control. *Transactions on Robotics and Automation*, 1996.
- Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for POMDPs. In *International Conference on Machine Learning (ICML)*, 2018.
- Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *Conference on Vision and Pattern Recognition (CVPR)*, 2017.
- Martin Jagersand, Olac Fuentes, and Randal Nelson. Experimental evaluation of uncalibrated visual servoing for precision manipulation. In *International Conference on Robotics and Automation (ICRA)*, 1997.
- Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V. Gool. Dynamic filter networks. In *Neural Information Processing Systems (NeurIPS)*, 2016.
- Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM International Conference on Multimedia*, 2014.
- Stefanie Tellex John Oberlin. Autonomously acquiring instance-based object models from experience. In *International Symposium on Robotics Research*, 2015.
- Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.
- Gregory Kahn, Adam Villafior, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*, 2017.
- Nal Kalchbrenner, Aäron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. In *International Conference on Machine Learning (ICML)*, 2017.

- Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. In *International Conference on Learning Representations (ICLR)*, 2017.
- Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.
- Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.
- Danica Kragic and Henrik I. Christensen. Survey on visual servoing for manipulation. *Computational Vision and Active Perception Laboratory, Fiskartorpsv*, 2002.
- Thomas Lampe and Martin Riedmiller. Acquiring visual servoing reaching and grasping skills using neural reinforcement learning. In *International Joint Conference on Neural Networks (IJCNN)*, 2013.
- Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *International Joint Conference on Neural Networks (IJCNN)*, 2010.
- Sascha Lange, Martin Riedmiller, and Arne Voigtlander. Autonomous reinforcement learning on raw visual input data in a real world application. In *International Joint Conference on Neural Networks (IJCNN)*, 2012.
- Anders Boesen Lindbo Larsen, Soren Kaae Sonderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In *International Conference on Machine Learning (ICML)*, 2016.
- Alex X. Lee, Sergey Levine, and Pieter Abbeel. Learning visual servoing with deep features and fitted q-iteration. In *International Conference on Learning Representations (ICLR)*, 2017.
- Alex X. Lee, Richard Zhang, Frederik Ebert, Pieter Abbeel, Chelsea Finn, and Sergey Levine. Stochastic adversarial video prediction. *arXiv preprint arXiv:1804.01523*, 2018.
- Alex X. Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv preprint arXiv:1907.00953*, 2019.
- Ian Lenz and Ashutosh Saxena. DeepMPC: Learning deep latent features for model predictive control. In *Robotics: Science and Systems (RSS)*, 2015.
- Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.
- Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 2016a.

- Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *International Journal of Robotics Research (IJRR)*, 2016b.
- Guang Li, B. Lennox, and Zhengtao Ding. Infinite horizon model predictive control for tracking problems. In *International Conference on Control and Automation*, 2005.
- Xiaodan Liang, Lisa Lee, Wei Dai, and Eric P. Xing. Dual motion GAN for future-flow embedded video prediction. In *International Conference on Computer Vision (ICCV)*, 2017.
- Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.
- Ziwei Liu, Raymond Yeh, Xiaoou Tang, Yiming Liu, and Aseem Agarwala. Video frame synthesis using deep voxel flow. In *International Conference on Computer Vision (ICCV)*, 2017.
- William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. In *International Conference on Learning Representations (ICLR)*, 2017.
- David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.
- Chaochao Lu, Michael Hirsch, and Schölkopf. Flexible spatio-temporal networks for video prediction. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.
- Ezio Malis, Francois Chaumette, and Sylvie Boudet. 2 1/2 D visual servoing. *Transactions on Robotics and Automation*, 1999.
- Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. In *International Conference on Learning Representations (ICLR)*, 2016.
- Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2018.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing Atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013.
- Kartik Mohta, Vijay Kumar, and Kostas Daniilidis. Vision-based control of a quadrotor for perching on lines. In *International Conference on Robotics and Automation (ICRA)*, 2014.
- Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *International Conference on Robotics and Automation (ICRA)*, 2018.

- Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Neural Information Processing Systems (NeurIPS)*, 2018.
- Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive separable convolution. In *International Conference on Computer Vision (ICCV)*, 2017.
- Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. doi: 10.23915/distill.00003. URL <http://distill.pub/2016/deconv-checkerboard>.
- Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L. Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in Atari games. In *Neural Information Processing Systems (NeurIPS)*, 2015.
- Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. In *Conference on Vision and Pattern Recognition (CVPR)*, 2016.
- Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *International Conference on Robotics and Automation (ICRA)*, 2016.
- MarcAurelio Ranzato, Arthur Szlam, Joan Bruna, Michael Mathieu, Ronan Collobert, and Sumit Chopra. Video (language) modeling: a baseline for generative models of natural videos. *arXiv preprint arXiv:1412.6604*, 2014.
- Scott E. Reed, Aäron van den Oord, Nal Kalchbrenner, Sergio Gomez Colmenarejo, Ziyu Wang, Yutian Chen, Dan Belov, and Nando de Freitas. Parallel multiscale autoregressive density estimation. In *International Conference on Machine Learning (ICML)*, 2017.
- Martin Riedmiller. Neural fitted Q iteration – First experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, 2005.
- Reuven Y. Rubinstein and Dirk P. Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. 2013.
- Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *International Conference on Computer Vision (ICCV)*, 2011.
- Mehdi Sadeghzadeh, David Calvert, and Hussein A. Abdullah. Self-learning visual servoing of robot manipulator using explanation-based fuzzy neural networks and Q-learning. *Journal of Intelligent and Robotic Systems*, 2015.
- Masaki Saito, Eiichi Matsumoto, and Shunta Saito. Temporal generative adversarial nets with singular value clipping. In *International Conference on Computer Vision (ICCV)*, 2017.

- Mehul P. Sapat, Zhou Wang, Shalini Gupta, Alan Conrad Bovik, and Mia K. Markey. Complex wavelet structural similarity: A new image similarity index. *Transactions on Image Processing (TIP)*, 2009.
- Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: a local SVM approach. In *International Conference on Pattern Recognition (ICPR)*, 2004.
- John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015.
- Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2016.
- Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.
- Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.
- Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind control suite. *arXiv preprint arXiv:1801.00690*, 2018.
- Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. MoCoGAN: Decomposing motion and content for video generation. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.
- Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.
- Joost van Amersfoort, Anitha Kannan, Marc’Aurelio Ranzato, Arthur Szlam, Du Tran, and Soumith Chintala. Transformation-based models of video sequences. *arXiv preprint arXiv:1701.08435*, 2017.
- Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International Conference on Machine Learning (ICML)*, 2016.
- Ruben Villegas, Jimei Yang, Seunghoon Hong, Xunyu Lin, and Honglak Lee. Decomposing motion and content for natural video sequence prediction. In *International Conference on Learning Representations (ICLR)*, 2017.
- Carl Vondrick and Antonio Torralba. Generating the future with adversarial transformers. In *Conference on Vision and Pattern Recognition (CVPR)*, 2017.
- Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. In *Neural Information Processing Systems (NeurIPS)*, 2016.

- Jacob Walker, Abhinav Gupta, and Martial Hebert. Dense optical flow prediction from a static image. In *International Conference on Computer Vision (ICCV)*, 2015.
- Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In *European Conference on Computer Vision (ECCV)*, 2016.
- Jacob Walker, Kenneth Marino, Abhinav Gupta, and Martial Hebert. The pose knows: Video forecasting by generating pose futures. In *International Conference on Computer Vision (ICCV)*, 2017.
- Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. In *Neural Information Processing Systems (NeurIPS)*, 2018.
- Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Neural Information Processing Systems (NeurIPS)*, 2015.
- Lee E. Weiss, Arthur C. Sanderson, and Charles P. Neuman. Dynamic sensor-based control of robots with visual feedback. *Journal on Robotics and Automation*, 1987.
- William J. Wilson, Carol C. Williams Hulls, and Graham S. Bell. Relative end-effector control using cartesian position based visual servoing. *Transactions on Robotics and Automation*, 1996.
- Shi Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Neural Information Processing Systems (NeurIPS)*, 2015.
- Tianfan Xue, Jiajun Wu, Katherine Bouman, and Bill Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *Neural Information Processing Systems (NeurIPS)*, 2016.
- Billibon H Yoshimi and Peter K. Allen. Active, uncalibrated visual servoing. In *International Conference on Robotics and Automation (ICRA)*, 1994.
- Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*, 2016.
- Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew J Johnson, and Sergey Levine. SOLAR: Deep structured latent representations for model-based reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2019.
- Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. In *European Conference on Computer Vision (ECCV)*, 2016.

- Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep networks as a perceptual metric. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.
- Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.
- Yipin Zhou and Tamara L. Berg. Learning temporal transformations from time-lapse videos. In *European Conference on Computer Vision (ECCV)*, 2016.
- Henry Zhu, Abhishek Gupta, Aravind Rajeswaran, Sergey Levine, and Vikash Kumar. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. In *International Conference on Robotics and Automation (ICRA)*, 2019.
- Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A. Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation. In *Neural Information Processing Systems (NeurIPS)*, 2017.
- Pengfei Zhu, Xin Li, Pascal Poupart, and Guanghui Miao. On improving deep reinforcement learning for POMDPs. *arXiv preprint arXiv:1804.06309*, 2018.
- Brian D Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. 2010.

Appendix A

Learning Visual Servoing with Deep Features and Fitted Q-Iteration

A.1 Linearization of the Bilinear Dynamics

The optimization of Equation (2.3) can be solved efficiently by using a linearization of the dynamics,

$$f_c^{(l)}(\mathbf{y}_{t,c}^{(l)}, \mathbf{u}) = f_c^{(l)}(\mathbf{y}_{t,c}^{(l)}, \bar{\mathbf{u}}) + \mathbf{J}_{t,c}^{(l)}(\mathbf{u} - \bar{\mathbf{u}}) = f_c^{(l)}(\mathbf{y}_{t,c}^{(l)}, \mathbf{0}) + \mathbf{J}_{t,c}^{(l)}\mathbf{u}, \quad (\text{A.1})$$

where $\mathbf{J}_{t,c}^{(l)}$ is the Jacobian matrix with partial derivatives $\frac{\partial f_c^{(l)}}{\partial \mathbf{u}}(\mathbf{y}_{t,c}^{(l)}, \bar{\mathbf{u}})$ and $\bar{\mathbf{u}}$ is the linearization point. Since the bilinear dynamics are linear with respect to the controls, this linearization is exact and the Jacobian matrix does not depend on $\bar{\mathbf{u}}$. Without loss of generality, we set $\bar{\mathbf{u}} = \mathbf{0}$.

Furthermore, the bilinear dynamics allows the Jacobian matrix to be computed efficiently by simply doing a forward pass through the model. For the locally bilinear dynamics of Equation (2.2), the j -th column of the Jacobian matrix is given by

$$\mathbf{J}_{t,c,j}^{(l)} = \frac{\partial f_c^{(l)}}{\partial \mathbf{u}_j}(\mathbf{y}_{t,c}^{(l)}, \mathbf{0}) = \mathbf{W}_{c,j}^{(l)} * \mathbf{y}_{t,c}^{(l)} + \mathbf{B}_{c,j}^{(l)}. \quad (\text{A.2})$$

A.2 Servoing Cost Function for Reinforcement Learning

The goal of reinforcement learning is to find a policy that maximizes the expected sum of rewards, or equivalently, a policy that minimizes the expected sum of costs. The cost should be one that quantifies progress towards the goal. We define the cost function in terms of the position of the target object (in the camera's local frame) after the action has been taken,

$$c(\mathbf{s}_t, \mathbf{u}_t, \mathbf{s}_{t+1}) = \begin{cases} \sqrt{\left(\frac{\mathbf{p}_{t+1}^x}{\mathbf{p}_{t+1}^z}\right)^2 + \left(\frac{\mathbf{p}_{t+1}^y}{\mathbf{p}_{t+1}^z}\right)^2 + \left(\frac{1}{\mathbf{p}_{t+1}^z} - \frac{1}{\mathbf{p}_*^z}\right)^2}, & \text{if } \|\mathbf{p}_{t+1}\|_2 \geq \tau \text{ and car in FOV} \\ (T - t + 1) c(\cdot, \cdot, \mathbf{s}_t), & \text{otherwise,} \end{cases} \quad (\text{A.3})$$

where T is the maximum trajectory length. The episode terminates early if the camera is too close to the car (less than a distance τ) or the car’s origin is outside the camera’s field of view (FOV). The car’s position at time t is $\mathbf{p}_t = (\mathbf{p}_t^x, \mathbf{p}_t^y, \mathbf{p}_t^z)$ and the car’s target position is $\mathbf{p}_* = (0, 0, \mathbf{p}_*^z)$, both in the camera’s local frame (z-direction is forward). Our experiments use $T = 100$ and $\tau = 4$ m.

A.3 Experiment Details

Task Setup

The camera is attached to the vehicle slightly in front of the robot’s origin and facing down at an angle of $\pi/6$ rad, similar to a commercial quadcopter drone. The robot has 4 degrees of freedom, corresponding to translation and yaw angle. Pitch and roll are held fixed.

In our simulations, the quadcopter follows a car that drives at 1 m s^{-1} along city roads during training and testing. The quadcopter’s speed is limited to within 10 m s^{-1} for each translational degree of freedom, and its angular speed is limited to within $\pi/2 \text{ rad s}^{-1}$. The simulator runs at 10 Hz. For each trajectory, a car is chosen randomly from a set of cars, and placed randomly on one of the roads. The quadcopter is initialized right behind the car, in the desired relative position for following. The image observed at the beginning of the trajectory is used as the goal observation.

Learning Feature Dynamics

The dynamics of all the features were trained using a dataset of 10000 triplets $\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1}$. The observations are 128×128 RGB images and the actions are 4-dimensional vectors of real numbers encoding the linear and angular (yaw) velocities. The actions are normalized to between -1 and 1 .

The training set was generated from 100 trajectories of a quadcopter following a car around the city with some randomness. Each trajectory was 100 steps long. Only 5 training cars were shown during learning. The generation process of each trajectory is as follows: First, a car is chosen at random from the set of available cars and it is randomly placed on one of the roads. Then, the quadcopter is placed at some random position relative to the car’s horizontal pose, which is the car’s pose that has been rotated so that the vertical axis of it and the world matches. This quadcopter position is uniformly sampled in cylindrical coordinates relative to the car’s horizontal pose, with heights in the interval 12 m to 18 m, and azimuthal angles in the interval $-\pi/2$ rad to $\pi/2$ rad (where the origin of the azimuthal angle is the back of the car). The radii and yaw angles are initialized so that the car is in the middle of the image. At every time step, the robot takes an action that moves it towards a target pose, with some additive Gaussian noise ($\sigma = 0.2$). The target pose is sampled according to the same procedure as the initial pose, and it is sampled once at the beginning of each trajectory.

We try the fully and locally connected dynamics for pixel intensities to better understand the performance trade-offs when assuming locally connected dynamics. We do not use the latter for the semantic features since they are too high-dimensional for the dynamics model to fit in memory.

The dynamics models were trained with Adam using 10000 iterations, a batch size of 32, a learning rate of 0.001, and momentums of 0.9 and 0.999, and a weight decay of 0.0005.

Learning Weighting of Feature Dynamics with Reinforcement Learning

We use CEM, TRPO and FQI to learn the feature weighting and report the performance of the learned policies in Table A.1. We use the cost function described in Appendix A.2, a discount factor of $\gamma = 0.9$, and trajectories of up to 100 steps. All the algorithms used initial weights of $\mathbf{w} = 1$ and $\lambda = 1$, and a Gaussian exploration policy with the current policy as the mean and a fixed standard deviation $\sigma_{\text{exploration}} = 0.2$.

For the case of unweighted features, we use CEM to optimize for a single weight \mathbf{w} and for the weights λ . For the case of weighted features, we use CEM to optimize for the full space of parameters, but we only do that for the pixel feature dynamics since CEM does not scale for high-dimensional problems, which is the case for all the VGG features. Each iteration of CEM performs a certain number of noisy evaluations and selects the top 20% for the elite set. The number of noisy evaluations per iteration was 3 times the number of parameters being optimized. Each noisy evaluation used the average sum of costs of 10 trajectory rollouts as its evaluation metric. The parameters of the last iteration were used for the final policy. The policies with unweighted features dynamics and the policies with pixel features dynamics were trained for 10 and 25 iterations, respectively.

We use TRPO to optimize for the full space of parameters for each of the feature dynamics we consider in this work. We use a Gaussian policy, where the mean is the servoing policy of Equation (2.3) and the standard deviation is fixed to $\sigma_{\text{exploration}} = 0.2$ (i.e. we do not learn the standard deviation). Since the parameters are constrained to be non-negative, we parametrize the TRPO policies with $\sqrt{\mathbf{w}}$ and $\sqrt{\lambda}$. We use a Gaussian baseline, where the mean is a 5-layer CNN, consisting of 2 convolutional and 3 fully connected layers, and a standard deviation that is initialized to 1. The convolutional layers use 16 filters (4×4 , stride 2) each, the first 2 fully-connected layers use 32 hidden units each, and all the layers except for the last one use ReLU activations. The input of the baseline network are the features (either pixel intensities or VGG features) corresponding to the feature dynamics being used. The parameters of the last iteration were used for the final policy. The policies are trained with TRPO for 50 iterations, a batch size of 4000 samples per iteration, and a step size of 0.01.

We use our proposed FQI algorithm to optimize for the weights \mathbf{w} , λ , and surpass the other methods in terms of performance on test executions, sample efficiency, and overall computation efficiency¹. The updates of the inner iteration of our algorithm are computationally efficient; since the data is fixed for a given sampling iteration, we can precompute $\phi(\mathbf{s}_t, \mathbf{u}_t)$ and certain terms of $\phi(\mathbf{s}_{t+1}, \cdot)$. The parameters that achieved the best performance on 10 validation trajectories were used for the final policy. The policies are trained with FQI for $S = 2$ sampling iterations, a batch size of 10 trajectories per sampling iteration, $K = 10$ inner iterations per sampling iteration, and a

¹Our policy based on conv4_3 features takes around 650 s to run $K = 10$ iterations of FQI for a given batch size of 10 training trajectories.

Feature Dynamics	Policy Optimization Algorithm				
	unweighted feature dynamics + CEM (1500)	feature dynamics + CEM (3250)	feature dynamics + TRPO (≥ 80)	feature dynamics + TRPO (≥ 2000)	ours, feature dynamics + FQI (20)
pixel, FC	8.20 ± 0.66	7.77 ± 0.66	9.56 ± 0.62	8.03 ± 0.66	7.92 ± 0.67
pixel, LC	8.07 ± 0.74	7.13 ± 0.74	10.11 ± 0.60	7.97 ± 0.72	7.98 ± 0.77
VGG conv1_2	2.22 ± 0.38		2.06 ± 0.35	1.66 ± 0.31	1.89 ± 0.32
VGG conv2_2	2.40 ± 0.47		2.42 ± 0.47	1.89 ± 0.40	1.40 ± 0.29
VGG conv3_3	2.91 ± 0.52		2.87 ± 0.53	1.59 ± 0.42	1.56 ± 0.40
VGG conv4_3	2.70 ± 0.52		2.57 ± 0.49	1.69 ± 0.41	1.11 ± 0.29
VGG conv5_3	3.68 ± 0.47		3.69 ± 0.48	3.16 ± 0.48	2.49 ± 0.35

(a) Costs when using the set of cars seen during learning.

Feature Dynamics	Policy Optimization Algorithm				
	unweighted feature dynamics + CEM (1500)	feature dynamics + CEM (3250)	feature dynamics + TRPO (≥ 80)	feature dynamics + TRPO (≥ 2000)	ours, feature dynamics + FQI (20)
pixel, FC	8.84 ± 0.68	8.66 ± 0.70	10.01 ± 0.62	8.75 ± 0.67	9.00 ± 0.70
pixel, LC	8.37 ± 0.75	7.17 ± 0.75	11.29 ± 0.57	8.25 ± 0.71	8.36 ± 0.79
VGG conv1_2	2.03 ± 0.43		1.79 ± 0.36	1.42 ± 0.33	1.78 ± 0.37
VGG conv2_2	2.01 ± 0.44		2.00 ± 0.45	1.26 ± 0.30	1.28 ± 0.30
VGG conv3_3	2.03 ± 0.47		2.08 ± 0.47	1.46 ± 0.37	1.04 ± 0.31
VGG conv4_3	2.40 ± 0.50		2.57 ± 0.53	1.48 ± 0.36	0.90 ± 0.26
VGG conv5_3	3.31 ± 0.45		3.55 ± 0.50	2.76 ± 0.42	2.56 ± 0.41

(b) Costs when using novel cars, none of which were seen during learning.

Table A.1: Costs on test executions of the dynamics-based servoing policies for different feature dynamics and weighting of the features. The reported numbers are the mean and standard error across 100 test trajectories, of up to 100 time steps each. We test on executions with the training cars and the novel cars; for consistency, the novel cars follow the same route as the training cars. We compare the performance of policies with unweighted features or weights learned by other methods. For the case of unweighted feature dynamics, we use the cross entropy method (CEM) to learn the relative weights λ of the control and the single feature weight w . For the other cases, we learn the weights with CEM, Trust Region Policy Optimization (TRPO) for either 2 or 50 iterations, and our proposed FQI algorithm. CEM searches over the full space of policy parameters w and λ , but it was only ran for pixel features since it does not scale for high-dimensional problems. We report the number of training trajectories in parenthesis. For TRPO, we use a fixed number of training *samples* per iteration, whereas for CEM and FQI, we use a fixed number of training *trajectories* per iteration. We use a batch size of 4000 samples for TRPO, which means that at least 40 trajectories were used per iteration, since trajectories can terminate early, i.e. in less than 100 time steps.


















































































































Dynamics	Observations from Test Executions										Cost
pixel, fully connected											24.74
											16.69
pixel, locally connected											24.92
											16.47
VGG conv1_2											15.91
											1.57
VGG conv2_2											7.53
											2.56
VGG conv3_3											6.01
											3.76
VGG conv4_3											5.94
											4.31
VGG conv5_3											15.51
											17.39

Table A.2: Sample observations from test executions in our experiments, and the costs for each trajectory, for different feature dynamics. We use the weights learned by our FQI algorithm. This table follows the same format as Table 2.1. Some of the trajectories were shorter than 100 steps because of the termination condition (e.g. the car is no longer in the image). The first observation of each trajectory is used as the target observation. The policies based on deeper VGG features, up to VGG conv4_3, are generally more robust to the appearance changes between the observations and the target observation, which are typically caused by movements of the car, distractor objects, and occlusions.

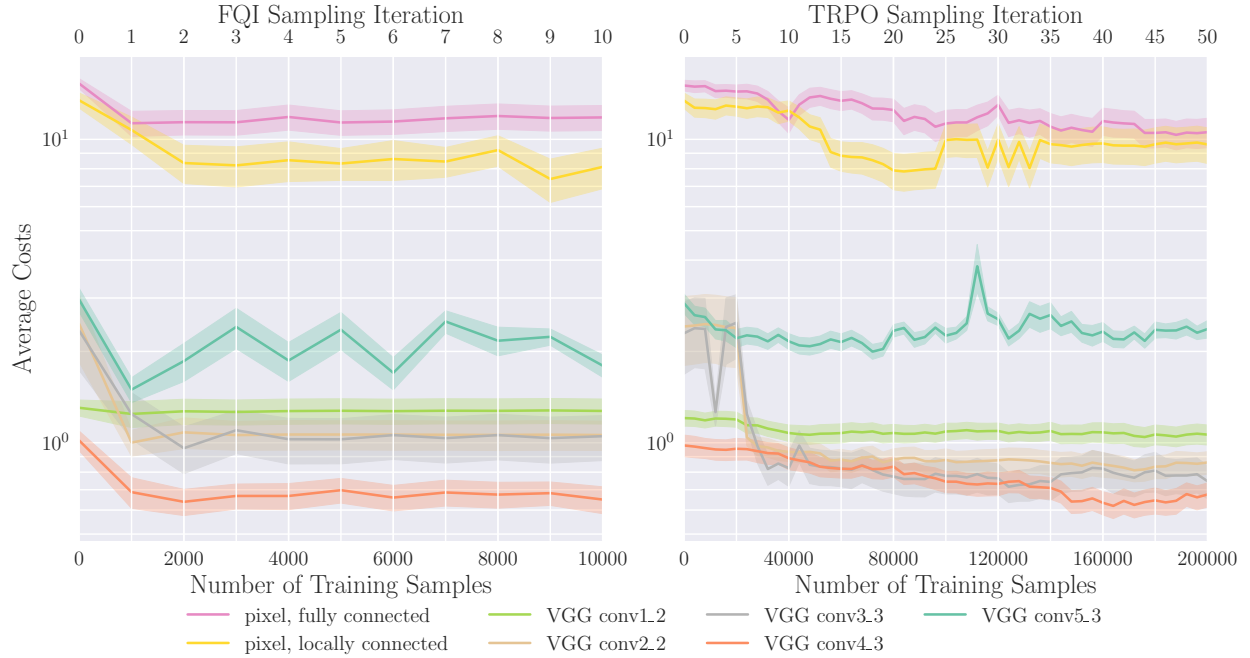


Figure A.1: Costs of validation executions using various feature dynamics models, where the feature weights are optimized with FQI (left plot) or TRPO (right plot). The reported values are the mean and standard error across 10 validation trajectories, of up to 100 time steps each.

regularization coefficient of $\nu = 0.1$. We found that regularization of the parameters was important for the algorithm to converge. We show sample trajectories of the resulting policies in Table A.2.

The FQI algorithm often achieved most of its performance gain after the first iteration. We ran additional sampling iterations of FQI to see if the policies improved further. For each iteration, we evaluated the performance of the policies on 10 validation trajectories. We did the same for the policies trained with TRPO, and we compare the learning curves of both methods in Figure A.1.

Learning End-to-End Servoing Policies with TRPO

We use TRPO to train end-to-end servoing policies for various observation modalities and report the performance of the learned policies in Table A.3. The policies are trained with the set of training cars, and tested on both this set and on the set of novel cars. The observation modalities that we consider are ground truth car positions (relative to the quadcopter), images of pixel intensities from the quadcopter’s camera, and VGG features extracted from those images. Unlike our method and the other experiments, no feature dynamics are explicitly learned for these experiments.

We use a Gaussian policy, where the mean is either a multi-layer perceptron (MLP) or a convolutional neural net (CNN), and the standard deviation is initialized to 1. We also use a Gaussian baseline, which is parametrized just as the corresponding Gaussian policy (but no parameters are shared between the policy and the baseline). For the policy that takes in car positions, the mean

Observation Modality	
ground truth car position	0.59 ± 0.24
raw pixel-intensity images	3.23 ± 0.22
VGG conv1_2 features	7.45 ± 0.40
VGG conv2_2 features	13.38 ± 0.53
VGG conv3_3 features	10.02 ± 0.49

(a) Costs when using the set of cars seen during learning.

Observation Modality	
ground truth car position	0.59 ± 0.24
raw pixel-intensity images	5.20 ± 0.40
VGG conv1_2 features	8.35 ± 0.44
VGG conv2_2 features	14.01 ± 0.47
VGG conv3_3 features	10.51 ± 0.65

(b) Costs when using a new set of cars, none of which were seen during learning.

Table A.3: Costs on test executions of servoing policies that were trained end-to-end with TRPO. These policies take in different observation modalities: ground truth car position or image-based observations. This table follows the same format as Table A.1. The mean of the first policy is parametrized as a 3-layer MLP, with tanh non-linearities except for the output layer; the first 2 fully connected layers use 32 hidden units each. For the other policies, each of their means is parametrized as a 5-layer CNN, consisting of 2 convolutional and 3 fully-connected layers, with ReLU non-linearities except for the output layer; the convolutional layers use 16 filters (4×4 , stride 2) each and the first 2 fully-connected layers use 32 hidden units each. All the policies are trained with TRPO, a batch size of 4000 samples, 500 iterations, and a step size of 0.01. The car position observations are not affected by the appearance of the cars, so the test performance for that modality is the same regardless of which set of cars are used.

is parametrized as a 3-layer MLP, with tanh non-linearities except for the output layer; the first 2 fully connected layers use 32 hidden units each. For the other policies, each of their means is parametrized as a 5-layer CNN, consisting of 2 convolutional and 3 fully-connected layers, with ReLU non-linearities except for the output layer; the convolutional layers use 16 filters (4×4 , stride 2) each and the first 2 fully-connected layers use 32 hidden units each.

The CNN policies would often not converge for several randomly initialized parameters. Thus, at the beginning of training, we tried multiple random seeds until we got a policy that achieved a relatively low cost on validation trajectories, and used the best initialization for training. The MLP policy did not have this problem, so we did not have to try multiple random initializations for it. All the policies are trained with a batch size of 4000 samples, 500 iterations, and a step size of 0.01. The parameters of the last iteration were used for the final policy.

Classical Image-Based Visual Servoing

Traditional visual servoing techniques (Feddema and Mitchell, 1989; Weiss et al., 1987) use the image-plane coordinates of a set of points for control. For comparison to our method, we evaluate the servoing performance of feature points derived from bounding boxes and keypoints derived from hand-engineered features, and report the costs of test executions on Table A.4.

We use bounding boxes from the C-COT tracker (Danelljan et al., 2016) (the current state-of-the-art visual tracker) and ground truth bounding boxes from the simulator. The latter is defined as the box that tightly fits around the visible portions of the car. We provide the ground truth bounding box of the first frame to the C-COT tracker to indicate that we want to track the car. We use the four corners of the box as the feature points for servoing to take into account the position and scale of the car in image coordinates.

We provide the ground truth depth values of the feature points for the interaction matrices. In classical image-based visual servoing, the control law involves the interaction matrix (also known as feature Jacobian), which is the Jacobian of the points in image space with respect to the camera’s control (see Chaumette and Hutchinson (2006) for details). The analytical feature Jacobian used in IBVS assumes that the target points are static in the world frame. This is not true for a moving car, so we consider a variant where the feature Jacobian incorporates the ground truth dynamics of the car. This amounts to adding a non-constant translation bias to the output of the dynamics function, where the translation is the displacement due to the car’s movement of the 3-dimensional point in the camera’s reference frame. Note that this is still not exactly equivalent to having the car being static since the roads have different slopes but the pitch and roll of the quadcopter is constrained to be fixed.

For the hand-crafted features, we consider SIFT (Lowe, 2004), SURF (Bay et al., 2006) and ORB (Rublee et al., 2011) keypoints. We filter out the keypoints of the first frame that does not belong to the car and use these as the target keypoints. However, we use all the keypoints for the subsequent observations.

The servoing policies based on bounding box features achieve low cost, and even lower ones if ground truth car dynamics is used. However, servoing with respect to hand-crafted feature points is significantly worse than the other methods. This is, in part, because the feature extraction and matching process introduces compounding errors. Similar results were found by Collewet and Marchand (2011), who proposed photometric visual servoing (i.e. servoing with respect to pixel intensities) and showed that it outperforms, by an order of magnitude, classical visual servoing that uses SURF features.

Classical Position-Based Visual Servoing

Position-based visual servoing (PBVS) techniques use poses of a target object for control (see Chaumette and Hutchinson (2006) for details). We evaluate the servoing performance of a few variants, and report the costs of test executions on Table A.5.

Similar to our IBVS experiments, we consider a variant that uses the car pose of the next time step as a way to incorporate the ground truth car dynamics into the interaction matrix. Since the

Observation Modality (Feature Points)		
corners of bounding box from C-COT tracker	(0.75)	1.70 ± 0.30
corners of ground truth bounding box	(0.75)	0.86 ± 0.25
corners of next frame's bounding box from C-COT tracker	(0.65)	1.46 ± 0.22
corners of next frame's ground truth bounding box	(0.65)	0.53 ± 0.05
SIFT feature points	(0.30)	14.47 ± 0.75
SURF feature points	(0.60)	16.37 ± 0.78
ORB feature points	(0.30)	4.41 ± 0.60

Table A.4: Costs on test executions when using classical image-based visual servoing (IBVS) with respect to feature points derived from bounding boxes and keypoints derived from hand-engineered features. Since there is no learning involved in this method, we only test with one set of cars: the cars that were used for training in the other methods. This table follows the same format as Table A.1. This method has one hyperparameter, which is the gain for the control law. For each feature type, we select the best hyperparameter (shown in parenthesis) by validating the policy on 10 validation trajectories for gains between 0.05 and 2, in increments of 0.05. The servoing policies based on bounding box features achieve low cost, and even lower ones if ground truth car dynamics is used. However, servoing with respect to hand-crafted feature points is significantly worse than the other methods.

Observation Modality (Pose)	Policy Variant					
	Use Rotation			Ignore Rotation		
car pose	(1.55)	0.58	± 0.25	(1.90)	0.51	± 0.25
next frame's car pose	(1.00)	0.0059 ± 0.0020		(1.00)	0.0025 ± 0.0017	

Table A.5: Costs on test executions when using classical position-based visual servoing (PBVS). Since there is no learning involved in this method, we only test with one set of cars: the cars that were used for training in the other methods. This table follows the same format as Table A.1. This method has one hyperparameter, which is the gain for the control law. For each condition, we select the best hyperparameter (shown in parenthesis) by validating the policy on 10 validation trajectories for gains between 0.05 and 2, in increments of 0.05. These servoing policies, which use ground truth car poses, outperforms all the other policies based on images. In addition, the performance is more than two orders of magnitude better if ground truth car dynamics is used.

cost function is invariant to the orientation of the car, we also consider a variant where the policy only minimizes the translational part of the pose error.

These servoing policies, which use ground truth car poses, outperforms all the other policies based on images. In addition, the performance is more than two orders of magnitude better if ground truth car dynamics is used.

Appendix B

Self-Supervised Visual Planning with Temporal Skip Connections

B.1 Hyperparameters

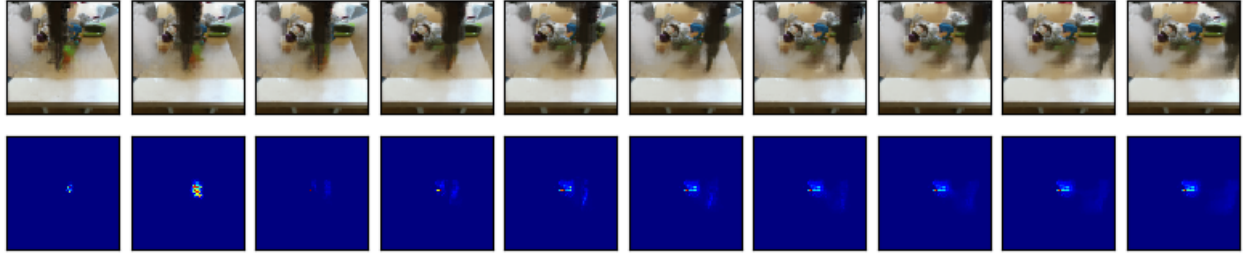
The video prediction network was trained on sequences of 15 steps taken from 44000 trajectories of length 30 by randomly shifting the 15-step long window thus providing a form of data augmentation. The model was trained on 66000 iterations with a minibatch-size of 32, using a learning rate of 0.001 with the Adam optimizer Kingma and Ba (2015). For visual MPC we use $n_{iter} = 3$ CEM-iterations. At every CEM iteration $M = 200$ action sequences are sampled and the best $K = 10$ samples are selected, fitting a Gaussian to these examples. Then new actions are sampled according to the fitted distribution and the procedure is repeated for n_{iter} iterations.

B.2 Prototyping in Simulation

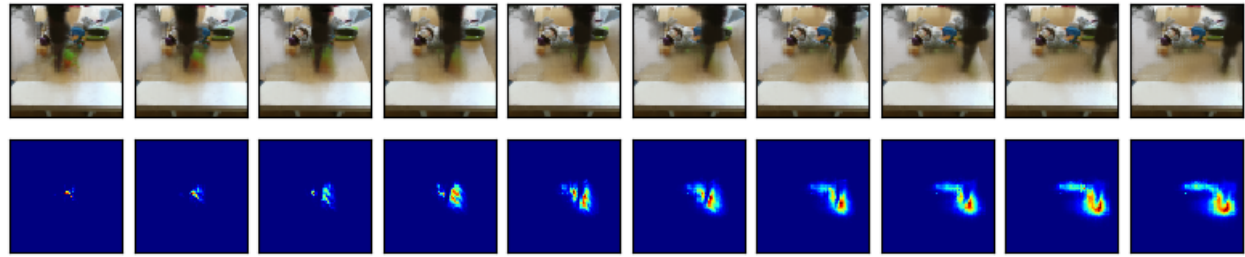
The visual MPC algorithm has been prototyped and tested on a simple block pushing task simulated in the MuJoCo physics engine. The code has been made available in the same repository. Using the simulator, training data could be collected orders of magnitude faster. Furthermore a benchmark was set up which does not require any manual reset or manual labeling of the objects' positions. The main downside of simulation however is that the complexity of both real-world dynamics and real-world visual scenes cannot be matched. (More realistic simulators exist, but require very large computational resources and large amounts of hand-engineering for setup).

B.3 Dependence of Model Performance on Amount of Training Data

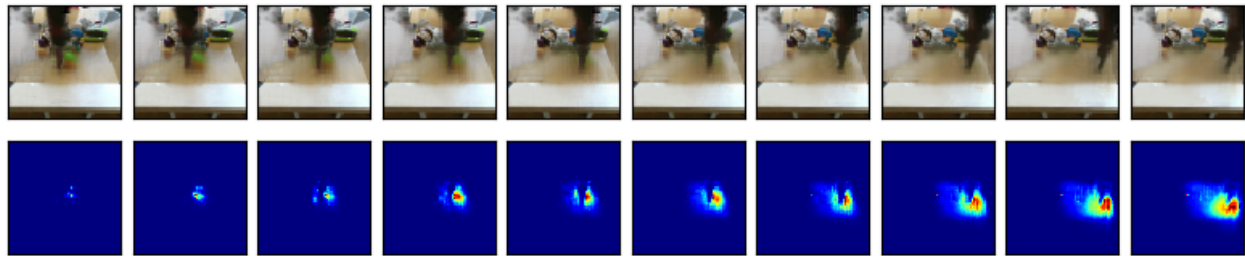
We tested the effect of using different amounts of training data and compare the results visually, see Figure B.1. The complete size of the data set is 44000 trajectories. For this evaluation all models



(a) using 5% of data



(b) using 10% of data



(c) using 100% of data

Figure B.1: Top rows: Predicted images. Bottom rows: Predicted probability distributions of designated pixel on the green object, marked in Figure 3.3. When using only 5% of the data (of 44000 trajectories in total), the model is not able to predict object movement at all (in fact the arm passes through the objects) and the distribution remains on the initial position of the object. For 30% percent the object distribution appears much more smeared than with 100% of the data.

were trained for 76000 iterations. The quality of the predictions for the probability distribution of the designated pixel vary widely with different amounts of training data and the quality of these predictions is crucial for the performance of visual MPC. The fact that performance still improves with very large amounts of training data suggests that the model has sufficient expressive power to leverage this amount of experience. In the future we will investigate the effect of using even more data and evaluate the effect of a larger number of different training objects.

B.4 Failure Cases and Limitations

Visual MPC fails when objects have vastly different appearance than those used in training. For example when objects are much bigger than any of the training objects these methods tend to perform poorly. Also we observed failure for one object with a very bright color which did not occur in the training set. However the model usually generalizes well to novel objects of similar size and appearance as the training objects. Apart from that we observed two main failure modes: The first one occurs when the probability mass of the designated pixel, $P_{t,d^{(i)}}$ drifts away from the object of interest (as discussed in the Section 3.8) since there is no feedback correcting the position of the designated pixel. To alleviate this problem a tracker can be integrated into the system enabling visual MPC to keep retrying when object behave differently than expected.

The second failure mode occurs when visual MPC does not find an action sequence which moves the designated pixel closer to the goal. This can happen, when the goalpoint for pushing the object is far away and it is very unlikely to sample a trajectory moving closer to the goal. In some cases the problem can be avoided simply by increasing the number of samples used when applying CEM. However this slows down the planning process. In order to enable more temporally extended action sequences a potential solution could be to increase the efficiency of the sampling process e.g. by introducing macro-actions which consist of a sequence of actions.

B.5 Robustness of the Model

We did not find any negative influence of clutter in the workspace, as long as the object to be moved has a free way path to its goal. When an obstacle is on the path it can be marked by a second designated pixel to avoid collision using an additional cost function. In this case the planner usually manages to find a way around the obstacle. However visual MPC with the current type of model is not yet capable of reasoning by itself that it has to push and object around obstacles when it is not marked explicitly. The reason is that the model has large uncertainty when multiple collisions occur (i.e. when the arm pushes object 1 and object 1 collides with object 2).

Our model is robust to small changes of the viewpoint (in the order of several centimetres in translation and several degrees in orientation). A likely reason is that during data collection the camera has been slightly displaced in position. Robustness could be improved by training a model from several viewpoints at the same time. As the prediction problem also becomes harder in this way more data might be required.

B.6 General skip connection neural advection model

We show a diagram of the general skip connection neural advection model in Figure B.2

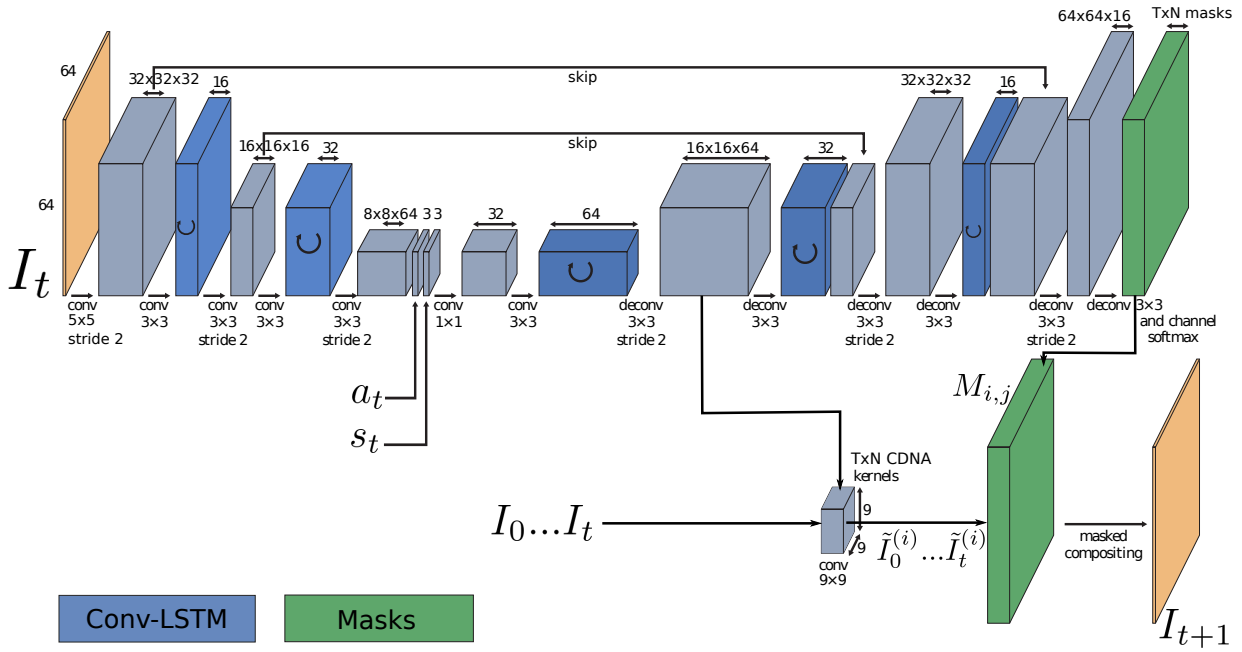


Figure B.2: General SNA model based on Equation (3.1).

Alternative Model Architectures

Instead of copying pixels from the first image of the sequence I_0 , the pixels in the first image can also be transformed and then merged together with the transformed pixels from the previous time step as

$$\hat{I}_{t+1} = \tilde{I}_0 \mathbf{M}_{N+1} + \sum_{i=1}^N \tilde{I}_t^{(i)} \mathbf{M}_i \quad (\text{B.1})$$

where \tilde{I}_0 is the transformed background. We tested this model on various pushing task and found that performance is comparable to the SNA model.

Appendix C

Stochastic Adversarial Video Prediction

C.1 Networks and Training Details

Network Details

Generator.

Our generator network, shown in Figure C.1, is inspired by the convolutional dynamic neural advection (CDNA) model proposed by (Finn et al., 2016a). The video prediction setting is a sequential prediction problem, so we use a convolutional LSTM (Hochreiter and Schmidhuber, 1997; Xingjian et al., 2015) to predict future frames. We initialize the prediction on the initial sequence of ground truth frames (2 frames for both datasets), and predict 10 future frames. The model predicts a sequence of future frames by repeatedly making next-frame predictions and feeding those predictions back to itself. For each one-step prediction, the predicted frame is given by a compositing layer, which composes intermediate frames with predicted compositing masks. The intermediate frames include the previous frame, transformed versions of the previous frame, and a frame with pixels directly synthesized by the network. The transformed versions of the frame are produced by convolving in the input image with predicted convolutional kernels, allowing for different shifted versions of the input. In more recent work, the first frame of the sequence is also given as one of the intermediate frames (Ebert et al., 2017).

To enable stochastic sampling, the generator is also conditioned on time-varying latent codes, which are sampled at training and test time. Each latent code \mathbf{z}_t is an 8-dimensional vector. At each prediction step, the latent code is passed through a fully-connected LSTM to facilitate correlations in time of the latent variables. The encoded latent code is then passed to all the convolutional layers of the main network, by concatenating it along the channel dimension to the inputs of these layers. Since they are vectors with no spatial dimensions, they are replicated spatially to match the spatial dimensions of the inputs.

We made a variety of architectural improvements to the original CDNA (Finn et al., 2016a) and SNA (Ebert et al., 2017) models, which overall produced better results on the per-pixel loss and similarity metrics. Each convolutional layer is followed by instance normalization (Ulyanov et al., 2016) and ReLU activations. We also use instance normalization on the LSTM pre-activations (i.e.,

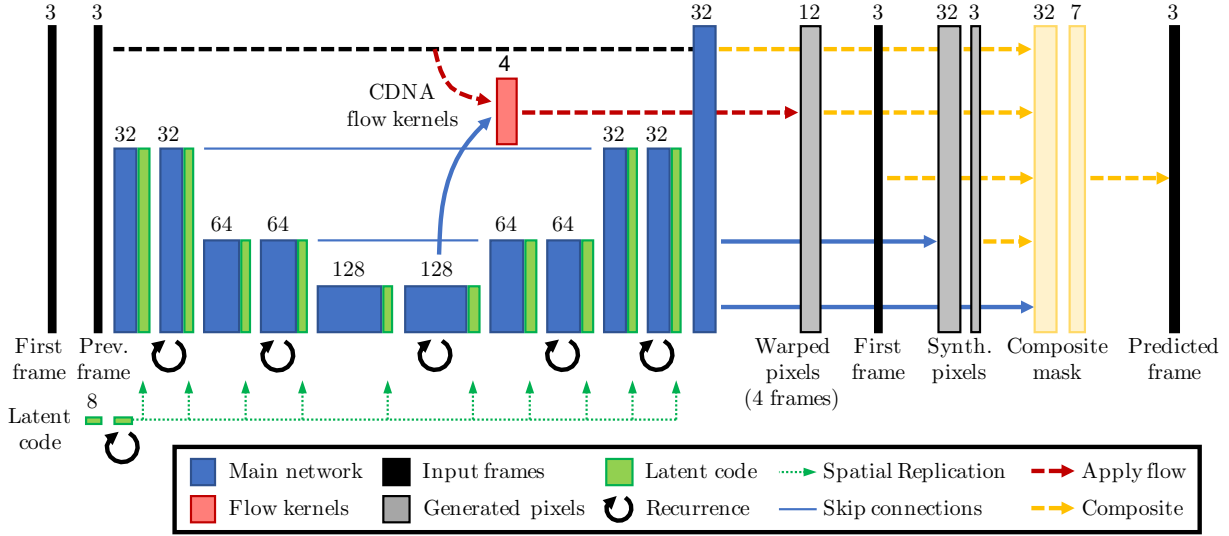


Figure C.1: **Architecture of our generator network.** Our network uses a convolutional LSTM (Hochreiter and Schmidhuber, 1997; Xingjian et al., 2015) with skip-connection between internal layers. As proposed by Finn et al. (2016a), the network predicts (1) a set of convolution kernels to produce a set of transformed input images (2) synthesized pixels at the input resolution and (3) a compositing mask. Using the mask, the network can choose how to composite together the set of warped pixels, the first frame, previous frame, and synthesized pixels. One of the internal feature maps is given to a fully-connected layer to compute the kernels that specify pixel flow. The output of the main network is passed to two separate heads, each with two convolutional layers, to predict the synthesized frame and the composite mask. These two outputs use sigmoid and softmax non-linearities, respectively, to ensure proper normalization. We enable stochastic sampling of the model by conditioning the generator network on latent codes. These are first passed through a fully-connected LSTM, and then given to all the convolutional layers of the the convolutional LSTM.

the input, forget, and output gates, as well as the transformed and next cell of the LSTM). In addition, we modify the spatial downsampling and upsampling mechanisms. Standard subsampling and upsampling between convolutions is known to produce artifacts for dense image generation tasks (Odena et al., 2016; Zhao et al., 2017; Niklaus et al., 2017). In the encoding layers, we reduce the spatial resolution of the feature maps by average pooling, and in the decoding layers, we increase the resolution by using bilinear interpolation. All convolutions in the generator use a stride of 1.

Encoder.

The encoder is a standard convolutional network that, at every time step, encodes a pair of images \mathbf{x}_t and \mathbf{x}_{t+1} into $\mu_{\mathbf{z}_t}$ and $\log \sigma_{\mathbf{z}_t}$. The latent variable \mathbf{z}_t is sampled at every time step and the same encoder network with shared weights is used at every step. The encoder architecture consists of three convolutional layers, followed by average pooling of all the spatial dimensions. Two separate fully-connected layers are then used to estimate $\mu_{\mathbf{z}_t}$ and $\log \sigma_{\mathbf{z}_t}$, respectively. The convolutional

layers use instance normalization, leaky ReLU non-linearities, and stride 2. This encoder architecture is the same one used in BicycleGAN (Zhu et al., 2017) except that the inputs are pair of images, concatenated along the channel dimension.

Discriminator.

The discriminator is a 3D convolutional neural network that takes in all the images of the video at once. We use spectral normalization and the SNGAN discriminator architecture (Miyato et al., 2018), except that we “inflate” the convolution filters from 2D to 3D. The two video discriminators, D and D_{VAE} , share the same architecture, but not the weights, as done in BicycleGAN (Zhu et al., 2017).

Training Details

Our generator network uses scheduled sampling during training as in Finn et al. (2016a), such that at the beginning the model is trained for one-step predictions, while by the end of training the model is fully autoregressive. We trained all models with Adam (Kingma and Ba, 2015) for 300000 iterations, linearly decaying the learning rate to 0 for the last 100000 iterations. The same training schedule was used for all the models, except for SVG, which was trained by its author. Our GAN-based variants used an optimizer with $\beta_1 = 0.5$, $\beta_2 = 0.999$, learning rate of 0.0002, and a batch size of 16. The VAE models (including SV2P from prior work) used an optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, learning rate of 0.001, and a batch size of 32. For our VAE-based variants, we linearly anneal the weight on the KL divergence term from 0 to the final value λ_{KL} during training, as proposed by Bowman et al. (2016), from iterations 50000 to 100000.

Appendix D

Stochastic Latent Actor-Critic: Deep Reinforcement Learning with a Latent Variable Model

D.1 Training and Evaluation Details

The control portion of our algorithm uses the same hyperparameters as SAC (Haarnoja et al., 2018a), except for a smaller replay buffer size of 100000 environment steps (instead of a million) due to the high memory usage of image observations. All of the parameters are trained with the Adam optimizer (Kingma and Ba, 2015), and we perform one gradient step per environment step. The Q-function and policy parameters are trained with a learning rate of 0.0003 and a batch size of 256. The model parameters are trained with a learning rate of 0.0001 and a batch size of 32. We use sequences of length $\tau = 8$ for all the tasks. Note that the sequence length can be less than τ for the first t steps ($t < \tau$) of each episode.

We use action repeats for all the methods, except for D4PG for which we use the reported results from prior work (Tassa et al., 2018). The number of environment steps reported in our plots correspond to the unmodified steps of the benchmarks. Note that the methods that use action repeats only use a fraction of the environment steps reported in our plots. For example, 3 million environment steps of the cheetah task correspond to 750000 samples when using an action repeat of 4. The action repeats used in our experiments are given in Table D.1.

Unlike in prior work (Haarnoja et al., 2018a;b), we use the same stochastic policy as both the behavioral and evaluation policy since we found the deterministic greedy policy to be comparable or worse than the stochastic policy.

Benchmark	Task	Action repeat	Original control time step	Effective control time step
DeepMind Control Suite	cheetah run	4	0.01	0.04
	walker walk	2	0.025	0.05
	ball-in-cup catch	4	0.02	0.08
	finger spin	2	0.02	0.04
OpenAI Gym	HalfCheetah-v2	1	0.05	0.05
	Walker2d-v2	4	0.008	0.032
	Hopper-v2	2	0.008	0.016
	Ant-v2	4	0.05	0.2

Table D.1: Action repeats and the corresponding agent’s control time step used in our experiments.

D.2 Reward Functions for the Manipulation Tasks

Sawyer Door Open

$$r = -|\theta_{\text{door}} - \theta_{\text{desired}}| \quad (\text{D.1})$$

Sawyer Drawer Close

$$\begin{aligned} d_{\text{handToHandle}} &= \|xyz_{\text{hand}} - xyz_{\text{handle}}\|_2^2 \\ d_{\text{drawerToGoal}} &= |x_{\text{drawer}} - x_{\text{goal}}| \\ r &= -d_{\text{handToHandle}} - d_{\text{drawerToGoal}} \end{aligned} \quad (\text{D.2})$$

Sawyer Pick-up

$$\begin{aligned} d_{\text{toObj}} &= \|xyz_{\text{hand}} - xyz_{\text{obj}}\|_2^2 \\ r_{\text{reach}} &= 0.25(1 - \tanh(10.0 * d_{\text{toObj}})) \\ r_{\text{lift}} &= \begin{cases} 1 & \text{if object lifted} \\ 0 & \text{else} \end{cases} \\ r &= \max(r_{\text{reach}}, r_{\text{lift}}) \end{aligned} \quad (\text{D.3})$$

D.3 Additional samples from our model

We show additional samples from our model in Figure D.1, Figure D.2, and Figure D.4. Samples from the posterior show the images \mathbf{x}_t as constructed by the decoder $p_\psi(\mathbf{x}_t|\mathbf{z}_t)$, using a sequence of latents \mathbf{z}_t that are encoded and sampled from the posteriors, $q_\psi(\mathbf{z}_1|\mathbf{x}_1)$ and $q_\psi(\mathbf{z}_{t+1}|\mathbf{x}_{t+1}, \mathbf{z}_t, \mathbf{a}_t)$. Samples from the prior, on the other hand, use a sequence of latents where \mathbf{z}_1 is sampled from $p(\mathbf{z}_1)$

and all remaining latents \mathbf{z}_t are from the propagation of the previous latent state through the latent dynamics $p_\psi(\mathbf{z}_{t+1}|\mathbf{z}_t, \mathbf{a}_t)$. These samples do not use any image frames as inputs, and thus they do not correspond to any ground truth sequence. We also show samples from the conditional prior, which is conditioned on the first image from the true sequence: for this, the sampling procedure is the same as the prior, except that \mathbf{z}_1 is encoded and sampled from the posterior $q_\psi(\mathbf{z}_1|\mathbf{x}_1)$, rather than being sampled from $p(\mathbf{z}_1)$.

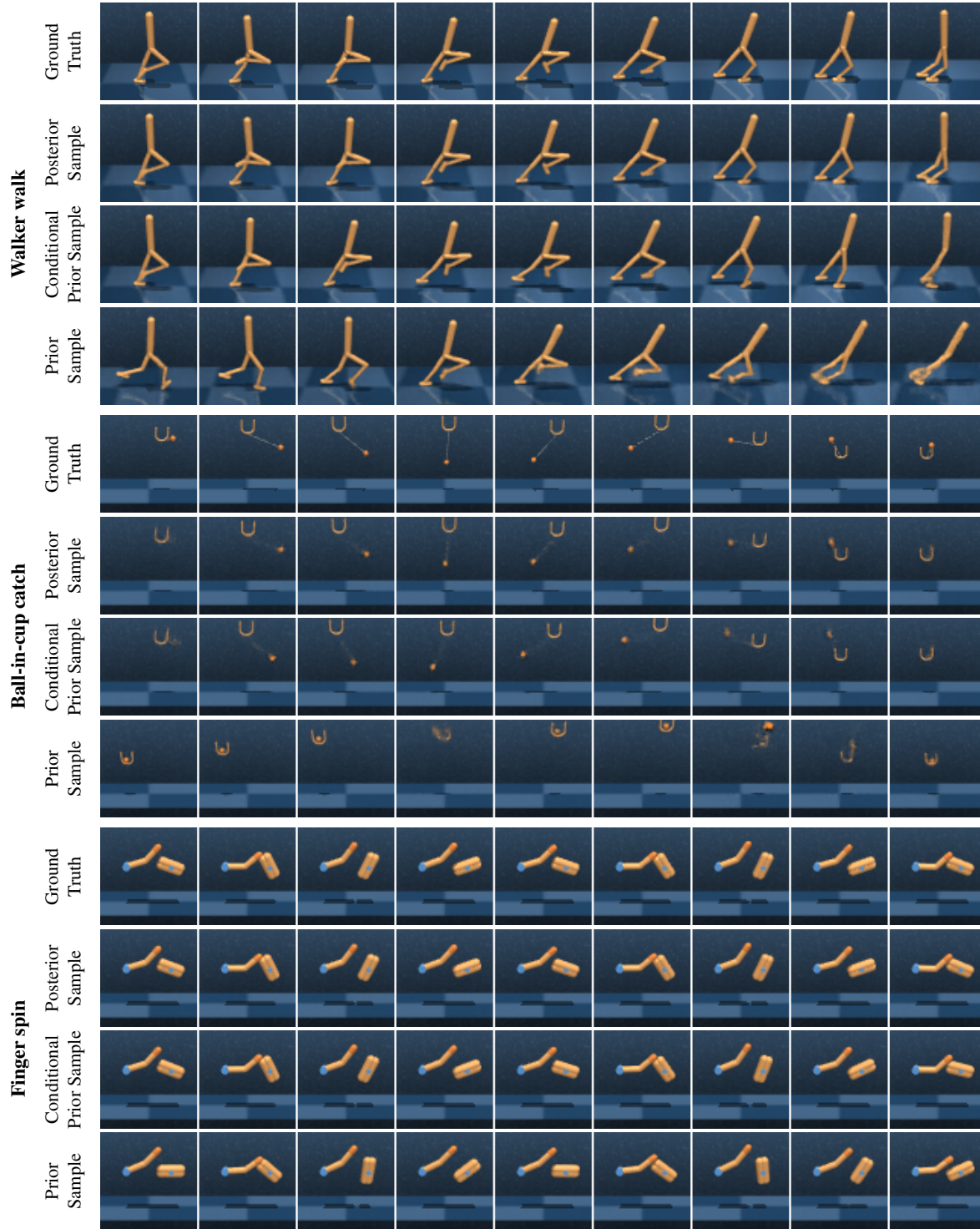


Figure D.1: Example image sequences, along with generated image samples, for three of the DM Control tasks that we used in our experiments. See Figure 5.9 for more details and for image samples from the cheetah task. Continues on Figure D.3.

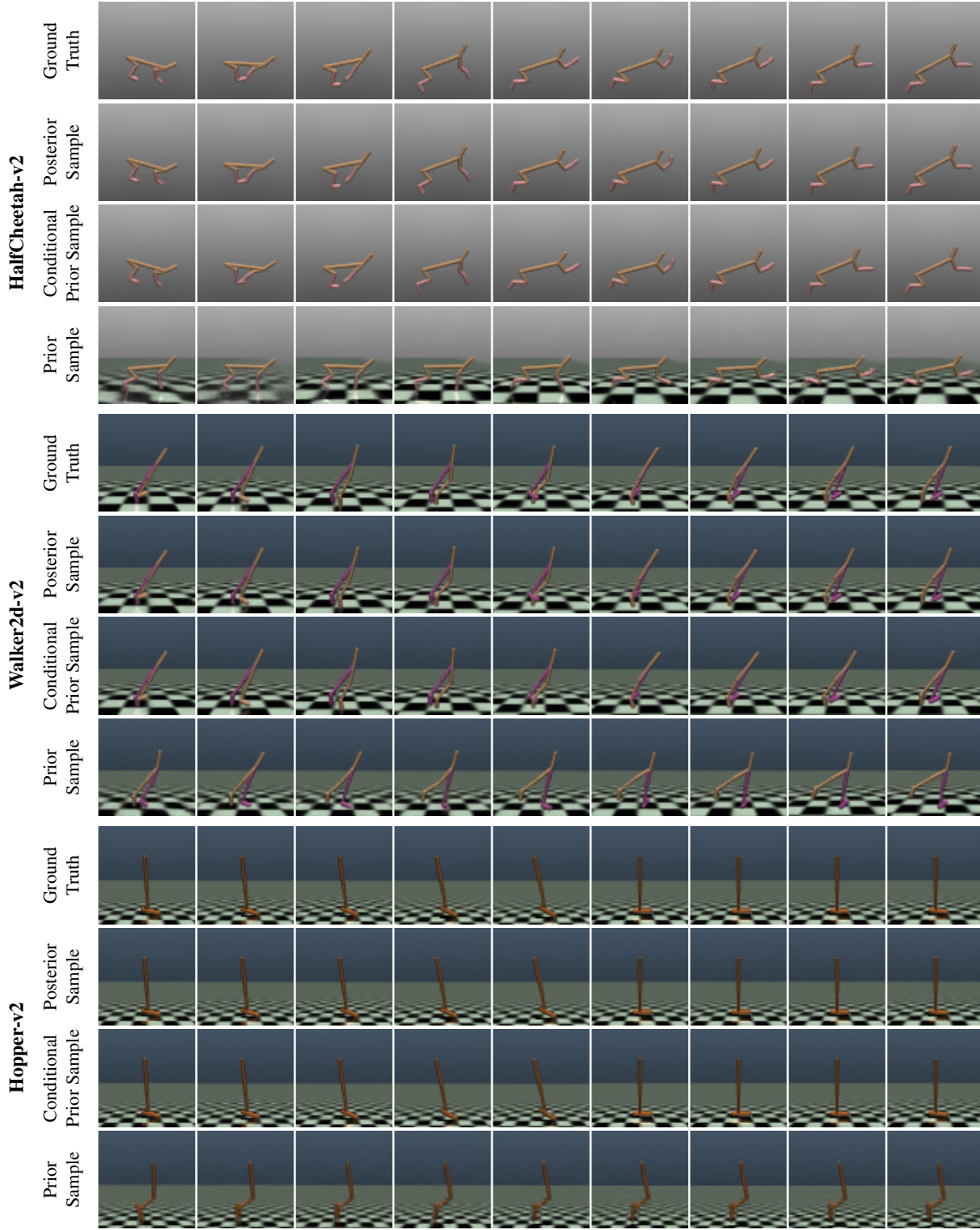


Figure D.2: Example image sequences, along with generated image samples, for the four OpenAI Gym tasks that we used in our experiments.

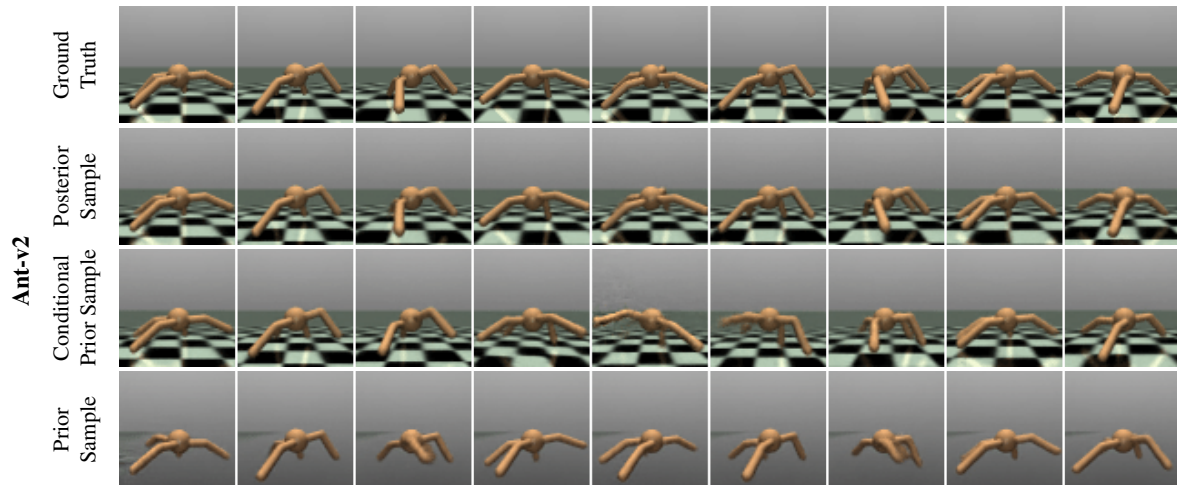


Figure D.3: Continuation of Figure D.2. Example image sequences, along with generated image samples, for the four OpenAI Gym tasks that we used in our experiments.

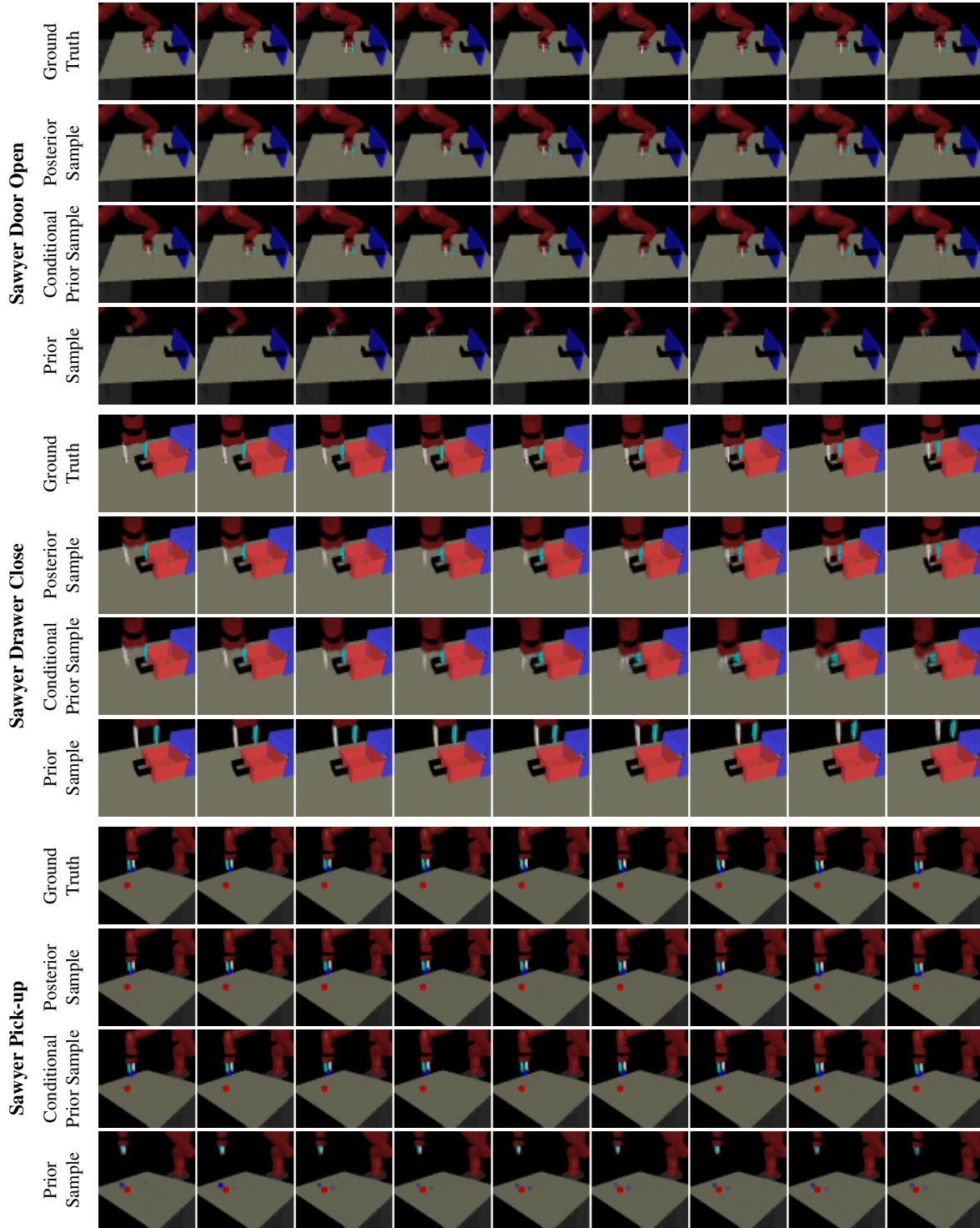


Figure D.4: Example image sequences, along with generated image samples, for the manipulation tasks that we used in our experiments. Note that these image samples are difficult to interpret as image strips, so we instead recommend the videos of these samples that are on the project website.