Structured Deep Visual Dynamics Models for Robot Manipulation

Arunkumar Byravan

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2019

Reading Committee:

Dieter Fox, Chair

Siddhartha Srinivasa

Byron Boots

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Structured Deep Visual Dynamics Models for Robot Manipulation

Arunkumar Byravan

Chair of the Supervisory Committee:
Professor Dieter Fox
Computer Science and Engineering

The emergence of deep learning, access to large amounts of data and powerful computing hardware have led to great strides in the state-of-the-art in robotics, computer vision, and AI. Unlike traditional methods that are strongly model-based with priors and explicit structural constraints, these newer learning approaches tend to be data-driven and often neglect the underlying problem structure. As a consequence, while they usually outperform their traditional counterparts on many problems, achieving good generalisation, interpretability, task transfer and data-efficiency has been challenging. Combining the strengths of the two paradigms, the flexibility of modern learning techniques, and the domain knowledge and structure of traditional methods should help bridge this gap.

In this thesis, we will present work that combines these two paradigms, specifically in the context of learning visual dynamics models for robot manipulation tasks. This thesis is divided into two parts. In the first part, we discuss a structured approach to designing visual dynamics models for manipulation tasks. We propose a specific class of deep visual dynamics models (SE3-Nets) that explicitly encode strong physical and 3D geometric priors (specifically, rigid body physics) in their structure. As opposed to deep models that reason about motion a pixel level, SE3-Nets model the dynamics of observed scenes at the object level - they identify objects in the scene and predict rigid body rotation and translation per object. This leads to an interpretable architecture that can robustly model the dynamics of

complex interactions. Next, we discuss SE3-Pose-Nets, an extension of SE3-Nets that additionally learns to estimate a latent, globally-consistent pose representation for objects and use the corresponding representation for real-time closed-loop visuomotor control of a Baxter robot. We show that the structure inherent in SE3-Pose-Nets allows them to be robust to visual perturbations and noise, generalizing to settings significantly different than seen during training. We also briefly discuss Dynamics-Nets, a recurrent extension to SE3-Pose-Nets that can be used for the control of dynamical systems.

In the second part of the thesis, we present an approach towards solving long-horizon manipulation tasks, using reinforcement learning; we combine the flexibility of modern model-free RL approaches with model-based reasoning. Our approach, Imagined Value Gradients (IVG), learns a predictive model of expected future observations, rewards and values from which a policy can be derived by following the gradient of the estimated value along imagined trajectories. We show how robust policy optimization can be achieved even with approximate models on robot manipulation tasks, learned directly from vision and proprioception. We evaluate the efficacy of our approach in a transfer learning scenario, re-using previously learned models on tasks with different reward structures and visual distractors. The structure inherent in our system, i.e. the model, allows us to transfer knowledge across tasks, achieving significant improvements in learning speed compared to strong model-free baselines.

Finally, we conclude with a discussion of the proposed work and future directions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First, I would like to thank my advisor Dieter Fox for his encouragement, guidance and advice throughout my doctoral studies. Dieter was instrumental in guiding me to work on the area of structured deep learning models and shaped my views on research, including how it should be done in a principled manner. I am constantly amazed by his enthusiasm for research, insight and knowledge of the field. It has been a privilege to work with and learn from him during my time at the University of Washington. I am indebted to him for his gracious support during many difficult periods over the course of my Ph.D. Additionally, I would like to thank the members of my committee, Byron Boots, who mentored me during my initial years at UW, Siddhartha Srinivasa, with whom I co-authored my first paper and the graduate school representative, Blake Hannaford.

I would also like to thank my fellow labmates from the Robotics and State Estimation Lab and my peers from the Systems Lab for the many interesting discussions, insights, moral support and help on solving many a problem that crept up during the course of my studies. They were also subject to my taste for Indian food whenever we went out for lunch; thanks so much for indulging and tolerating me!

Next, I would like to acknowledge the support given to me by everyone at the University of Washington, especially Lindsay Michimoto, who was extremely helpful during the early years of my Ph.D. I also thank the National Science Foundation for funding much of the work developed in this thesis.

I would like to give a special note of thanks to my Masters advisors Daniel Koditschek and Shai Revzen during my time at the University of Pennsylvania – their guidance during my first research projects were invaluable and really pushed me to pursue a career in research.

Finally, I would like to thank my family, especially my dad and mom, for their love, understanding and encouragement throughout my doctoral studies; I am forever grateful for their support.

DEDICATION

This thesis is dedicated to the memory of my loving father, Byravan Muthusubramaniam, without whose encouragement, care, support, and unwavering faith in me none of this would have been possible.

Chapter 1

INTRODUCTION

Manipulation is the act of making an intentional change to the environment or objects in the environment through selective contact [108, 111]. Manipulation played a significant role in human evolution and is surprisingly widespread among animals; many everyday human activities involve manipulation skills like pushing, grasping, lifting and tool use. Manipulation is also a fundamental challenge in robotics; for a robot to operate alongside a human, it needs to be able to manipulate its environment freely.

The ability to predict how an environment changes based on forces applied to it is fundamental for a robot to achieve desired manipulation goals. For instance, in order to arrange objects on a table into a desired configuration, a robot has to be able to reason about where and how to push individual objects, which requires some understanding of physical quantities such as object boundaries, mass, surface friction, and their relationship to forces. This understanding of the environment, often endowed with a capability to predict future outcomes of actions, is referred to as a "**model**"; this thesis explores the benefits of learning (structured) models for enabling robot manipulation.

Humans begin to develop an intuitive understanding of their environment, i.e. a **model**, right from a young age; for example, infants as young as 5 months old have a notion of objects and their permanence when occluded by other objects [12]. Similarly, infants around 6.5 months old have an intuitive concept of physics; they can already reason about support phenomena and object collisions [11]. As infants grow and interact further with their environment, their models improve; adults have a significantly better and more general model of their environment, including an approximate understanding of physical concepts such as mass, friction and contact. Studies have shown that human intuitive models of physics, while

approximate, can accurately predict the outcomes of complex physical tasks, reconciling well with results generated from probabilistic physics simulators [15, 130]. There is also evidence that models help bootstrap learning on related and similar tasks; by leveraging prior knowledge from their internal models, human players can learn novel, complex visual games in minutes [154].

Just as intuitive human models have played a significant role in development of robust manipulation skills, models have many demonstrable advantages for robotics. Models formalize our intuition of how the physical world works and enable us to take advantage of domain knowledge and problem structure when designing solutions for manipulation tasks. A standard approach in robot manipulation has been to use a physical model of the environment and perform optimal control to find a policy that leads to the goal state [110]. For instance, extensive work utilizing the Mujoco physics engine [150] has shown how strong physics models can enable solutions to control problems in complex and contact-rich environments [59]. Similarly, closed-form models have been proposed to model the mechanics of many manipulation skills in simple, structured environments; combined with planning approaches these have shown impressive results on tasks such as grasping [109, 38, 127], planar pushing [176, 107, 93], non-prehensile manipulation [87] and even on tasks involving contact and collisions [159]. A shortcoming of such models is, however, that they rely on very accurate estimates of the state of the system [180, 92, 34]. Unfortunately, estimating values such as the mass distribution, surface friction and even the pose of an unseen, unknown object using visual information and force feedback is extremely difficult. This is one of the main reasons why humans are still far better than robots at performing even simple tasks such as pushing an arbitrary object along a desired trajectory, even though their control policies are informed only by approximate, intuitive notions of physics. Robot manipulation in unstructured, non-industrial settings is particularly hard due to the aforementioned problems, making it a great avenue for research in robotics.

Recently, machine learning models, specifically deep neural networks, have achieved tremendous success in computer vision [94, 105], natural language processing [143, 9] and in

reinforcement learning [112, 140]. Deep networks have achieved the state of the art results in many visual tasks that are relevant to robotics such as object detection and recognition [60, 71], semantic segmentation [105, 69], pose estimation [164, 171] and dense optical flow prediction [51, 79]. These successes have in turn led to a growing body of work applying machine learning techniques to robotics problems. Of particular interest are video prediction models [19, 47, 54, 174], much akin to the predictive models we described above, visuomotor control approaches that use a learned predictive model along with optimal control techniques [49, 160] and end-to-end policy learning approaches which learn closed-loop control policies directly from visual observations [101, 112, 126]. A crucial advantage of these methods is their capacity to learn complex causal relationships and behaviours directly from raw sensor data while minimizing the amount of expert domain knowledge and modeling needs, much unlike traditional robotics techniques. On the other hand, these data-driven methods suffer from poor sample complexity, requiring significant amounts of data to train, and can have weaker generalization, interpretability and robustness compared to more model-based approaches prevalent in standard robotics pipelines.

To summarize, there is an interesting dichotomy between model-based and data-driven methods for robotics; traditional models are general formalisms that transfer easily across domains but make strong assumptions and require accurate system identification. The opposite is true for deep networks which are flexible learners with substantial capacity; unfortunately, they struggle to generalize across domains and tasks. Given these seemingly contrasting differences between these paradigms, can we benefit from a principled combination of the two?

1.1 Model-Based vs Data-Driven

An interesting perspective on model learning can be gleaned from considering the bias-variance trade off [166] inherent in machine learning (see Fig. 1.1). As the complexity of the model increases, the model's bias (or the assumptions inherent in the model) reduces, thereby allowing it to fit the training data well. Unfortunately, this can lead to a reduction in

generalization performance, or an increase in the variance (the performance on held out test data). This is the problem of overfitting, where the model captures the noise in the data as opposed to the underlying causal structure. The opposite, underfitting, occurs when the model makes erroneous assumptions about the data. The minimum error point is reached when these two conflicting sources of error are balanced.

We posit that model-based approaches in traditional robotics fall on the left half of the trade off curve, i.e. they have high bias and (potentially) low variance. Physics models are often approximate, making several assumptions about the mechanics of manipulation especially in cases of object interactions [38], contact [43] and collisions [151]. At the same time, a manipulation algorithm driven by a physics model can easily generalize to different objects, robots and sensors; physical principles (e.g. Newton's laws) are broadly applicable. On the other end of the spectrum, we have data-driven methods such as deep learning which have high variance and low bias. Neural networks are universal function approximators [77], with a capacity to fit a variety of data extremely well. The caveat, though, is that deep networks are prone to overfitting. Indeed, deep networks are shown to be sensitive to noise addition [117, 142], mis-labeling examples even in the presence of a small amount of input noise. Similar behaviors can be observed when training deep networks for robotic tasks; the resulting systems are brittle and task-specific [101], performing poorly even under small changes to the data distribution.

Both model-based and data-driven approaches have their strengths and weaknesses. Traditional model-based approaches are broadly applicable, but make many assumptions and usually require accurate state estimates. Data-driven approaches such as deep networks can learn complex behaviors directly from raw data but generalize poorly. Can we combine these paradigms to get the best of both worlds? The first half of this thesis explores this question in detail, primarily in the context of learning predictive models of environment dynamics directly from raw visual data and optionally, proprioception information (e.g. joint angles and velocities). We integrate model-based structural priors, specifically rigid body motion, into the architecture of deep networks for modeling visual dynamics; the resulting architectures

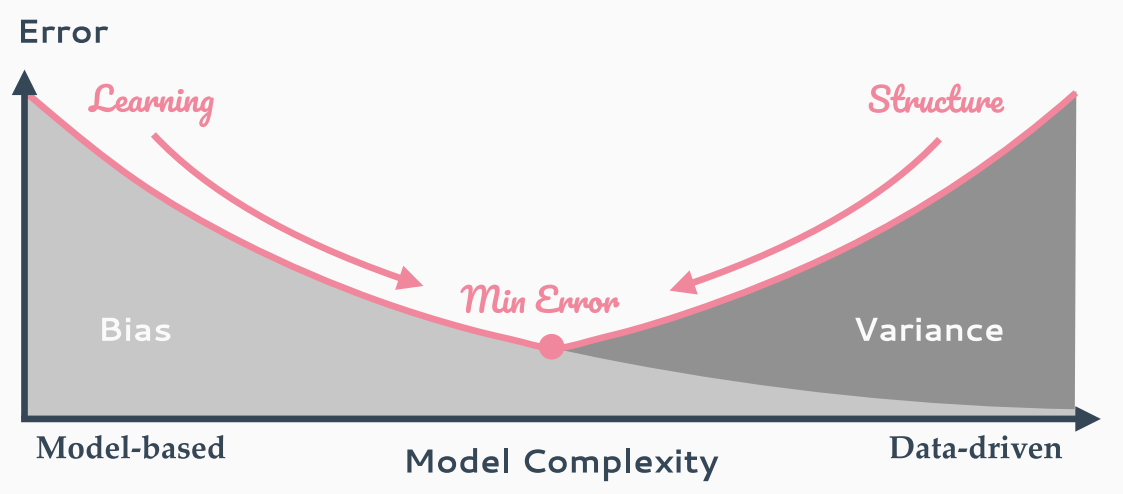

Figure 1.1: Bias-Variance trade off curve. As the model's complexity increases, it can fit the training data well (thereby reducing bias), but generalization error, or the variance (measured on held out data), increases. The minimum error point achieves the best trade off between training performance and generalization. Traditional model-based robotics approaches have high bias and low variance; physics models tend to capture general, overarching principles but can struggle to fit observed data due to inaccuracies in modeling and system identification. On the other hand, (deep) learned models appear have low bias and high variance; they fit observed data extremely well but can struggle to learn the underlying causal structure in the data leading to poor generalization. Combining the **structure** of model-based approaches with the **learning** capabilities of modern deep networks can give us the best of both worlds, driving our algorithms towards the minimum error point. Figure courtesy: Jeannette Bohg

SE3-Nets and SE3-Pose-Nets are interpretable, object centric, deep visual dynamics models that can predict the dynamics of complex object interactions directly from raw visual observations. We also present applications of these models to the task of visuomotor control.

Reinforcement Learning (RL) is a general sequential learning approach that has recently been successfully applied to many problems in AI [140, 112] and robotics [36, 101, 126]. There are two major paradigms in RL: model-based methods, which like traditional robotics approaches, estimates a dynamics model of the environment and model-free approaches, which learn direct mappings of states to actions, usually through a function approximator such as a deep network. Going back to the bias-variance tradeoff (Fig. 1.1), model-based RL appears on the left side; the models learned by these methods are (hypothetically) generalizable, but they are limited by their reliance on local planners and sensitivity to model approximation errors. On the other hand, model-free RL can flexibly learn task-specific complex policies, but cannot generalize to novel settings; they have low bias and high variance. Interestingly, this is the same contrast between model-based and data-driven methods; can we hope to gain similar advantages when we combine these two RL paradigms? We explore this question in the second half of the thesis where we propose a hybrid reinforcement learning agent that learns a model in tandem with a parameterized policy. We evaluate this agent on a set of long-horizon manipulation tasks; model-based RL would fail on these tasks due to the length of the planning horizon while our approach succeeds by virtue of the model-free policy. Crucially, we also show that learning a model allows us to transfer knowledge across related tasks, much akin to human learning of manipulation skills.

The rest of this chapter is organized as follows. We first describe the structured model learning problem tackled in the first half of the thesis and briefly describe our approach in Sec. 1.2. Next, we discuss the reinforcement learning problem explored in the second half of the thesis and our approach towards combining model-based and model-free RL in Sec. 1.3. Finally, we present an overview of this thesis in Sec. 1.4.

1.2 The role of structure

From the discussion above, we can summarize a few key advantages of model-based methods. First, model-based approaches are **interpretable**, i.e. the state of the system can be summarized succinctly by the model parameters, which for many manipulation pipelines involve representations of the robot's kinematics, object poses, geometry and physical parameters. Second, model-based approaches are **generalizable**. For example, a manipulation system that operates by planning on an underlying physics model can be applied for many different manipulators, objects and environments, assuming that the model's parameters are accurately identified. Lastly, model-based approaches are **composable**. Many traditional robotics pipelines follow a modular architecture, usually inspired by the sense-plan-act paradigm where a state estimator is used for identifying the system state followed by a planning via a physics model and execution through a local controller. On the other hand, machine learning models, specifically deep networks, tend to be monolithic, black box systems with hundreds of network layers and millions of parameters [71, 177].

By integrating model-based reasoning with deep networks, we can hope to inherit many of the advantages described above while retaining the general flexibility of learned models. While there are many ways to achieve this, we propose to combine these paradigms by explicitly structuring the network architecture of standard deep networks. We focus on the problem of learning predictive models of environment dynamics directly from raw visual data and corresponding applications of these models for visuomotor control, where we control the manipulator to reach a target visual state. We make the following structural constraints and assumptions, each of which contribute towards an addition of the properties described above. First, we limit the scope of the dynamics learning problem to modeling rigid bodies and their dynamics. This allows us to introduce several constraints into our network architecture which improve performance and interpretability, resulting in a modular network architecture. Second, we force our deep network to learn an intermediate representation of object segmentation and 6D poses. This leads to an interpretable model that mimics a state estimator in traditional

systems while retaining the capacity to operate on raw high-dimensional observations, with fewer parameters compared to standard deep architectures. Third, we learn to model the dynamics of objects as opposed to pixels. This nicely constrains the learning problem to a low-dimensional interpretable space, exposing opportunities to use the resultant model for global data association and real-time control, even in the presence of noise. Last, we introduce a differentiable transformation layer that models the motion of systems of rigid objects, with no trainable parameters. The structure inherent in this layer allows us to learn complex object dynamics with self-supervised data association labels, significantly less supervision than would be needed otherwise. In general, we see that the **structure** and assumptions inherent in models, when used properly, can act as a regularizer for data-driven learning, potentially reducing overfitting and aiding generalization (see Fig. 1.1).

1.3 Models in Reinforcement Learning

Reinforcement Learning (RL) is a general, powerful approach towards learning complex skills for a variety of manipulation problems. Over the past few years, model-free reinforcement learning, where one directly learns a policy mapping from states to actions, has seen tremendous success on many challenging control problems [37, 90], leading to state of the art results even when learning directly from images [112, 103, 133, 126]. Much of this success has been driven by advances in machine learning, enabling policies to be represented with deep networks with millions of parameters. In spite of these advancements, several key challenges exist that limit their applicability. Firstly, the poor sample complexity of model-free RL means it requires large amounts of real-world interaction. More importantly, model-free methods transfer poorly; the learned policies are task-specific and hard to transfer even on related tasks.

In contrast to model-free RL, model-based approaches to reinforcement learning estimate the dynamics of observations, i.e., learn a model, leveraging optimal control and planning algorithms for generating actions. This can lead to improved data efficiency [36, 101] and crucially, fast adaptation when transferring to related domains [144]. Traditionally these

methods were limited to state-space tasks, but with the advent of deep learning there has been significant progress on learning image-based predictive models [174, 23, 39] and (structured) low-dimensional latent representations [160, 65], leading to model-based RL being applied to continuous visual control tasks [160, 157, 65, 178, 85, 161], Atari games [83] and tabletop manipulation [39]. A key drawback of many of these approaches is their reliance on planning through the learned model for generating actions at execution time, which can both be difficult in the presence of model errors (e.g. on long-horizon tasks and sparse reward tasks), and expensive in terms of computation.

Interestingly, the dichotomy between model-based and model-free RL is very similar to the tradeoffs between model-based and data-driven approaches to robotics (see Fig. 1.1); we can expect to gain similar benefits from a principled combination of the two. To this end, we propose a hybrid RL agent that jointly learns a model and a parametric policy for solving complex long-horizon manipulation tasks (e.g. lifting, stacking), directly from high-dimensional visual observations and proprioception. Our model learns a deterministic mapping of observations to abstract, latent states, evolving the transition dynamics in this low-dimensional representation; the learned policy also operates directly on this latent state. We train the model, policy, value and reward functions jointly from scratch and show that stable policy optimization can be achieved even with approximate models. Crucially, we show that the models learned by our RL agent can be transferred to related tasks; learning with a model pre-trained on a related task can result in a significant boost in learning speed compared to learning from scratch. We also present a multi-task version of our RL agent that trains the model simultaneously on multiple related tasks; models trained on multiple tasks exhibit significantly better generalization to related tasks, further speeding up learning in the transfer setting.

1.4 Thesis overview

This thesis is split into two parts, each of which explore a combination of model-based and data-driven approaches to solving manipulation problems in robotics, directly from raw

visual data. In the first half of the thesis, we explore learning structured visual predictive models from RGBD data. Chapter 2 presents our first structured deep model, the SE3-Net, which models the dynamics of systems of rigid bodies by explicitly segmenting them and predicting an $\mathbb{SE}(3)$ motion per object. We present results applying this model to simulated tabletop settings with multi-object interactions, simulated robot motion and real data where a robot pushes an object on a table. Next, in Chapter 3, we present the SE3-Pose-Net, an extension of the SE3-Net architecture that additionally predicts a globally consistent pose for each manipulable object in the scene. By leveraging the structured latent pose space of the SE3-Pose-Net we can tackle the problem of visuomotor control, where we control a robot manipulator to servo to a target visual state using only raw RGBD observations. We also briefly touch on some limitations of SE3-Pose-Nets and discuss ways to extend our architecture to handle these limitations. We briefly discuss Dynamics-Nets, a recurrent extension to SE3-Pose-Nets that can handle dynamic interactions and present initial results applying this model to the 2D cartpole task. In the second half of the thesis, we explore the integration of models with reinforcement learning (chapter 4). We develop Imagined Value Gradients (IVG), a model-based policy optimization algorithm that learns a predictive model of expected future observations, rewards and values from which a policy can be derived by following the gradient of the estimated value along imagined trajectories. We evaluate the efficacy of our approach in a transfer learning scenario, re-using previously learned models on tasks with different reward structures and visual distractors, and show a significant improvement in learning speed compared to strong off-policy baselines. Finally, Chapter 5 concludes with a discussion and future outlook for this line of work.

Part I

STRUCTURED DEEP VISUAL DYNAMICS MODELS

Chapter 2

SE3-NETS

Humans, and some animals, can effectively and efficiently manipulate objects in their environment, quickly learning, adapting and mastering many complex tasks while generalizing to potentially unseen situations. Much of this knowledge is transcribed into "models" of the environment, capturing intuitive notions of physics [15, 130]. There is evidence that these intuitive models are learned right from a young age through interactions with the environment; even infants understand concepts such as objects, their dynamics and object permanence [11, 10, 12].

Models are similarly prevalent in robotics and are used widely to formalize our intuitions about the physical world. For example, analytical and approximate models of many physical phenomena (e.g. dynamics, contact and collisions) are used as parts of many traditional robotics approaches. A standard approach in robot control has been to use a physical model of the environment and perform optimal control to find a policy that leads to the goal state. This has shown impressive results on many tasks, even in complex and contact-rich environments [59, 38, 127]. Unfortunately, many of our traditional model-based approaches have a few limitations: 1) They can be biased, as they make many assumptions and simplifications for tractability [38, 107, 127], 2) They require accurate state estimates which can be hard for parameters such as mass distribution and friction which are not directly observed [180, 92]. This is one of the main reasons why humans are still far better than robots at manipulation, especially in unstructured environments, even though their control policies are informed only by approximate, intuitive notions of physics.

An alternative approach to explicitly modeling the environment via an analytical model is to "learn" an implicit model of the world using interaction data. Recently, machine learning

model (such as deep neural networks) have achieved tremendous success in many fields [94, 105, 143, 112] and significant work has been done on applying these techniques for problems relevant to robotics, such as video prediction [19, 47, 54, 174], visuomotor control [49, 160] and end-to-end policy learning [101]. An advantage of these approaches is their ability to learn complex behaviors directly from raw visual data and ease of use as little domain knowledge is needed to learn these models. A few drawbacks include poor data efficiency, limited interpretability and weak generalization performance.

In this chapter, we look at the intersection of these two paradigms: we would like to retain the generality and structure of traditional "model-based" robotics approaches while having the flexibility to learn complex behaviours directly from raw perceptual data. We explore the use of deep learning to model the concept of "physical intuition", learning a model that predicts changes to the environment based on specific actions. We focus on modeling the motion of systems of rigid bodies, such as predicting how an object on a table moves when being pushed by a robot manipulator. Importantly, we want to learn predictive models from sequences of *raw 3D point clouds* observed with a depth camera along with continuous action vectors (such as the velocities or torques applied to a robot's joints). Supervision for learning is only provided via point-wise associations between consecutive point clouds, no higher level information such as object segmentation is provided to the learner. While a standard deep network architecture can be trained on such data to predict the 3D motion of individual observed points, such a vanilla network architecture is not able to learn and represent an explicit notion of *objects and their motion*, which is very useful for control tasks and for higher-level reasoning.

To overcome this limitation, we introduce SE3-Nets, which learn to segment a scene into "salient" objects and predict the motion of these objects under the effect of applied actions. SE3-Nets represent motion in the environment as a set of $\mathbb{SE}(3)$ transforms[1], which are widely used in robotics, computer vision, and graphics to model rigid body motion. Key

[1] $\mathbb{SE}(3)$ refers to the Special Euclidean Group representing 3D rotations and translations: $\{R, t | R \in \mathbf{SO}(3), t \in \mathbf{R}^3\}$

to SE3-Nets is the notion of disentangling the motion of the objects (the **What?**) from their location in the environment (the **Where?**). SE3-Nets do this by explicitly predicting a set of "k" $\mathbb{SE}(3)$ transforms to encode the motion, and "k" dense pointwise masks that specify the contribution of each $\mathbb{SE}(3)$ towards a 3D point. Finally, the network combines the $\mathbb{SE}(3)$s, their masks, and the 3D input through a differentiable *transform* layer that blends the motions to produce a predicted output point cloud.

In the absence of any constraints, SE3-Nets can represent an arbitrary per-point 3D motion. To bias the network to model rigid motion, we adapt a weight sharpening technique used in [165] and show that this results in a segmentation of the environment into distinct objects along with their rigid motion. We show results on three simulated scenarios where our system predicts the motion of a varying number of rigid objects under the effect of applied forces, and a robot arm with four actuated joints. We present experiments testing different parts of our network and show results highlighting the robustness of SE3-Nets to different types of noise, similar to those found in real world data. We also show the capability of SE3-Nets to operate on real world data collected using the Baxter robot pushing objects on a table.

The main contributions in this chapter are as follows. We introduce SE3-Nets, a deep neural network architecture that models scene dynamics by segmenting the scene into distinct objects and jointly predicting their rigid body ($\mathbb{SE}(3)$) motion. We show that SE3-Nets can learn to do this solely based on sequences of control, raw depth camera data and point-wise associations, without the need for explicit segmentation information in the training data. We also provide substantial experimental evidence that SE3-Nets outperform standard deep learning baselines, and can be applied to real robot data.

This chapter is organized as follows. After discussing related work, we introduce SE3-Nets in Section 2.2, followed by an experimental evaluation in Section 2.3 and discussion.

2.1 Related Work

2.1.1 Robotics

Traditionally, robotics systems are pipelines consisting of modules for state estimation, planning and control [141, 147]. Many of these modules are hand-crafted and require significant knowledge of the specific problem domain. As mentioned before, many planning and optimal control techniques require a "dynamics" model that predicts the effect of actions on the state of the system [153, 152, 168]. Early work on learning dynamics models from data focused on low-dimensional state and control representations [36]. More recent work has looked at learning forward rigid-body dynamics models assuming tracking information is available [91]. In contrast to these methods, Boots et al. [19] learn a model using Predictive State Representations to predict raw depth images given a history of prior images and control. There is also a large body of work investigating how robots can interact with objects for manipulation or object exploration tasks, as summarized in a recent survey [18]. The focus of this area, however, is not on learning predictive motion models from raw data, as we present in this chapter. Recently, deep models have been used for learning dynamics models in robotics and reinforcement learning, by mapping raw pixel images to low-dimensional (latent) encodings on top of which standard optimal control methods are applied [160, 157, 49]. In a similar flavor, SE3-Nets explicitly model the dynamics of the scene as rigid body motion of salient objects, jointly learning object segmentation and motion prediction.

2.1.2 Physics prediction

Recent work in machine learning has looked at the problem of physics prediction, for example to predict the stability of stacked blocks from images using deep networks [100, 102], predicting ball motion from RGB images [54], learning to predict object dynamics in images [113], learning action conditional video predictions [118] and learning contact models from state data [43]. These methods mostly predict low-dimensional outputs like ball velocity [54] or the probability of falling [102]. One exception is the work by Lerer et al. [100], which predicts

images of a tower of blocks, but their network operates on RGB images and has no specific notion of an action or forces.

In particular, work by Agrawal et al. [4] and Finn et al. [47] are closely related, both focusing on modeling the effect of robot actions on a scene, albeit using RGB images. Unlike Agrawal et al. [4], we explicitly predict a dense point cloud through our forward model, encoding the motion using $\mathbb{SE}(3)$ transforms and masks. Similar to Finn et al., [47], we composite motion using predicted masks with two main differences: we use $\mathbb{SE}(3)$ transforms in 3D to capture complex out of plane motions and our masks are sharpened to enforce the rigid-body assumption while improving prediction accuracy.

Additionally, there has been recent work on "neural simulation engines", where the idea is to implement a differentiable physics simulator using neural network layers [14, 26]. Similar to these ideas, our network architecture can be thought of as a physics simulator that explicitly learns to model the observed dynamics as rigid body $\mathbb{SE}(3)$ motions of visible objects in the scene.

2.1.3 Geometry and Deep Learning

Related work in the computer vision literature has looked at the problem of integrating geometric priors with deep networks, specifically for tasks such as predicting 3D rotations between pairs of images [96, 175], scene flow prediction [155, 135] and video prediction [81]. Differing from most of this work, we operate on 3D data, incorporate continuous actions, and explicitly predict rigid body motion and object masks. We describe the differences to a few key related papers below.

Independently in parallel to our work, Handa et al. [66] proposed deep models using rigid body transforms for depth image registration and alignment, but do not model object motion or the effect of actions. Yang et al. [175] proposed a deep recurrent model that predicts the change in an encoded pose vector through the effect of a one-hot action vector to render rotated color images of objects. Differing from their work, we operate on 3D data in complex multi-object settings and real world scenes, use a continuous action vector, and explicitly

predict rigid body motion and object masks.

A related line of work to ours is the idea of attentional mechanisms [62, 7] which focus on parts of the environment related to task performance and the concept of disentangling representations [96, 175], which aim to separate variations in the environment. Our model has a differentiable, dense, point-wise attender that learns to focus on parts of the environment where motion occurs, using these cues to segment objects. Also central to our model is the idea of disentangling the motion of the object from its location.

Finally, the Spatial Transformer Network (STN) architecture [81] is quite related to our approach - the STN is a differentiable transformation module that allows for affine transformations of inputs or intermediate feature representations. This is achieved through the use of a localization module that predicts the affine transform to be applied to the data, followed by a grid generator and a bilinear sampler that apply the transformation to the data to generate the output. Spatial Transformers can be added anywhere within the network, allowing the system to be robust to affine transformations of the input data and noise to a certain extent. Our network architecture SE3-NETScan be thought of as an extension of the STN architecture with a few key differences – unlike STNs which operate on 2D grid or 3D voxel data (which can be extremely costly to explicitly represent in 3D), we apply rigid $\mathbb{SE}(3)$ transforms to an ordered input 3D point cloud (from a 2.5D depth camera). Also, we learn explicit segmentation masks which attend to different objects in the scene as opposed to the explicit grid generator which cannot easily deform to the shape of the objects in the scene.

2.2 Network Architecture

Given a 3D point cloud (X) from a depth sensor and an n-dimensional continuous action u as input, SE3-NETS model the scene dynamics as rigid body motions of the constituent objects to generate a transformed point cloud Y:

$$Y = f(X, u) \mid f := \{R_i, t_i, M_i\}, i = 1 \dots k \tag{2.1}$$

In essence, SE3-NETS decompose the scene into k rigid objects, predicting per object a

mask M that attends to parts of the scene containing the object and a rigid body transform $[R, t] \in \mathbb{SE}(3)$ that quantifies the object's motion. Note that in our setting, k is a pre-specified network parameter that limits the number of distinctly moving objects or parts (including background which has no motion).

Fig. 2.1 shows the general architecture of SE3-Nets . There are three major components: an encoder that generates a joint latent state given the input point cloud X and the control u, a decoder that predicts the object masks with the corresponding transforms and a final transformation layer that generates the transformed point cloud Y.

2.2.1 Encoder

The encoder has two parts: a convolutional encoder which generates a latent state from the point cloud X (represented as a 3-channel image) and a fully connected network that encodes the control vector u. We adopt a late fusion architecture and concatenate the outputs of these two parts to produce the final encoding, which is used by the decoder for further predictions.

2.2.2 Decoder

The decoder decomposes the motion prediction problem into two sub-problems: identifying and grouping together points that move together (we call this grouping an "object" or a "motion-class") and subsequently predicting the $\mathbb{SE}(3)$ transformation parameters that quantify the object's motion.

Predicting motion masks: The mask decoder attends to parts of the scene that exhibit motion, grouping points that move together to form objects. As an example, all points belonging to a rigid object can be grouped together as they move with it. Presupposing that the scene has k distinctly moving objects, we can formulate this as a k-class labeling problem where each input point can belong to one of the k motion-classes. Unfortunately, this formulation is non-differentiable due to the discreteness of the labeling. Instead, we relax it to allow each point to belong to multiple motion classes, quantified by a per-point probability distribution M^j over the k motion-classes:

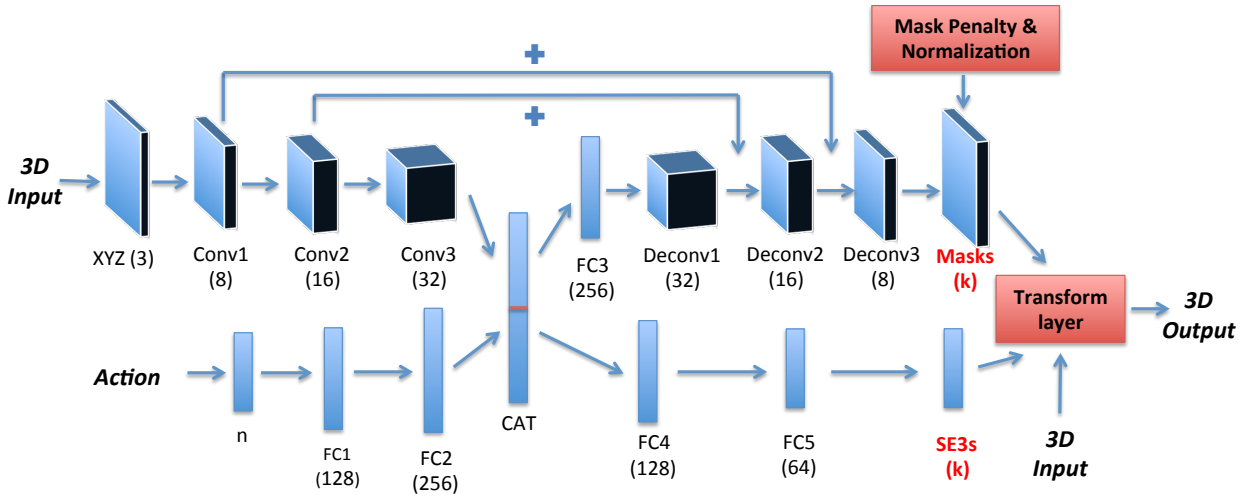

Figure 2.1: SE3-NET architecture. The inputs are a 3D point cloud and an n-dimensional action vector (bold-italics), both of which are encoded and concatenated to a joint feature vector (CAT). The decoder uses this encoding to predict "k" object masks M and "k" $\mathbb{SE}(3)$ transforms which are used to transform the input cloud via the **Transform layer** to generate the output. This layer has no trainable parameters. Mask weights are sharpened and normalized via weight sharpening before use for prediction. Conv = Convolution, FC = Fully Connected, Deconv = Deconvolution, CAT = Concatenation. Each Conv, FC and Deconv layer (except the last) is followed by a PReLU activation function. We use batch normalization after the activations of the Conv and Deconv layers.

$$M^j = \{m_{1j}, m_{2j}, \ldots, m_{kj}\} \mid \sum_{i=1}^{k} m_{ij} = 1; \tag{2.2}$$

allowing each point j to smoothly interpolate between multiple motions.

We use a de-convolutional architecture to compute dense object masks, generating k masks at the input resolution. Following recent work [105], we use a skip-add architecture wherein we add the convolutional layer outputs to the de-convolutional layer inputs. This gives us sharper reconstructions of object shapes and contours, improving overall performance.

Predicting rigid transforms: As mentioned before, we represent motion using 3D rigid body transforms. A 3D rigid body transform $[R, t] \in \mathbb{SE}(3)$ can be specified by a rotation $R \in \mathbf{SO(3)}$ and a translation $t \in \mathbf{R}^3$. A 3D point x affected by this transformation moves to: $x' = Rx + t$. We represent rotations using a 3-parameter axis-angle transform $a \in \mathbf{R}^3$, with $||a||_2 = \theta$, the magnitude of rotation. The $\mathbb{SE}(3)$ decoder predicts k $\mathbb{SE}(3)$ transforms, one for each of the k motion-classes (including background). We use a fully connected network to predict these transforms.

2.2.3 Transform layer

Given the predicted $\mathbb{SE}(3)$ transforms and masks, the transform layer produces a blended output point cloud from the input points:

$$y_j = \sum_{i=1}^{k} m_{ij} \left(R_i x_j + t_i\right) \tag{2.3}$$

where y_j is the 3D output point corresponding to input point x_j. Eqn. 2.3 computes a convex combination of the transformed input points, transformed by each of the k $\mathbb{SE}(3)$ transforms with weights given by the object mask. As a consequence of our relaxation for the mask (Eqn. 2.2), the effective transform on a given point is generally not in $\mathbb{SE}(3)$ as Eqn. 2.3 blends in 3D space rather than in the space of $\mathbb{SE}(3)$ transforms. On the other hand, we now have the flexibility to represent both rigid and non-rigid motions through a combination of the transforms and the oject masks. Additionally, we avoid the potential

singularities that can arise from blending in $\mathbb{SE}(3)$ space. In spite of the advantages, using the current framework to model rigid motion without any explicit regularization can lead to over-fitting and blurry predictions. To avoid this, we encourage the network to predict rigid motions through a form of regularization on the object mask.

Enforcing Rigidity: A simple way to restrict the network to predict rigid motions is to force the per-point mask probability vector M^j to make a binary decision over the k predicted transforms. As mentioned before, a naive formulation can lead to non-differentiability. Instead, we smoothly encourage the mask weights towards a binary decision using an approach known as weight sharpening [165]:

$$\begin{aligned} \forall i = 1 \dots k,\ {m_{ij}}' &= (m_{ij} + \epsilon)^{\gamma}; \quad \epsilon \sim \mathcal{N}(0, \sigma^2) \\ {m_{ij}}'' &= \frac{{m_{ij}}'}{\sum_k {m_{kj}}'} \end{aligned} \tag{2.4}$$

where the noise ϵ is sampled from a Gaussian. Typically, we set $\sigma^2 = 0$ and $\gamma = 1$ at the start of training and slowly increase them as the number of training iterations increase (usually, γ increases by 1 every 500 training iterations till it reaches a maximum of 100 and σ increases by 0.01 every 1000 iterations to a maximum of 0.1). In practice, the combination of the noise and growing exponent forces the network to push its decisions apart, resulting in nearly binary distributions at the end of training. Finally, at test time, the network segments the input point cloud X into k distinct objects, predicts their motion and applies a rigid transform to each input point (Eqn. 2.3), to generate the transformed output point cloud Y.

2.3 Evaluation

We evaluate SE3-Nets on how well they predict motion on multiple simulated tasks using the Gazebo physics simulator and real world data of a Baxter robot poking at objects on a table. We first present results on simulated data, followed by tests on the robustness of the network to different types of noise and hyper-parameter choices and finally discuss results on real world data.

2.3.1 Simulated data collection

We set up four simulated tasks using the Gazebo physics simulator, consisting of scenes where a fixed camera looks at rigid bodies moving under the effect of applied forces. In all our settings, the network takes a 3D point cloud, and an n dimensional continuous control vector as input and predicts the resulting point cloud 0.15 seconds in the future. We assume that the control is held fixed for this duration. Note that our input point cloud is fed to the network as a 3-channel image ($3\times240\times320$ resolution) with the X, Y and Z values for a given point concatenated along the channels axis, similar to feeding in RGB images to deep networks. We detail the data collection next.

Single Box: This dataset has 9000 random scenes where a ball collides with a box placed at a random position on a table. In each scene, we place the ball at a random location in front of the box and continuously apply a randomly chosen constant force to the ball, directing it to collide with the box. For a given scene, we run the simulation for one second, record data and discard frames where the box falls off the table. Across scenes, we vary the start pose of the objects and the applied force while keeping the objects' size and mass constant. We also vary the table size to introduce background variations. Across all scenes, we have a total of $\sim$170,000 examples (each example consists of an input control and input & target point clouds). The control vector u is 10-dimensional, consisting of the ball pose (position & quaternion) and the applied force ($n = 10$). We add the ball's orientation to the control to model cases where the ball undergoes spin.

Multiple Boxes: To test the generalization of the system to different object sizes, masses and number of objects, we generated a second dataset that varies all three at random. Each scene has anywhere from 1-3 objects of varied sizes and proportional mass, where the ball collides with a randomly chosen box. We only consider examples where a single box and the ball are in collision and discard those that involve multiple collisions as it is hard for the system to model such motion without any temporal information. This dataset has $\sim$12,000 different scenes, with a total of $\sim$210,000 examples. The controls are represented the same as

the Single Box dataset ($n = 10$).

Baxter: Our third dataset consists of sequences of depth images looking at a Baxter robot being controlled to move its right arm randomly. In each scene, we apply a constant, randomly chosen velocity to 1-4 randomly chosen joints on the robot's right arm. Each scene lasts for one second, after which we bring the arm to a rest. We randomly reset the pose of the arm once every 20 scenes. In total, this dataset has ∼11,000 scenes with ∼220,000 examples. The controls for this task are the commanded joint velocities, a 14-dimensional vector in which 1-4 values are non-zero ($n = 14$).

Household Objects: The final simulated dataset tests the generalization of the system to irregular object shapes. We use 11 household objects from the LineMOD dataset [74]. Each scene has 1-4 objects randomly placed on a table, with the ball colliding against a randomly chosen object. In total, we have 10,000 random scenes with ∼200,000 examples. The controls are similar to the box datasets ($n = 10$). This dataset is particularly challenging as it has significant amounts of toppling and fast rotations due to the objects' irregular shapes.

2.3.2 Training

We implemented our system using the deep learning package Torch [35]. We trained our networks using the ADAM optimization method [88] along with Google's Batch Normalization technique [80] to speed up training. At the start of training, we initialize the layer predicting $\mathbb{SE}(3)$ transforms to predict the identity transform which we found to improve convergence. We initially set the weight sharpening penalty to zero and slowly ramp up the noise parameter σ and the exponent γ till they reach a preset maximum. The number of objects k is chosen apriori: $k = 5$ for the Baxter dataset (4 joints + background) and $k = 3$ for all other datasets (ball + object + background).

Comparison: We train three variants of SE3-Nets and compare against two baseline networks that predict 3D scene flow as well as a baseline that always predicts zero motion:

- *Ours:* SE3-Net from Fig. 2.1, with 3-Conv/Deconv layers and a total of ∼1.4 million

parameters.

- *Ours (Large):* Bigger version of the SE3-NET with 5-Conv/Deconv layers and ∼6x as many parameters.

- *No Penalty:* SE3-NET without any weight sharpening to enforce binarization of the object masks.

- *Flow:* Network trained to predict dense 3D optical flow directly using a Conv/Deconv architecture similar to the original SE3-NET, but without the $\mathbb{SE}(3)$ prediction module and the transform layer. As a consequence, this network learns pixel motions as opposed to the underlying structure in the scene – object motion.

- *Flow (Large):* Bigger version of the flow network with 5-Conv/Deconv layers and ∼8x as many parameters as the original SE3-NET.

- *No Motion:* Baseline that always predicts zero motion.

Both the small networks use strided convolution with no pooling, while the large networks use max pooling instead of striding. All networks use the Parametric-ReLU non-linearity. We trained the networks for 50k iterations on the single box and Baxter datasets and for 75k iterations on the other two datasets with a 70:30 train/test split. Depending on the network size, training takes anywhere between a few hours to half a day on an NVIDIA Titan X GPU.

Training targets: Given an input point cloud X and control u, SE3-NETS predict the output point cloud Y by transforming the input points (Eqn. 2.3). We would like these predictions to match the ground truth targets Y', which specify the true position of our input points at a future time. In order to compute these targets though, we need to track and associate a given input point x_j across multiple depth images. In this work, we assume that these associations are given to us at training time, either by the physics simulator or an object tracker (in case of real data).

Task	Ours	Ours (Large)	No Penalty	Flow	Flow (Large)	No Motion
Single Box	3.73	**1.65 ± 0.17**	4.39	10.1	2.48 ± 0.22	23.24
Multiple Boxes	3.22	**1.29 ± 0.14**	2.83	6.2	1.69 ± 0.17	21.84
Baxter	0.074	**0.057 ± 0.002**	0.074	0.11	0.063 ± 0.001	0.33

Table 2.1: Average per-point flow MSE (cm) across tasks and networks. Our (Large) network achieves the best flow error compared to baselines even though it is not directly trained to predict flow. "No Motion" results quantify the average magnitude of motion in the datasets (>20 cm for box datasets, < 1 cm for Baxter datasets).

Evaluation Metric: We report Mean Squared Error (MSE) between the predicted 3D scene flow (computed as the difference between the input and the predicted point clouds) and ground truth, averaged across points with non-zero ground truth flow. This metric takes into account errors in both the mask and $\mathbb{SE}(3)$ predictions, and is also the loss function used to train the flow networks.

2.3.3 Results on simulated data

Table 2.1 reports test results on the first three simulated datasets. Our networks significantly outperform their counterpart small and large flow networks on all tasks. We also did a 5-fold cross validation for the large networks and found that our improvements over the flow baselines are statistically significant. Our networks also achieve a large reduction in prediction error compared to the zero motion baseline (Table 2.1, last column), even for very large motions (> 20cm per point). In practice, we've seen similar performance on shorter horizons (0.06 sec) with smaller, subtle motions.

Fig. 2.2 shows some representative predictions made by the networks on three simulated datasets. These results highlight yet another advantage of our approach as compared to a naive flow baseline - the consistency and sharpness of our predictions. By segmenting the scene into distinct objects (Fig. 2.3) and predicting individual $\mathbb{SE}(3)$ transforms, our network

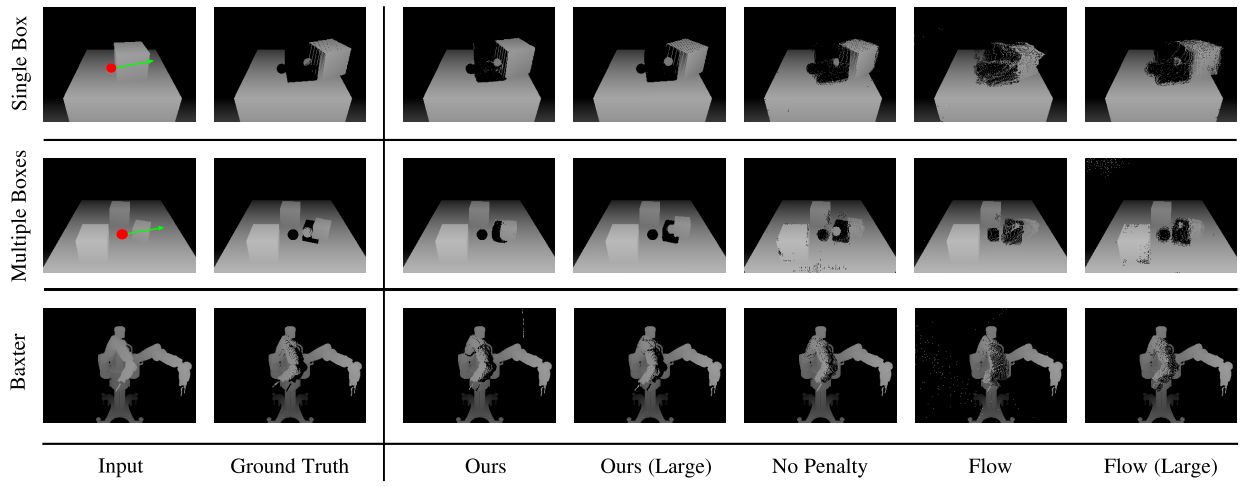

Figure 2.2: Prediction results for three simulated datasets. All images (except first column on the left) were rendered by projecting the predicted 3D point cloud to 2D using the camera parameters and rounded off to the nearest pixel without any interpolation. (From left to right) Input point cloud with the ball highlighted in red and applied force shown in green; ground truth; predictions generated by the different networks. 3D point clouds for the flow networks were computed by adding the predicted flow to the input. The black regions in the images correspond to parts that were occluded in the input and later became visible due to the objects' motion (none of the networks can fill in missing data). Image best viewed in high resolution. For a better understanding of the results, please refer to the supplementary video on our webpage.

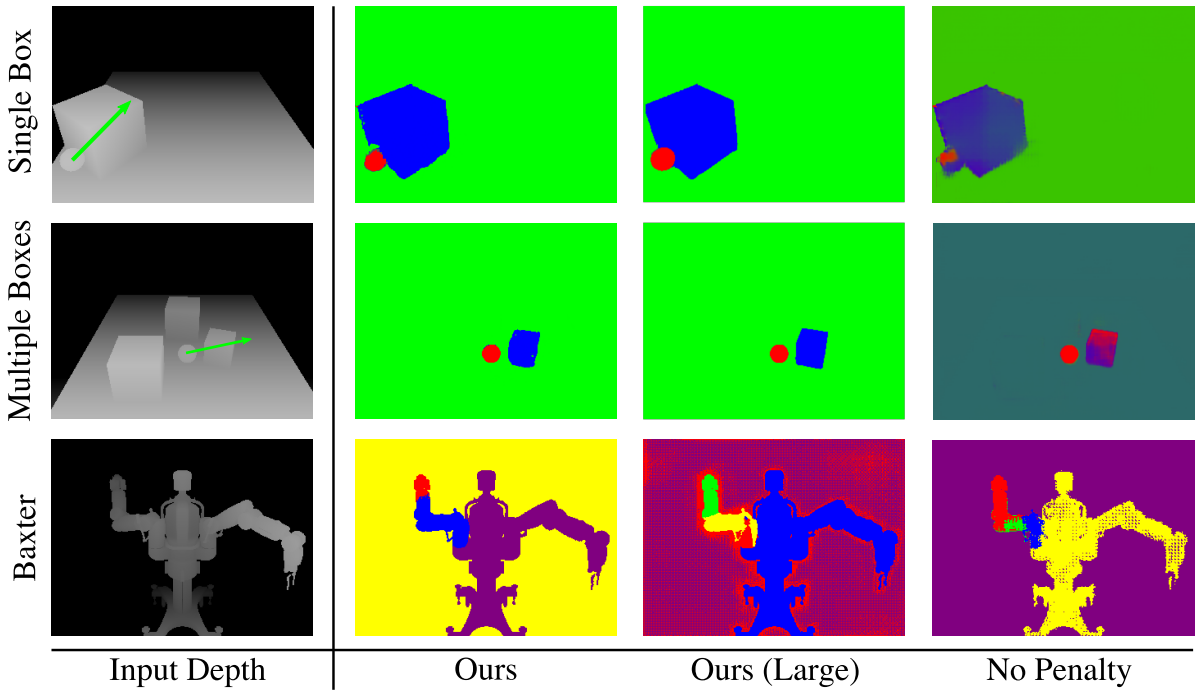

Figure 2.3: Object masks predicted by our networks. Box masks (k=3) are rendered directly as RGB images while Baxter masks (k=5) are colored based on an arg-max operation across the k-mask channels. SE3-NET predictions (middle two columns) are near-binary as seen by the distinct coloring for each distinctly moving object (eg: blue for the box, red for the ball). **No Penalty** masks (right column) have mixed coloring across the scene (eg: indigo for the box), indicating that the masks are not binary. Image best viewed in color.

ensures that points which belong to an object rigidly move together in an interpretable manner. This results in a sharp prediction with very little noise, as compared to the flow networks which do not have any such constraints. With increasing layer depth, the flow networks can somewhat compensate for this, but there is still a significant amount of noise in the predictions resulting in smearing across the canvas (Fig. 2.2). Surprisingly, we noticed that the flow networks perform quite poorly on examples where only a few points move such as when just the ball moves in the scene while our networks are able to predict the ball's motion quite accurately.

We also present mask predictions made by our networks in Fig. 2.3. The colors indicate that the masks predicted by our networks for the box datasets are near binary (we render the 3-channel masks directly as RGB images). Our network successfully segments the ball and box as distinct objects without any explicit supervision. In practice, we found that it is crucial to give the network examples where the ball moves independently as this provides implicit knowledge that the ball and the box are distinct objects. In cases where the training examples always have the ball in contact with the box, the network had a hard time separating the objects, often masking them out together. For the Baxter dataset, depending on the motion, our network usually segments the arm into 2-3 distinct parts (Fig. 2.3), often with a split at the elbow.

In comparison, the "No Penalty" SE3-NET rarely predicts binary masks, often blending across different $\mathbb{SE}(3)$s (rightmost column, Fig. 2.3). While this leads to some overfit on datasets with large motion such as the Single Box dataset (4.4 cm MSE compared to 3.7 cm for the small SE3-NET$\tilde{)}$, it still performs quite well on the other datasets. Interestingly, the "No Penalty" networks significantly outperform the smaller flow networks, hinting that our mask/$\mathbb{SE}(3)$ decomposition structure (even without the rigidity penalty) better models the underlying problem structure for dense motion prediction.

Fig. 2.4 shows two representative results from testing on the household objects dataset where the network has to deal with complex shaped objects, some with holes. As it is clear, the network can model the dynamics of these objects well, with the resulting predictions being significantly sharper compared to the large flow network, which does very poorly. We have also seen that our network is able to gracefully handle cases where objects topple or undergo large motions.

Finally, we test the consistency of our network in modeling sequential data by allowing the network to predict scene dynamics multiple steps in the future. Fig. 2.5 shows these results for a Baxter sequence where we feed the network's predictions back in as input and fix the control vector for 5 steps into the future. We compare against ground truth and see that our predictions remain consistent across time without significant noise addition. In comparison,

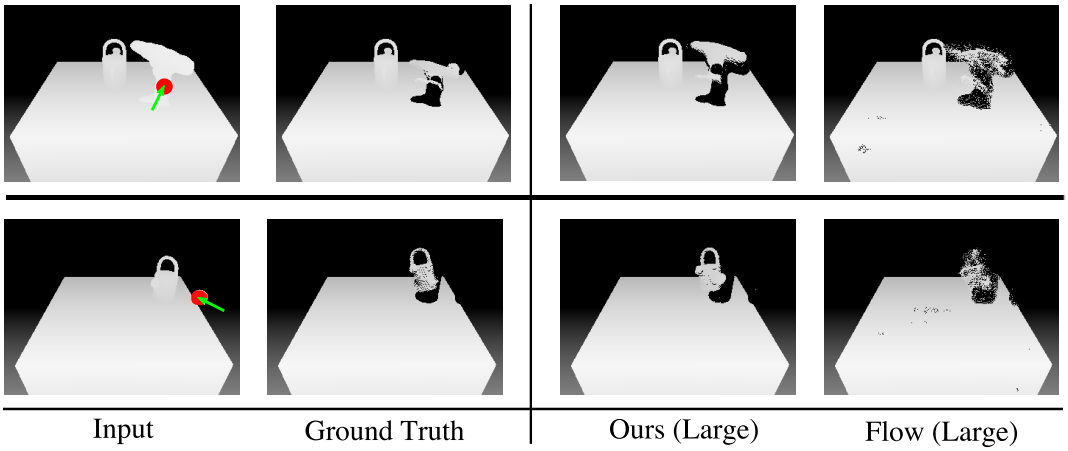

Figure 2.4: Two results from the "Household Objects" dataset, showing topping and sliding motion. Our network segments the objects correctly and predicts consistent motion leading to sharp images while the flow baseline smears the object across the image. Ball highlighted in red and the applied force is shown as a green arrow.

the predictions from the large flow network degrade over time as the noise cascades. Similar behavior can be observed when looking at the quantitative prediction errors for a sequence on the Baxter dataset (see Fig. 2.6). To get a better understanding of our results, we encourage the readers to look at the video on our webpage, where we show prediction results for many sequences.

2.3.4 Robustness

We perform a few additional experiments to test the robustness of our networks to noisy data.

Robustness to depth noise: To test whether our network is capable of handling the types of noise seen in real depth sensors, we trained networks under two types of depth noise: First, we added gaussian noise with a standard deviation (SD) of 0.75 cm, and scaled the noise by the depth (farther points get more noise) as is common in commodity depth sensors. Second, we increased the noise SD to 1.5cm without scaling by the depth. Table 2.2 shows the performance of the two large networks under both types of noise: while our performance

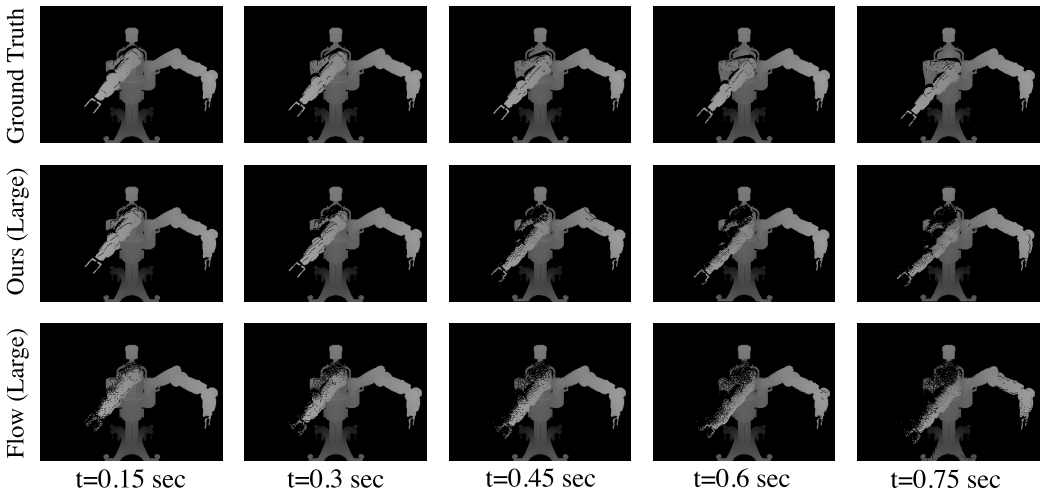

Figure 2.5: Multi-step prediction results obtained by feeding back the output of the network as the input for four consecutive times. At the end of each frame, the predicted SE3-Net flow is added to the input point cloud to generate the next input point cloud. Note that we do not back-project the point cloud on the image plane to generate the new input. Ground truth is reset at the end of each frame.

Task	Depth Noise (SD = Noise standard deviation)			
	SD = 0.75 cm		SD = 1.5cm, No depth scaling	
	Ours (Large)	Flow (Large)	Ours (Large)	Flow(Large)
Single Box	**2.61**	6.87	**2.70**	4.31
Multiple Boxes	**1.95**	6.10	**3.56**	4.42
Baxter	0.073	**0.066**	**0.44**	0.63

Table 2.2: Average per-point flow MSE (cm) for networks trained with noise added to the input depth (which in turn translates to a noise in the point cloud). Our networks are structured which allows them to be robust to noise additions; their performance degrades gracefully with increasing noise as compared to large errors for the flow baseline.

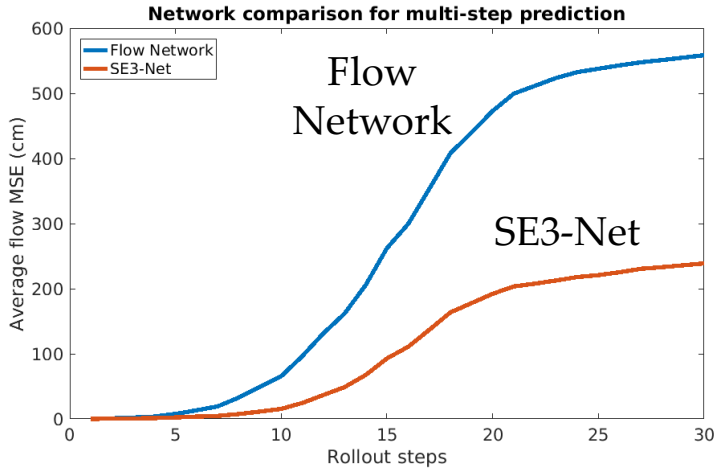

Figure 2.6: Multi-step prediction error for a single sequence from the Baxter dataset, showing the average flow MSE (cm) against the open-loop rollout step. SE3-NETS generate significantly better predictions and far lower prediction errors in the long term compared to the baseline flow networks, even though they are trained only on single step predictions.

Task	Data Association Noise			
	9×9, threshold = ±10cm		15×15, threshold = ±20 cm	
	Ours (Large)	Flow (Large)	Ours (Large)	Flow(Large)
Single Box	**1.79**	3.15	**2.80**	5.32
Multiple Boxes	**1.05**	1.95	**2.48**	4.26
Baxter	**0.10**	0.15	**0.30**	0.43

Table 2.3: Average per-point flow MSE (cm) for networks trained with noise added to the data associations used for generating the flow targets used in training the networks. The error is measured w.r.t the true data associations (i.e. the ground truth flow without any noise). Once again, the structure inherent in our networks allow them to be robust to noise; their performance degrades gracefully with increasing noise as compared to large errors for the flow baseline.

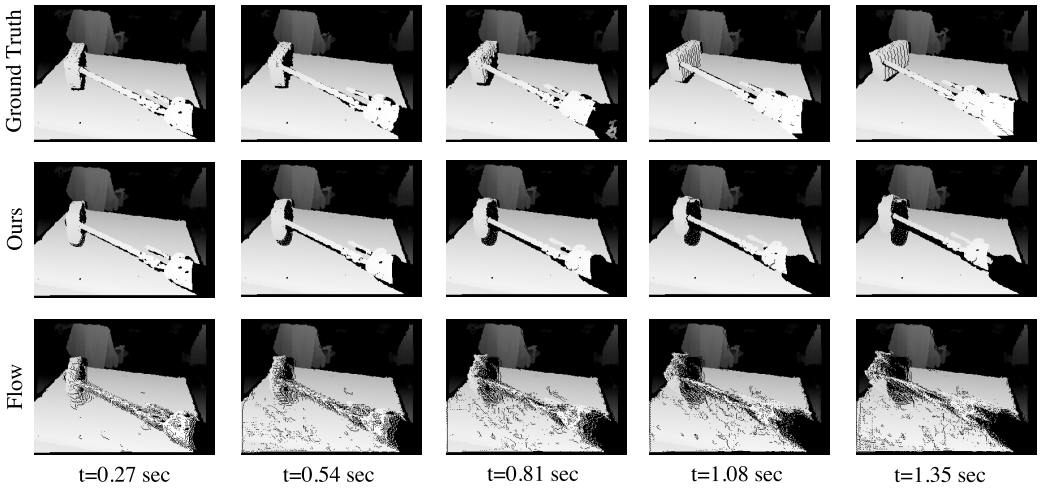

Figure 2.7: Multi-step prediction results on real world data collected by poking objects with the Baxter robot. Our network predictions are sharper than the flow baselines highlighting the consistency of the learned segmentation and transforms.

degrades, we significantly outperform the baseline flow network. Additionally, our network is still able to segment the objects properly in most of our tests.

Robustness to noise in data association: We test how well our networks respond to uncertainty in data association by allowing spurious ground-truth associations when computing our training targets. We allow each point to be randomly associated to any other point in a $m \times m$ window around it, as long as their depth differences are no larger than a threshold. We train in two increasingly noisy settings: associating in a 9×9 window with a threshold of ± 10 cm and in a 15×15 window with a threshold of $\pm$ 20 cm. Table 2.3 shows the results of these tests. Our network strongly outperforms the flow baseline, with errors almost half of the flow baseline. While this test does not simulate a systematic association bias, it still shows that our network is robust to uncertain associations.

We believe that the strong structural constraints inherent in our network allow it to average over noise, leading to robust predictions even in highly noisy settings.

2.3.5 Hyperparameter choices

We also performed tests to evaluate the sensitivity of our model to various hyperparameter choices. We detail a few below:

Number of objects: In all prior experiments, we have chosen the number of predicted $\mathbb{SE}(3)$s (k) apriori with our knowledge of the datasets. To test the sensitivity of our algorithm to this parameter, we trained our networks setting k to a large number ($k = 8$ for the Baxter dataset and $k = 6$ for the rest). In most cases, we found that the network automatically segments the scene into the correct number of objects, with the remaining mask channels assigned to identity. We also saw little to no performance drop in these experiments.

Prediction horizon: To test the robustness of our network to the prediction horizon (0.15 seconds so far), we tested our networks on two smaller horizons (0.06 & 0.03 seconds). We saw similar performance for both these cases, with slightly poor results on the smallest horizon (0.03 seconds) due to difficulty in disambiguating distinct objects from small motions.

Initialization: As mentioned before, we initialize the $\mathbb{SE}(3)$ decoder to predict identity at the start of training. When relaxing this by doing a random initialization, we saw only a small drop in performance, with an increase in the convergence time.

Rotation representation: We also evaluated the following $\mathbb{SE}(3)$ rotation representations in addition to the 3-parameter Axis-Angle transform used in all our experiments: 1) A 4-parameter quaternion representation where the network predicts 4 values that are normalized to generate a unit quaternion, 2) A 3-parameter XYZ-Euler angle representation (in radians), and 3) A 3-parameter stereographic projection of the quaternion rotation representation (see [146] for more details). In practice, we found that the Axis-Angle representation outperformed all other rotation representations; the 4-parameter quaternion representation was second best and others performed significantly worse across all our experiments.

$\mathbb{SE}(3)$ **parameterization:** We evaluated a variant of the $\mathbb{SE}(3)$ parameterization presented so far wherein we add a pivot point p that captures the center of rotation; the resulting $\mathbb{SE}(3)$ is 9-dimensional, the network predicts the translation, rotation and pivot position.

With this parameterization, the transform operation in (2.3), now becomes:

$$y_j = \sum_{i=1}^{k} m_{ij} \left(R_i(x_j - p_i) + p_i + t_i \right) \tag{2.5}$$

where p_i is the predicted pivot position for the ith transform. Decomposing the transformation with a pivot allows additional degree of freedoms to specify centers of rotation other than the camera center (e.g. the center of the object). In practice though, we found that this parameterization has no advantages compared to the standard 6-parameter $\mathbb{SE}(3)$; empirically they perform about the same.

2.3.6 Results on real data

Finally, we present results from a preliminary evaluation of SE3-Nets on real world data obtained with the Baxter robot interacting with three objects in a tabletop scene (a cheezeit box, mustard bottle and pringles can), poking them with a stick attached to it's end-effector. At the start of each poking action, we randomly place a single object on the table. Similar to [5], we choose a random direction to poke in, while keeping the end-effector level and at a constant orientation. Each poking action lasts around 5-7 seconds during which we record point clouds from a depth camera mounted on the torso of the robot along with joint encoder data. As a preliminary effort, we collected 67 poke actions with mostly sliding and rotational object motion for a total of 7700 examples. We use the DART motion tracker [132] to generate ground truth data associations for training.

We trained the small SE3-Net and flow networks to predict a frame 0.27 seconds in the future to allow for large motions. We use the commanded joint angles and velocities as the control ($n = 14$). Due to the limited quantity of data, we make a modification to speed up training: we provide the SE3-Net with the ground truth mask of the robot arm. While this makes the problem easier, the network still has to segment the object and jointly predict the arm and the object's motion.

Fig. 2.7 shows a representative sequence from testing on a held out set of pokes - we cascade our predictions forward for more than one second. While the errors increase over

time, our network is able to segment and predict consistent motion for the object and the arm. In comparison, the flow network performs very poorly (though it does have a harder problem as the ground truth arm label is not fed to it). Once again, we suggest the reader to look at the supplementary video on our webpage for more results. Overall, we believe that this is a strong proof of concept showing: 1) we can easily generate training data needed for our networks and 2) SE3-Nets are able to learn scene dynamics from limited real world data. In the future, we plan to collect more data and train larger networks to handle complicated dynamics such as toppling and falling.

2.4 Discussion and Future Directions

Learning "intuitive" models of the physical world from raw data is a promising alternative to explicitly designed physics-based models. This is due to the fact that models learned from data are tightly coupled to perception and thus well suited for closed-loop control. Toward this long term goal, we introduced SE3-Nets, a deep learning model that predicts changes in an environment based on applied actions, parameterized as a series of rigid transforms applied to 3D points in the environment. SE3-Nets selectively learn to focus on parts of the scene where motion occurs, segmenting the scene into objects and predicting $\mathbb{SE}(3)$ motions for each distinct object. We showed that this separation works well in practice and results in strong performance on four simulated and one real robot task with multiple rigid bodies in motion. SE3-Nets are able to generalize across different scenes and produce results that are very consistent with the observed rigid motion, as compared to traditional flow networks.

There are several promising directions for future work. First, while our experiments indicate that SE3-Nets can learn predictive models from real data, we are confident that a larger data collection effort would enable us to train networks that generalize across many types of objects and scenes. A key area for improvement is in learning data associations, which we currently provide as part of the training setup. A first step towards this would be to formulate a loss function based on Iterative Closest Point matching, which is able to align close-by depth data. Another exciting direction is to use the "No Penalty" version of

SE3-Nets to model non-rigid motion, exploring the use of strong regularization and locality priors in the masks [116] to improve efficiency and generalization.

A limitation of the current SE3-Net architecture is its inability to integrate information across time as the model reasons only over single-steps as opposed to sequences of data. One potential way to add this capability is to extend the SE3-Net architecture to include recurrent layers (such as LSTM [76], GRU [29]) right after the bottleneck concatenation layer (denoted CAT in Fig. 2.1) to capture temporal correlations in the data. Initial tests showed us that using two separate recurrent layers at the start of the mask and $\mathbb{SE}(3)$ prediction decoders worked better than a single recurrent layer at the bottleneck; this allows additional flexibility to model temporal correlations in mask and $\mathbb{SE}(3)$ predictions independently. Exploring the use of this recurrent multi-step SE3-Net architecture for sequential prediction is an interesting area of future work.

Finally, we would like to use the learned SE3-Net for control. We address this question in Chapter 3 where we discuss a potential limitation of the SE3-Net architecture when applied to the task of visuomotor control. We propose an extension to the SE3-Net architecture that fixes this problem and present both simulated and real-world results using the proposed model for fast, real-time visuomotor control.

Chapter 3

SE3-POSE-NETS

In the previous chapter, we looked at modeling the visual dynamics of a robot's environment based on actions applied by the robot, e.g. pushing an object. Our model, SE3-Nets, reasons about objects in the environment and their motion, resulting in a structured predictive model that performs well across multiple scenarios. While predicting the effect of a robot's actions is desirable for many reasons (see Chapter 4 for a discussion on this), it is in of its own, not an end goal in manipulation – instead we would like our robot to be able to execute actions and behaviors in order to achieve desired goals, such as move its arm to a target configuration or manipulate objects in its environment.

In this chapter, we focus on this control problem. Specifically, we look at the visuomotor control setting where we are receiving observations of a scene from a camera and we would like to control our robot to reach a target, specified as a visual observation (i.e. RGB/D image). Note that this is a very general formulation of a manipulation problem; for example, we can specify a task of cleaning with a target image of a cleaned surface, for a block stacking task we can specify the task by providing the image of the stacked blocks and so on.

Much work has been done in the robotics literature on the visuomotor control problem [39, 160], and also on the related problem of visual servoing [78]. Most of these approaches decompose this problem into two parts: data-associating the current scene to the target and modeling the effect of applied actions to changes to the scene, combining these in a tight loop to servo to the target. Data association is usually handled through the use of features (e.g. SIFT [106], SURF [16]) or via external tracking systems [132], while the visual predictive model that models environment dynamics based on applied actions, may be analytically determined [78], or can be learned directly from data [23, 49].

Recent work on deep learning has looked at learning visual predictive models directly in the space of observations, relating changes in pixels or 3D points to the applied actions [19, 47, 23]. Given a target scene, we can use this predictive model to generate suitable controls to visually servo to the target using model-predictive control [49, 160]. As described above though, for this pipeline to work, we need an external system (such as [132, 6] or a feature matching approach) capable of providing long range data-associations to measure progress.

As we showed in the previous chapter, instead of reasoning about raw pixels, we can predict scene dynamics by decomposing the scene into objects and predicting object dynamics instead (see Sec. 2.2). While this significantly improves prediction results, it still does not provide a clear solution to the data association problem that we encounter during control - we still lack the capability to explicitly associate objects/parts across scenes. We observe three key points: 1) We can data-associate across scenes by learning to predict a "pose" representation of detected objects/parts in the scene (the pose implicitly provides tracking), 2) We can model the dynamics of an object directly in the learned low-dimensional pose space, and 3) We can predict scene dynamics by combining the dynamics predictions of each detected part.

We combine these ideas in this chapter to propose SE3-Pose-Nets, an extension of the SE3-Net architecture that learns to data-associate across long term sequences and can be used for efficient visuomotor control. Our network detects parts of the scene by generating a semantic segmentation mask and models data associations by embedding these parts into a latent pose embedding that is globally consistent across long sequences. By modeling dynamics in this latent pose embedding, we can perform efficient control in a low-dimensional space rather than in the raw observation space. While our approach is general and can be applied for any manipulation task, in this work we present results on real-time reactive control of a Baxter manipulator directly from raw depth images.

We make the following contributions:

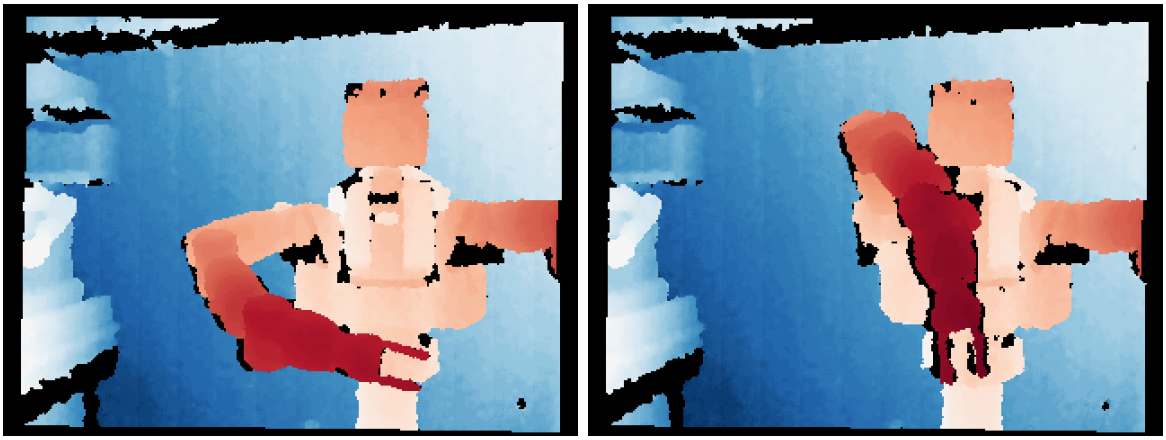

Figure 3.1: An example scenario showing the initial (left) and target depth clouds (right). SE3-Pose-Nets can be used to control the robot to reach the target state based on raw depth (and optionally, color) data. Depth images colorized for display purposes only.

- We show how to learn predictive models that detect parts of the scene and jointly learn a consistent pose representation for these parts with minimal supervision.

- We demonstrate how a deep predictive model can be used for reactive visuomotor control using simple gradient backpropagation and a more sophisticated Gauss-Newton optimization, reminiscent of approaches in inverse kinematics [22].

- We present results on real-time reactive control of a Baxter arm using raw depth/color images and velocity control, both in simulation and on real data.

- Finally, we present results on testing the robustness and generalization of the real-world controller under various changes to imaging conditions.

Fig 3.1 shows an example scenario where our proposed method can be applied to control the robot to reach the target state (right) from the initial state (left).

3.1 Related Work

Visual Servoing & Visuomotor control: There have been multiple approaches to the traditional problem of "visual servoing" over the years [78, 42, 27]. These approaches decompose this problem into two parts: data-associating the current scene to the target (usually through the use of pre-specified image features such as SIFT [106], SURF [16] etc.) and modeling the effect of applied actions to changes to these features, combining these in a tight loop to servo to the target.

Recently, there has been a renewed interest in this area, primarily due to the advent of deep learning for learning good feature representations [160, 157, 101, 5, 49, 82]. These methods can be broadly classed into those that directly regress to controls from visual data [101], generate controls by planning on learned forward dynamics models [49, 160, 157], through inverse dynamics models [5] or by reinforcement learning [82, 98]. We look at a few of these in more detail below:

End-to-End learning: Levine et al. [101] proposed one of the earliest approaches towards visuomotor control using deep neural networks - an end-to-end policy learning approach that maps RGB images and proprioceptive information directly to torques applied to a PR2 robot's joints. The key idea of the approach was to convert the deep network policy learning problem into one of supervised learning where the supervisory signal was provided through a trajectory centric model-based reinforcement learning algorithm – this approach was termed "Guided Policy Search". By alternating these two algorithms in tandem, the approach was able to learn complex policies in contact-rich environments directly on the real robot with minimal supervision. An added advantage of this method was that it needed full state information during training but could operate with partial information at test time as the policy could be trained directly on raw images without explicit knowledge of object poses. The approach was tested on tasks such as cube insertion, screwing a bottle cap, mounting a hangar etc and showed impressive results while being trained with a fairly small amount of real-world data. While this approach is general and requires very little domain knowledge, a

key drawback of such end-to-end learning methods is that they do not generalize well and have to be explicitly re-trained for every single task. Unlike this method, our approach uses deep learning for state estimation and forward dynamics prediction and combines tools from optimal control for generating plans to solve tasks [24] – while this has the drawback of having to plan at test time, we hope to gain improved generalization performance and robustness to noise and changes to input conditions.

Planning with forward dynamics models: As opposed to end-to-end policy learning, there are many approaches towards visuomotor control that learn a forward dynamics model directly in the observation space followed by the use of optimal control techniques to generate trajectories to reach the target. Watter et al. [160] proposed Embed to Control (E2C), an approach that learns a forward dynamics model from images by encoding them (through a variational auto encoder) to a low-dimensional latent state where the dynamics are assumed to be locally linear. Both the auto encoder and the locally-linear dynamics model are learned jointly by minimizing a reconstruction loss (at time t and t+1) and a consistency loss (at t+1) that measures the mismatch between the latent state embeddings predicted by the dynamics model and the latent state predicted through the encoder (at t+1). At test time, given a target image, the trained encoder and locally-linear dynamics model are used together with the iterative Linear Quadratic Gaussian (iLQG) algorithm [152] to find a sequence of controls that reach the target in a closed-loop Model Predictive Control (MPC) fashion. Watter et al. tested their approach on simulated toy problems such as balancing a cartpole and controlling a 3-DOF arm to reach a target configuration. This method was one of the earliest to incorporate the structure from the intended optimal control algorithm to learn good state representations for visuomotor control. Our work [24] borrows some ideas from this paper – specifically, the idea of learning a latent space for control and the use of a consistency loss to enforce long-term consistency of the dynamics model predictions. But unlike this work, we use a structured dynamics model that incorporates 3D geometric priors and object segmentation, our latent state space is very structured (we interpret them as the poses of objects in the scene) and we use a loss that measures motion as opposed to a restrictive reconstruction loss.

Similar to the E2C approach, Finn et al. [49] proposed an approach that also generates controls through MPC with learned forward dynamics models. First, they learn an unsupervised recurrent visual predictive model [47] that models the effect of actions on RGB images. This model takes as input the RGB image and the applied action and predicts a set of convolutional kernels that parameterize the motion followed by compositing masks that weigh the effect of the motion kernels. Combining the two results in a prediction of the image at the next timestep and by feeding this prediction back as input (along with an intermediate LSTM layer in the network) allows the network to make predictions multiple timesteps into the future. This model serves as the forward dynamics model which is then combined with a stochastic optimization algorithm called the Cross Entropy Method (CEM) [129] to find a sequence of actions that drives the network to the target state. Interestingly, the target could be specified as an explicit image or by providing pixel targets to points on objects through a click and drag interface. This method was tested on real-world manipulation tasks where the robot had to push objects in a bin towards target locations and showed good performance for mostly planar motion targets. While our approach shares many similarities with this work such as the use of a visual dynamics model and optimal control for finding the actions there are a few key differences: our network architecture SE3-Nets are more structured and interpretable, unlike their approach which controls in the observation space through sampled actions (at $\approx$ 5Hz), our controller runs gradient optimization on a learned low-dimensional pose embedding in real-time ($>$ 30 Hz). Mainly, their approach requires an external tracker to measure progress when running the controller while we explicitly learn to data-associate through our latent pose space and have no need for an external tracker.

Inverse Dynamics Models: Another class of approaches for visuomotor control generate actions by training inverse models which take in a start and goal state and generate a feasible action that moves the robot towards the goal. Agrawal et al. [5] propose a model that jointly learns a forward predictive model that predicts the effect of applied actions in a learned latent embedding and an inverse model that generates the applied action given the latent embedding of a pair of images separated by that action. The encoder uses a

siamese architecture to learn the latent embedding followed by fully conntected layers for the forward and inverse models. This model was trained on real-world data of the robot poking multiple objects in a tabletop setting. At test time, the inverse model was used to generate poke actions given a target image and the current image and showed good performance on manipulating a range of different objects including a few which were unseen during training. While inverse models have been shown to help regularize representation learning by allowing the network to focus on task-relevant features, in practice they are hard to train as the action space can be multi-modal, high-dimensional and continuous.

Data association: The data association problem, also known as correspondence estimation, is a fundamental problem for many computer vision systems. Traditional approaches to solving data association has relied on matching hand-engineered feature representations such as SIFT [106], SURF [16], etc. With the advent of machine learning, there has been significant progress in learning data associations directly from data. Some of the earliest work in this area focused on learning random decision forests for human and object pose estimation [20, 138, 145]. Using deep networks this was extended towards learning dense pixel-wise semantic segmentation labels [105, 67], again with explicit supervision of dense labels. In contrast to this, recent work has focused on using self-supervision for learning dense visual descriptors for data association [131, 169, 158, 30], typically through the use of a contrastive loss on correspondences over short training sequences. In contrast to many of these approaches, we use correspondences only between pairs of frames to learn a consistent embedding space (i.e. latent poses) that can data-associate across significantly longer sequences.

Pose estimation: There has also been a lot of recent work on object and camera pose estimation from RGB/D data using learning methods [122, 86, 20] including approaches that use structured deep networks [171]. Most, if not all these methods, require explicit pose supervision and work primarily on a fixed set of objects such as the YCB object dataset [25]. While our approach does not generalize to unknown objects, we learn a low-dimensional pose representation purely through self-supervised consistency losses – this resulting pose

Algorithm 1: A simple gradient-based visuomotor controller

Given: Target point cloud ($\mathbf{x}_T$), Pre-trained dynamics model (h_{dyn})

Given: Number of optimization iterations N, Step size α

while Error (E) $> \epsilon$ **do**

  Receive current observation: $\mathbf{x}_t$

  Sample a control: $\mathbf{u}_t$

  for i = 1 to N **do**

    Predict next state: $\hat{\mathbf{x}}_{t+1} = h_{dyn}(\mathbf{x}_t, \mathbf{u}_t)$

    Compute error E between prediction $\hat{\mathbf{x}}_{t+1}$ and target $\mathbf{x}_T$

    Compute gradient of error w.r.t control: $\frac{dE}{d\mathbf{u}_t}$

    Update control: $\mathbf{u}_t = \mathbf{u}_t - \alpha \frac{dE}{d\mathbf{u}_t}$

  end for

  Execute final control $\mathbf{u}_t$ on the robot

end while

space can be used for robust control but does not directly correspond to the canonical pose of objects in the scene.

The rest of this chapter is organized as follows. We first introduce the visuomotor control problem and present an initial gradient-based algorithm in Sec. 3.2 that requires access to privileged information in the form of data associations. To solve this problem, we propose SE3-Pose-Nets in Sec. 3.3; these networks learn a latent pose embedding which can be used in a visuomotor control pipeline without explicit data associations. We evaluate SE3-Pose-Nets in Sec. 3.5, present a discussion in Sec. 3.6 and conclude with a brief summary of Dynamics-Nets, a recurrent extension of SE3-Pose-Nets in Sec. 3.7.

3.2 A First Attempt at the Visuomotor Control Problem

Given a target scene, represented as a point cloud $\mathbf{x}_T$[1], the visuomotor control problem involves finding a sequence of controls $\mathbf{u}_t, \mathbf{u}_{t+1}, \ldots, \mathbf{u}_T$, applying which we can get the robot to reach the target scene, from any initial (or current) scene $\mathbf{x}_t$. As mentioned earlier, in this chapter we will focus on the problem of controlling the robot to reach a target configuration sans any interaction with objects in the scene; many components of our approach are still applicable in the general setting. Fig. 3.1 shows an example setting of this problem, we would like to control the Baxter robot to reach the goal scene (right) from the start scene (left).

With this problem definition in place, we now present a reactive, gradient-based solution to the visuomotor control task, much akin to traditional approaches in visual servoing [78] (Alg. 1). We assume that a pre-trained dynamics model of the environment h_{dyn} is available; this model takes in a point cloud $\mathbf{x}$, control $\mathbf{u}$ and predicts the resulting outcome of taking the action as a point cloud $\hat{\mathbf{x}}$. This model could take many forms such as the baseline flow network or the SE3-NET presented in the previous chapter (Sec. 2.3) and we assume that it has been trained on data relevant to the control problem.

The control algorithm proceeds as follows: At the start of each outer loop iteration, we get the latest observation $\mathbf{x}_t$. Next, we sample a control action $\mathbf{u}_t$ – this sampling procedure can be implemented in many ways but for the purposes of this description we assume sampling from a zero-mean Gaussian distribution with a standard deviation estimated from the data used for training the dynamics model. This is followed by an optimization procedure that refines the control action before it is executed on the robot. We repeat this outer loop until convergence.

Fig. 3.2 illustrates a single step of the optimization procedure used to refine our initial control sample $\mathbf{u}_t$. First, the current observation $\mathbf{x}_t$ and control $\mathbf{u}_t$ are input to the pre-trained dynamics model h_{dyn}, which predicts the resulting future point cloud $\hat{\mathbf{x}}_{t+1}$. Second, we compute the error E between the predicted point cloud $\hat{\mathbf{x}}_{t+1}$ and the target $\mathbf{x}_T$; we discuss

[1]In this chapter, bold fonts denote collections of items

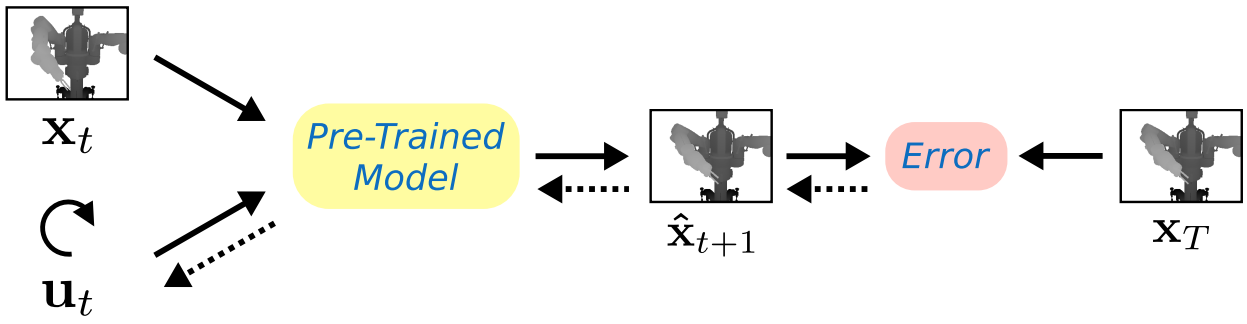

Figure 3.2: A single step of the inner optimization in the gradient-based visuomotor control algorithm (see Alg. 1), where the target is specified as a point cloud $\mathbf{x}_T$. Given a current observation $\mathbf{x}_t$, we hypothesize a random control $\mathbf{u}_t$ and use the pre-trained model (yellow block) to predict the outcome, represented as a point cloud $\hat{x}_{t+1}$. We can iteratively update the controls $\mathbf{u}_t$ by back-propagating the gradients (dotted lines) of the error (red block) between the prediction $\hat{x}_{t+1}$ and the target $\mathbf{x}_T$; the resulting control can be applied to the robot and the process repeated till convergence to the target scene.

how this can be done later. Third, we compute the gradient of this error w.r.t the control $\mathbf{u}_t$ that is input to the dynamics model ($\frac{dE}{d\mathbf{u}_t}$). This can be done by back propagating through the model as done during model training (except we care about the gradients w.r.t the inputs as opposed to the network parameters). Finally, we take a step in the direction of this gradient to update the control. This process is repeated for a fixed number of N iterations, resulting in a control that better minimizes the error between the predicted and target point clouds. We note that sophisticated gradient-based procedures like L-BFGS [104], Conjugate Gradient descent [136] etc., or gradient-free sample based methods such as the Cross Entropy Method [129] can be used in place of the optimization procedure described above to improve the convergence properties; in spite of this the algorithm presented above is hard to implement, as we discuss next.

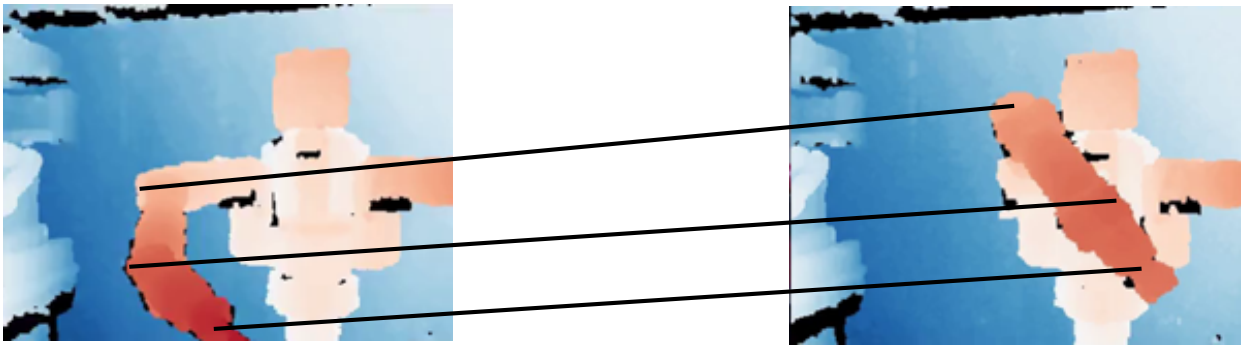

Figure 3.3: The data association problem. Given a pair of frames, the data association problem is to find corresponding points/pixels belonging to the same object in both scenes. In this example, black lines indicate pixels that correspond to the same point on the Baxter robot in both scenes. Correspondence estimation or data association across large motions is an open problem in computer vision.

3.2.1 The Data Association Problem

A key assumption in the gradient-based approach presented in Alg. 1 is the availability of a procedure to compute the error E between a pair of point clouds. A simple error metric we can consider is to measure the norm of the difference between the point clouds – i.e. the norm of the pixelwise difference between the 3-channel HxW representation of the point clouds used by our network. We argue that this error metric is a poor measure of the true distance between these point clouds as it has no knowledge of the underlying structure of the scene – as the arm moves directly towards the target configuration this error need not go down monotonically. The fundamental problem in this, or any error metric directly in the observation space, is the lack of correspondences between these frames i.e. also known as the correspondence estimation or data association problem. Fig. 3.3 illustrates this problem. Given the correspondences (black lines in the figure) we can compute the error E by measuring the norm of the difference in position between the corresponding points – this error should monotonically reduce as we get closer to the target and should also result in proper controls

when used as part of Alg. 1.

Unfortunately, finding correspondences between pairs of frames with arbitrary motion is hard; data association is an open area of research in computer vision [131, 169]. Traditionally, this problem has been solved through the use of sparse features [106, 16]. In the context of deep visuomotor control, optical-flow based trackers have also been used [39]. which have small, local basins of attraction. In contrast to these methods, learning-based methods have looked at learning representations for visual servoing and data associations [27, 131, 160], usually through the use of short-sequences of correspondence labels or unsupervised reconstruction losses. In much the same way, we propose an extension to SE3-Nets that learns a latent, globally-consistent pose representation for data association; by measuring error directly in this learned pose embedding, we present an improvement to Alg. 1 that does not need an explicit data association system unlike prior work.

3.3 Network Architecture

Our approach to data association free visuomotor control begins by proposing an extension to the existing SE3-Nets architecture, which we call SE3-Pose-Nets. Our deep dynamics model SE3-Pose-Nets decomposes the problem of modeling scene dynamics into three sub-problems: a) modeling scene structure by identifying parts of the scene that move distinctly and by encoding their latent state as a 6D pose, b) modeling the dynamics of individual parts under the effect of the applied actions as a change in the latent pose space (parameterized as an $\mathbb{SE}(3)$ transform), and finally c) combining these local pose changes to model the dynamics of the entire scene. Each sub-problem is modeled by a separate component of the SE3-Pose-Net:

- **Modeling scene structure:** An ***encoder*** (h_{enc}) that decomposes the input point cloud ($\mathbf{x}$) into a set of K rigid parts, predicting per part a 6D pose (p^k, $k = 1 \dots K$) and a dense segmentation mask (m^k) that highlights points belonging to that part

- **Modeling part dynamics:** A ***pose transition*** network (h_{trans}) that models dynamics

in the pose space, taking in the current poses ($\mathbf{p}_t$) and action ($\mathbf{u}_t$) to predict the change in poses ($\Delta\mathbf{p}_t$)

- **Predicting scene dynamics:** A ***transform*** layer (h_{tfm}) that generates the next point cloud ($\hat{\mathbf{x}}_{t+1}$) given the current point cloud ($\mathbf{x}_t$), predicted object masks ($\mathbf{m}_t$) and the predicted pose deltas ($\Delta\mathbf{p}_t$) by explicitly applying 3D rigid body $\mathbb{SE}(3)$ transforms on the input point cloud.

Fig. 3.4 shows the network architecture of the SE3-Pose-Net. Next, we present the details of the three sub-components and outline a training procedure for training the SE3-Pose-Net end-to-end with minimal supervision.

3.3.1 Modeling scene structure

Given a 3D point cloud $\mathbf{x}$ from an RGBD sensor (represented as a 3-channel H x W image), the **encoder** (blue block in Fig. 3.4, top half) segments the scene into distinctly moving parts ($\mathbf{m}$) and predicts a 6D pose ($\mathbf{p}$) per part:

$$(\mathbf{p}, \mathbf{m}) = h_{enc}(\mathbf{x}) \tag{3.1}$$

The encoder structure is similar to the SE3-Net architecture (Fig. 2.1); it takes in the RGB/D image and generates a k-channel object segmentation mask of the scene (Sec. 2.2.2) along with k 6D poses.

The encoder has three components, the first is a convolutional network that generates a latent representation of the input point cloud ($\mathbf{x}$). This network has five convolutional layers, each followed by a max pooling layer. The latent representation is further used as input for the mask and pose predictions.

Object masks: We use a fully-convolutional network with five de-convolutional layers and a skip-add architecture (similar to SE3-Nets) to predict a dense pixel-wise segmentation of the scene into its constituent parts ($\mathbf{m}$). The masks predicted by this network are at full resolution with K channels (K x H x W), where K is a pre-specified hyper-parameter that is

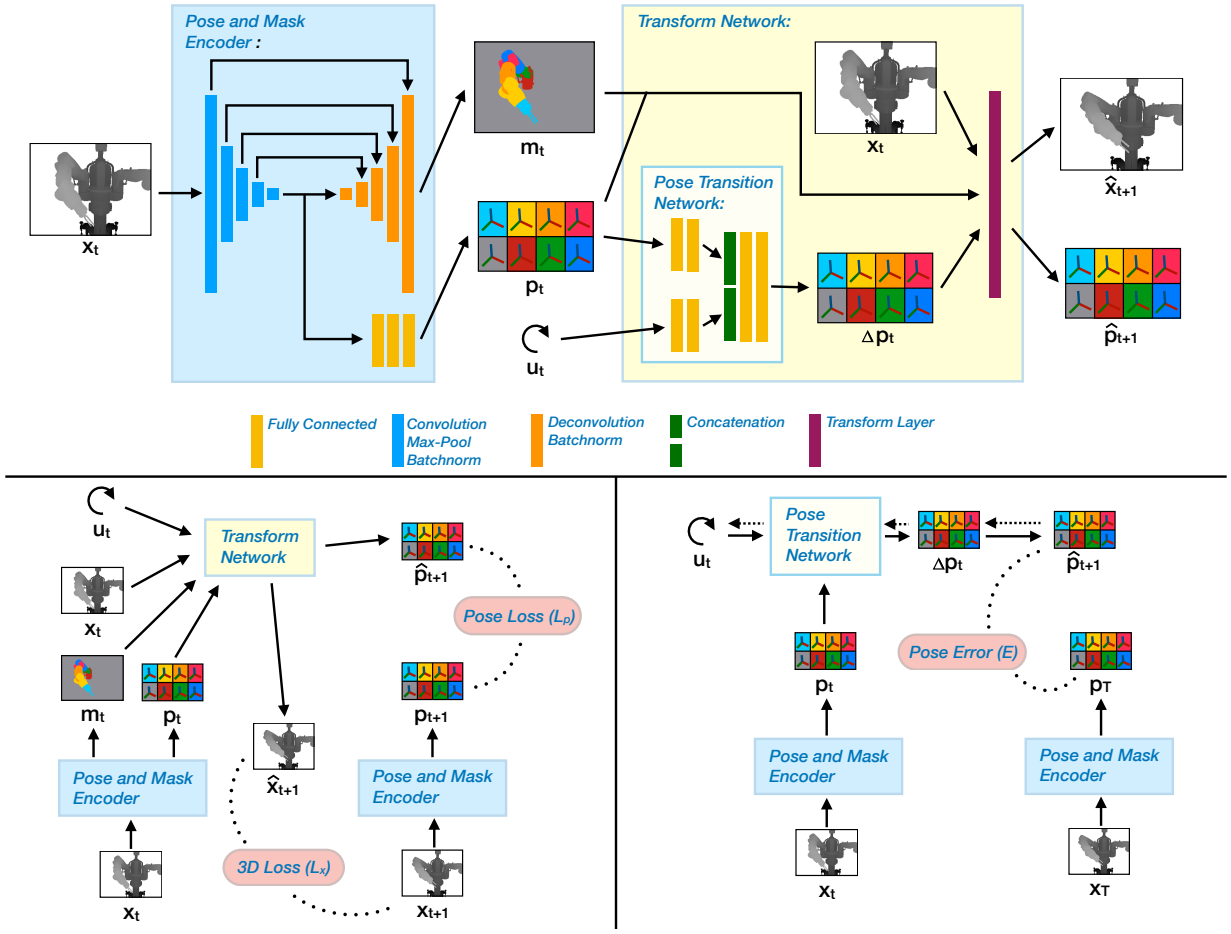

Figure 3.4: ***Top:*** SE3-POSE-NET architecture consisting of three components: the **encoder** (h_{enc}, shown in blue) models scene structure (explained in Sec. 3.3.1), the **pose transition** net (h_{trans}) models object/part dynamics (Sec. 3.3.2) and the **transform layer** (h_{tfm}) which transforms point clouds based on the predicted object dynamics (Sec. 3.3.3). ***Bottom Left:*** Schematic showing the training procedure for SE3-POSE-NET (Sec. 3.3.4). ***Bottom Right:*** Procedure for closed loop control using the SE3-POSE-NET (Sec. 3.4). Gradients shown as dotted lines; initial gradient flows from Pose Error (red block) to predicted poses $\hat{\mathbf{p}}_{t+1}$.

greater than or equal to the number of moving parts in the scene (including background). The predicted segmentation mask learns to attend to parts of the scene that move together, representing areas of the scene that can move independently as different parts. As described in the previous chapter (Sec. 2.2.2), we formalize mask prediction as soft-classification where the network outputs a k-length probability distribution which we sharpen to get a binary segmentation mask.

Object poses: In addition to the predicted segmentation, the SE3-POSE-NETencoder predicts the 6D poses of each segmented part. Given the encoded latent representation, we use a three layer fully-connected network to predict the 6D pose p^k of each of the K segmented parts. We represent each pose by 6 numbers: a 3D position ($y \in \mathbb{R}^3$) and an orientation ($R \in \mathbb{SO}(3)$), represented as a 3-parameter axis-angle vector. As we show later, our pose network learns to predict consistent poses which can be used to data-associate observations over long sequences of motions.

At a high level, the **encoder** implicitly learns the structure of observed scenes by persistently identifying parts, and by predicting consistent poses for parts across multiple scenes.

3.3.2 Modeling part dynamics

Next, we reason about the effect of applied actions on the identified parts. We model this notion of "part dynamics" through a fully-connected **pose transition** network that takes predicted poses from the encoder ($\mathbf{p}$) and applied actions ($\mathbf{u}$) as input to predict the change in pose ($\Delta\mathbf{p}$) for all K segmented parts:

$$\Delta\mathbf{p} = h_{\text{trans}}(\mathbf{p}, \mathbf{u}) \tag{3.2}$$

where $\Delta\mathbf{p} = [\mathbf{R}, \mathbf{T}]$ is represented as an $\mathbb{SE}(3)$ transform per part with a rotation $\mathrm{R}^k \in \mathbb{SO}(3)$ (parameterized as an axis-angle transform) and a translation vector $\mathrm{T}^k \in \mathbb{R}^3$. The transition network first applies two fully connected layers to both inputs, concatenates their outputs followed by two final fully-connected layers to predict the pose-deltas. Fig. 3.4 (top half) shows the architecture of the transition network. As we show later in Sec. 3.4 we rely on

good predictions of pose-deltas through the pose-transition network for efficient control.

3.3.3 Predicting scene dynamics

Finally, given the predicted segmentation ($\mathbf{m}_t$) and change in poses ($\Delta\mathbf{p}_t$), we model the dynamics of the input scene ($\mathbf{x}_t$) due to the applied action ($\mathbf{u}_t$). We do this through the **Transform** layer (h_{tfm}) which applies the predicted rigid rotations ($\mathbf{R}_t$) and translations ($\mathbf{T}_t$) to the input depth cloud, weighted by the predicted mask probabilities ($\mathbf{m}_t$). We predict the transformed point cloud as:

$$\hat{x}_{t+1}^j = \sum_{k=1}^{K} m_t^{kj} \left(R_t^k \, x_t^j + T_t^k \right) \tag{3.3}$$

where $\hat{x}_{t+1}^j$ is the 3D output corresponding to input point x_t^j. In effect, we apply the kth rotation and translation ($\Delta p^k = [R^k, T^k]$) to all points x^j that belong to the corresponding object as indicated by the kth mask channel m^k (assuming binary masks) to predict the transformed points $\hat{x}^j$ for that object. Repeating this for all objects gives us the transformed output point cloud ($\hat{\mathbf{x}}$). Note that this part has no trainable parameters. For more details, please refer to Sec. 2.2.3 in the previous chapter.

3.3.4 Training

We now outline a procedure to train the SE3-Pose-Net end-to-end, using supervision in the form of point-wise data associations between a pair of point clouds ($\mathbf{x}_t$, $\mathbf{x}_{t+1}$), related by an action ($\mathbf{u}_t$) i.e. for each input point (x_t^i), we know its corresponding point (x_{t+1}^i) if it is visible. No other supervision is given for learning the masks, poses, and the change in poses. Fig. 3.4 (bottom left) shows a schematic of this procedure.

Given two point clouds $\mathbf{x}_t, \mathbf{x}_{t+1}$, we use the **encoder** to predict the corresponding masks and poses:

$$\mathbf{p}_t, \mathbf{m}_t = h_{enc}(\mathbf{x}_t) \;;\; \mathbf{p}_{t+1}, \mathbf{m}_{t+1} = h_{enc}(\mathbf{x}_{t+1}) \tag{3.4}$$

Next, the predicted pose ($\mathbf{p}_t$) and control ($\mathbf{u}_t$) are used as input to the **pose transition** net to predict the change in pose from t to $t+1$:

$$\Delta\mathbf{p}_t = h_{trans}(\mathbf{p}_t, \mathbf{u}_t) \tag{3.5}$$

Finally, we predict the next point cloud using the **transform** layer (3.3):

$$\hat{\mathbf{x}}_{t+1} = h_{tfm}(\mathbf{x}_t, \mathbf{m}_t, \Delta\mathbf{p}_t) \tag{3.6}$$

The predicted mask ($\mathbf{m}_{t+1}$) at time $t+1$ is discarded. We use two losses used to train the entire pipeline end to end:

- A **3D loss** (L_x) that penalizes the error between the predicted point cloud ($\hat{\mathbf{x}}_{t+1}$) and the data associated target point cloud ($\tilde{\mathbf{x}}_{t+1}$). We use a normalized version of the mean-squared error (MSE) that scales based on the target magnitude:

$$L_x = \frac{1}{N}\sum_{i=1}^{HW}\frac{(\hat{x}^i_{t+1} - \tilde{x}^i_{t+1})^2}{\alpha\tilde{f}^i + \beta} \tag{3.7}$$

  where ($\tilde{f}^i = \tilde{x}^i_{t+1} - x^i_t$) denotes the ground truth motion for point i relative to the input point cloud $\mathbf{x}_t$, N is the number of points that actually move between t and $t+1$ and α & β are hyper-parameters ($\alpha = 0.5, \beta = 1e-3$ in all our experiments). This loss is aimed to tackle two main issues with a standard MSE loss: a) By normalizing the loss by a separate scalar per dimension ($\tilde{f}^i$) that depends on the target magnitude we make the loss scale invariant, allowing us to treat equally parts that move less (such as the end-effector when only the wrist rotates) as those that have large motion (e.g. the elbow), and b) By dividing the total error by the number of points (N) that move in the scene, we treat scenes where very few points move equally as those where large parts move.

- An **pose consistency** loss (L_p) that encourages consistency between the poses predicted by the encoder ($\mathbf{p}_t, \mathbf{p}_{t+1}$) and the change in pose predicted by the pose transition network ($\Delta\mathbf{p}_t$):

$$\hat{\mathbf{p}}_{t+1} = \mathbf{p}_t \oplus \Delta\mathbf{p}_t$$
$$L_p = \frac{1}{I}\sum_{i=1}^{I}(\hat{p}^i_{t+1} - p^i_{t+1})^2 \tag{3.8}$$

where $\oplus$ refers to composition in $\mathbb{SE}(3)$ pose space, $\hat{\mathbf{p}}_{t+1}$ is the expected pose at $t+1$ from composing the current pose ($\mathbf{p}_t$) and the predicted pose change from the transition model ($\Delta\mathbf{p}_t$) and I is the cardinality of $\mathbf{p}_t$. In essence, this loss constrains the encoder to predict poses that are consistent with the pose-deltas predicted by the transition model. This loss encourages global consistency in the pose space by enforcing local consistency over pairs of frames and is crucial for learning a pose space that is consistent across long term motions.

From an optimization standpoint, this loss can be optimized in two ways: by modifying the poses ($\mathbf{p}_t, \mathbf{p}_{t+1}$) to match the predicted change in pose ($\Delta\mathbf{p}_t$) or vice-versa. We note that our 3D loss (Eq. 3.7) directly provides the right gradient signals for the deltas ($\Delta\mathbf{p}_t$). Adding in the consistency loss will cause competition between the two loss gradients which we want to avoid. To avoid this, we ensure that there is no gradient path between the consistency loss and the predicted deltas during training: the consistency loss only helps train the encoder poses.

The total loss for training $L = L_x + \gamma L_p$, where γ controls the relative strengths of the two losses. We set $\gamma = 10$ in all our experiments.

A key point to note is that we do not provide any explicit supervision to learn the pose space. While the consistency loss ensures that the poses are more or less globally consistent, it does not anchor them to a specific reference frame such as the object's center and its principal axes. As such, the poses learned by the network need not correspond directly to the canonical 6D pose of the parts making direct comparisons to pose estimation networks hard. Providing more constraints to physically ground the pose space is an interesting area for future work.

Algorithm 2: DA-free visuomotor control using SE3-Pose-Nets

Given: Target point cloud ($\mathbf{x}_T$), Pre-trained encoder (h_{enc}) & transition model (h_{trans})

Given: Control magnitude: u_{max}

Compute target pose: $\mathbf{p}_T = h_{enc}(\mathbf{x}_T)$

while Pose Error (E) $> \epsilon$ **do**

  Receive current observation ($\mathbf{x}_t$)

  Predict current pose: $\mathbf{p}_t = h_{enc}(\mathbf{x}_t)$

  Initialize control to all zeros: $\mathbf{u}_t = 0$

  Predict change in pose: $\Delta\mathbf{p}_t = h_{trans}(\mathbf{p}_t, \mathbf{u}_t)$

  Predict next pose: $\hat{\mathbf{p}}_{t+1} = \mathbf{p}_t \oplus \Delta\mathbf{p}_t$

  Compute pose error: $E = \frac{1}{I}\sum_{i=1}^{I}(\hat{p}_{t+1}^i - p_T^i)^2$

  Compute gradient of error w.r.t. control: $g = \frac{dE}{dU_t}$

  Compute control: $\mathbf{u}_t = -u_{max} \times \frac{g}{||g||}$

  Execute control $\mathbf{u}_t$ on the robot

end while

3.4 Closed-Loop Visuomotor Control using SE3-Pose-Nets

We now show how an SE3-Pose-Net can be used for closed-loop visuomotor control to reach a target specified as a RGB/D image, essentially performing visual servoing [78]. As we discussed in Sec. 3.2, a crucial component of every visual servoing system is to perform data association (DA) between the current and target images, which can then be used to generate controls that reduce the corresponding offsets. In contrast to Alg. 1 which needs an explicit external data association system, SE3-Pose-Nets solve this problem by making use of the learned, low-dimensional latent pose space. By enforcing frame-to-frame consistency in the pose space through the consistency loss (Eqn. 3.8), the pose space becomes consistent, that is, our encoder network learns to data-associate observations to unique poses which are consistent under the effect of actions. Importantly, these data associations are generated at

the mask, or object level, resulting in an ability akin to object detection in computer vision. Unlike prior work [48, 23] which is restricted to operate in the observation space, we can now directly minimize error between the poses $\mathbf{p}_0$ and $\mathbf{p}_T$, which are automatically extracted from the initial and the target observations, to recover the sequence of actions that takes the robot from $\mathbf{p}_0$ to $\mathbf{p}_T$. Additionally, unlike prior work [48], we do not need an external tracking system to measure progress toward the goal as our learned **encoder** implicitly tracks in the pose space.

3.4.1 Control

Algorithm 2 presents a simple algorithm for reactive control using SE3-Pose-Nets that efficiently computes a closed-loop sequence of controls that takes the robot from any initial state $\mathbf{x}_0$ to the specified target $\mathbf{x}_T$ (Fig. 3.4, bottom right). Given a target point cloud, $\mathbf{x}_T$, the algorithm uses the learned encoder to predict the poses of the constituent parts $\mathbf{p}_T = h_{enc}(\mathbf{x}_T)$. This becomes the target to the controller.

At every time step, the algorithm computes the pose embedding $\mathbf{p}_t$ of the current observation $\mathbf{x}_t$. We would like to find controls that move these poses closer to the target poses. To do this, the algorithm makes a prediction through the learned **pose transition** model using the current poses ($\mathbf{p}_t$) and an initial guess for the controls (here we use $\mathbf{u}_t = 0$), resulting in a predicted change in poses ($\Delta\mathbf{p}_t$) and the corresponding predicted next pose ($\hat{\mathbf{p}}_{t+1}$) [2]. We measure the mean-squared error (E) between the predicted poses ($\hat{\mathbf{p}}_{t+1}$) and the target poses ($\hat{\mathbf{p}}_T$), compute its gradient with respect to the control inputs (g) and use it to generate the next control. We propose two ways of computing this update:

- **Backpropagation:** A simple approach to compute the update is to backpropagate the gradients of the pose error E through the transition model. Via chain rule, we have:

$$g = \frac{dE}{d\mathbf{u}_t} = \frac{d\hat{\mathbf{p}}_{t+1}}{d\mathbf{u}_t}\frac{dE}{d\hat{\mathbf{p}}_{t+1}} = J^T \frac{dE}{d\hat{\mathbf{p}}_{t+1}} \tag{3.9}$$

[2] Even when using a zero control initialization, this forward pass through the network is necessary to get the correct gradients for the backward pass.

where $\frac{dE}{d\hat{\mathbf{p}}_{t+1}}$ is the gradient of the pose error w.r.t the predicted poses and J is the Jacobian of the transition model w.r.t the input controls ($\mathbf{u}_t$). Unlike backpropagation during training, where we compute gradients w.r.t. the network weights, here we fix the weights and compute gradients over the input controls. The resulting control scheme is analogous to the Jacobian Transpose method from inverse kinematics [22].

- **Gauss-Newton:** A better approach is to compute the Gauss-Newton update:

$$g = (J^T J + \lambda I)^{-1} J^T \frac{dE}{d\hat{\mathbf{p}}_{t+1}} \tag{3.10}$$

where J is the Jacobian of the transition model, and $\frac{dE}{d\hat{\mathbf{p}}_{t+1}}$ is the gradient of the pose error (E) w.r.t the predicted poses $\hat{\mathbf{p}}_{t+1}$. This update conditions the pose error gradient through the Jacobian's pseudo-inverse (the jacobian can be computed efficiently by finite differencing), where λ controls the strength of the conditioning (set to 1e-4 in all our experiments). In practice, this leads to significantly faster convergence with little to no additional overhead in computation compared to the backpropagation method as the Jacobian can be computed efficiently through finite differencing. We do this by running a single forward propagation with perturbed control inputs (perturbation set to 1e-3) stacked along the batch dimension to take advantage of GPU parallelism. Eqn. 3.10 is analogous to the Damped Least Squares technique from inverse kinematics [22] and traditional visual servoing [78].

Finally, the algorithm computes the unit-vector in the direction of the computed update and scales this by a pre-specified control magnitude u_{max} (1 radian) to get the next control $\mathbf{u}_t$. We execute this control on the robot and repeat in a closed-loop either until **convergence**, measured by reaching a small error in the pose space ($E < \epsilon$) or a maximum number of iterations, whichever comes first. Fig. 3.4 (bottom right) illustrates a single iteration of our visuomotor control algorithm using SE3-Pose-Nets – in contrast to the initial observation space controller in Alg. 1 and the corresponding optimization procedure in Fig. 3.2, we now

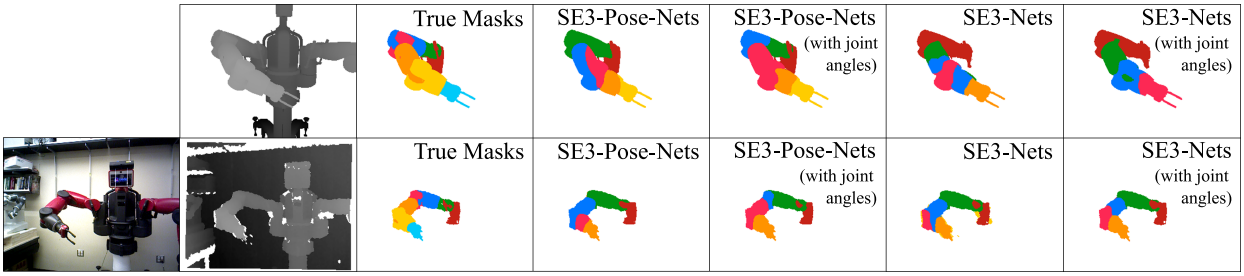

Figure 3.5: Masks generated by different networks on simulated (top) and real data (bottom, with additional color input shown). Both SE3-Nets and SE3-Pose-Nets segment the arm into multiple "physically consistent" parts without any explicit supervision, on both simulated and real data. *Note:* Colors are for display only. Predicted mask colors **do not** need to match the colors of ground truth masks.

operate in the learned latent pose space and the pose error can be evaluated directly without the need for an external data association system.

3.5 Evaluation

We first evaluate SE3-Pose-Nets on predicting the dynamics of a scene where a Baxter robot moves its right arm in front of the RGBD camera, both in simulation and in the real world. We also present results on control performance where the task is to control the joints of the Baxter's right arm to reach a specified target observation.

3.5.1 Task and Data collection

We first provide details on the task setting in simulation. Our simulator uses OpenGL to render depth images from a camera pointed towards the robot (see Fig. 3.5) and is kinematic with little to no dynamics in the motion and no depth noise. We use this as a test bed to parse the effectiveness of the proposed algorithm and compare it to various baselines. We collected around 800K training images (from a single viewpoint) in the simulator where the robot moves all joints on its right arm. Around half of the examples are whole arm motions

where the robot plans a trajectory to reach a target end-effector position sampled randomly in front of the robot. The remaining motions are perturbations to individual joints from various initial configurations sampled to be within the viewpoint of the camera. These additional motions help de-correlate kinematic dependencies, improving performance especially on joints lower down the kinematic chain.

Additionally, we collected (RGBD) data from the real robot where the Baxter moves its right arm in front of an ASUS Xtion Pro camera placed around 2.5 meters from the robot. Data associations, ground truth masks, and ground truth flows are determined via the DART tracker [132]. We collected around 7 hours of training data on the real robot with a 1:1 mix of whole arm and single joint motions (specifically of the lower joints of the arm). This data has some (unintended) variations in the background and minor changes in the camera pose and lighting. Unlike the simulated data, the depth data in the real world is quite noisy and there are significant physical and dynamic effects. For both the simulated and real world settings, our controls are joint velocities ($\mathbf{u}$).

3.5.2 Baselines

We compare against five different baselines:

- **SE3-Pose-Nets + Joint Angles:** Our proposed network with the robot's joint angles given as an extra input to the **encoder**. This is a strong baseline that uses significant extra information to inform pose prediction. As we discuss later in Sec. 3.6, we would ideally combine this state information with visual data when we look at interacting with objects in the real world.

- **SE3-Nets:** Prior work from chapter 2 which directly predicts masks and change in poses given input point clouds and control. As there is no explicit pose space, we control in the point cloud space with this network (Alg. 1).

- **SE3-Nets + Joint Angles:** SE3-Nets that additionally take in joint angles as

inputs.

- **Flow Net:** Baseline flow model from prior work ([23] and chapter 2). This network directly regresses to a per-point 3D flow without any $\mathbb{SE}(3)$ transforms or masks. Similar to SE3-Nets we control in point cloud space with this network (Alg. 1).

- **Flow Net + Joint Angles:** Baseline flow network that additionally takes in joint angles as input.

All baseline networks are trained on the same data as the SE3-Pose-Nets using the 3D normalized loss (L_x, Eqn. 3.7).

3.5.3 Training details

We implemented our networks in PyTorch using the RMSprop/ADAM optimizers for training with a learning rate of 1e-4 (baselines work best with ADAM). All our networks use Batch Normalization [80] and the PReLU non-linearity [70]. At the start of training, we initialize the layers predicting the poses and the change in poses to close to identity for the SE3-Pose-Nets and SE3-Nets and the layer predicting flows to identity for the baseline flow networks. We set the maximum number of moving objects $K = 8$ for all our experiments (7 joints + background). Each network is trained for 100K iterations, and we use the one with the least 3D loss (L_x) on a separate validation set for our results. Additionally for the real-world experiments, we use RGB images as an extra input to the pose-mask encoder (concatenated with the depth image) for training all networks including baselines.

3.5.4 Results on modeling scene dynamics

First, we present results on the prediction task used for training all networks. Table 3.1 shows the average per-point flow RMSE (cm) across all baselines on both simulated and real data. SE3-Nets achieve the best results while the baseline flow network performs slightly worse. Unsurprisingly, networks that have access to the joint angles do better than those

Setting	SE3-Pose-Nets	SE3-Pose-Nets + Joint Angles	SE3-Nets	SE3-Nets + Joint Angles	Flow	Flow + Joint Angles
Simulated	0.027	0.017	0.022	**0.012**	0.037	0.021
Real	0.209	0.200	0.182	**0.170**	0.201	0.193

Table 3.1: Average per-point flow RMSE (cm) across tasks and networks, normalized by the number of points M with ground truth motion $>$ 1mm. Our network achieves results comparable/better than baseline networks on simulated data and performs slightly worse on real data. However, it is also solving additional tasks necessary for control.

which do not, as they have strictly more information that is highly correlated with the sensor data. To our initial surprise, SE3-Pose-Nets had the largest prediction errors among all baseline models (on real data). However, this makes sense given the following considerations: a) SE3-Pose-Nets are trained to explicitly embed the observations in a pose space from which they predict the scene dynamics, rather than using the input point cloud directly. While this provides more structure and is necessary for the control task, it also restricts the prediction to go through an information bottleneck which generally makes training hard. b) SE3-Pose-Nets additionally have to optimize the consistency loss, which enforces constraints that are different from those of the prediction problem evaluated in this experiment. As we show later, while there is a slight drop in prediction performance, SE3-Pose-Nets are able to learn a consistent pose representation which significantly improves control performance as compared to the baseline models (3.5.6).

Fig. 3.5 visualizes the masks predicted by SE3-Pose-Nets and the baseline SE3-Neton an example each from the simulated and real data along with the ground truth masks. Even without any supervision, SE3-Pose-Nets and SE3-Nets learn a detailed segmentation of the arm into multiple salient parts, most of which are consistent with ground truth segments on both the simulated and real data.

Multi-step prediction: We also tested the performance of all networks on multi-step

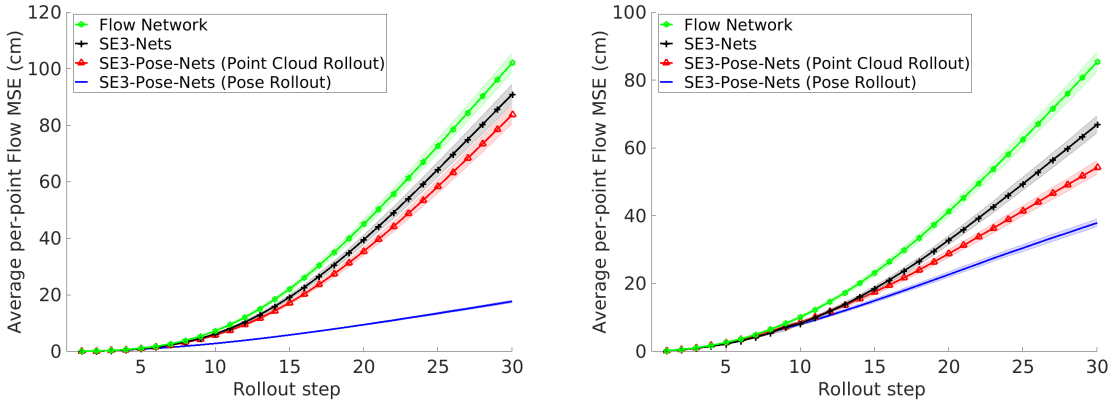

Figure 3.6: Multi-step prediction performance showing per-point flow RMSE (in cm) against the number of rollout steps into the future for different network architectures (without joint angles). **Left:** Simulated data, **Right:** Real data. Shaded regions indicate 95% confidence bounds. All networks were trained for single-step prediction only.

prediction, where the task is to predict a sequence of future point clouds $(\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, ..., \hat{\mathbf{x}}_M)$ given an initial point cloud $(\mathbf{x}_0)$ and a sequence of controls $(\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_{M-1})$. We do this in two ways:

- **Point cloud rollouts:** Given the initial point cloud and control $(\mathbf{x}_0, \mathbf{u}_0)$, we make a prediction using the trained single-step model: $\hat{\mathbf{x}}_1 = f(\mathbf{x}_0, \mathbf{u}_0)$. We use this predicted point cloud (without any re-projection) as the input for the next prediction step: $\hat{\mathbf{x}}_2 = f(\hat{\mathbf{x}}_1, \mathbf{u}_1)$ and repeat this for all future predictions. This is applicable to all our networks: SE3-Pose-Nets, SE3-Nets and the baseline flow networks.

- **Pose rollouts:** As an alternative to rollouts in the observation space, we make use of the learned low-dimensional pose space in SE3-Pose-Nets for multi-step predictions. Given the initial point cloud $(\mathbf{x}_0)$, we first predict the initial pose and mask using the encoder: $\mathbf{p}_0, \mathbf{m}_0 = h_{enc}(\mathbf{x}_0)$. Similar to computing rollouts in point cloud space, we use the learned pose transition model (h_{trans}) in a sequential manner to predict all future poses $(\hat{\mathbf{p}}_1, \hat{\mathbf{p}}_2, ...\hat{\mathbf{p}}_M)$ given the predicted initial pose $(\mathbf{p}_0)$ and the sequence of controls

$(\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_{M-1})$. Finally, to predict a point cloud at any future step t, we use the **transform** layer (3.3): $\hat{\mathbf{x}}_{t+1} = h_{tfm}(\mathbf{x}_0, \mathbf{m}_0, \hat{\mathbf{p}}_t \ominus \mathbf{p}_0)$ where $\hat{\mathbf{p}}_t \ominus \mathbf{p}_0$ denotes the total change in pose from the initial pose $\mathbf{p}_0$ to the current pose $\hat{\mathbf{p}}_t$.

Fig. 3.6 presents multi-step prediction results on both simulated and real data for all networks (without joint angle inputs). Note that while we evaluate the networks on multi-step prediction performance all the networks are trained for the single-step prediction task described above. Overall, SE3-Pose-Nets perform significantly better than baseline models in multi-step prediction. As expected, error increases as we predict further into the future in an open-loop manner due to cascading noise and data-distribution mismatch. Flow networks (green) show the largest increase, followed by SE3-Nets (black) and SE3-Pose-Nets with point cloud rollouts (red). SE3-Pose-Nets with pose space rollouts (blue) achieve the best performance with large reductions in error on simulated data and to a lesser extent on real data. We hypothesize that this benefit comes primarily from our structured pose space where noise propagates slowly as opposed to the high-dimensional observation space (similar to results from [28]). This leads us to believe that using Model Predictive Control (MPC) style approaches (such as iterative LQG [152]) with pose space rollouts should work well for solving long horizon planning problems which we will investigate further in future work. These results can be further improved by optimizing our models for multi-step prediction performance (optionally, with memory), which is also scope for future work. A video showing multi-step prediction results on simulated data can be found on our webpage.

3.5.5 Pose consistency

We evaluated the consistency of the latent pose embedding learned by SE3-Pose-Nets. We consider sequences of observations $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_T$ from our test set, and generate corresponding pose embeddings $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_T$ for each step using the pose-mask encoder from our trained network. We compute the distance to the target pose $\mathbf{p}_T$ for all poses (i.e. the pose error in Alg. 2) and plot this against the normalized distance between the corresponding joint

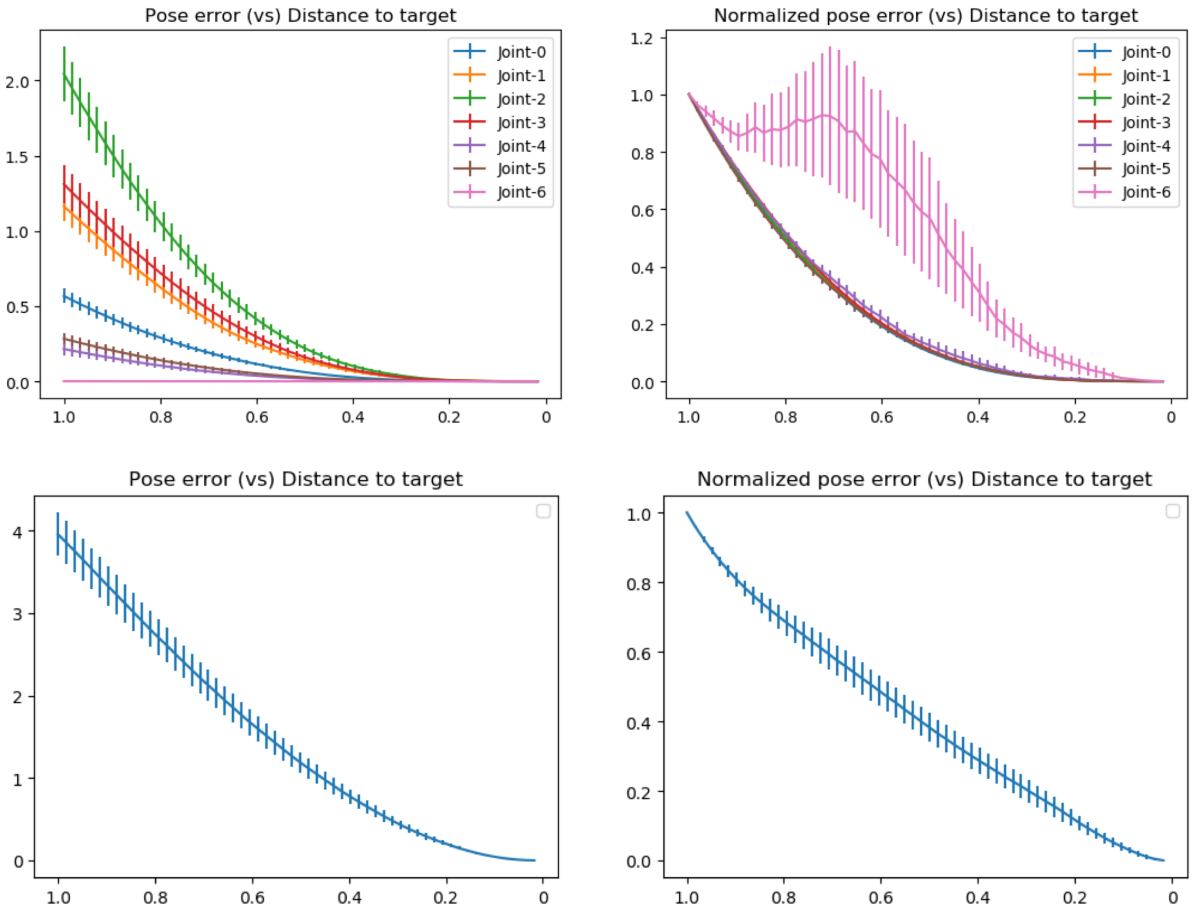

Figure 3.7: Plots evaluating the long-term consistency of the learned pose embedding. **Top row:** Error in the learned pose space (y-axis) as a function of the normalized error in the joint angles (x-axis), for motion along a single joint of the Baxter. **Bottom row:** Pose error (y-axis) vs normalized joint angle error (x-axis), for simultaneous motion across all joints of the Baxter. Errors averaged over 100 sequences and error bars show standard error. In both rows, the *left* plot displays the raw, un-normalized MSE between the current and target poses while the *right* plot displays errors normalized by the pose error at the start of motion. The normalized error reduces monotonically as a function of the joint angle error for all joints (except the end-effector whose motion is hard to distinguish from low-resolution images), lending strength to its use as the metric for our DA-free visuomotor controller.

angles and the joint angles of the target configuration q_T. Fig. 3.7 presents these results; the left column plots the raw, un-normalized pose errors while the right column plots the pose errors normalized by the error at the initial timestep $t = 0$, which, ideally should be the largest value in the sequence. Additionally, the plots are separated based on the type of sequence considered: the top row plots errors from sequences where a single joint of the robot is actuated and all other joints are fixed while the bottom row presents results from sequences where all 7 joints of the robot are actuated.

We describe a few key points: 1) Errors in the latent pose space reduce monotonically as a function of the error in the joint angles, both in the single and multi joint motion settings. This can be seen on both the plots on the right column of Fig. 3.7; the normalized errors in the pose space reduce monotonically w.r.t the joint angle error. This result highlights the consistency of our learned pose space; even when learning only from single step data associated point clouds the structure underlying SE3-Pose-Nets allows them to learn an embedding that is globally consistent and can be used as a proxy for joint angle error. 2) The magnitude of the pose error depends on the joint being actuated (see Fig. 3.7, top left); as one would expect, joints higher up the kinematic chain (such as the base and elbow joints) have a larger effect on the underlying pose space than those lower down. 3) The end-effector usually occupies a tiny fraction of the observed image and small motions of the end-effector (under its own actuation) are hard to distinguish directly from visual observations. As a consequence, the learned pose space struggles to model the motion of the end-effector, as observed by the small magnitude of the pose errors (Fig. 3.7, top left) and the non-monotonic nature of the normalized pose error (Fig. 3.7, top right) under actuations of the end-effector (joint-6). Although our pose space struggles to model the end-effector motion well, we believe that the results so far present compelling evidence that our latent pose embedding is globally consistent and can be used as a metric for visuomotor control. We test this further in the next few experiments, where we attempt to control the Baxter robot (sans the end-effector) solely from visual observations using the procedure outlined in Alg. 2.

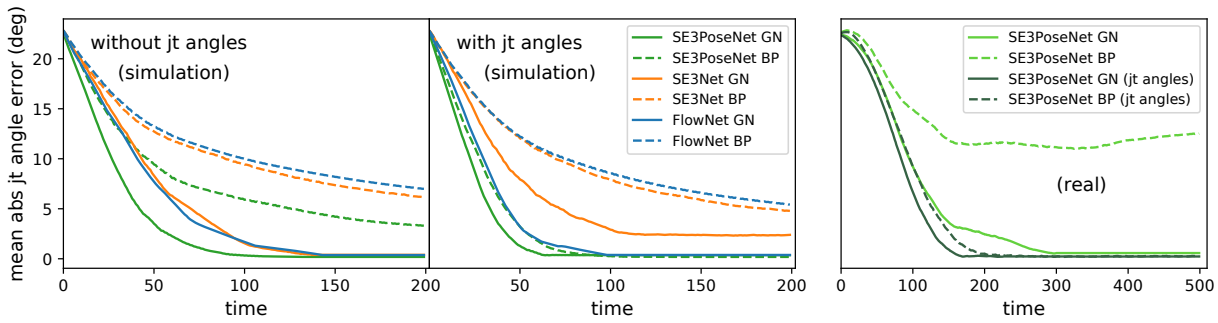

Figure 3.8: Convergence of mean absolute joint angle error (averaged across 11 different examples, 6 joint control) for simulated Baxter control tasks (left - w/o joint angle input & center - with joint angle input) and real robot control (right). Dotted lines use backprop gradient updates (BP), solid lines use Gauss-Newton updates (GN). SE3-Pose-Nets perform as well or better than baseline methods even though baseline models have additional information in the form of ground truth-associations. All plots share the y-axis.

3.5.6 Control performance

Next, we test the performance of the networks on controlling the first six joints of the Baxter's right arm to reach a target configuration, specified as a point cloud $\mathbf{x}_T$. We test both the gradient update schemes from Sec. 3.4, comparing their performance on a set of 11 distinct servoing tasks (each with an average initial error of ˜25 degrees per joint).

Control with baseline models: As most of our baseline models operate directly in the observation space (unlike the learned pose space in SE3-Pose-Nets), they **require external data associations** to be able to do any control at all. For the simulation experiments, we provide the baselines with ground-truth associations and use the procedure outlined in Alg. 1 using the MSE between the predicted point cloud $\hat{\mathbf{x}}_{t+1}$ and the data-associated target $\mathbf{x}_T$ as the error to be minimized for generating controls. It is important to keep in mind that the baseline models have an advantage over SE3-Pose-Nets for the control task as they get strictly more information in the form of ground-truth data associations.

Metric and Task specification: We use the mean absolute error in the joint angles

as the metric for measuring control performance. We run all models to convergence (based on the pose error for SE3-Pose-Nets and 3D point/flow error for the baseline models) or for a maximum of 200/500 iterations (sim/real). We integrate joint velocities forward to generate position commands both in simulation and the real world.

Simulation results: Fig. 3.8 plots the error in joint angles as a function of the number of control iterations. The first two plots (from the left) show results in simulation, both for networks that do not use joint angles (leftmost) and otherwise (middle). In general, SE3-Pose-Nets achieve excellent performance compared to the baseline models, converging quickly to an almost zero error even **in the absence any external data associations**. We highlight a few key results: 1) For all networks, Gauss-Newton based optimization (GN) leads to faster convergence than backprop (BP). This is to be expected as Gauss-Newton conditions the gradient based on pseudo-second order information. 2) Baseline SE3-Net models perform worse given joint angles than without. This is due to an issue of credit assignment during gradient computation - the networks learn erroneous causations (when there are only correlations) between the input joint angles and the predicted flows which diminishes the control's contribution to the prediction problem and subsequently affects the gradient. Additionally, SE3-Nets performance is affected by the lack of a good control initialization (needed to ensure that they get a good segmentation) - given zero controls the SE3-Net can choose not to segment the arm at all 3) All models struggle with the motion of the final wrist joint due to increasing correlations along the kinematic chain that result in a small contribution of the joint's own motion to the full movement of the wrist. SE3-Pose-Nets (and other baselines) do overcome this problem given input joint angles in a separate experiment, achieving < 1 deg error across all 11 examples when controlling the entire right arm. This provides encouraging proof that adding in the joint state supplements information that is hard to parse directly from vision, establishing the case for including them systematically, and finally 4) Good performance of SE3-Pose-Nets indicate that the learned pose space is consistent across large motions and can be used for fast, robust, and reactive control.

Result	Lighting	Camera	Occlusion	Motion
Average Error	3.8	1.4	1.9	2.5
Failures	0/11	1/11	1/11	2/11

Table 3.2: Average final joint angle error (degrees) and number of failures (divergent examples) out of 11 tasks across different perturbations. SE3-Pose-Nets can still robustly control the arm in the presence of different types of observation noise. The reported errors are averaged over non-divergent examples.

Real robot results: We further test the control performance of SE3-Pose-Nets in the real world (RGBD input) on the same set of configurations. We do not compare to any baselines as they need an explicit external data association system to be feasible. Fig. 3.8 (right) shows the results - SE3-Pose-Nets converge very quickly to nearly zero error on all examples indicating that our network can control robustly even in the presence of sensor noise and unmodeled dynamics. Once again, Gauss-Newton significantly outperforms the backprop update which fails to converge in the absence of joint angles. Additionally, adding joint angles does allow us to control the wrist, albeit not as robustly as in simulation. A video showing real-robot control results can be found on our webpage.

Generalization/Robustness results: Finally, we tested the generalization performance and robustness of real-robot control using SE3-Pose-Nets (GN, no joint angles) to novel perturbations of the scene. We tested control performance across all 11 servoing tasks for the following four variations: *1) Change in lighting:* We considered a low-light setup, significantly darker than the training set (Fig. 3.9, top left), *2) Varying camera pose:* We considered three variations to the camera pose: moving forward by 10 cm and rotating by 10 degrees to the left & right (Fig. 3.9, bottom right), *3) Occlusion:* We tested three settings where we added multiple fixed occluders to the robot: cloth attached to the robot torso only, additionally occluding the base joint and adding occluders to the torso, base joint and end-effector (Fig. 3.9, bottom left) and finally, *4) Motion:* We tested four different

Figure 3.9: Examples of different generalization tests, unseen during training. Clockwise from top left: 1) Low light setting, 2) Motion in the background (person on the chair moves), 3) Fixed distractor(s) in the foreground (white towels on the robot) and 4) Change in camera pose (rotated ∼10 degrees to the right). SE3-Pose-Nets are robust to these changes, achieving good control performance on a majority of our tests with these perturbations.

settings with moving distractors (people, books) in the foreground (in front of the robot) and background Fig. 3.9, top right).

Table 3.2 summarizes the control results under perturbations. In general, SE3-Pose-Nets can control robustly even in the presence of scene changes, achieving low joint angle error at the end of the control optimization. There is a caveat though: we observed divergent behavior for joints lower down in the kinematic chain (primarily joints 5,6) for 1-2 servoing examples under some perturbations (Table 3.2, last row). While improvements in robustness and a more thorough evaluation (also in simulation) are necessary, we believe that these results serve as a good proof of concept of the strengths of SE3-Pose-Nets; the structure inherent in our networks allow them to be robust to noise in the inputs. Also, these results highlight some of the advantages of learning methods as opposed to more traditional model-based tracking methods (such as DART [132]) which need significant additional work to handle such variations. A video showing real-robot control (with and without scene variations) can be found on our webpage.

Speed: SE3-Pose-Nets optimize errors directly in the low-dimensional pose space for control. This leads to significant speedups: while both the flow and SE3-Nets can operate at around 10Hz (excluding data association), SE3-Pose-Nets run in real-time (30Hz) including pose detection.

3.6 Discussion and Future Directions

In this chapter we presented SE3-Pose-Nets, a framework for learning predictive models that enable control of objects in a scene. In the context of control of a robot manipulator, we showed how to solve this problem by learning a predictive model for the individual parts of the manipulator, along the lines of previous work (Chapter 2). Additionally, SE3-Pose-Nets learn a consistent *pose space* for these parts, essentially learning to *detect* the 6D poses of manipulator parts in the raw depth images. This detection capability enables SE3-Pose-Nets to solve the data association problem that is crucial for relating the current observation of the manipulator to a desired target observation. The difference between these poses can be

used to generate control signals to move the manipulator to its target pose, similar to visual servoing applied to an image of the manipulator. We also showed how the learned network can be used to determine the gradients needed for the control signals. Our experiments show that SE3-Pose-Nets generate control superior to representations learned by previous techniques, even when these are provided with external data associations. Furthermore, in addition to providing data associations, SE3-Pose-Nets allow us to compute controls directly in the low dimensional pose space, enabling far more efficient control than techniques that operate in the raw perception space. Additionally, control using SE3-Pose-Nets is robust to some perturbations to imaging conditions. Crucially, all these abilities are learned in a single framework based on raw data traces solely annotated with frame-to-frame point cloud correspondences.

Overall, the control performance shown by our SE3-Pose-Nets is extremely encouraging and provides strong evidence that such networks can learn a consistent pose space that provides long-range correspondences for fast reactive control. While this provides reason to rejoice, there are multiple areas for improvement:

- As shown in the generalization results, SE3-Pose-Nets can exhibit non-convergent behavior under some perturbations - this is primarily for joints lower down the kinematic chain (joints 5–7) whose observed motions are significantly correlated to the motions of the joints higher up in the kinematic chain. Additionally, all networks struggle to control the end-effector without access to joint angle information (which also has poor visibility on depth images). There are potentially multiple ways to tackle this problem, including curriculum and active learning, and through the use of domain randomization [149] techniques to facilitate sim-to-real transfer and generalization to novel scenarios. Another way to handle the dynamics of kinematic chains could be to explicitly structure our networks to learn the relationships between the joints, similar to ideas proposed in recent work [172].

- The poses learned by SE3-Pose-Nets are globally consistent, but are not grounded to

the physical objects in the world, i.e. their positions do not correspond to the object centers and their orientations are not aligned with the principal axes. There are multiple ways the network can be structured to allow for this grounding; for example, we can constrain the predicted pose centers (or alternatively, the center of rotation) to be near the median point of the segmented objects in a given observation. Leveraging such priors to better model the dynamics of the physical world is a promising avenue for future research.

- The dynamics predicted by SE3-Pose-Nets, especially in the multi-step open loop prediction setting, need not be entirely physically consistent. For example, different links of the arm can self-collide, extend beyond joint limits or even split into pieces as there is no explicit constraint that restricts the occurrence of such behavior (except the statistics of the training data), unlike in the real world. On the other hand, these physical constraints can also be exploited to provide interesting self-supervision for training data for our networks; figuring out which constraints are relevant (e.g., contact, self-collision, joint limits) and ways to use them to streamline training is a promising area for future work.

- SE3-Pose-Nets predict rotations close to identity, choosing to encode rotations in terms of translations by stretching the latent pose embedding space. This is because rotations are non-linear and locally resemble translations; SE3-Pose-Nets are trained on single-step prediction tasks where the non-linearity of the rotation is not easily reflected in the observed scene flows. One potential way to correct this is to train SE3-Pose-Nets on multi-step sequences (with supervised data association labels) which can help de-correlate the effects of translation and rotation. We did some preliminary work along this direction; our architecture Dynamics-Nets which we briefly present at the end of this chapter (Sec. 3.7), operates on sequences of observations from dynamic tasks, predicting positions and rotations that are highly consistent with ground truth data, without explicit supervision.

- SE3-Pose-Nets are limited to modelling kinematic motion of systems of rigid bodies as the pose-mask encoder takes in a single image observation to predict a pose; higher level dynamics such as velocities, accelerations and other information obtained potentially through integrating temporal information are currently omitted. A key area for future work is to extend our architecture to handle truly dynamic settings; as mentioned before, we have seen encouraging initial results with our Dynamics-Nets architecture (see Sec. 3.7) which can model the dynamics of complex under actuated 2D systems such as the cartpole and a 2-link manipulator. Dynamics-Nets take advantage of multi-step data associations, a recurrent, highly structured dynamics model [160] and finite-differenced velocity estimates [82] to learn a pose space that is highly correlated with the true pose space of relevant objects. When combined with model predictive controllers such as Model-Predictive Path Integral control (MPPI) [167], Dynamics-Nets achieve good control performance, even on highly dynamic under actuated tasks. For a brief discussion on the architecture of Dynamics-Nets please refer to the end of the chapter (Sec. 3.7).

- As described above, SE3-Pose-Nets detect objects in the scene, predicting a globally consistent pose for each relevant object; such detection systems struggle to model unseen and novel objects as the network internally has no reference to compare the novel object to, thereby making the detection and assignment problem hard. In concurrent work [99], we looked at a reformulation of the detection problem to instead tracking relative poses of objects between pairs of temporally close frames in a video sequence. This reformulation allows us to focus the attention on differences between pairs of frames which in turn generalizes well to unseen objects. Our proposed architecture, Motion-Nets, has a separate network each for predicting segmentation, rotation and translation between pairs of frames, which are cropped to center around the object of interest. By considering pairs of frames and looking at relative predictions, the system can easily generalize to unseen, novel objects; as we show in Leeb et al. [99] this architecture

can also be used for long-range visuomotor control of arbitrary unknown objects in simulated environments. We refer the reader to Leeb et al. [99] for a detailed discussion of the approach and results.

- A potential limitation of the current visuomotor control approach is the specification of the target/goal as an image. This is a chicken-and-egg problem as in order to get an image of the goal we need to have reached the corresponding scene, which amounts to already solving the task. An alternative approach is to specify the goal through the use of natural language commands, which can then be translated to images or latent space embeddings for visuomotor control. We encourage the reader to look at our work in collaboration with Paxton et al. [123], where we propose a joint learning system that tackles the task of stacking blocks in simulation from natural language commands.

- Finally, a key area for future work is in extending our system to interact with and manipulate external objects, as opposed to visual servoing of a manipulator. Several key challenges need to be tackled to achieve this: 1) Most object interaction tasks such as grasping, lifting and stacking require multi-step planning; local control algorithms such as the one described in Alg. 2 are susceptible to local optima and would most likely fail on such long-horizon tasks. 2) SE3-Pose-Nets and even the earlier SE3-Nets rely on the availability of a pre-collected training set of relevant data; while this is relatively easier when controlling just the manipulator, it becomes much harder when interacting with objects. For example, trackers like DART [132], used in our approach for generating supervision, can struggle in the presence of interactions & occlusions and setting up a manipulation system to generate useful data for tasks like stacking can involve significant work. This makes it hard to extend such approaches to complicated real world tasks. 3) Lastly, even when there is adequate data available, closed-loop control with a pre-trained model may not succeed due to data distribution mismatch, i.e. covariate shift [128]. This necessitates the need to generate on-policy samples which can then be used to further improve the model, in a self-reinforcing loop.

As an alternative to this supervised learning setup, we explore the use of reinforcement learning for learning manipulation tasks. Recently, reinforcement learning (RL) has been shown to be successful on many challenging control problems [37, 90], leading to state of the art results even when learning directly from images [112, 103, 83, 126]. RL approaches can learn from scratch by collecting experience on-policy and can also represent complex closed-loop policies through the use of neural networks. In the next chapter (Chapter 4), we explore integrating deep visual predictive models (albeit unstructured unlike SE3-Pose-Nets) with model-free RL approaches, showing robust performance on learning long-horizon manipulation tasks like stacking, both from scratch and in transfer settings.

Before concluding this chapter, we briefly discuss one extension to the SE3-Pose-Net architecture, the Dynamics-Net, which extends the capability of our network to handle dynamic tasks. This work was done in collaboration with an undergrad whom I advised – Felix Leeb.

3.7 Dynamics-Nets

As discussed earlier, the SE3-Net and SE3-Pose-Net architectures are limited by their single-step predictive nature; while SE3-Pose-Nets can handle multi-step rollouts via the learned latent pose space, it is not sufficient for handling dynamic settings, where higher order information such as velocity may be needed. To overcome this limitation, we propose Dynamics-Nets, a recurrent, multi-step extension to the SE3-Pose-Net architecture for handling dynamic tasks.

3.7.1 Architecture and Training

Figure. 3.10 shows the architecture of Dynamics-Nets. We state a few key points:

- Similar to SE3-Pose-Nets, the network has three components: 1) a pose-mask encoder that predicts k-object masks $\mathbf{m}$ and their corresponding object poses $\mathbf{p}$ given the input

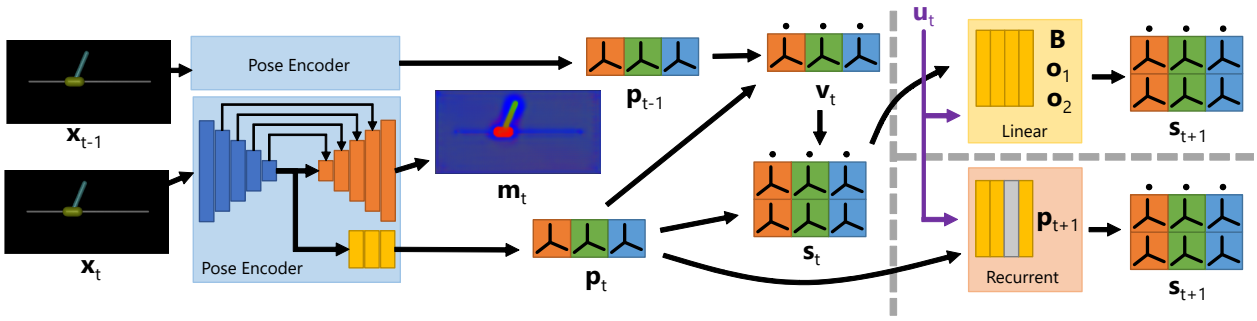

Figure 3.10: Outline of the network architecture of DYNAMICS-NETS. Similar to SE3-POSE-NETS, we use a pose-mask encoder which predicts the pose of all objects in the scene $\mathbf{p}$ and their corresponding segmentation masks $\mathbf{m}$. These poses along with the controls $\mathbf{u}$ are fed into the pose transition network. We show two different transition networks: 1) The feed-forward *Linear* model (top-right), takes in the state $\mathbf{s}_t$ which consists of the pose $\mathbf{p}_t$ and a finite-differenced delta pose $\mathbf{v}_t$ along with the control to predict the next pose $\mathbf{p}_{t+1}$. 2) The *Recurrent* transition model (bottom-right) takes in the pose $\mathbf{p}_t$ and the control to predict the next pose $\mathbf{p}_{t+1}$. This network has a recurrent layer (GRU) which allows it to integrate temporal information across sequences of input observations. Similar to SE3-POSE-NETS, these networks are trained via next scene prediction using supervised data-associations and the consistency loss, albeit across sequences. The blue layers are convolutional, red are deconvolutional, yellow are fully-connected, and the grey is recurrent.

observation, 2) a transition model that operates in the latent pose space that takes in a pose $\mathbf{p}$ and an action $\mathbf{u}$ to predict the next pose $\mathbf{p}_{t+1}$, or a pair of poses $(\mathbf{p}_t, \mathbf{p}_{t-1})$ (which can be finite differenced to generate a velocity estimate $\mathbf{v}_t = \frac{\mathbf{p}_t - \mathbf{p}_{t-1}}{\alpha}$) and an action $\mathbf{u}$ to predict the next pose and 3) an $\mathbb{SE}(3)$ transform layer (see Sec. 3.3.3 for more details).

- Unlike SE3-Pose-Nets, we train Dynamics-Nets on sequences of observations, also predicting a sequence of future observations. This training allows us to: 1) Handle dynamic scenes such as those with interactions that cannot be predicted with information from a single frame alone, 2) Train a significantly more consistent pose space that is physically correlated with the real pose space, and can predict much better rotation estimates (as can be seen in Sec. 3.7.2), and 3) Combine the resulting model with long-range planner that can generate controls even for highly dynamic environments.

- We use the flow loss and the pose consistency loss from SE3-Pose-Nets (see Sec. 3.3.4) to train Dynamics-Nets. Unlike SE3-Pose-NETSthough, we take in a sequence of input observations and controls and make open-loop multi-step future predictions – we compute the loss at each of these future predictions and backpropagate the gradients through time to train our model parameters.

- We experiment with two different transition models:

  Linear: This model (see Fig. 3.10, top-right) takes as input a pair of poses $(\mathbf{p}_t, \mathbf{p}_{t-1})$ and computes a state estimate $\mathbf{s}_t := (\mathbf{p}_t, \mathbf{v}_t)$ where the velocity $\mathbf{v}_t$ is estimated via finite differencing. It uses this state estimate $\mathbf{s}_t$ and the input control $\mathbf{u}_t$ to predict an acceleration which is linear with respect to the control, and then integrates that, together with some offsets to compute the next position and velocity. Instead of directly predicting these though, the model estimates the parameters $\hat{\mathbf{w}}_t, \hat{\vec{b}}_t, \hat{\vec{o}}_t$ which are used

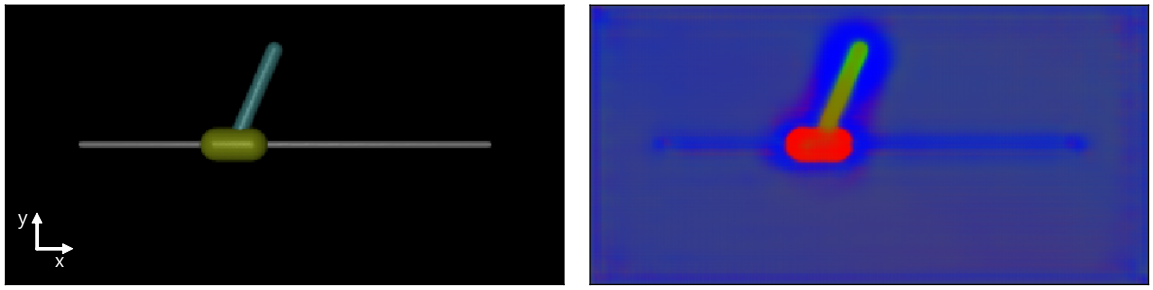

Figure 3.11: Left: Snapshot of the cartpole environment. Right: The learned mask output by the pose encoder for the snapshot on the left, where each channel corresponds to one mask. Even though the motion of the cart and pole are highly correlated, the encoder is able to differentiate the objects, albeit with some slight overlap near the joint.

in the following way:

$$\begin{aligned} \hat{\mathbf{w}}_t, \hat{\vec{b}}_t, \hat{\vec{o}}_t &= h^{\text{lin}}(\mathbf{s}_t) \\ \hat{\mathbf{a}}_{t+1} &= \hat{\mathbf{w}}_t * \mathbf{u}_t + \hat{\vec{b}}_t \\ \hat{\mathbf{v}}_{t+1} &= \mathbf{v}_t + \alpha \hat{\mathbf{a}}_{t+1} \\ \hat{\mathbf{p}}_{t+1} &= \mathbf{p}_t + \alpha \hat{\mathbf{v}}_{t+1} + \hat{\vec{o}}_t \end{aligned} \tag{3.11}$$

Recurrent: A recurrent transition model (see Fig. 3.10, bottom-right) that takes in a pose $\mathbf{p}_t$ and control $\mathbf{u}_t$ to directly generate the next pose. Unlike the transition model in SE3-Pose-Nets, this network has a recurrent layer (GRU) which allows it to integrate information temporally – using the sequence based training mentioned earlier we can implicitly capture information such as velocity and acceleration that can help us make better predictions for dynamic settings.

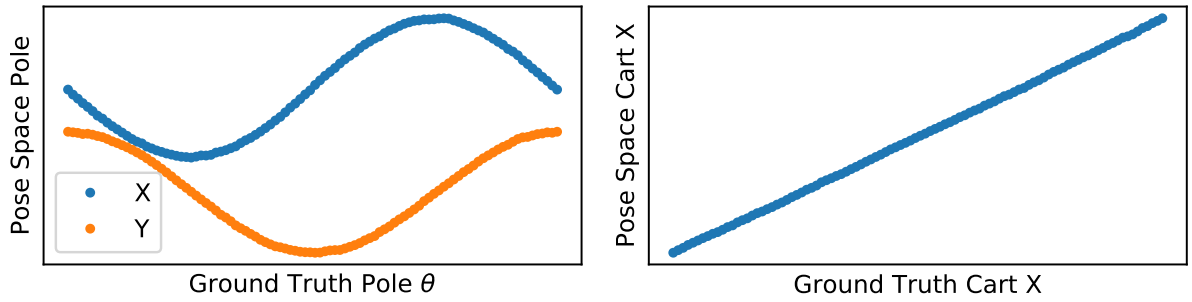

Figure 3.12: Plot showing the dimensions in the pose space corresponding to the predicted pole X and Y position as the ground truth pole angle is varied (left), and the predicted cart X position while varying the ground truth cart X position. As can be seen, the learned pose space is very consistent with the true positions of the cart and pole.

Model	# of Rollouts	Length	Av. Loss	Speed (Hz)
Linear	20000	8	0.37	**36**
Recurrent	20000	8	0.43	11
Ground Truth	50	20	**-0.58**	4

Table 3.3: Comparison of the control performance using MPPI [167] with both the transition models. All hyper parameters in MPPI are the same for the linear and recurrent models. Although neither of our transition models can match the ground truth model, there is a significant speed up in control when using a neural network as opposed to the simulator, especially for the linear model.

3.7.2 Experimental Results

We apply DYNAMICS-NETS to model the well-known 2D cartpole task, which has very dynamic interactions. We used the Mujoco [151] simulation environment to simulate the task, with a timestep of 0.02 seconds. Our observations are RGBD images of 320 x 240 resolution. We use a dataset of approximately 25000 samples where each sample is a short sequence generated by sampling a random initial state and applying random actions to train our model – we also added a few samples showing the cart being balanced. We set the number of objects $k = 3$ and train across multi-step sequences as discussed earlier. We present a few key results:

- Similar to SE3-POSE-NETS, DYNAMICS-NETS can segment the objects in the scene extremely well, even without direct supervision on the segmentation (see Fig. 3.11 for an example).

- Interestingly, the pose space predicted by DYNAMICS-NETS is highly correlated with the true pose space, as can be evidenced by the results from Fig. 3.12 which plots parts of the learned pose space as a function of the cart's motion. DYNAMICS-NETS can learn good rotation estimates (Fig. 3.12, left) even when not given explicit supervision – this is mainly possible due to the multi-step training of these networks. Similarly, the learned positions are also well correlated with the true observed motions (Fig. 3.12, right).

- We also tested the performance of DYNAMICS-NETS when combined with a sample-based forward planning algorithm – Model-Predictive Path Integral Control (MPPI) [167]. Similar to SE3-POSE-NETS we used latent space losses for the optimization – the task is to balance the pole vertically. Unlike SE3-POSE-NETS we used multi-step trajectory optimization with multiple rollouts, all executed in parallel, for the optimization. Table. 3.3 presents these results where "Ground Truth" indicates using the Mujoco simulator as the model. In general, both our models succeed in lifting the pole with varying degrees of success when balancing – this is especially sensitive when the cart

hits the ends of the rail where prediction error can be high. While our approach is not as robust compared to MPPI with the true simulator, it is significantly faster – MPPI with the linear model is 9x faster with little loss in performance.

Overall, we have seen very encouraging initial results with DYNAMICS-NETS: they can be used for controlling complex dynamical systems such as the cartpole directly from raw observations. There are a few interesting questions and areas for future work: 1) Extending the paradigm to 3D manipulation tasks such as the ones explored earlier in the chapter, 2) Reducing data needs and removing dependence on data-association supervision for training, 3) Exploring the use of other model-based optimizers such as iLQG [152], and 4) Investigating additional priors and structure to robustify learning.

Part II

MODELS IN REINFORCEMENT LEARNING

Chapter 4

IMAGINED VALUE GRADIENTS[1]

The last decade has seen significant progress in reinforcement learning techniques. The field has matured from primarily solving problems with short temporal horizons or low-dimensional policy representations to a state where RL can now solve challenging simulated motor control problems [72, 120] or games [139, 156], in many cases directly from high-dimensional inputs such as images [112, 83, 41]. In particular, significant advancements have been made in off-policy model-free algorithms suitable for high-dimensional continuous action spaces and motor control tasks [103, 73, 3]. These have improved in robustness and data-efficiency to an extent that it is now becoming possible to solve non-trivial contact-rich tasks directly on robotics hardware [2, 64, 181, 84].

Model-free techniques directly learn a policy mapping states to actions (and value function mapping states to expected rewards) from environment interactions. Their simplicity, generality and versatility has been a major factor behind their recent successes. At the same time, these techniques do not entirely satisfy the basic intuition that a learning agent should be able to acquire approximate knowledge of the dynamics of its environment, i.e. a model, in a task-independent manner. Such a model would then be transferable to new tasks in the same domain in a more effectively way than task-specific objects such as policies. It is this intuition and, more practically speaking, the desire to further improve on the sample-efficiency and transferability of model-free solutions that has driven much of the recent research in model-based RL.

A growing body of literature is concerned with learning dynamics models for physical control problems [36, 40], including approaches that learn latent models from images [160, 65,

[1]WORK DONE DURING AN INTERNSHIP AT GOOGLE DEEPMIND.

178]. Although it has has been possible to achieve data efficient control in some cases [36, 101], more generally, model-based methods have not yet been able to achieve the same robustness and versatility as model-free techniques, especially in the presence of model inaccuracies and long planning horizons. There has also been work combining learned dynamic models with policy learning [144, 63, 163, 44, 162], however, in such cases the advantages over purely model-free techniques can be less clear.

As we discussed in chapter 1, this dichotomy between the flexibility of model-free methods and the structure and potential transferability of model-based approaches is similar to the bias-variance tradeoff in machine learning. In this work, we explore combining these two paradigms by explicitly learning the dynamics of the environment, i.e. a model, in addition to stable learning of a stochastic parametric policy and value function as is common in modern model-free reinforcement learning. We build on the idea of Stochastic Value Gradients (SVG) [73] which provide a way of computing low-variance policy gradients by directly differentiating a model-based value function. We extend this work in two ways: Firstly, since the true low-dimensional system state is not directly observed, we develop a latent state space model that allows us to reliably predict expected future reward and value as a function of the current policy and is suitable for domains with high-dimensional observation spaces such as images. Secondly, rather than using the model only for credit assignment along observed trajectories we use the model directly to produce "imagined" rollouts and show that stable policy optimization can still be achieved in this setting.

Unlike the previous chapter where we considered controlling the manipulator, we apply our algorithm to several challenging long-horizon vision-based object manipulation tasks (e.g. lifting and stacking) in simulation and demonstrate the following: (a) Our combined approach is as robust and achieves the same asymptotic performance as competitive model-free baselines; (b) in several cases it also significantly improves data-efficiency. (c) Our approach can effectively transfer the learned model to novel tasks with different reward distributions or visual distractors, leading to a dramatic gain in data-efficiency in such settings. (d) Our approach is particularly effective in a multi-task setup where the models learned on multiple

tasks learn faster and generalize better to new tasks, successfully solving problems that cannot be learned in isolation. We believe that our work presents a structured way to combine model-based reasoning with learning complex interaction policies from raw data, potentially paving the way for a general hybrid approach to solving robot manipulation.

4.1 Related Work

Model-free RL: Model-free Reinforcement learning (RL) has recently seen tremendous success on challenging problems [37, 90], achieving human level performance in Atari and Starcraft games [112, 156], beating the world champion at Go [139], and generating state of the art results in non-linear control and manipulation tasks in contact-rich environments [103, 133, 3, 126]. With recent advances in deep learning [94], these methods can now scale to high-dimensional state and observation spaces [112, 126, 3], often representing policies, value, Q-value and reward functions as large neural networks with potentially millions of parameters. In spite of these advancements, several key challenges exist that limit their applicability to general real world robotics tasks. First, model-free RL requires large amounts of data to learn, leading to poor data efficiency. Second, model-free methods transfer poorly; the learned policies, value and reward functions are task-specific and cannot be used to quickly transfer, even across related tasks.

Model-based RL: On the other end of the spectrum, model-based methods learn a model of the environment and use a planner to generate actions. One of the recent successes in this area is work by Deisenroth et al. [36] which learned uncertainty aware policies using Gaussian-Process models and showed impressive data efficiency on multiple low-dimensional tasks. Using deep neural networks, model based RL has been extended to vision-based problems such as Atari games [83] and complex manipulation tasks with rich contact dynamics [101, 40]. Further work has also looked at learning vision-based latent representations for RL [160, 157, 178, 161, 65]; finding that learned low-dimensional representations can provide good, task-specific, feature spaces in which learning and planning can proceed significantly faster compared to learning directly from images. A major drawback of these methods,

though, is their reliance on the model to generate accurate predictions of the future; in the presence of model approximations and inaccuracies these methods tend to perform poorly. Additionally, many of these methods rely on a forward planning algorithm (e.g. iLQR [152], CEM [129]) to generate policies; planning can be difficult for long-horizon tasks and in sparse reward settings.

Model-free + Model-based: Several papers have focused on combining the strengths of model-free and model-based RL. The work on Dyna [144] was among the first. It integrates an action model and model-based imagined rollouts with policy learning, interleaving planning, learning and execution in a tight loop, albeit applied only to toy RL problems. Recent work has further explored combining these paradigms through the use of imagined rollouts generated with learned models for accelerating learning of model-free policies [63, 163] – or indirectly speeding up learning via model-based value estimates [44, 21]. These approaches propose different means to handle model approximation errors, such as using short rollouts to avoid cascading model errors [44], uncertainty estimation through model ensembles [21] and using the rollouts as policy conditioning variables only [163]. Contrasting these methods, Stochastic Value Gradients (SVG) [73] re-evaluates rollouts with a learned model from off-policy data, accelerating learning through value gradients back-propagated through time via a learned model. In this work, we extend SVG to use imagined rollouts in latent spaces, accounting for model approximation errors via gradient averaging.

There has also been work on combining latent space models and model-free RL. DeepMDP [58], MERLIN [162], CRAR [55] and VPN [119] combine representation learning with policy optimization via auxiliary losses such as observation reconstruction [162], predicting the next latent state [58] or future values [119]. Along these lines, we learn a latent representation that can be used for predicting (expected) future latent states, observations, rewards and values conditioned on the observed actions.

Transfer: A major argument in support of model-based RL is the potential of the learned model to transfer across tasks in the same environment. Sutton et al. [144] discuss early examples of this type of transfer on simple problems. Recently, Francois et al. [55] showed

encouraging results with a vision-based RL agent on 2D labyrinth tasks. Nagabandi et al. [114, 115] meta-learn predictive dynamics models to enable a model-based agent to rapidly adapt to changes in its environment. Other work on transfer in RL has focused on learning reusable skills e.g. in the form of embedding spaces [68, 53, 148], successor representations [13], transferable priors [57] or meta-policies [45, 32]. As they learn "behaviors" rather than dynamics models, they are in a sense orthogonal and complementary to the ideas presented in this work.

The rest of this chapter is organized as follows. We first present some background in Sec. 4.2 followed by our approach, which is split into two parts: we discuss learning the model, value and reward functions in Sec. 4.3 followed by the policy learning in Sec. 4.4. We then present our experimental results in Sec. 4.6 and conclude with a discussion in Sec. 4.7.

4.2 Background

We are interested in solving motor control problems such as robotic manipulation tasks from vision. This setup can be formalized as a partially observed Markov decision process (a POMDP) with observation $\mathbf{o}^t \in R^{N_O}$, states $\mathbf{s}^t \in R^{N_S}$, actions $\mathbf{a}^t \in R^{N_A}$ transition probabilities $p(\mathbf{s}^{t+1}|\mathbf{s}^t, \mathbf{a}^t)$, and reward function $r^t = r(\mathbf{s}^t, \mathbf{a}^t)$. Let $\tau_{\leq t} = (\mathbf{o}^{1:t}, \mathbf{a}^{1:t-1})$ be the sequence of observations and actions in a trajectory up to decision point t. The optimal policy and the value function at time step t in this setting are a function of the posterior of the system state $p(\mathbf{s}_t|\tau_{\leq t})$ given the interaction history. However, this posterior is usually intractable and many reinforcement learning approaches resort, instead, to directly optimizing a parametric policy that is a function of the history $\tau_{\leq t}$, i.e. they consider policies of the form $\pi(\mathbf{a}_t|\tau_{\leq t})$. Thus, they aim to maximize the sum of discounted rewards:

$$J(\theta) = \mathbb{E}_{p_\pi(\mathbf{s}^{t:T}, \mathbf{a}^{t:T}, \mathbf{o}^{t+1:T}|\tau_{\leq t})} \left[\sum_{t=0}^{T} \gamma^t r^t \right] \tag{4.1}$$

with discount factor $\gamma \in [0, 1)$, where the trajectory distribution is assumed to decompose into transition and action probabilities as

$$p_\pi(\mathbf{s}^{t:T}, \mathbf{a}^{t:T}, \mathbf{o}^{t+1:T} | \tau_{\leq t}) = p(\mathbf{s}^t | \tau_{\leq t}) \prod_{t' \geq t}^{T-1} \pi(\mathbf{a}^{t'} | \tau_{\leq t'}) p(\mathbf{s}^{t'+1} | \mathbf{s}^{t'}, \mathbf{a}^{t'}) p(\mathbf{o}^{t'+1} | \mathbf{s}^{t'+1}). \tag{4.2}$$

From this, we can also also define the value function of an observed trajectory as

$$V^\pi(\tau_{\leq t}) = \mathbb{E}_{p_\pi(\mathbf{s}^{t:T}, \mathbf{a}^{t:T}, \mathbf{o}^{t+1:T} | \tau_{\leq t})} \left[\sum_{k=t}^{T} \gamma^{k-t} r^t \right], \tag{4.3}$$

where, below, we consider the common infinite horizon case; $\lim_{T \to \infty}$.

4.3 A Predictive Model of Observations and Rewards

Naively, a model-based evaluation of 4.3 would require an accurate predictive model of future observations and rewards given a a partial trajectory $\tau_{\leq t}$. Especially in high-dimensional observation spaces such a model can be difficult to learn. Instead we consider an approximate model suitable for partially observed domains with limited stochasticity. We develop a latent state-space model whose latent state at time step t is optimized to represent a sufficient statistic of the interaction history $\tau_{\leq t}$. The model is trained to predict expectations of future observations and reward by approximately modeling the evolution of the summary statistic via a deterministic transition model.

We express the policy as a function of the latent state and the model allows us to construct a surrogate model $\tilde{V}$ that expresses the value $V^\pi(\tau_{\leq t})$ as a recursive function of the policy, the deterministic transition function, and a learned approximation to the reward. This surrogate model can be used to compute approximate policy gradients.

More specifically, we let

$$\mathbf{h}^t = f_{\text{enc}}(\mathbf{o}^{1:t}; \phi)$$

be a deterministic mapping (with parameters ϕ) extracting a summary statistic from a history

of observations[2]; and we let

$$\mathbf{h}^{t+1} \triangleq f_{\text{trans}}(\mathbf{h}^t, \mathbf{a}^t; \phi)$$

denote a latent transition function. The assumption that the latent dynamics are well described by a deterministic transition function is our primary simplification. We further define the approximate reward function based on latent states as $\hat{r}(\mathbf{h}^t, \mathbf{a}^t; \phi)$. Finally, we denote with $\pi_\theta(\mathbf{a}|\mathbf{h})$ a stochastic policy, parameterized by θ.

Using these definitions we construct an approximation of the expectation in Eqn. 4.3 as

$$V^\pi(\tau_{\leq t}) \approx \tilde{V}^\pi(\mathbf{h}^t) = \mathbb{E}_{\pi_\theta}\left[\sum_{k=t}^{\infty} \gamma^{k-t} \hat{r}^k | \mathbf{h}^t = \mathbf{h}\right] \tag{4.4}$$

or, written recursively:

$$V^\pi(\tau_{\leq t}) \approx \tilde{V}^\pi(\mathbf{h}^t) = \int \left[\hat{r}(\mathbf{h}^t, \mathbf{a}; \phi) + \gamma \tilde{V}^\pi(\mathbf{h}^{t+1}) \middle| \mathbf{h}^{t+1} = f_{\text{trans}}(\mathbf{h}^t, \mathbf{a}; \phi)\right] \pi_\theta(\mathbf{a}|\mathbf{h}^t) \text{d}\mathbf{a}, \tag{4.5}$$

where the initial latent state is given as $\mathbf{h}^t = f_{\text{enc}}(\mathbf{o}^{1:t}; \phi)$ and we have used the fact that the transition dynamics is assumed to be deterministic, i.e.: $\mathbf{h}' \triangleq \mathbf{h}^{t+1} = f_{\text{trans}}(\mathbf{h}^t, \mathbf{a})$.

In the following we will describe our approach in two steps: We first describe how to learn the predictive model for f_{enc}, f_{trans}, $\hat{r}$, and thus $\tilde{V}$ in Sec. 4.3.1. We then explain in Sec. 4.4 how the model can be used to optimize the policy.

4.3.1 Model Learning

We need to estimate all quantities comprising Equation 4.5; i.e., we need to estimate the following parametric functions (for brevity we use a single set of parameters ϕ for all model parameters):

Encode $f_{enc}(\mathbf{o}^{1:t}; \phi) = \mathbf{h}^t \approx \mathbb{E}_{p(\mathbf{s}|\mathbf{o}^{1:t})}[\mathbf{s}^t]$

Transition $f_{trans}(\mathbf{h}^t, \mathbf{a}; \phi) = \mathbf{h}^{t+1}$

Decode $f_{dec}(\mathbf{h}^t; \phi) \approx \mathbb{E}_{p(\mathbf{o}|\mathbf{s}^t, \tau_{\leq t})}[\mathbf{o}^t]$

Value $\hat{V}^\pi(\mathbf{h}^t; \phi) \approx V^\pi(\tau_{\leq t})$

Reward $\hat{r}(\mathbf{h}^t, \mathbf{a}^t) = \hat{r}(\mathbf{h}^t, \mathbf{a}^t; \phi) \approx \mathbb{E}_{p(\mathbf{s}^t|\tau_{\leq t}, \mathbf{a}^t)}[r(\mathbf{s}^t, \mathbf{a}^t)]$

[2]We drop the dependency on actions here and in the following, assuming that the state $\mathbf{s}$ is retrievable from a history of observations only.

We briefly describe each of these models below, but defer to Sec. 4.5 for a detailed discussion on their architecture and implementation details:

- The **Encoder** maps a history of observations $\mathbf{o}^{1:t}$ to a summary statistic or an abstract, unstructured latent state representation $\mathbf{h}^t$, via a recurrent neural network. Recurrence allows us to handle partially observable settings (eg: occlusion). Note that while this model is not structured similar to prior work (SE3-Nets, SE3-Pose-Nets), much of that work is complementary and can be integrated with this approach for further improvements in performance and generalization. We discuss this further in Sec. 4.7.

- The **Transition** function predicts the next latent state $\mathbf{h}^{t+1}$ given the current state $\mathbf{h}^t$ and action $\mathbf{a}$, evolving dynamics in the low-dimensional latent space.

- The **Decoder** maps the latent $\mathbf{h}^t$ back to an expected observation $\mathbf{o}^t$ and is primarily used as self-supervision for training the encoder [162].

- The **Value** function, predicts the sum of expected rewards (the value) as a function of a latent $\mathbf{h}^t$.

- Lastly, the **Reward** function predicts the immediate, expected reward for a given latent state-action pair.

For the model-based value function $\tilde{V}$ in Eqn. 4.5 to form a good approximation of the true value V^π (and its gradient) we train the model components on trajectories collected while interacting with the environment. The main approximation of our approach is the assumption that the evolution of the latent state is well modeled by a deterministic transition function. In partially observed and stochastic environments this is not guaranteed. For the relevant quantities to be well approximated despite this simplification we employ a number of losses that satisfy the following desiderata:

i) We want to ensure that $\mathbf{h}^t$ is a sufficient statistic of the history of observations $\mathbf{o}^{1:t}$ and the system is thus Markov in the latent state $\mathbf{h}$.

ii) We want to minimize the discrepancy between the predicted and observed evolution of the latent state $\mathbf{h}$, i.e., our open-loop predictions through the transition model should match the expected latent states generated by our encoder (which has access to the entire history of observations).

iii) Given a latent state $\mathbf{h}^t$ we want to accurately predict the expected reward and value (of policy π) at time step $t+k$ after executing some action sequence $\mathbf{a}^{t:t+k}$.

Let $\mathcal{B}$ denote a set of trajectory data collected while executing some behavior policy $\mu(\mathbf{a}|\mathbf{h})$. Let us define the full model loss after an initial "burn-in" of H steps (to ensure that the encoder has sufficient information) as

$$\mathcal{L}^N = \mathbb{E}_{p_\pi,\mu}[\sum_{t=H+1}^{H+N-1} \mathcal{L}_e(\mathbf{h}^t, \mathbf{a}^t, r^t, \mathbf{o}^{1:t+1})|f_{\text{trans}}, f_{\text{enc}}] \tag{4.6}$$

where we approximate the expectation w.r.t. $p_\pi(r^{t:T}, \mathbf{a}^{t:T}, \mathbf{o}^{t+1:T}|\tau_{\leq t})$ with samples from the buffer $\mathcal{B}$,

$$\mathcal{L}^{\mathcal{N}} \approx \mathbb{E}_{\tau\sim\mathcal{B}}\left[\sum_{t=H+1}^{H+N-1} \mathcal{L}_e(\mathbf{h}^t, \mathbf{a}^t, r^t, \mathbf{o}^{1:t+1})\Big|\mathbf{h}^{t+1} = f_{\text{trans}}(\mathbf{h}^t, \mathbf{a}^t; \phi), \mathbf{h}^H = f_{\text{enc}}(\mathbf{o}^{1:H}; \phi)\right], \tag{4.7}$$

with $\tau := (\mathbf{o}^{1:H+N}, \mathbf{a}^{1:H+N}, r^{1:H+N})$ and where the per example loss is defined as

$$\mathcal{L}_e = \mathcal{L}_f(\mathbf{h}^t, \mathbf{o}^{1:t}) + \alpha\mathcal{L}_r(\mathbf{h}^t, \mathbf{a}^t, r^t) + \beta\mathcal{L}_V(\mathbf{h}^t, \mathbf{a}^t, r^t, \mathbf{o}^{1:t+1}) \tag{4.8}$$

where α and β are coefficients that determine the relative contribution of the loss components. The per example transition model loss is given as

$$\mathcal{L}_f(\mathbf{h}^t, \mathbf{o}^{1:t}) = \|f_{\text{dec}}(\mathbf{h}^t; \phi) - \mathbf{o}^t\|_2^2 + \zeta\|f_{\text{enc}}(\mathbf{o}^{1:t}; \phi) - \mathbf{h}^t\|_2^2, \tag{4.9}$$

where the first term measures the error between the observations $\mathbf{o}$ and reconstructions from the open-loop latent state predictions ($\mathbf{h}^{t>H}$) and the second term enforces consistency

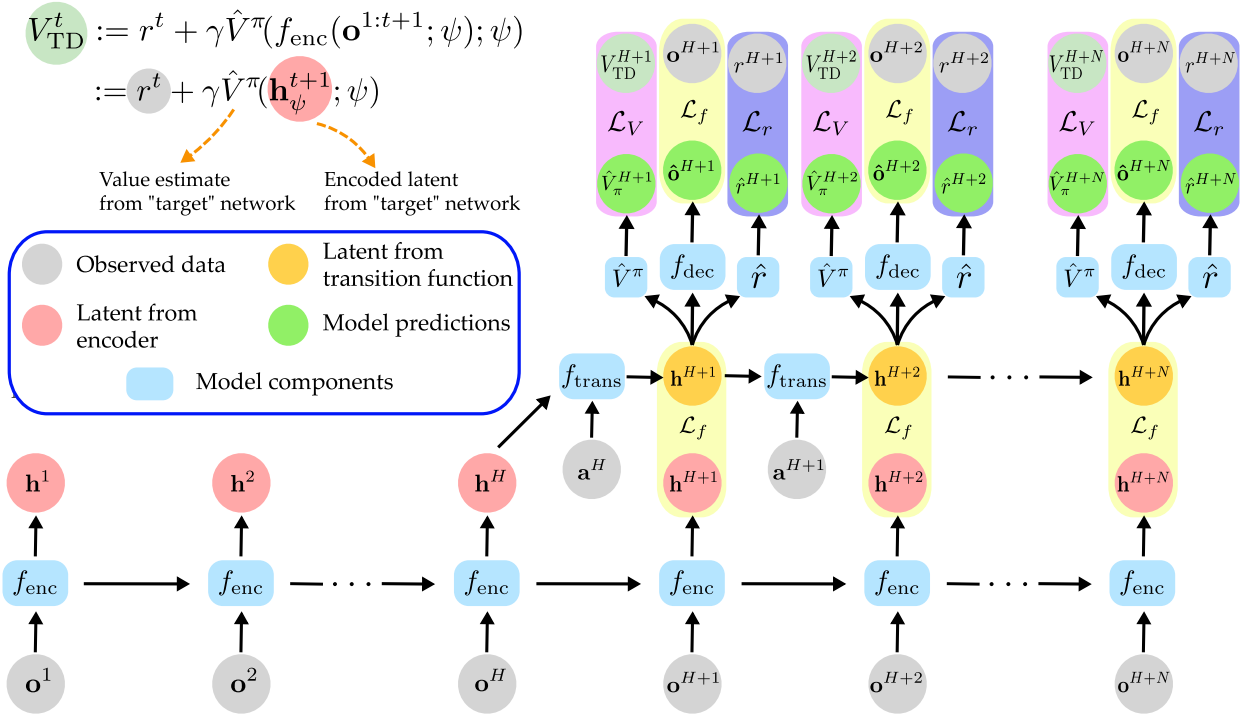

Figure 4.1: Schematic showing the rollout from on a trajectory sampled from the replay buffer $(\mathbf{o}^{1:H+N}, \mathbf{a}^{1:H+N}, r^{1:H+N}) \sim \mathcal{B}$ through the model (blue rectangles), predicting latent states (red & orange circles for encoder and transition predictions respectively) and their corresponding reconstructed observations, value and reward predictions (green circles). First, the encoder f_{enc} encodes the observations $\mathbf{o}^{1:H+N}$ to latents $\mathbf{h}^{1:H+N}$. The open-loop rollout begins after a history of H observations have been encoded to generate the latent $\mathbf{h}^{H}$. From this latent, the rollout is computed through the transition model f_{trans} using the true actions $\mathbf{a}^{H:H+N-1}$, generating the latents $\mathbf{h}^{H+1:H+N}$ (orange circles; note that these are different from the latents generated by the encoder). The transition model latents are passed through the decoder f_{dec}, value $\hat{V}^{\pi}$ and reward $\hat{r}$ estimators to generate (expected) reconstructed observations $\hat{\mathbf{o}}^{H+1:H+N}$, (expected) values $\hat{V}_{\pi}^{H+1:H+N}$ and (expected) rewards $\hat{r}^{H+1:H+N}$ which are used to compute losses for training the model (losses are highlighted with the rectangular color patches with the labels $\mathcal{L}_{(.)}$). Of special mention is the loss for training the value estimator; this uses V-trace and Temporal Difference (TD) style value targets based on a "target" network. The targets are generated by the "target" value estimator $\hat{V}^{\pi}(.\,;\psi)$ which takes in latents encoded by the "target" encoder $f_{\text{enc}}(.\,;\psi)$ along with the observed rewards r^{t}.

between the latent state representation from the encoder f_{enc} and the predictions from the transition model f_{trans}; this encourages the latent state to stay close to encodings of observed trajectories thus addressing points i) and ii) above. ζ is a coefficient that weights the two loss terms. The per example reward loss is

$$\mathcal{L}_r(\mathbf{h}^t, \mathbf{a}^t, r^t) = \|\hat{r}(\mathbf{h}^t, \mathbf{a}^t; \phi) - r^t\|_2^2, \tag{4.10}$$

and the value-loss is given by the, importance weighted, squared Bellman error

$$\mathcal{L}_V(\mathbf{h}^t, \mathbf{a}^t, r^t, \mathbf{o}^{1:t+1}) = \frac{\pi(\mathbf{a}^t|\mathbf{h}^t)}{\mu(\mathbf{a}^t|\mathbf{h}^t)} \left(r^t + \gamma \hat{V}^\pi(f_{\text{enc}}(\mathbf{o}^{1:t+1}; \psi); \psi) - \hat{V}^\pi(\mathbf{h}^t; \phi) \right)^2, \tag{4.11}$$

where the next state value $\hat{V}^\pi(f_{\text{enc}}(\mathbf{o}^{1:t+1}; \psi); \psi)$ is calculated via a "target network", whose parameters ψ are periodically copied from ϕ, to stabilize training (see e.g. [112] for a discussion). In practice we use v-trace [41] to calculate a better target value. That is we set

$$\hat{V}^\pi(f_{\text{enc}}(\mathbf{o}^{1:t+1}; \psi); \psi) \hat{=} \hat{V}^\pi_{\text{trace}}(f_{\text{enc}}(\mathbf{o}^{1:t+1}; \psi); \psi) \tag{4.12}$$

where the v-trace target is given as:

$$\hat{V}^\pi_{\text{trace}}(\mathbf{h}^t; \psi) = \hat{V}^\pi(\mathbf{h}^t; \psi) + \sum_{i=t}^{t+N-1} \gamma^{i-t} \left(\Pi_{j=t}^{i-1} \rho_i \right) \delta_i^V \tag{4.13}$$

with

$$\delta_i^V = \rho_i (r_i + \gamma \hat{V}^\pi(\mathbf{h}_{i+1}; \psi) - \hat{V}^\pi(\mathbf{h}^t; \psi)) \tag{4.14}$$

being the temporal difference error multiplied by importance weight $\rho_i = \min\left(\rho_{\text{clip}}, \frac{\pi(a_i|s_i)}{\mu(a_i|s_i)}\right)$ with $\mu(a|s)$ denoting the behaviour policy and we set $\rho_{\text{clip}} = 1$. We refer to Espeholt et al. [41] for additional details regarding v-trace. Note that both loss terms are evaluated for *predicted* latent states with gradients flowing backwards through the transition model and eventually the encoder, which addresses point iii).

A schematic showing the model-based rollout process can be seen in Fig. 4.1; given a sampled trajectory from the buffer $\tau \sim \mathcal{B}$, we encode the observations through the encoder to generate the states, run an open loop rollout through the transition model and decode the

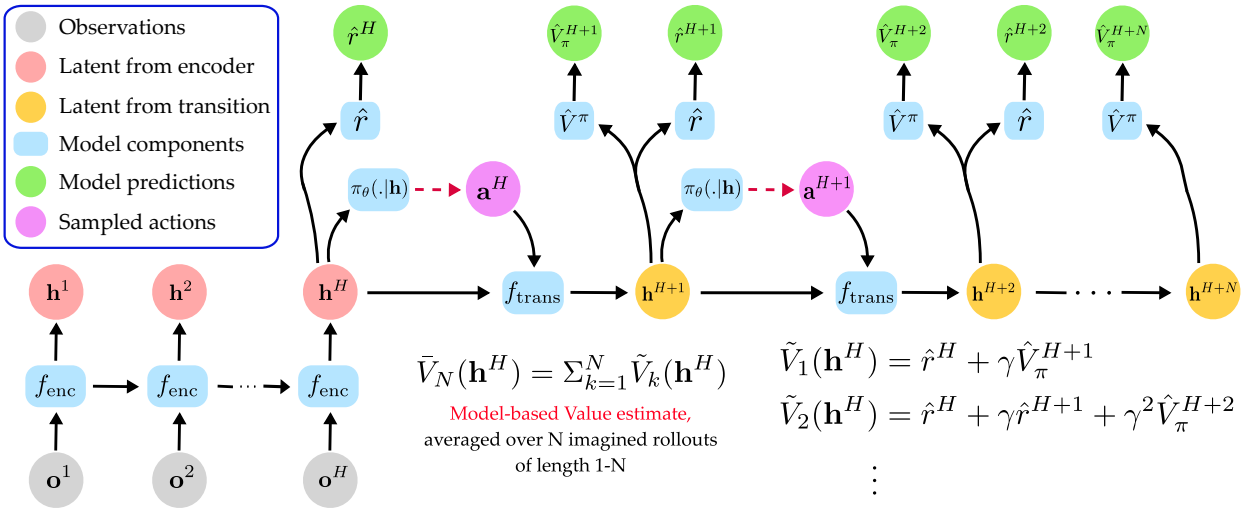

Figure 4.2: Imagined policy gradient computation. Given a history H of observations from the buffer $\mathcal{B}$, we encode a latent state $\mathbf{h}^H$ through the recurrent encoder f_{enc}, followed by an "imagined" rollout of length N – using a sequence of actions $\mathbf{a}^{t>H}$ sampled from the policy $\pi_\theta(.|\mathbf{h})$ and rolled out through the transition model f_{trans}. This leads to imagined states $\mathbf{h}^{t>H}$ with corresponding value and reward estimates (from the trained value $\hat{V}_\pi$ and reward $\hat{r}$ models). We average cumulative rewards over N horizons – computing the estimate from Eqn. 4.24 and update the policy via the gradient of this estimate. These policy gradients are back propagated through the transition model back to the policy parameters, thereby conditioning the update through the dynamics of the latent state.

observations, values and rewards from these imagined states. We then compute the model loss from Eqn. 4.7 and use its gradient to update the model parameters ϕ (see Alg. 3 for more details). We also present implementation specific details about the network architectures, loss functions used for training the model and training hyper-parameters in Sec. 4.5. Next, we discuss the policy optimization procedure.

4.4 Imagined Value Gradients in Latent Spaces

Given a model, we optimize a parametric policy $\pi_\theta(\mathbf{a}|\mathbf{h})$ by maximizing the N-step surrogate value function which is a recursive composition of the policy, the transition, reward and value function:

$$\tilde{V}_N(\mathbf{h}^t) = \mathbb{E}_{\mathbf{a}^k\sim\pi}\left[\sum_{k=t}^{t+N-1}\gamma^{k-t}\hat{r}(\mathbf{h}^k,\mathbf{a}^k) + \gamma^N\hat{V}^\pi(\mathbf{h}^{t+N};\phi)\Big|\mathbf{h}^{k+1} = f_{\text{trans}}(\mathbf{h}^k,\mathbf{a}^k)\right], \tag{4.15}$$

This N-step value can be computed by performing an "imagined" rollout of length N in the latent state-space using our model, using the discounted sum of rewards from the first $N-1$ states followed by a final bootstrapping step using the learned parametric value function (see Fig. 4.2 for a schematic of this rollout). We can further write this N-step value estimate[3] recursively in the following manner:

$$\begin{aligned}
\tilde{V}_N(\mathbf{h}^t) &= \mathbb{E}_{(\mathbf{a}^t,\mathbf{a}^k)\sim\pi}\left[\hat{r}(\mathbf{h}^t,\mathbf{a}^t) + \sum_{k=t+1}^{t+N-1}\gamma^{k-t}\hat{r}(\mathbf{h}^k,\mathbf{a}^k) + \gamma^N\hat{V}^\pi(\mathbf{h}^{t+N};\phi)\right] \\
&= \mathbb{E}_{(\mathbf{a}^t,\mathbf{a}^k)\sim\pi}\left[\hat{r}(\mathbf{h}^t,\mathbf{a}^t) + \gamma\left(\sum_{k=t+1}^{t+N-1}\gamma^{k-(t+1)}\hat{r}(\mathbf{h}^k,\mathbf{a}^k) + \gamma^{N-1}\hat{V}^\pi(\mathbf{h}^{t+N};\phi)\right)\right] \\
&= \mathbb{E}_{\mathbf{a}^t\sim\pi}\left[\hat{r}(\mathbf{h}^t,\mathbf{a}^t) + \gamma\tilde{V}_{N-1}(\mathbf{h}^{t+1};\phi)\;\middle|\;\mathbf{h}^{t+1} = f_{\text{trans}}(\mathbf{h}^t,\mathbf{a}^t)\right]
\end{aligned} \tag{4.16}$$

where we have dropped the dependency on the transition dynamics $\mathbf{h}^{k+1} = f_{\text{trans}}(\mathbf{h}^k,\mathbf{a}^k)$ in the first two equations.

We can train our policy by maximizing this value estimate by gradient ascent, exploiting the so called "value gradient" $\nabla_\theta\tilde{V}_N(\mathbf{h}^t)$ [73]; which can often be computed recursively, taking advantage of the reparameterization trick [89, 125] for sampling from π_θ and calculating analytic gradients via backpropagation backwards through time. We start by expressing the policy as a deterministic function $\pi_\theta(\mathbf{h}^t,\epsilon)$ that transforms a sample ϵ from a canonical noise distribution $p(\epsilon)$ into a sample from $\pi_\theta(\mathbf{a}|\mathbf{h})$. In the following we will consider Gaussian policy

[3] To clarify further, we denote by $\hat{V}^\pi$ the learned parametric value function and by $\tilde{V}_N$ the value estimate from an N-step rollout through our learned model.

distributions, i.e. $\pi_\theta(\mathbf{a}|\mathbf{h}) = \mathcal{N}(\mu_\theta(\mathbf{h}), \sigma^2_\theta(\mathbf{h}))$, for which the reparameterization is given as

$$\pi_\theta(\mathbf{h}, \epsilon) = \mu_\theta(\mathbf{h}) + \epsilon\sigma_\theta(\mathbf{h}) \tag{4.17}$$

with $p(\epsilon) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, where $\mathbf{I}$ denotes the identity matrix. Using this reparameterization, we can further rewrite the recursive expression for the value estimate from Eqn. 4.16 as:

$$\tilde{V}_N(\mathbf{h}^t) = \mathbb{E}_{p(\epsilon)}\left[\hat{r}\big(\mathbf{h}^t, \pi_\theta(\mathbf{h}^t, \epsilon)\big) + \gamma\tilde{V}_{N-1}(\mathbf{h}^{t+1}; \phi) \;\middle|\; \mathbf{h}^{t+1} = f_{\text{trans}}\big(\mathbf{h}^t, \pi_\theta(\mathbf{h}^t, \epsilon)\big)\right] \tag{4.18}$$

where $\mathbf{a}^t = \pi_\theta(\mathbf{h}^t, \epsilon)$ and we used the reparameterization trick to replace the expectation w.r.t action samples from the policy $\mathbf{a}^t \sim \pi$ with the expectation over the noise distribution $p(\epsilon)$.

We can now define the N-step value gradient $\nabla_\theta\tilde{V}_N(\mathbf{h}^t)$ for any state $\mathbf{h}^t$ as

$$\begin{aligned}\nabla_\theta\tilde{V}_N(\mathbf{h}^t) = \mathbb{E}_{p(\epsilon)}\big[&\nabla_\theta\hat{r}\big(\mathbf{h}^t, \pi_\theta(\mathbf{h}^t, \epsilon)\big) + \\ &\gamma\nabla_{\mathbf{h}^{t+1}}\tilde{V}_{N-1}(\mathbf{h}^{t+1})\nabla_\theta f_{\text{trans}}\big(\mathbf{h}^t, \pi_\theta(\mathbf{h}^t, \epsilon)\big) + \\ &\gamma\nabla_\theta\tilde{V}_{N-1}(\mathbf{h}^{t+1})\big]\end{aligned} \tag{4.19}$$

where $\mathbf{h}^{t+1} = f_{\text{trans}}(\mathbf{h}^t, \pi_\theta(\mathbf{h}^t, \epsilon))$ and we dropped the dependencies of all functions on ϕ for brevity. The partial value gradient $\nabla_{\mathbf{h}^{t+1}}\tilde{V}_{N-1}(\mathbf{h}^{t+1})$ w.r.t a state $\mathbf{h}^{t+1}$ in the above expression can be defined recursively as:

$$\begin{aligned}\nabla_{\mathbf{h}^k}\tilde{V}_N(\mathbf{h}^k) = \mathbb{E}_{p(\epsilon)}\Big[&\nabla_{\mathbf{h}^k}\hat{r}\big(\mathbf{h}^k, \mathbf{a}^k\big) + \nabla_{\mathbf{a}^k}\hat{r}\big(\mathbf{h}^k, \mathbf{a}^k\big)\nabla_{\mathbf{h}^k}\pi_\theta\big(\mathbf{h}^k, \epsilon\big) + \\ &\gamma\nabla_{\mathbf{h}^{k+1}}\tilde{V}_{N-1}\big(\mathbf{h}^{k+1}\big)\nabla_{\mathbf{h}^k}f_{\text{trans}}\big(\mathbf{h}^k, \mathbf{a}^k\big) \;\Big|\; \mathbf{a}^k = \pi_\theta\big(\mathbf{h}^k, \epsilon\big)\Big]\end{aligned} \tag{4.20}$$

through a standard application of the chain rule.

The case $\nabla_\theta\tilde{V}_1(\mathbf{h}^{t+N-1})$ is established by assuming the policy is fixed for all steps after N; i.e. bootstrapping with $\nabla_{\mathbf{h}^k}\tilde{V}_0(\mathbf{h}^k) = \nabla_{\mathbf{h}^k}\hat{V}^\pi(\mathbf{h}^k; \phi)$. We note that, to calculate these gradients, only an initial state $\mathbf{h}^t$ (encoded from a history of observations $\mathbf{o}^{1:t} \sim \mathcal{B}$) is required in addition to the learned model (see Fig. 4.2 for a schematic). Eqn. 4.19 computes N policy gradient contributions; using both the encoded state $\mathbf{h}^t$ and the imagined states $\mathbf{h}^{t+1} \ldots \mathbf{h}^{t+N}$. This ensures that the policy can be evaluated on either kind of latent state. Our derivation

here is thus analogous to the N-step stochastic value gradient definition from SVG [73]; but replacing observed states with imagined latent states – and assuming a deterministic transition model.

4.4.1 Stable Regularized Policy Optimization

Given the definition of the value gradient in the previous section (Eqn. 4.19) we can make a few interesting observations relevant to its use in practice.

First, we can realize that, in principle, the single-step gradient estimate $\nabla_\theta \tilde{V}_1(\mathbf{h})$ (and hence a model trained by minimizing $\mathcal{L}^1$) is sufficient for performing policy optimization. However, in this case we would obtain a biased value gradient after one gradient step in the direction of $\nabla_\theta \tilde{V}_1(\mathbf{h})$ – since at that point $\hat{V}^\pi(\mathbf{h}) \neq \tilde{V}^{\pi_\theta}(\mathbf{h})$ – which only becomes unbiased again once the dynamic programming updates from Eqn. 4.11 have converged.

To counteract this bias we could consider using the N-step gradient $\nabla_\theta \tilde{V}_N(\mathbf{h})$ for large N. Such an estimate is not affected by the above described bias for the first $N-1$ steps (since the equivalence of π and π_θ is only assumed for steps after time N). As a result, it facilitates faster learning (as also demonstrated in our experiments). A downside of this approach is that it can be more heavily affected by modelling errors; i.e. a latent state-space model predicting rewards N-steps into the future is harder to learn than a 1-step model.

As a compromise, trading-off bias with modelling errors, we found that using a simple average gradient estimate – over N horizons – worked well in practice

$$\nabla_\theta \bar{V}_N(\mathbf{h}) = \frac{1}{N} \sum_{k=1}^{N} \nabla_\theta \tilde{V}_k(\mathbf{h}). \tag{4.21}$$

This averaging linearly down weights the contributions from states further along the trajectory – the first state appears N times in the sum, N-1 times for the second state and so on; as opposed to a discount based weighting that decays slowly this can drastically reduce the effect of model errors later in the sequence. We note that, in principle, we could also use weighting terms based on the variance of different horizon estimates, but opted for an average for simplicity here.

Second, even with the averaged model-gradient from above, gradient based optimization is prone to exploiting modelling errors (in both the transition dynamics and reward/value estimates), yielding overly optimistic policies. This is a well known problem in model based RL; see e.g. [65] for a recent discussion. To counteract such effects it is hence desirable to further regularize the policy optimization step. Similar to many existing policy optimization methods [133, 3, 64] we adopt a relative-entropy (KL) regularization scheme. We augment the estimated reward with a sample based likelihood ratio term (a sample based estimate of the KL)

$$\hat{r}_{\mathrm{KL}}(\mathbf{h}, \mathbf{a}, \pi_\theta) = \hat{r}(\mathbf{h}, \mathbf{a}) + \lambda \log \frac{\pi_\theta(\mathbf{a}|\mathbf{h})}{p(\mathbf{a}|\mathbf{h})}, \tag{4.22}$$

where $p(\mathbf{a}|\mathbf{h})$ is a the prior action probability (we use $p(\mathbf{a}|\mathbf{h}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ throughout) and λ is a multiplier trading-off reward and regularization. Replacing $\hat{r}$ in Equation 4.19 with $\hat{r}_{\mathrm{KL}}$ – noting that $\hat{r}_{\mathrm{KL}}$ is differentiable w.r.t. the policy parameters θ – results in the regularized value gradient:

$$\begin{aligned} \nabla_\theta \tilde{V}_N^{\mathrm{KL}}(\mathbf{h}) = \mathbb{E}_{p(\epsilon)} \Big[& \nabla_\theta \hat{r}_{\mathrm{KL}}\big(\mathbf{h}, \pi_\theta(\mathbf{h}, \epsilon), \pi_\theta\big) + \\ & \gamma \nabla_{\mathbf{h}'} \tilde{V}_{N-1}^{\mathrm{KL}}(\mathbf{h}') \nabla_\theta f_{\mathrm{trans}}\big(\mathbf{h}, \pi_\theta(\mathbf{h}, \epsilon)\big) + \\ & \gamma \nabla_\theta \tilde{V}_{N-1}^{\mathrm{KL}}(\mathbf{h}') \Big], \end{aligned} \tag{4.23}$$

where $\mathbf{h}' = f_{\mathrm{trans}}(\mathbf{h}, \pi_\theta(\mathbf{h}, \epsilon))$ and $\nabla_{\mathbf{h}'} \bar{V}_{N-1}^{\mathrm{KL}}(\mathbf{h}')$ is, analogously, given by inserting $\hat{r}_{\mathrm{KL}}$ into Equation 4.20. To ensure that the bootstrap value for $\tilde{V}_0^{\mathrm{KL}}(\mathbf{h})$ is compatible with this regularized reward we additionally change the loss for the value function (the Bellman error); by, again, replacing $\hat{r}$ with $\hat{r}_{\mathrm{KL}}$ in Equation 4.11. The total derivative estimate we use in practice is then given as

$$\nabla_\theta \mathbb{E}_{p_\pi} \Big[\bar{V}_N^{\mathrm{KL}}(\mathbf{h}^t) \Big] \approx \mathbb{E}_{\mathbf{o}^{1:t} \sim \mathcal{B}} \Big[\frac{1}{N} \sum_{k=1}^{N} \nabla_\theta \tilde{V}_k^{\mathrm{KL}}(f_{\mathrm{enc}}(\mathbf{o}^{1:t})) \Big], \tag{4.24}$$

where we use batches of samples from the replay buffer $\mathcal{B}$ to optimize the policy on all visited states. That is we perform stochastic gradient ascent combining the gradient from Equation 4.24 with any optimization method (we use Adam [88]).

Algorithm 3: Imagined Value Gradients in Latent Spaces

Given: Empty experience dataset $\mathcal{B}$, burn-in H, rollout length N, episode length T

Each actor do:

while True **do**

Fetch policy / model parameters (θ, ϕ) from learner

Initialize empty trajectory $\tau := \emptyset$

for t = 0 to T **do**

Encode history and Apply control $\pi_\theta(f_{enc}(\mathbf{o}^{t-H:t}; \phi), \epsilon),\ \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

Observe reward r and next observation $\mathbf{o}'$

Insert $(\mathbf{o}, \mathbf{a}, r, \mathbf{o}')$ into τ

end for

Save τ in the dataset $\mathcal{B}$

end while

Learner do:

while True **do**

Sample sub-trajectory of length $H + N$ from buffer: $(\mathbf{o}^{1:H+N}, \mathbf{a}^{1:H+N}, r^{1:H+N}) \sim \mathcal{B}$

Compute $\nabla_\phi \mathcal{L}^{\mathcal{N}}$, the gradient of the model loss (Eqn. 4.7) w.r.t model, reward and value parameters ϕ

Compute the policy gradient $\nabla_\theta \mathbb{E}_{p_\pi}\left[\bar{V}_N^{\mathrm{KL}}(\mathbf{h})\right]$ (Eqn. 4.24) w.r.t policy parameters θ

Gradient ascent to update the policy: $\theta \leftarrow \mathrm{Step}_{\mathrm{adam}}(\theta, \nabla_\theta \mathbb{E}_{p_\pi}\left[\bar{V}_N^{\mathrm{KL}}(\mathbf{h})\right])$

Gradient descent to update the model: $\phi \leftarrow \mathrm{Step}_{\mathrm{adam}}(\phi, -\nabla_\phi \mathcal{L}^N)$

end while

The full learning procedure involves executing the most recent policy to populate the replay buffer $\mathcal{B}$ in parallel with model and policy optimization using the model loss from Eqn. 4.7 and the averaged policy gradient from Eqn. 4.24. Please see Algorithm 3 for more details on this procedure.

4.5 Experimental Setup and Implementation Details

4.5.1 Environment Setup

We evaluate our approach on several challenging, simulated long-horizon manipulation tasks using the Mujoco environment [151]. The environment consists of a Sawyer robot, equipped with a two-finger Robotiq gripper, interacting with objects on a tabletop; in all our experiments the robot interacts with 2-3 wooden blocks and balls of different colors (see Fig. 4.3 for an example).

Simulation details: We ran the simulation with a numerical time step of 10 milliseconds, integrating 5 steps, to get a control interval of 50 milliseconds for the agent. In this way we can resolve all important properties of the robot arm and the object interactions in simulation. All the objects used were based on wooden toy blocks and balls. For the majority of our experiments we used two cubic blocks with side lengths of 5 cm, colored red and blue. For a few additional experiments where we added visual distractors, we used a yellow cubic block of size 6 cm and a yellow ball of diameter 4 cm as the distractor.

We used a table with sides of 60 cm x 30 cm in length as the workspace of the robot in all our experiments. Objects were spawned randomly on the table surface. The robot hand is initialized randomly above the table-top with a height offset of up to 20 cm above the table (minimum 10 cm) and the fingers in an open configuration. All experiments are run on episodes with 200 steps length (which gives a total simulated real time of 10 seconds per episode).

Control: The sawyer robot is controlled via a 5D position controller based on an inverse kinematics model. That is, that agent outputs are 5D velocity commands – 3 for the robot end effector position, one for the gripper and one for rotating the gripper to change its orientation. We integrate this to generate positions which are fed into the inverse kinematics model that computes joint angles for control. The action space for all our tasks, therefore, is 5 dimensional.

Observations: We use visual observations (RGB images) and robot proprioception as

Entry	Dimensions	Unit
arm joint pos	7	rad
arm joint vel	7	rad / s
finger joint pos	1	rad
finger joint vel	1	rad / s
finger grasp	1	Binary

Entry	Dimensions	Unit
camera 1 (front-left)	3 x 64 x 64	RGB
camera 2 (front-right)	3 x 64 x 64	RGB

Table 4.1: *Left*: Proprioceptive observations used in all simulation experiments. *Right*: Image observations used in all simulation experiments (except the state based ones).

Entry	Dimensions	Unit
object i pose	7	m, au
object i velocity	6	m/s, dq/dt
object i relative pos	3	m

Table 4.2: Object feature observations, used in the state based MPO experiment. Note that these are not used for any vision based experiment. The pose of the objects is represented as world coordinate position and quaternions. In the table m denotes meters, q refers to a quaternion which is in arbitrary units (au). i denotes the id of the object; features from all objects are used as input.

the observations for all our experiments. Table 4.1 (left) shows the list of proprioception observations we use for all our experiments. These observations are concatenated to produce a 17 dimensional vector which is used as input to our models. In addition to proprioception, we use RGB images from two cameras located to the left and right of the table (in the front of the table, pointing towards the robot) as visual observations (see table 4.1, right and Fig. 4.18

in the results section). These two images (3x64x64 dimensions each) are concatenated along the channel dimensions to generate a 6x64x64 input visual observation to our model. It is worth noting that the availability of two camera views helps disambiguate most occlusions. We will explore switching to a single central view in future experiments; this significantly increases occlusions and makes the tasks strongly partially observable.

Additionally, for some state-based baseline experiments we used features of the objects (see table 4.2) in addition to the proprioceptive features as inputs to the policy.

4.5.2 Tasks & Rewards

We evaluate our approach on several challenging simulated long-horizon manipulation tasks where the robot interacts with multiple objects on a tabletop; in all our experiments the robot interacts with 2-3 wooden blocks and balls of different colors (see Fig. 4.3 for an example).

Each task is accompanied by a reward function that specifies the task; in this work we used shaped rewards for specifying all our tasks as we wanted to measure transfer efficiency rather than the capability to handle sparse reward settings. In principle, our approach can be extended to the sparse reward setting easily, in a manner similar to the Scheduled Auxiliary Control framework [126].

Below, we present details on the tasks considered in the paper. For more details on their rewards, please refer to Sec. A.1 in the appendix. We consider the following tasks:

Lift: In the lift task, the agent has to pick-up an object on the table and lift it above a certain height. We consider two variants of the task: the *Lift-R* task requires the robot to lift the red block (see Fig. 4.3, left) while the *Lift-B* task involves lifting the blue block. The reward specification for this task includes auxiliary rewards that encourage reaching the target object, grasping it and lifting it once grasped; these auxiliary rewards provide shaping to the task which makes the exploration problem significantly easier (see Sec. A.1 in the appendix for details on the reward).

Stack: In addition to the lift task, we consider a block stacking task where the robot has to place a block on top of another. Similar to the lift task there are two variants of the

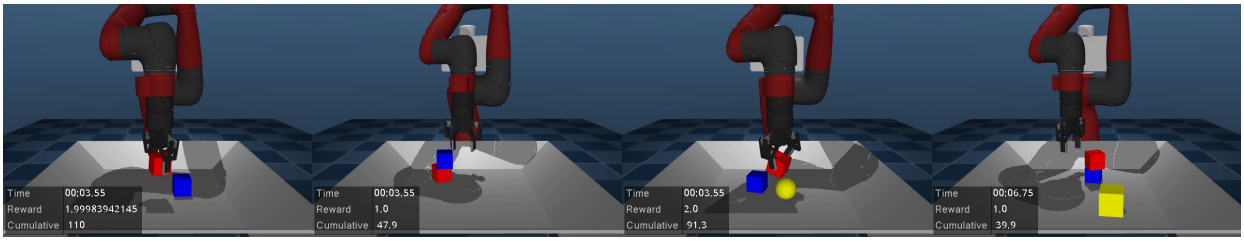

Figure 4.3: *Left*: **Lift-R**: Example scene from the lift task where the robot lifts the red block. *Center-Left*: Example scene from the **Stack-B** task. *Center-Right* and *Right*: Tasks with unseen **distractors** (yellow sphere and yellow cube) added to the scene for the Lift-R and Stack-R tasks. **Note:** The camera view shown is not the one used for generating the observations. We use two cameras located to the left and right of this current view to generate our observations (see Fig. 4.18 for an example).

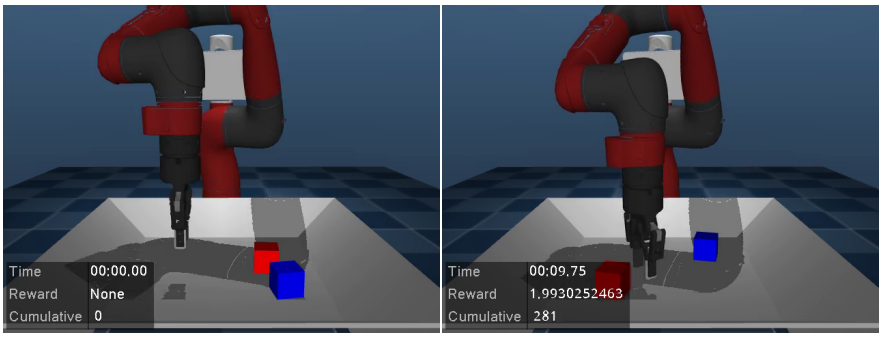

Figure 4.4: Example scenes from the match positions task. On the left is an initial image showing the blocks and the arm initialized to random starting positions. On the right, we show an image where the blocks are in their respective target positions (the blocks are always required to go to this target configuration in the task).

stack task: 1) *Stack-R*, where the agent has to stack the red block on the blue block and 2) *Stack-B*, where the agent does the opposite (see Fig. 4.3, center-left). We again introduce shaping in the reward by first encouraging the agent to lift the object – the lift reward is a part of the reward for the stack task. Additionally, once the object has been lifted we encourage the agent to move towards the target, align it with the target block and release the grasped object (see appendix Sec. A.1 for more details)

Match Positions: Unlike the lift and stack tasks where the robot primarily interacts with a single object, we consider a task where the robot has to move both blocks to a fixed target position. Fig. 4.4) shows an example scenario for this task; the left image shows a random starting configuration of the blocks and the right image shows the target configuration of the blocks in all our experiments. As this task involves moving both objects it is a nice setting for testing the generalization of our learned models as we show later in our results. Additionally, the reward is not shaped to encourage motion towards an object; there is no change in the reward unless one of the objects is moved. This means that the agent has a significantly harder exploration and credit assignment task to solve compared to the shaped lift and stack tasks, making it a substantially difficult task to learn from scratch.

Visual Distractors: We additionally consider introducing visual distractors into the environment to test the generalization of our approach to novel environments. We consider variants of the *Lift* and *Stack* tasks where we add in a third object (yellow block or ball). Fig. 4.3 (center-right and right) show examples of this task. Note that the rewards for this task are the same as the original lift and stack tasks.

Partially Observable Tasks: We also considered two tasks where we introduced randomness in the environment to model partial observability. In the first instance we added a delay to the proprioceptive features (2 timestep lag) in the Stack-R task. This results in an environment where our model now has to integrate observations temporally to estimate the robot's arm position (and velocities). In the second experiment we created a variant of the Stack-R task in which the red block (that needs to be lifted and placed on top of the blue block) changes color (switching from blue to red at random every 2 frames). We present

results on these tasks in Fig. 4.12 of the results section. Note that the rewards for these tasks are the same as their unperturbed counterparts.

Multi-task setup: Finally, we also consider a multi-task framework where our approach learns simultaneously on several tasks at once, inspired by the Scheduled Auxiliary Control framework [126]. In this setup, we have a main "extrinsic" task which is the central task to be solved. In order to facilitate learning this task, we introduce several auxiliary, "intrinsic" tasks which can be solved in addition to the main task. Conceptually, learning an auxiliary task can help make progress towards learning the main task. In this work, we consider this multi-task framework as a way towards learning models that generalize well across multiple tasks and can be transferred quickly and robustly to novel, but potentially related, manipulation tasks.

To test this, we setup a multi-task agent with the *Stack-B* task as the extrinsic task with the intrinsic tasks encouraging the robot to reach, lift and move the blue block and an additional intrinsic task that encourages the robot to reach the red block (this is unlike the other single tasks where the robot either lifts or stacks the blue or red block, but doesn't interact with both). For more details on the rewards and the different task details for the multi-task setup, please refer to Sec. A.1 in the appendix.

To train in a multi-task setup we use a task-conditioned policy, value- and reward-function. On the other hand, the learned model (i.e. $f_{\text{enc}}, f_{\text{trans}}, f_{\text{dec}}$) is not conditioned on the task – it hence has to learn consistent dynamics across tasks. We note that even though the multi-task largely consists of the same rewards as for individual experiments the data distribution is very different as the model is trained on episodes from all tasks. We additionally present more details on training the model and policy in the multi-task setup in Sec. 4.5.5; results from this setup are presented in Sec. 4.6.

4.5.3 Model & Policy

Next, we present some details on the architecture of the model and policy networks. As mentioned earlier, our latent space is unstructured unlike most of our prior work (SE3-Nets and SE3-Pose-Nets); much of that work can be integrated into this framework, leading to

additional benefits in performance and generalization.

Encoder: The encoder f_{enc} uses a recurrent, deterministic, convolutional neural network (CNN) to encode the observations $\mathbf{o}^t$ to a low-dimensional latent state representation $\mathbf{h}^t$ (see Fig. 4.5). Our observation is a pair of RGB images (3x64x64 each), concatenated along the first dimension, and a proprioception vector. The images are passed through a CNN with an initial convolutional layer followed by three residual blocks with strided convolutions [177] and average pooling to generate a vector of outputs. In parallel, the proprioception input is passed through a 2-layer multilayer perceptron (MLP) to generate a feature vector. These are concatenated and passed through a 3-layer MLP and an LSTM which outputs the latent state $\mathbf{h}^t$. As an initial pre-processing step, we normalized all our images to be between 0-1 and proprioception to -1 to 1.

Transition: The transition model f_{trans} is deterministic, taking a latent state $\mathbf{h}^t$ and action

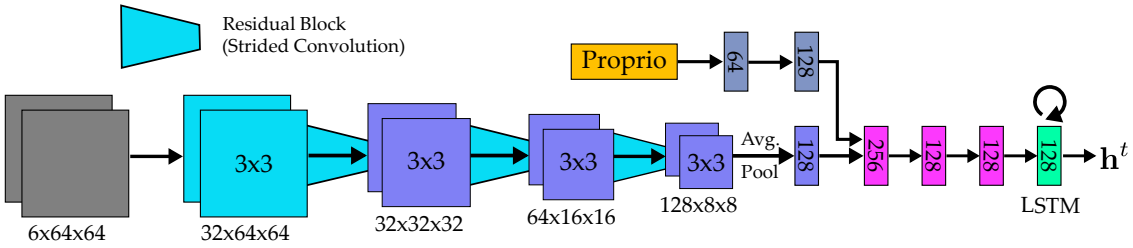

Figure 4.5: Network architecture of the encoder. The encoder takes in a pair of 64x64 RGB images concatenated along the channels axis and the proprioception observation and returns a 128-dimensional latent state vector ($\mathbf{h}$) as output. It is implemented as a recurrent residual CNN with a final LSTM layer that integrates information across time.

$\mathbf{a}^t$ to predict the next latent state $\mathbf{h}^{t+1}$ (see Fig. 4.6). Both the inputs are first passed through 2-layer MLPs. The outputs of these MLPs are concatenated and passed through another 2-layer MLP which predicts the change in latent state $\delta\mathbf{h}$. To ensure that the transition model outputs are well conditioned for long rollouts, we pass this delta change through a *tanh* layer to normalize the result to -1 to 1. This is further scaled by a linear transform and added to the input state $\mathbf{h}^t$ to generate the prediction $\mathbf{h}^{t+1}$. In practice, we saw a significant improvement

in performance when predicting the change in state as opposed to directly predicting the next state. **Decoder:** The decoder f_{dec} predicts the (expected) input observation $\mathbf{o}^t$ from

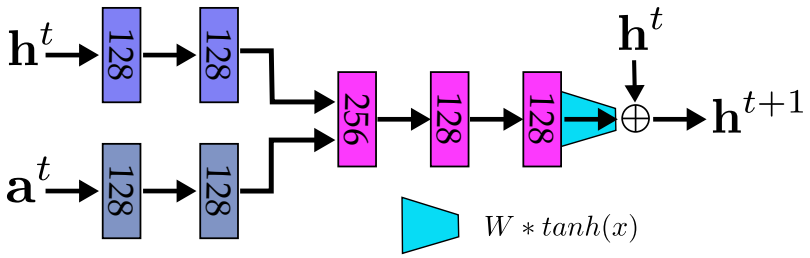

Figure 4.6: Network architecture of the transition model. The transition model takes a state ($\mathbf{h}$) and action ($\mathbf{a}$) as input and returns a prediction of the next state ($\mathbf{h}'$). It is implemented as an MLP that predicts a delta change to the state ($\delta\mathbf{s}$) which is added to the input state to predict the output.

the latent state $\mathbf{h}^t$ (see Fig. 4.7). We have two parts to the decoder: 1) To reconstruct the proprioception input, we pass the latent state through a 2-layer MLP. 2) For reconstructing the images, we first use a linear layer to transform the latent state to a 2048 dimensional vector which is reshaped into a 64x8x8 feature tensor. This feature tensor is passed through three upsampling layers, each using a bilinear additive upsampling layer [170] followed by a convolution; the output is at the same resolution as the input images. Finally, the output features are passed through a 1x1 convolution layer to get the correct number of channels and a sigmoid layer to ensure that the outputs are normalized. We also experimented with using a de-convolutional architecture for the image upsampling but found that it reduced the reconstruction quality.

Value & Reward: The value $\hat{V}^\pi$ and reward $\hat{r}$ modules predict the (expected) value $\hat{V}^t_\pi := \hat{V}^\pi(\mathbf{h}^t;\phi)$ and reward $\hat{r}^t$ from a given state $\mathbf{h}^t$ (see Fig. 4.8). Both these modules are implemented as 3-layer MLPs, with a layer norm [8] after the output of first layer.

Policy: Lastly, the policy $\pi_\theta(\mathbf{a}|\mathbf{h})$ network (Fig. 4.9) predicts a distribution over actions $\mathbf{a}^t$ from the corresponding latent state $\mathbf{h}^t$. As mentioned earlier, we consider Gaussian

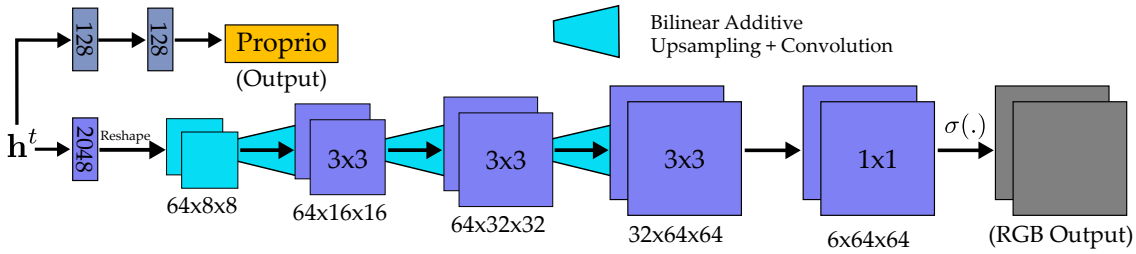

Figure 4.7: Network architecture of the decoder. The decoder takes a state ($\mathbf{h}$) as input and returns a reconstruction of the corresponding RGB images and proprioception. We use an MLP to predict the proprioception output and a mix of bilinear upsampling and convolutional layers to generate the RGB reconstructions (which are normalized to 0-1 via a sigmoid).

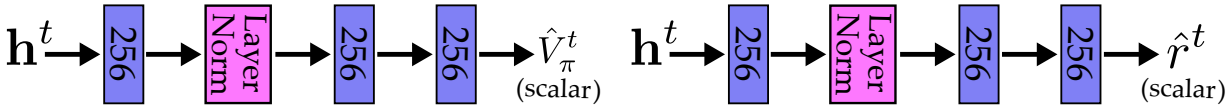

Figure 4.8: *Left*: Network architecture of the value model $\hat{V}^\pi$. The network takes the latent state $\mathbf{h}^t$ as input and uses a three fully-connected layers to predict the expected value $\hat{V}_\pi^t := \hat{V}^\pi(\mathbf{h}^t; \phi)$ (scalar). *Right*: The reward model uses the same architecture as the value model but predicts the immediate reward $\hat{r}^t$ from the state $\mathbf{h}^t$.

policy distributions i.e. $\pi_\theta(\mathbf{a}|\mathbf{h}) = \mathcal{N}(\mu_\theta(\mathbf{h}), \sigma_\theta^2(\mathbf{h}))$, from which we can sample through the reparameterization trick as $\pi_\theta(\mathbf{h}^t, \epsilon) = \mu_\theta(\mathbf{h}^t) + \epsilon\sigma_\theta(\mathbf{h}^t)$, with $p(\epsilon) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, where $\mathbf{I}$ denotes the identity matrix. We implement the policy network as a 3-layer MLP similar to the **value** and **reward** modules. Unlike those, the policy outputs the mean μ_θ and log-standard deviation $log(\sigma_\theta)$ from which we can sample an action using the reparameterization trick.

4.5.4 Training details

We implemented all our models in Python using the Tensorflow neural network package [1]. Below, we present some details on the loss functions used for training and the hyper-parameter settings.

Model Loss: We defined the per example model loss as $\mathcal{L}_e = \mathcal{L}_f(\mathbf{h}^t, \mathbf{o}^{1:t}) + \alpha\mathcal{L}_r(\mathbf{h}^t, \mathbf{a}^t, r^t) +$

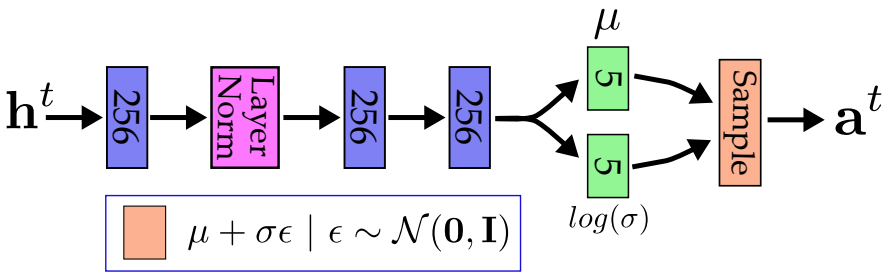

Figure 4.9: Network architecture of the policy $\pi_\theta(\mathbf{a}|\mathbf{h})$. The policy takes as input the state $\mathbf{h}^t$ and predicts the mean μ and log-variance $log(\sigma)$ of a Gaussian distribution over actions. We use the reparameterization trick to sample from this distribution by sampling $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The output is a sampled action $\mathbf{a}^t$.

$\beta\mathcal{L}_V(\mathbf{h}^t, \mathbf{a}^t, r^t, \mathbf{o}^{1:t+1})$ (see Eqn. 4.7). α and β are coefficients that determine the relative contribution of the loss components. As explained earlier, we use a squared error term for the reward loss $\mathcal{L}_r$ and a squared error to a V-trace target for the value loss $\mathcal{L}_V$. The per example transition model loss is given as

$$\mathcal{L}_f(\mathbf{h}^t, \mathbf{o}^{1:t}) = \|f_{\text{dec}}(\mathbf{h}^t; \phi) - \mathbf{o}^t\|_2^2 + \zeta\|f_{\text{enc}}(\mathbf{o}^{1:t}; \phi) - \mathbf{h}^t\|_2^2, \tag{4.25}$$

where the first term measures the error between the observations $\mathbf{o}$ and reconstructions from the open-loop latent state predictions ($\mathbf{h}^{t>H}$), the second term enforces consistency between the latent states predicted by the encoder f_{enc} and the transition model f_{trans} and ζ is a coefficient that determines the relative contribution of the two loss terms. The reconstruction loss is split into two parts (weighted equally), an image reconstruction loss and a proprioception reconstruction loss. We use a squared error term for the proprioception loss and a binary cross entropy loss term for image reconstruction; in practice we found this to result in better image reconstructions than a squared error term.

Hyper-parameters: We used ADAM [88] with default settings and a fixed learning rate of 5e-5 for all our experiments. We used the ELU [33] non-linearity as the activation function in all our networks. We initialized the final layers of our policy to predict values close to zero at the start of training; we found that this improved stability, especially in the

early stages of learning. We used a latent state dimension of $N_S = 128$ for all our experiments ($|\mathbf{h}| = 128$). We found this to be low-dimensional enough to be used for fast RL while still allowing room for expressivity.

We found that setting $\alpha = \beta = \zeta = 1.0$ gave the good results and kept this setting throughout all experiments. For the policy optimization, we set the weight of the KL regularizer to $\lambda = 0.01$ based on a hyper-parameter sweep.

We used a batch size of 32 (two learners each with a batch size of 16) to train our model and policy in all our experiments; we initially experimented with larger batch sizes of 128 but found that lowering the batch size made learning more stable. We fixed the history length $H = 3$ and experimented with different rollout lengths $N = 1, 5, 20$; as shown in our experiments $N = 5$ performed best, we use that as the default. We ran experiments for a fixed number of episodes (per actor).

Actor data generation: We used 8 asynchronous actors for data generation in all our experiments (Alg. 3, actor loop). At the start of each episode, the actor retrieves the most recent model and policy parameters. It then executes this policy a fixed time horizon of $T = 10$ seconds (episode lasts 10 seconds). The resulting trajectory is split up into smaller sub-trajectories of length $H + N$, the length needed for learning, and added to a central replay buffer which collects experience from all actors. We used a buffer containing up to $100,000$ sequences (randomly deleting old sequences when full) for all our experiments. Both our learners sample from this replay buffer, compute the gradients for the model components and policy and perform synchronized updates to the parameters.

4.5.5 Learning in the Multi-task setup

We discuss a few key differences when learning in the multi-task setup; we learn a task-specific policy, value and reward function while the model is task agnostic. Additionally, throughout learning we collect experience from all tasks in the multi-task setup giving us a diverse dataset to train a generalizable representation and model.

Model architecture: We introduce a few additional changes to the network architecture

of a few model components and the policy in the multi-task learning setup. In this setting, each task has a unique **task ID** id associated with it – this is represented as a 1-hot vector of length M (M is the number of tasks). This task ID is fed as an additional input to both the **policy** ($\pi_\theta(\mathbf{a}|\mathbf{h}, \text{id})$) and **value** modules ($\hat{V}^{\pi,\text{id}}$), thereby conditioning their predictions based on the task that is currently being considered. On the other hand, the **reward** predictor now predicts the rewards $\hat{r}^t$ for all these tasks; its output is now M-dimensional as opposed to a scalar from before. This further encourages the latent state to capture features relevant to all the learned tasks, leading to better generalization performance as witnessed in our experiments. Lastly, the architectures of the encoder f_{enc}, decoder f_{dec} and transition model f_{trans} are unchanged; these components are task agnostic and can integrate & transfer knowledge across tasks.

Actor data generation: For the multi-task experiments, at the start of each episode, the actor chooses a task to execute at random out of the M available tasks. This task is executed for the full length of the episode ($T = 10$ seconds). Random sampling of tasks can help generate diverse trajectories for training, facilitating learning of an expressive latent representation.

4.5.6 Baselines

We consider the following baselines for our experiments:

1. **SVG(0)**: this refers to the model-free version of SVG [73], i.e. it does not estimate a model of the environment. As our approach (termed Imagined Value Gradients – **IVG** from here on out) builds on SVG we expect to improve on SVG(0).

2. **MPO**: Maximum a Posteriori Policy Optimisation [3], a state-of-the-art model-free approach. Both MPO and SVG(0) operate on the same set of observations as IVG, but do not learn an explicit model.

3. **MPO-State**: To obtain an upper bound on the performance and learning speed, we

also consider a version of MPO with access to the full system state (incl. objects); this uses the proprioception information (Table. 4.1) and object features (Table. 4.2) as inputs to the policy.

Hyper-Parameters: For the baseline experiments that used pixel observations we constructed policy and value networks that are equivalent in architecture to applying π_θ after f_{enc}, similar to the networks in our IVG approach (to ensure a fair comparison).

For the state-based baselines we concatenated the true object positions, velocities and orientations (see table 4.2) to the proprioceptive robot features (see table 4.1, left). This is fed as input to 3-layer MLP policy networks (ELU activations, 200 hidden units each, layer normalization [8] after the first layer) and 3-layer MLP Q-value networks (ELU activations, 300 units each, layer normalization [8] after the first layer) which additionally take the actions (concatenated to other features) as input. To train SVG(0) we used the same relative entropy regularization technique as in for our method (using $\lambda = 1e-3$). For MPO we used the hyper-parameters from [3], which performed well across all our tests. We tuned the learning rate for both MPO and SVG(0) for performance; a rate of 1e-4 worked best.

To ensure a fair comparison between algorithms in an asynchronous setting, we ensured all algorithms ran at the same frequency of learning steps per second (which we set to 10).

4.5.7 Transferring learned models:

A key advantage of our approach compared to the baseline model-free methods is that we estimate a model of the environment; unlike task-specific constructs like the policy or value functions, models should be able to transfer to related tasks. We explore this idea in this work by learning models on source task(s), either in the single-task or multi-task learning setup, and transfer these models to novel, related tasks.

IVG model transfer: We use the following three-step procedure for transferring our models to new tasks:

1. We first train our approach, IVG, from scratch on a source task (or) a set of source

tasks in the multi-task setting. From the trained modules, we choose the following model components: the encoder f_{enc}, transition model f_{trans}, and decoder f_{dec}. Only these components are transferred to the target task as they are task-agnostic.

2. We initialize the parameters of the encoder f_{enc}, transition model f_{trans} and decoder f_{dec} using the pre-trained networks from the source task. The parameters of the policy, value and reward functions are initialized to their default values; we train these from scratch on the target task.

3. We train IVG in the usual fashion on the target task. An important point to note is that the encoder, transition and decoder networks are fine-tuned on the target task; they just have a significantly better initialization (that is generalizable to the target task). As we show next in our experiments (Sec. 4.6), this transfer leads to a significant increase in learning speed, particularly when using a model that has been trained on multiple source tasks.

Baseline transfer: To test how well the model-free MPO baseline can transfer knowledge across tasks, we considered transferring the learned weights of the policy & value function, trained on a single source task, to related target tasks. As in IVG, the baseline MPO is also trained on a single source task. From the trained networks, we copy all the weights of the policy & value function (except the last layer), which we use to initialize training on the target task. As before, MPO is trained on the target task, albeit with a better initialization of parts of the policy & value function. We present comparisons to this baseline in our experiments.

4.6 Evaluation

We present results evaluating the performance of IVG on several long-horizon simulated manipulation tasks (see Sec. 4.5.2), learning tasks from scratch and also in the transfer setting. As mentioned earlier in Sec. 4.5.1, tasks involve the agent controlling a Sawyer manipulator equipped with a Robotiq gripper (action dim $N_A = 5$). Observations are 64x64px RGB

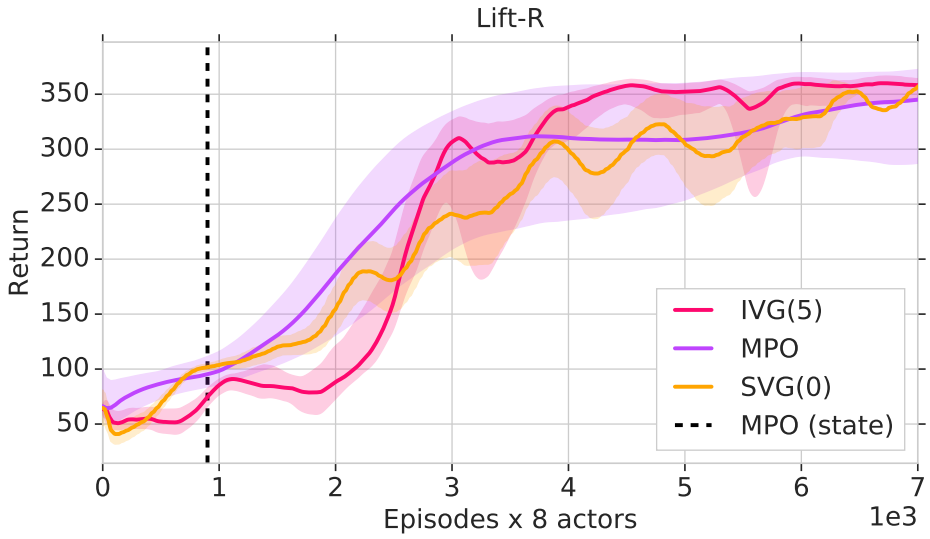

Figure 4.10: IVG(5), with $N = 5$, performs similarly to the model-free pixel-based baselines MPO and SVG(0) on the **Lift-R** task. Black dotted line represents the state-based MPO baseline scoring higher than a reward threshold for 50 consecutive episodes.

images from two cameras located in front of the robot and proprioceptive features. The latent representation is 128-dimensional (N_H=128) and unless otherwise noted we use a history of $H = 3$ observations and a rollout horizon of $N = 5$. Video results for our approach can be found at https://sites.google.com/view/ivg-corl19.

4.6.1 Learning from scratch

We first compare IVG and the baselines when learning the **Lift-R** and **Stack-R** manipulation tasks *from scratch*: The model-based IVG(5)[4] learns the simpler *lift* task stably and performs on par with the pixel-based MPO and SVG(0) baselines (see Fig. 4.10), empirically validating that stable learning can be achieved via imagined rollouts through a learned, approximate model. All these approaches learn significantly slower than the much informed MPO (State) baseline which operates directly from object features and proprioception.

[4]IVG(5) refers to IVG with a rollout length of $N = 5$. More generally, IVG(N) refers to IVG with a rollout length of N.

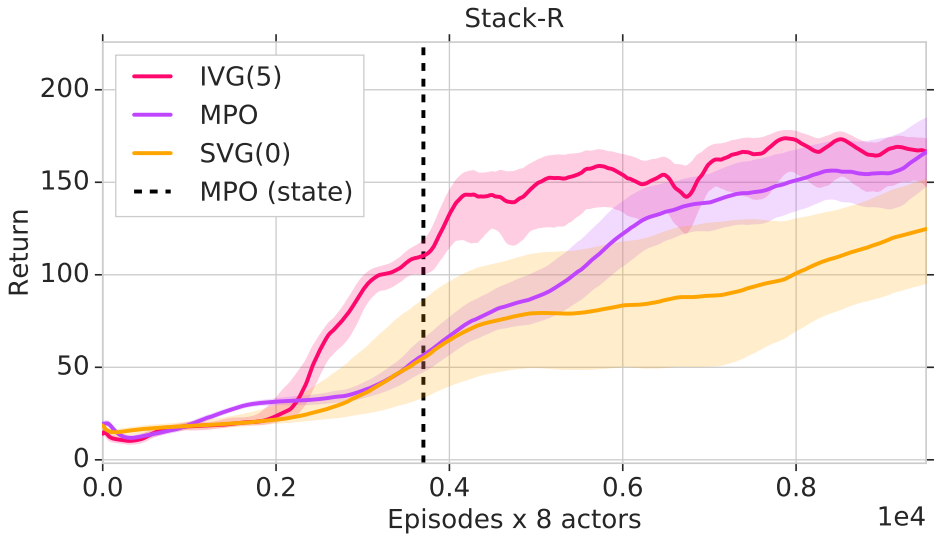

Figure 4.11: IVG(5) outperforms the baseline model-free pixel-based baselines MPO and SVG(0) on the **Stack-R** task. Compared to SVG(0), IVG(5) reaches a higher performance while it learns much faster compared to the model-free MPO. IVG(5) is worse than the baseline state-based MPO but the difference is much smaller than on the Lift-R task.

On the harder stack task (Fig. 4.11), IVG(5) learns significantly ($\sim$ 2x) faster than both MPO and SVG(0), and also outperforms SVG(0) in terms of final performance. Interestingly, the difference between the state-based MPO baseline and IVG(5) is much reduced for this task, highlighting the potential benefits in data-efficiency that can be gained from estimating a model. Even when learning directly from pixels, the structure inherent in IVG via the model and the corresponding latent space rollouts allows it to learn complex tasks faster than strong model-free baselines.

We also tested the ability of IVG to handle slightly more stochastic and partially observed environments (see 4.5.2 for details). Fig. 4.12 presents these results on learning the Stack-R task in environments with: 1) delayed proprioception (2 timestep lag) and 2) noisy observations where one of the blocks switches colors randomly every 3 frames. Even in these settings, IVG(5) successfully learns to stack in both cases (albeit more slowly), achieving a final performance close to the agent operating on the unperturbed environment. This further

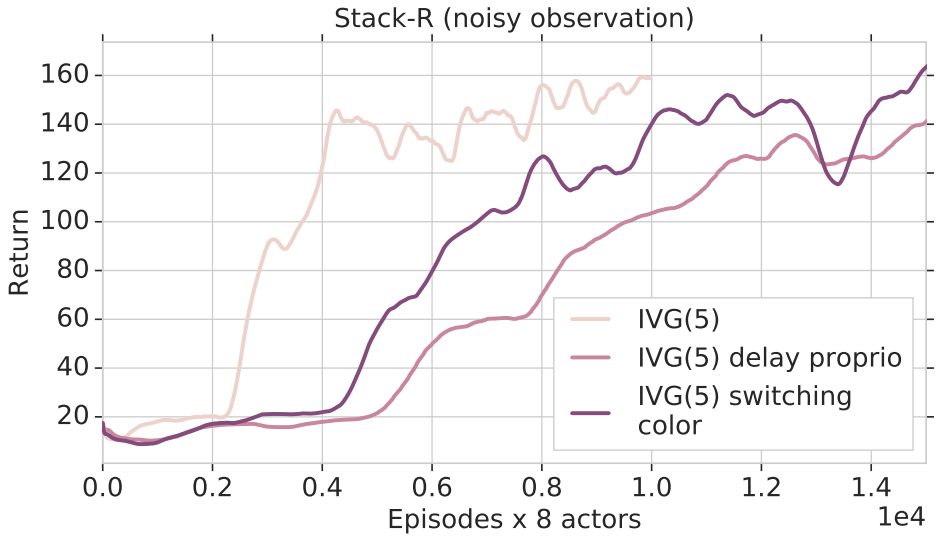

Figure 4.12: IVG(5) learns well even in environments with noisy observations such as with delays in proprioception and switches in block colors, highlighting the strength of our model in integrating temporal information. While it learns slower than IVG(5) on the unperturbed environment, it still is able to achieve on-par performance.

highlights the strength of our approach; the recurrent nature of our encoder architecture allows it to effectively integrate information across temporal sequences of observations, thereby gracefully handling noisy and partially observable settings.

4.6.2 Transferring learned models to related tasks

Unlike model-free methods, IVG learns a model of the environment which we may be able to transfer and thus accelerate the learning of related tasks. In the following, we evaluate this possibility. We follow the three-step process outlined in Sec. 4.5.7 to transfer our models trained on source task(s) to a related, but novel, target task; we transfer only the parameters of the encoder f_{enc}, transition model f_{trans} and decoder f_{dec} trained on the source task to the target task learner. We further include the transfer version of the model-free MPO algorithm (see Sec. 4.5.7) as a baseline for these experiments.

A model trained on multiple tasks should transfer better due to the diversity in transitions

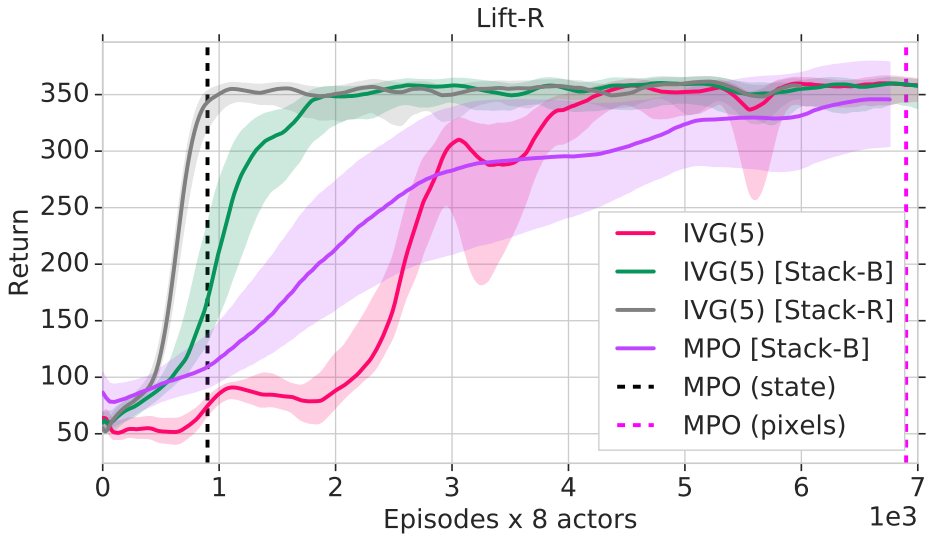

Figure 4.13: Transfer performance of IVG(5) on the **Lift-R** target task. Task names inside square brackets indicate the source task. Transferring an IVG model learned on the source Stack-R task has the largest improvement in performance, followed by transferring from the source Stack-B task. Both of these learn significantly faster compared to learning from scratch, the baseline pixel-based MPO and even the transfer version of MPO (named MPO [Stack-B]). IVG when transferring from Stack-R learns almost as fast as MPO (State).

observed. We test this hypothesis by comparing the multi-task variant of IVG (Sec. 4.5.5), trained on the multi-task setup described in Sec. 4.5.2 in our transfer experiments.

Transfer results: We present results on transferring IVG models to the following target tasks:

1) **Lift-R:** Fig. 4.13 shows the transfer performance of IVG(5) on the Lift-R target task, when transferring models trained on different source tasks. With a model pre-learned on *Stack-B*, IVG(5) learns ∼2x faster than from scratch and ∼4x faster than MPO, irrespective of whether MPO is learning from scratch or has been pre-learned on the Stack-B task. Interestingly, a model pre-trained on *Stack-R* accelerates the learning speed of IVG(5) further since the model has already observed many relevant transitions, achieving speed comparable to the *MPO (state)* baseline. These results highlight the significant benefits of a model when

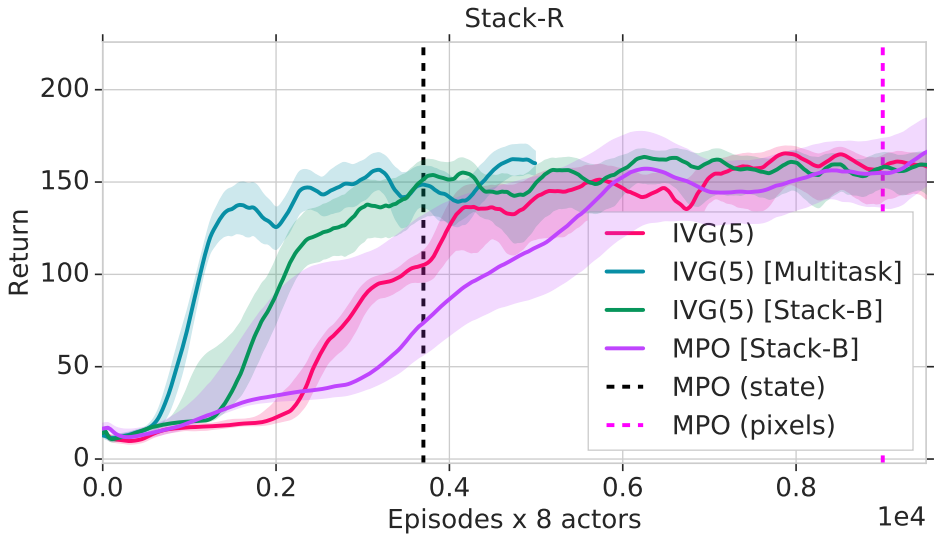

Figure 4.14: Transfer performance of IVG(5) on the **Stack-R** target task. Task names inside square brackets indicate the source task. All transfer versions of IVG(5) perform better than IVG(5) from scratch and the pixel-based MPO baselines (with or without transfer). Transferring the model learned on the multi-task setup outperforms all other comparisons, learning faster than even state-based MPO.

transferring to related tasks, even in cases where the model has only observed a subset of relevant transitions; the representations and dynamics learned by the model are generalizable to related tasks leading to significant gains in learning speed.

2) **Stack-R:** We further tested the performance of IVG when transferring models trained on different source tasks to the target Stack-R task. Results with transferred models for Stack-R are similar to those for Lift-R (see Fig. 4.14); transferring models helps speed up learning. In addition to transferring models trained on a single source task, we also compared the performance when transferring a model trained on multiple source tasks, i.e. using the *Multi-task* setup. This multi-task variant further accelerates learning speed; it is ∼1.5x faster than transferring from a single-task and about $\sim 3x$ faster than learning from scratch. Surprisingly, transferring a multi-task model leads to learning speeds that are comparable, or even faster, than the state-based MPO baseline, owing in part to the strong representation

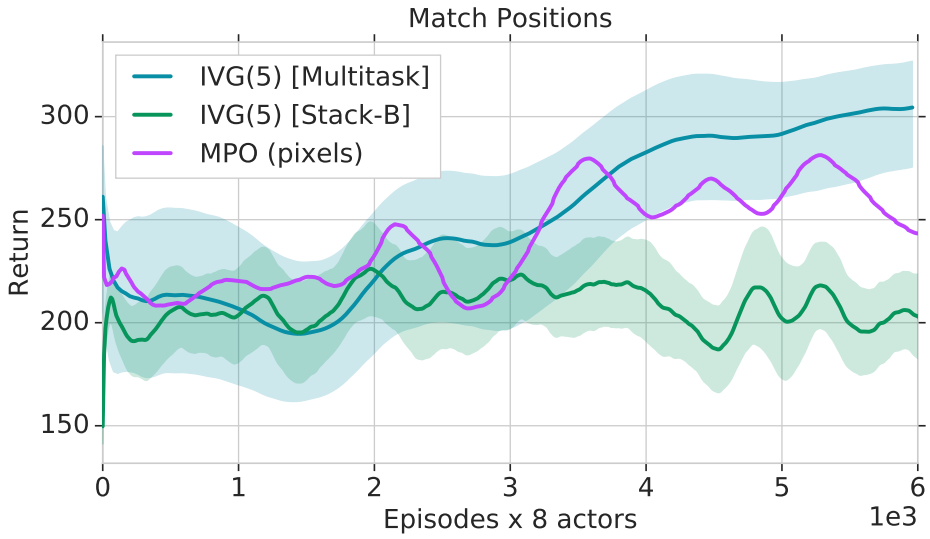

Figure 4.15: Transfer performance of IVG(5) on the **Match Positions** target task. Task names inside square brackets indicate the source task. Transferring the IVG model learned on the multi-task setup succeeds on learning this task while all other baselines including IVG(5) [Stack-B],where we transfer the model trained on the Stack-B task, fails to learn this task. This is due to the mixed sparse-dense nature of the task for which significant exploration is needed; having access to a model of the environment can reduce the effort needed for this.

and dynamics learning capabilities of our approach. This trend is reflected in later tasks as well; as we will see from subsequent results, models trained on multiple tasks greatly accelerate transfer.

3) **Match Positions:** Next, we tested the generalization of our learned models on the *match positions* task, which differs significantly from all the other tasks considered so far due to the lack of shaping in its reward (see Sec. 4.5.2 for a description of this task). Fig. 4.15 presents the results when transferring models trained on other source tasks to the match positions target task. All agents except multi-task IVG fail on this task; this includes transferring IVG from a single task (Stack-B) and the pixel-based MPO. This is likely due to the sparse structure of the reward; significant exploration is needed to figure out the reward substructure. This further demonstrates the benefits of multi-task training; representations

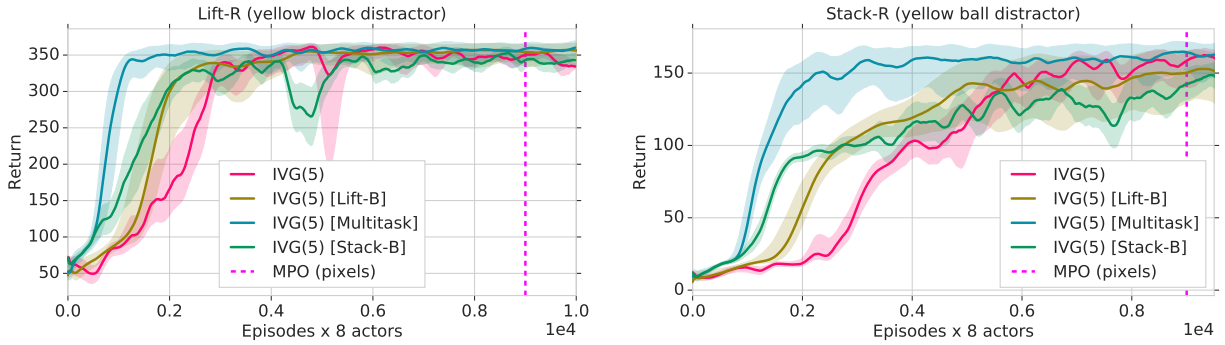

Figure 4.16: Transfer performance when a novel visual distractor is introduced into the scene. *Left*: Results on the Lift-R target task when a third object – a yellow block is added to the scene and *Right*: Results on the Stack-R target task with a yellow ball added as a visual distractor. As before, transferring IVG models learned on the multi-task setup significantly outperforms all other baselines and learning from scratch, learning up to 3-4x faster. Our models can transfer even in the presence of visual changes, highlighting the robustness of the learned IVG models and representations.

learned on multiple tasks need to be predictive of diverse rewards and values, this enables robust and expressive latent space learning that transfers extremely well to new tasks while potentially helping speed up exploration.

4) **Visual distractors:** Lastly, we analyzed the generalization of IVG when visual distractors in the form of a yellow cube or ball are added to the scene (Fig. 4.16, left & center respectively). Even in this setting, transfer from one or multiple source tasks remains effective. Similar to earlier results, transferring a multi-task model significantly outperforms all other methods, learning ∼3x faster than from scratch and ∼4x faster than MPO. Thus, even though CNNs are known to be sensitive to changes in visual inputs [117, 142], jointly training a predictive model with the policy allows it to locate task relevant features, optimizing for task performance in addition to prediction and reconstruction. This can lead to models and representations that are robust even to visual changes and can be transferred quickly across different tasks.

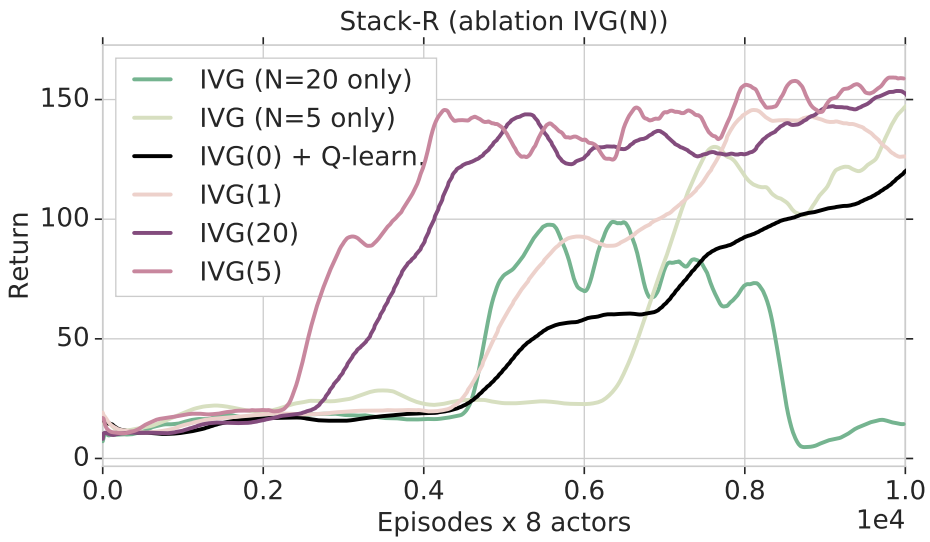

Figure 4.17: Ablation tests with IVG, using horizon lengths of $N = 0, 1, 5, 20$ & using a single N-step horizon for computing the policy gradient (Eqn. (4.23) instead of Eqn. (4.24)). IVG(0) uses the model loss (with horizon 5) only as an auxiliary signal (and otherwise corresponds to SVG(0)). Averaging across different horizons leads to robust policy optimization even with longer imagined rollouts (N=20), albeit with slower learning. Learning fails for long horizons without averaging (N=20 only) and is slow even for short horizons (N=5 only).

4.6.3 Ablation experiments

We performed two sets of ablation experiments to validate the design choices in our algorithm and models:

Rollout horizon: To evaluate the effect of the rollout horizon N on the policy gradient (Eqn. 4.24) we tested IVG across multiple settings of the rollout horizon N=0,1,5,20. Fig. 4.17 presents these results when learning the *Stack-R* task from scratch. Note that IVG(0) uses the model loss (with a rollout length of 5), only as an auxiliary signal to learn the representation – it learns a Q-function for policy optimization akin to SVG(0). IVG is robust to the choice of the rollout horizon and learns stably for all settings of N; even in the presence of cascading model errors for large N, IVG learns robustly. However, the choice

of N has a marked effect on the speed of learning. Increasing N speeds up learning up to a point after which no additional speedup is obtained (compare $N = 0, 1, 5$). Importantly, these results suggest that the benefit of the model in IVG is not just one of representation learning in partially observed domains and fast transfer but that using the model for policy optimziation is beneficial.

Averaging value gradients: To quantify the effect of averaging value gradients across multiple rollout horizons (c.f. Eq (4.24)), we ran experiments using a single imagined rollout of length N instead. Fig. 4.17 presents these results when learning the *Stack-R* task from scratch. For short horizons e.g. N=5, using a single rollout horizon leads to lower learning speed but learning still succeeds. On the other hand, learning completely fails for longer horizons N=20 (unlike with averaging) potentially due to cascading model errors on imagined rollouts, validating our averaging approach. Interestingly, averaging over multiple rollouts has little to no computational overhead compared to using a single rollout (as the computations for all intermediate rollouts of length $1 - N$ need to be performed to generate a single rollout of length N), further strengthening the case for rollout averaging to be a standard approach in computing model-based value estimates.

4.6.4 Reconstructions

Finally, we present a sequence of open-loop predictions and their corresponding image and proprioception reconstructions using a multi-task IVG(5) model, trained from scratch. We generate these as follows: First, the model encodes a history of H observations via the encoder f_{enc} to generate the latent state $\mathbf{h}^H$. Second, we apply the sequence of actions $a^{H+1:H+N}$ to generate the latent states $\mathbf{h}^{H+1:H+N}$ through an open loop rollout using the transition model f_{trans}. Finally, the decoder f_{dec} reconstructs the observations from these states.

Fig. 4.18 shows a 45-step open loop rollout from an IVG(5) model, trained on the *Multi-task* setting, from scratch. Even when tested on significantly longer sequences than it was trained on (in training N=5, testing $N = 45$. H=3 in both cases), the model predictions remain consistent, capturing salient details even after an open-loop prediction horizon of 30

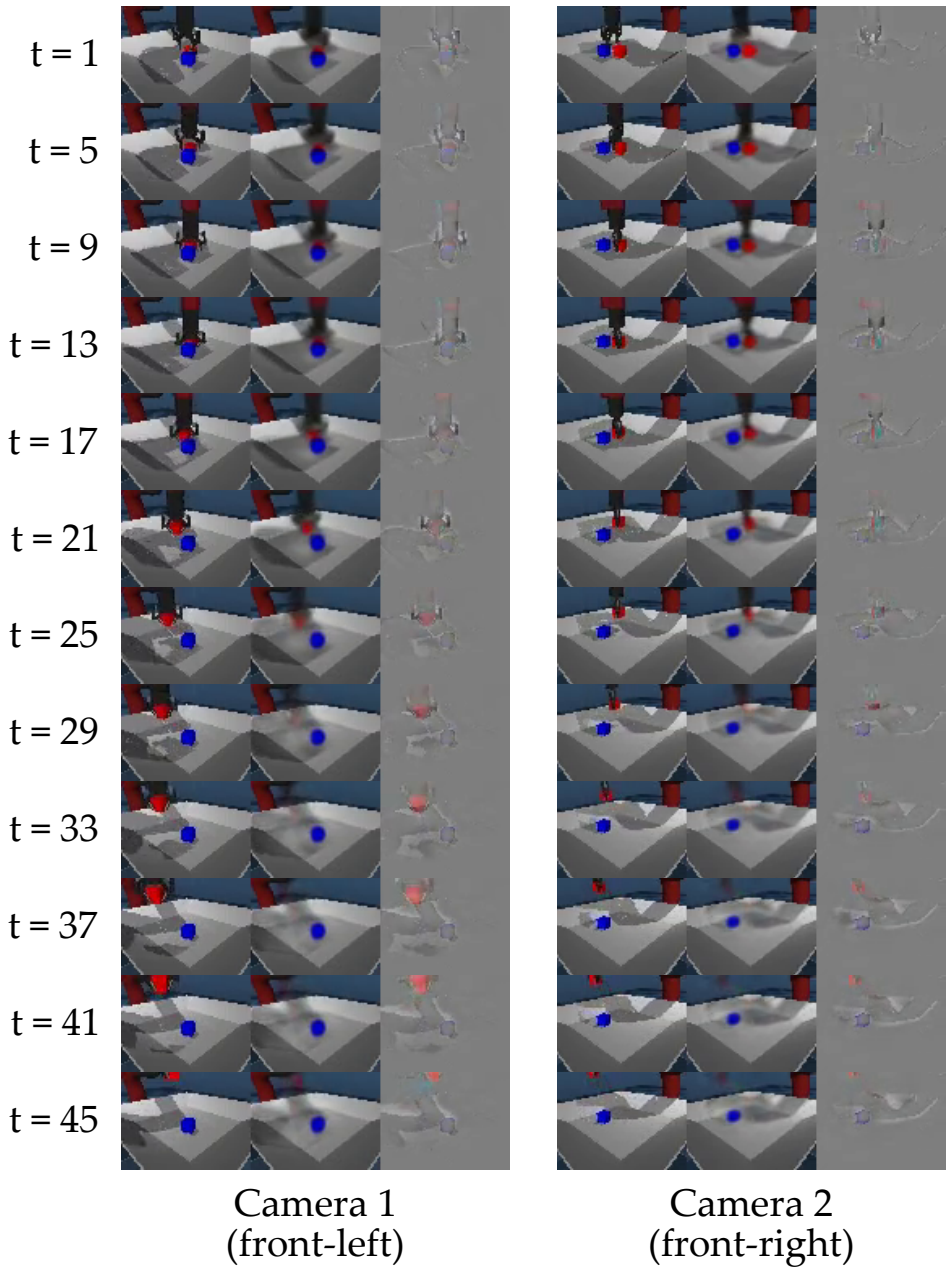

Figure 4.18: Example from the Lift-R task, showing open-loop reconstructions predicted by a multi-task IVG(5) model, trained from scratch. *Left*: Images from the left camera, reconstructions and error. *Right*: Images from the right camera, reconstructions and error. H=3 and N=45 for the sequence above. Even when tested on much longer sequences than in training (where N=5), the predictions remain consistent; the arm and objects are predicted well, albeit blurry, till around 30 frames. After 30 frames, the red block is reconstructed poorly but the rest are still well predicted.

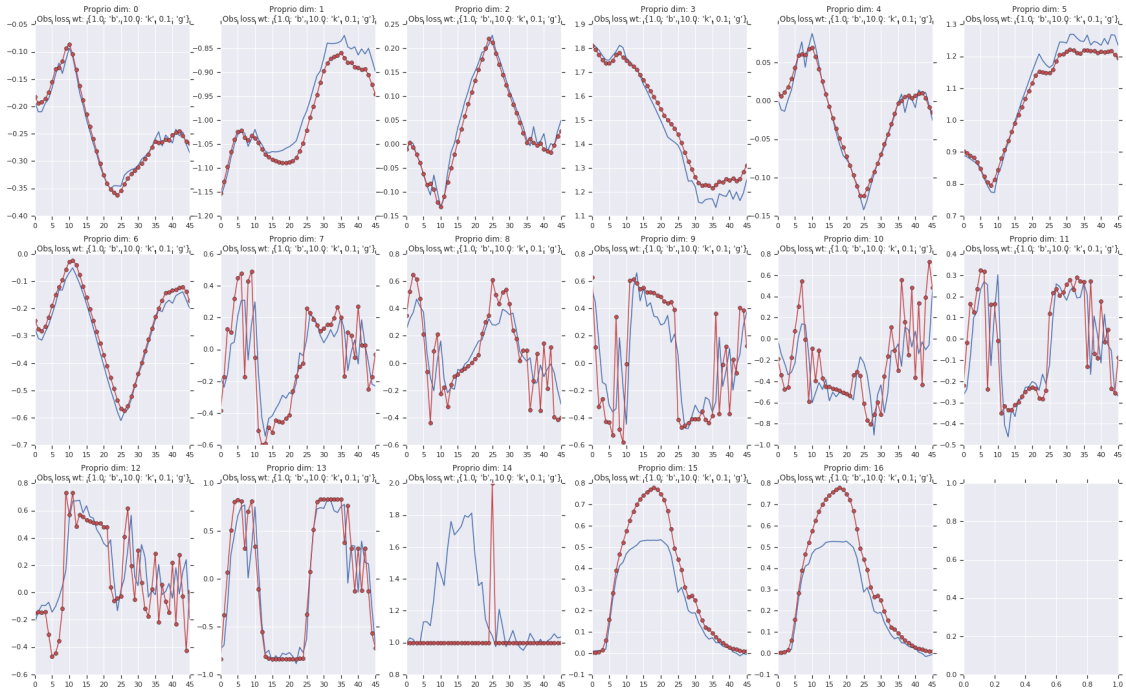

Figure 4.19: Example showing open-loop proprioception predictions from an IVG(5) model trained on the multi-task setting. Similar to the image reconstruction setting, we use a sequence of $H = 3$ observations and an open loop sequence of $N = 45$ actions ($\mathbf{a}^{H+1:H+N}$) to generate the proprioception predictions (blue line). The targets are plotted in red. The predictions from the IVG model are largely consistent, even for dimensions that are highly noisy; the learned latent state is able to encode the proprioceptive features and the transition model can recover their dynamics well.

frames. This also highlights an important point; even though the decoder predictions are not of high quality, the learned latent space can be used for robust control as it captures high-level task relevant details fairly well. Additionally, while model approximation errors can cascade, especially over longer horizons, rollout averaging enables us to be robust to these approximations as shown in the results from Sec. 4.6.3.

Lastly, Fig. 4.19 shows the result of a 45-step open loop prediction of proprioceptive features; the predictions are largely consistent with the observed results showing the strength of the learned model.

4.7 Discussion and Future Directions

We presented an approach for model-based RL where a predictive latent space model is trained jointly with policy, value and reward functions that operate on the learned latent space. To achieve efficient policy optimization, we introduced Imagined Value Gradients (IVG), an extension of the SVG algorithm [73] using imagined rollouts and N-step horizon averaging for stable learning. We demonstrated that IVG can learn complex, long-horizon manipulation tasks like lifting and stacking, directly from high dimensional visual and proprioceptive observations. We further demonstrated in several transfer experiments on related tasks that transferring a model learned via IVG can significantly improve data efficiency compared to off-policy baselines. Crucially, transferring with models trained on multiple tasks further accelerates learning, even succeeding on tasks where single-task transfer fails.

We feel that our approach is a promising first step towards designing RL methods that combine learning of closed loop policies with the generalization capabilities that learned approximate models can provide, thereby integrating the best of the model-based and model-free RL paradigms. While our initial results are promising there are multiple areas for future work:

- **Structured Model:** Our approach, IVG, used an encoder-transition-decoder style model that learns an unstructured, abstract latent representation which is trained via

observation reconstruction, consistency and task-specific value and reward prediction losses. By adding structure to this model in a manner similar to our prior work SE3-Nets and SE3-Pose-Nets and other structured predictive models [160, 40, 95] we can get the added benefits of interpretability, generalization and potentially faster learning through the use of a structured representation.

- **Exploration:** We have shown that models learned on single and multiple source tasks can be used to transfer quickly to related tasks. In addition to assisting transfer, models can be used to assist exploration. Recent work on curiosity guided exploration [121] has shown that better exploration can be achieved by finding actions that maximize model prediction error i.e., we find actions that take us to places unseen by the model. Similar heuristics can be used in our algorithm to also improve exploration; we believe that exploration is a big open problem in RL and coming up with more principled ways to approach this problem through the use of models is a big area for future work.

- **Exploitation:** In much the same way as exploration, we can also consider exploiting the knowledge in the model to get more targeted data that allows us to learn tasks faster. Several model-based approaches in the literature have looked at using planning and search algorithms in tandem with the learned models to generate simulated experience in addition to real-world data [83, 65, 139]. This can assist in reducing the data needs of the RL agent while also simultaneously improving learning speed. Trading off model-based exploitation with the current policy based rollouts in a principled manner is an open area for future work.

- **Data Efficiency:** While IVG has led to significant improvements in learning speed in the transfer setting, it still requires a prohibitive amount of samples to learn well. This is especially limiting in the real-world where it can be hard to generate millions of samples. An initial direction to explore for improving data efficiency could be to use existing samples efficiently by reusing them multiple times for gradient updates; we

currently use a single sample from the replay buffer ∼3-4 times (in comparison, the state based learner uses a sample ∼500 times). More general approaches to improving sample complexity is a key focus area for future work.

- **Real-World Experiments:** In this work, we presented results testing IVG on simulated manipulation experiments. We would like to test our approach on real-world tasks as a better validation of IVG; learning stably in the real-world involves many challenges including sample complexity, model complexity and being robust to visual perturbations such as lighting changes.

- **Observation changes:** Finally, in this chapter, we presented results on training IVG with two camera views. As mentioned earlier, this significantly reduces the amount of partial observability in the tasks, making them easier. We would like to explore learning from a single central camera view in future work. Additionally, we would like to explore learning from egocentric camera views; this brings additional control challenges as the viewpoint is dependent on the actions being taken.

Chapter 5
DISCUSSION

With the advent of machine learning, especially deep learning, there have been great advances in many areas such as robotics, computer vision and reinforcement learning. This has led to huge progress towards the goal of autonomous manipulation in unstructured environments. Learning has many advantages, such as the flexibility to model complex causal relationships and the potential to learn directly from raw, unstructured sensory information. This contrasts with many of our traditional approaches which usually make limiting assumptions on the problem structure and the availability of state estimates to make solutions tractable. On the other hand, many of our traditional models are broadly applicable across domains, initial conditions and environments; this is not the case for most deep learning methods which lack interpretability and struggle to generalize well. There is much to be gained by combining the strengths of these paradigms; the flexiblity of learning with the structure of traditional model-based reasoning.

In this thesis, we have explored ways to combine these paradigms, specifically in the context of learning to estimate the dynamics of the robot's environment, i.e. a model, directly from visual observations. We presented two major directions of work, applied to manipulation settings:

1. In Chapters 2 & 3, we presented a structured approach to designing deep visual dynamics models and their consequent application to the problem of visuomotor control. First, we proposed SE3-Nets (chapter 2), which are deep networks that model the dynamics of scenes under applied actions by learning to identify salient objects and predicting rigid body motion per object. By reasoning about objects in the scene, as opposed to pixels, SE3-Nets learn an interpretable deep model that can model complex object

interactions directly from raw depth observations, outperforming baseline deep networks on a variety of simulated and real world settings. Next, we discussed an improvement on the SE3-NET architecture, the SE3-POSE-NET, a network that additionally predicts a globally-consistent pose representation for each detected object (chapter 3). We further showed that our model can be used for closed-loop visuomotor control directly in the learned low-dimensional pose space, where the actions are computed by minimizing error in the pose space using gradient-based methods, similar to traditional model-based control; this is unlike many related methods which need an external system for providing data associations for measuring progress. We presented results on real-time, reactive control of a Baxter robot from raw RGBD data in simulation and in the real world; the structure inherent in SE3-POSE-NETS enables them to achieve robust control even in the presence of large amounts of noise in the visual observations. We also briefly presented DYNAMICS-NETS, a recurrent version of SE3-POSE-NETS that can be applied to model and control robots in dynamic tasks; we showed good prediction and control performance on the 2D cartpole task. The results and papers from this part of the thesis can be seen on our webpage.

2. While in the earlier chapters we presented initial results on controlling just the robot, in the second half of the thesis we considered long-horizon manipulation tasks with object interactions (e.g. lifting, stacking). Reinforcement learning has recently seen large success in many continuous control tasks and is a general approach towards learning manipulation tasks from scratch. In Chapter 4 we explored combining the strengths of model-free reinforcement learning approaches, which learn a policy directly mapping states to actions, with model-based RL approaches which estimate a dynamics model of the environment and plan to generate behaviors. We proposed Imagined Value Gradients (IVG), an algorithm that learns a predictive model of expected future observations, rewards and values from which a policy can be derived by following the gradient of the estimated value along imagined trajectories. We show how robust policy

optimization can be achieved even with approximate models on robot manipulation tasks, learned directly from vision and proprioception. We evaluate the efficacy of our approach in a transfer learning scenario, re-using previously learned models on tasks with different reward structures and visual distractors, and show a significant improvement in learning speed compared to strong off-policy baselines. Videos with results from our method can be found at this webpage.

Overall, we have seen many advantages to combining model-based reasoning with more recent deep network architectures such as: 1) an improvement in performance and interpretability (SE3-Nets, chapter 2), 2) robustness and the capability to solve tasks without test-time supervision (SE3-Pose-Nets, chapter 3) and, 3) transferrability of learned concepts to related tasks (IVG, chapter 4). There are many more benefits to be gained from a principled combination of these paradigms; this is a very promising area for future work.

5.1 Future Directions

There are many interesting directions for extending the work in this thesis, and in the general area of learning based models for manipulation. We list a few below:

Model architecture: We presented SE3-Nets and SE3-Pose-Nets which use a latent object-centric pose embedding and segmentation masks as their internal representation. There are a few key challenges towards using them for more general manipulation tasks: First, the latent representation in SE3-Pose-Nets captures the kinematics of the objects in the scene; for many manipulation tasks we want representations that model the dynamics, physical properties like mass, inertia etc., and visual properties like color, shape and texture. While Dynamics-Nets presented some initial ideas in this direction, extending these architectures, and other visual predictive models towards a more general representation that can be learned in a task specific manner is a key next step; there is some initial work along these lines which might be provide a good starting point [173, 179]. Second, SE3-Nets and SE3-Pose-Nets need to know the maximum number of moving objects in the scene apriori and cannot

gracefully handle varying number of objects (new work on handling arbitrary number of objects with similar models might help us alleviate this problem [135]). Third, while we used rigid body motion as the prior for SE3-Nets and SE3-Pose-Nets, many additional priors and constraints such as contact, self-collision and inter-penetration can be used to generate self-supervised signals for training more physically consistent predictive models. Lastly, we used an unstructured model in our IVG approach; integrating a more structured latent representation from a better model can significantly improve performance and generalization of IVG.

Model hierarchy: In this thesis, we presented multiple approaches towards learning visual dynamics models – all the models considered in this work make predictions based on low-level motor commands (such as joint velocities). In much the same way, we can imagine a model operating at a higher level, predicting the effect of meta-actions, options or even low-level policies (e.g., a grasp action, pick action, place action etc.). In fact, one could imagine the hierarchy extending to multiple levels; the actions at the highest level might be abstract and have effects across longer timescales. There are several challenges towards learning and using this model hierarchy for solving complex manipulation problems: 1) How do we automatically learn decompositions of a task into a meaningful hierarchy, potentially in an unsupervised manner? There is some initial work on task decompositions for learning from demonstration that could be a good starting point [137, 124]. 2) Given a decomposition, what are the right abstractions and model architectures? In some related work, we explored learning predictive models at a task level [123]; in practice these models tend to look very different than their low-level counterparts. How do we come up with a principled way to discover and learn such hierarchies? and lastly, 3) How do we effectively combine long-horizon search, planning algorithms with low-level reactive control policies for tasks with such a model hierarchy? Again, initial work in this area has shown some promising results with pre-specified models [139]; extending this to general learned models (and hierarchies) is an open area of research.

Generalizable models: We showed how structured model-based reasoning can help

our controller to be robust to visual perturbations (SE3-Pose-Nets Sec. 3.5.6) and transfer quickly to novel tasks with visual changes (IVG, Sec. 4.6). While these are interesting starting points, much work is to be done towards learning models that distill and generalize knowledge across a broader range of tasks. For example, can we have a single model that can predict well across different tasks in the Arcade Learning Environment [17]? Human models are able to do this quite easily as they can learn higher level concepts that translate well to even novel looking games [154]. There are several promising areas in the literature we can gain inspiration from for tackling these problems, such as meta-learning based methods which learn models and behaviors that translate quickly to new settings [46, 114] and visual tracking approaches that generalize to unseen and unknown objects [61].

Learning with less supervision: SE3-Nets and SE3-Pose-Nets use single-step data-association labels for training while IVG uses a hand-coded reward function to specify the task being learned; providing data-association labels and a shaped reward in real-world settings is hard. Reducing the amount of supervision needed for training our models is a must if we want them to scale to complex real-world tasks in unstructured environments. There are several interesting ways to tackle this problem: 1) We can replace our dense rewards with sparse reward specifications; while RL can succeed in these settings [126], it often requires millions of samples to learn properly. 2) An alternative could be to use demonstrations to bootstrap learning. Several promising approaches have looked at learning policies direclty from targeted demonstrations [75, 50]; this often leads to learning from fewer samples. 3) This idea could be further extended to unsupervised representation learning [95] and learning from video streams such as YouTube; such data is cheap to get and can be very informative if used correctly, and 4) Recent work has also looked at learning goal specifications for reinforcement learning directly from images, with minimal labeling [134, 52]. Extending these ideas to more general settings is an interesting area for future work.

Uncertainty estimates: A key requirement for many model-based robotic systems is a representation of uncertainty, especially for estimates of the state of the system. Uncertainty estimates can inform the optimization of robust policies via downstream planning and control

algorithms [31], even leading to large improvements in data efficiency during learning [36]. Unfortunately, most learning based systems, especially methods which involve deep learning, lack these estimates; this is true for the work presented in this thesis and much of the model learning work in the literature. Recent work in the Computer Vision literature has looked at approximating model uncertainty through techniques such as Dropout [56] and via model ensembles [97]. Extending these ideas to predictive models and enabling downstream planning, control and reinforcement learning algorithms to exploit these estimates is a promising area for future research.

Models in RL: In this thesis, we presented IVG, an approach that jointly learns a predictive model alongside a policy, value function and a reward estimator, using imagined rollouts through the learned model to achieve robust policy optimization. We demonstrated the benefits of IVG in a transfer learning scenario where reusing previously learned models leads to large improvements in learning speed. We discuss two other ways in which models can be integrated with RL: 1) A key open area in RL is exploration – recent work has shown the advantages of a model in enabling targeted exploration [121]. 2) Similarly, models can be exploited for faster learning of related tasks; we can generate goal-directed experience by combining a model with a search algorithm such as MCTS [139] or a sample-based planner like CEM [129]. In general, tightly integrating model-based reasoning with a general reinforcement agent can help us improve the sample complexity of learning on general manipulation tasks; this is one of the major bottlenecks preventing the application of RL to real-world tasks.

Over the course of this thesis, we have seen several advantages of the integration between traditional model-based and recent learning-based approaches to robotics. While these are promising, more evaluation is needed to quantify the benefits of this hybrid approach. Furthermore, a central open problem is the question of identifying the right mix between learning and structure – answering this question might lead us to a principled approach for tackling much of robotic manipulation!

Appendix A

APPENDIX FOR IMAGINED VALUE GRADIENTS

A.1 Tasks and Rewards

We used shaped rewards for specifying all our tasks as we wanted to measure transfer efficiency rather than the capability to handle sparse reward settings. In principle, the multi-task version of IVG can be extended to the sparse reward setting easily, in a manner similar to SAC [126]. Below, we discuss the reward setup for the four major task setups considered in the paper, the **Lift**, **Stack** and **Match Positions** tasks and the **Multi-task** setup. The addition of a *visual distractor* does not change the reward setup for a given task.

Lift: In the lift task, the agent has to pick-up one of either the red (Lift-R) or blue (Lift-B) objects on the table and lift it above a certain height. We introduce additional shaping to the task through auxiliary rewards that encourage reaching the target object, grasping it and lifting it once grasped. These are specified in turn as:

- *REACH(O)*: $tol(d(TCP, O), 0.02, 0.15)$:
  Minimize the distance of the TCP to the target cube.

- *GRASP*:
  Activate grasp sensor of gripper ("inward grasp signal" of Robotiq gripper)

- *HEIGHT(O, x)*: $lin(O, x, 0.10)$
  Increase z coordinate of an object more than $x = 0.03m$ relative to the table.

Where the $d(x, y)$ is the Euclidean distance between a pair of 3D points, and the tolerance

and linear reward functions terms are defined as:

$$tol(v, \epsilon, r) = \begin{cases} 1 & \text{iff } |v| < \epsilon \\ 1 - tanh^2(\frac{atanh(\sqrt{0.95})}{r}|v|) & \text{else,} \end{cases} \tag{A.1}$$

$$lin(v, \epsilon_{min}, \epsilon_{max}) = \begin{cases} 0 & \text{iff } v < \epsilon_{min} \\ 1 & \text{iff } v > \epsilon_{max} \\ \frac{v-\epsilon_{min}}{\epsilon_{max}-\epsilon_{min}} & \text{else.} \end{cases} \tag{A.2}$$

The final reward is a weighted sum of all these sub-rewards:

$$LIFT(O) = REACH(O) + 0.5 * (GRASP + HEIGHT(O, 0.03)),$$

which, overall cannot exceed a value of two.

Stack: Similar to the lift task, there are two variants of the stack task: 1) Stack-R, where the agent has to stack the red block on the blue block and 2) Stack-B, where the agent does the opposite. We again introduce shaping by first encouraging the agent to lift the object – the lift reward is a part of the reward for the stack task. Additionally, once the object has been lifted we encourage the agent to move towards the target, align it with the target block and release the grasped object. The total reward is:

$$STACK(O1, O2) = \begin{cases} LIFT(O1) & \text{iff } HEIGHT(O1, 0.03) \leq 0.8 \\ STACKED(O1, O2) & \text{else.} \end{cases} \tag{A.3}$$

where:

$$STACKED(O1, O2) = ABOVE(O1, O2) * NOTGRASP,$$

where $NOTGRASP$ is detemined by the grasp sensor and:

$$ABOVE(O1, O2) = tol(d(O1, O2), 0.02, 0.15) * HEIGHT(O1, 0.03)$$

Match Positions: In this task, the agent has to move both the red and blue blocks to a fixed target position (see Fig. 4.4 in the main text). As this task involves moving both

objects it is a nice setting for testing the generalization of our learned models. Additionally, the reward is not shaped to encourage motion towards an object; there is no change in the reward unless one of the objects is moved. We specify the reward as:

$$MP(O1, O2) = tol(d(O1, t1), 0.02, 0.15) + tol(d(O2, t2), 0.02, 0.15) \tag{A.4}$$

where $t1, t2$ denote the target 3D positions of the red and blue block respectively.

Multi-task setup: In the multi-task setup we introduce several auxiliary tasks that are solved in addition to a main extrinsic task. We consider the following tasks and rewards in all our multi-task experiments:

- $REACH(O_{\text{Red}})$
- $REACH(O_{\text{Blue}})$
- $LIFT(O_{\text{Blue}})$
- $STACK(O_{\text{Blue}})$
- $MOVE(O_{\text{Blue}})$

where the move reward is given by $MOVE(O) = tol(d(vel_O, v), 3, 0)$ where vel_O is the object's velocity. We note that even though the multi-task largely consists of the same rewards as for individual experiments the data distribution is very different as the model is trained on episodes from all tasks. As can be seen in the experiments (Sec. 4.6), this results in significant improvements in the transfer learning experiments.

BIBLIOGRAPHY

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Abbas Abdolmaleki, Jost Tobias Springenberg, Jonas Degrave, Steven Bohez, Yuval Tassa, Dan Belov, et al. Relative entropy regularized policy iteration. *arXiv preprint arXiv:1812.02256*, 2018.

[3] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018.

[4] Pulkit Agrawal, Joao Carreira, and Jitendra Malik. Learning to see by moving. In *ICCV*, pages 37–45, 2015.

[5] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *arXiv preprint arXiv:1606.07419*, 2016.

[6] Robert Anderson, David Gallup, Jonathan T Barron, Janne Kontkanen, Noah Snavely, Carlos Hernández, Sameer Agarwal, and Steven M Seitz. Jump: virtual reality video. *ACM Transactions on Graphics (TOG)*, 2016.

[7] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*, 2014.

[8] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[10] Renee Baillargeon. A model of physical reasoning in infancy. In *In C. Rovee-Collier, & LP Lipsitt (Eds.), Advances in infancy research*. Citeseer, 1995.

[11] Renée Baillargeon. Infants' physical world. *Current directions in psychological science*, 13(3):89–94, 2004.

[12] Renee Baillargeon, Elizabeth S Spelke, and Stanley Wasserman. Object permanence in five-month-old infants. *Cognition*, 20(3):191–208, 1985.

[13] Andre Barreto, Will Dabney, Remi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. In *NeurIPS 30*, 2017.

[14] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[15] Peter W Battaglia, Jessica B Hamrick, and Joshua B Tenenbaum. Simulation as an engine of physical scene understanding. *PNAS*, 2013.

[16] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[17] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[18] J. Bohg, K. Hausman, B. Sankaran, O. Brock, D. Kragic, S. Schaal, and G. Sukhatme. Interactive perception: Leveraging action in perception and perception in action. *arXiv preprint arXiv:1604.03670*, 2016.

[19] Byron Boots, Arunkumar Byravan, and Dieter Fox. Learning predictive models of a depth camera & manipulator from raw execution traces. In *ICRA*, pages 4021–4028. IEEE, 2014.

[20] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6d object pose estimation using 3d object coordinates. In *ECCV*, pages 536–551. Springer, 2014.

[21] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *NeurIPS*, 2018.

[22] Samuel R Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, 2004.

[23] Arunkumar Byravan and Dieter Fox. Se3-nets: Learning rigid body motion using deep neural networks. In *ICRA*, pages 173–180. IEEE, 2017.

[24] Arunkumar Byravan, Felix Leeb, Franziska Meier, and Dieter Fox. Se3-pose-nets: Structured deep dynamics models for visuomotor planning and control. *arXiv preprint arXiv:1710.00489*, 2017.

[25] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *2015 international conference on advanced robotics (ICAR)*, pages 510–517. IEEE, 2015.

[26] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

[27] Francois Chaumette, Seth Hutchinson, and Peter Corke. Visual servoing. In *Springer Handbook of Robotics*, pages 841–866. Springer, 2016.

[28] Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. *arXiv preprint arXiv:1704.02254*, 2017.

[29] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[30] Christopher B Choy, JunYoung Gwak, Silvio Savarese, and Manmohan Chandraker. Universal correspondence network. In *Advances in Neural Information Processing Systems*, pages 2414–2422, 2016.

[31] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.

[32] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. In *CoRL*, 2018.

[33] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[34] Alvaro Collet, Dmitry Berenson, Siddhartha S Srinivasa, and Dave Ferguson. Object recognition and full pose registration from a single image for robotic manipulation. In

2009 IEEE International Conference on Robotics and Automation, pages 48–55. IEEE, 2009.

[35] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[36] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *ICML*, pages 465–472, 2011.

[37] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[38] Mehmet Dogar and Siddhartha Srinivasa. A framework for push-grasping in clutter. *Robotics: Science and systems VII*, 1, 2011.

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

[40] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex X. Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *CoRR*, abs/1812.00568, 2018.

[41] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *ICML*, 2018.

[42] Bernard Espiau, François Chaumette, and Patrick Rives. A new approach to visual servoing in robotics. *Geometric reasoning for perception and action*, pages 106–136, 1993.

[43] Nima Fazeli, Samuel Zapolsky, Evan Drumwright, and Alberto Rodriguez. Learning

data-efficient rigid-body contact models: Case study of planar impact. *arXiv preprint arXiv:1710.05947*, 2017.

[44] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I. Jordan, Joseph E. Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *CoRR*, abs/1803.00101, 2018.

[45] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

[46] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[47] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *NIPS*, 2016.

[48] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. *arXiv preprint arXiv:1610.00696*, 2016.

[49] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *ICRA*, pages 2786–2793. IEEE, 2017.

[50] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

[51] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick Van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015.

[52] Carlos Florensa, Jonas Degrave, Nicolas Heess, Jost Tobias Springenberg, and Martin

Riedmiller. Self-supervised learning of image embedding for continuous control. *arXiv preprint arXiv:1901.00943*, 2019.

[53] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. In *ICLR*, 2017.

[54] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. *arXiv preprint arXiv:1511.07404*, 2015.

[55] Vincent François-Lavet, Yoshua Bengio, Doina Precup, and Joelle Pineau. Combined reinforcement learning via abstract representations. *arXiv preprint arXiv:1809.04506*, 2018.

[56] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[57] Alexandre Galashov, Siddhant Jayakumar, Leonard Hasenclever, Dhruva Tirumala, Jonathan Schwarz, Guillaume Desjardins, Wojtek M. Czarnecki, Yee Whye Teh, et al. Information asymmetry in KL-regularized RL. In *ICLR*, 2019.

[58] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare. Deepmdp: Learning continuous latent space models for representation learning. *ICML*, 2019.

[59] Perle Geoffroy, Nicolas Mansard, Maxime Raison, Sofiane Achiche, and Emo Todorov. From inverse kinematics to optimal control. In *Advances in Robot Kinematics*, pages 409–418. Springer, 2014.

[60] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[61] Daniel Gordon, Ali Farhadi, and Dieter Fox. Re3: Real-time recurrent regression networks for visual tracking of generic objects. *IEEE Robotics and Automation Letters*, 3(2):788–795, 2018.

[62] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv:1502.04623*, 2015.

[63] Shixiang Gu, Timothy P. Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *ICML*, 2016.

[64] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[65] Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *ICML*, 2019.

[66] Ankur Handa, Michael Bloesch, Viorica Pătrăucean, Simon Stent, John McCormac, and Andrew Davison. gvnn: Neural network library for geometric computer vision. In *Computer Vision–ECCV 2016 Workshops*, pages 67–82, 2016.

[67] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 447–456, 2015.

[68] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *ICLR*, 2018.

[69] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[70] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.

[71] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[72] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[73] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *NeurIPS*, 2015.

[74] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, , and N. Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *ACCV*, 2012.

[75] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[76] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8), 1997.

[77] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[78] Seth Hutchinson, Gregory D Hager, and Peter I Corke. A tutorial on visual servo control. *IEEE transactions on robotics and automation*, 12(5):651–670, 1996.

[79] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2462–2470, 2017.

[80] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[81] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *NIPS*, pages 2008–2016, 2015.

[82] Rico Jonschkowski, Roland Hafner, Jonathan Scholz, and Martin Riedmiller. Pves: Position-velocity encoders for unsupervised learning of structured state representations. *arXiv preprint arXiv:1705.09805*, 2017.

[83] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, et al. Model-based reinforcement learning for atari. *CoRR*, abs/1903.00374, 2019.

[84] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *CoRL*, 2018.

[85] Nan Rosemary Ke, Amanpreet Singh, Ahmed Touati, Anirudh Goyal, Yoshua Bengio, Devi Parikh, and Dhruv Batra. Modeling the long term future in model-based reinforcement learning. In *ICLR*, 2019.

[86] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *ICCV*, pages 2938–2946. IEEE, 2015.

[87] Jennifer E King, Joshua A Haustein, Siddhartha S Srinivasa, and Tamim Asfour. Nonprehensile whole arm rearrangement planning on physics manifolds. In *2015 IEEE*

International Conference on Robotics and Automation (ICRA), pages 2508–2515. IEEE, 2015.

[88] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[89] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[90] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[91] Marek Kopicki, Sebastian Zurek, Rustam Stolkin, Thomas Moerwald, and Jeremy L Wyatt. Learning modular and transferable forward models of the motions of push manipulated objects. *Autonomous Robots*, pages 1–22, 2016.

[92] Michael C Koval, Mehmet R Dogar, Nancy S Pollard, and Siddhartha S Srinivasa. Pose estimation for contact manipulation with manifold particle filters. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4541–4548. IEEE, 2013.

[93] Michael C Koval, Nancy S Pollard, and Siddhartha S Srinivasa. Pre-and post-contact policy decomposition for planar contact manipulation under uncertainty. *The International Journal of Robotics Research*, 35(1-3):244–264, 2016.

[94] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[95] Tejas Kulkarni, Ankush Gupta, Catalin Ionescu, Sebastian Borgeaud, Malcolm Reynolds, Andrew Zisserman, and Volodymyr Mnih. Unsupervised learning of object keypoints for perception and control. *arXiv preprint arXiv:1906.11883*, 2019.

[96] Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *NIPS*, pages 2530–2538, 2015.

[97] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413, 2017.

[98] Alex X Lee, Sergey Levine, and Pieter Abbeel. Learning visual servoing with deep features and fitted q-iteration. *ICLR*, 2017.

[99] Felix Leeb. Robots reasoning over rigid-body motion for visuomotor control, 2019.

[100] Adam Lerer, Sam Gross, and Rob Fergus. Learning physical intuition of block towers by example. *arXiv preprint arXiv:1603.01312*, 2016.

[101] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 17(39):1–40, 2016.

[102] Wenbin Li, Seyedmajid Azimi, Aleš Leonardis, and Mario Fritz. To fall or not to fall: A visual approach to physical stability prediction. *arXiv preprint arXiv:1604.00066*, 2016.

[103] 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. *arXiv preprint arXiv:1509.02971*, 2015.

[104] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[105] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, pages 3431–3440, 2015.

[106] David G Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[107] Matthew T Mason. Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research*, 5(3):53–71, 1986.

[108] Matthew T Mason. Toward robotic manipulation. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:1–28, 2018.

[109] MT Mason and JK Salisbury Jr. *Robot hands and the mechanics of manipulation*. The MIT Press, Cambridge, MA, 1985.

[110] David Q Mayne. Model predictive control: Recent developments and future promise. *Automatica*, 50(12):2967–2986, 2014.

[111] DW Miller. Nasa technology roadmaps, ta4: Robotics and autonomous systems. *National Aeronautics and Space Administration (http://www. nasa. gov), Office of the Chief Technologist,(http://www. nasa. gov/offices/oct/home/roadmaps/index. html)*, 2015.

[112] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[113] Roozbeh Mottaghi, Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. Newtonian scene understanding: Unfolding the dynamics of objects in static images. In *CVPR*, pages 3521–3529, 2016.

[114] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, et al. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

[115] Anusha Nagabandi, Chelsea Finn, and Sergey Levine. Deep online learning via meta-learning: Continual adaptation for model-based RL. *CoRR*, abs/1812.07671, 2018.

[116] R. Newcombe, D. Fox, and S. Seitz. DynamicFusion: Reconstruction and tracking of non-rigid scenes in real-time. In *CVPR*, 2015.

[117] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*, 2015.

[118] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *NIPS*, pages 2845–2853, 2015.

[119] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *NeurIPS*, 2017.

[120] OpenAI. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018.

[121] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *CVPR*, 2017.

[122] Georgios Pavlakos, Xiaowei Zhou, Aaron Chan, Konstantinos G Derpanis, and Kostas Daniilidis. 6-dof object pose from semantic keypoints. In *ICRA*, pages 2011–2018. IEEE, 2017.

[123] Chris Paxton, Yonatan Bisk, Jesse Thomason, Arunkumar Byravan, and Dieter Fox. Prospection: Interpretable plans from language by predicting the future. *arXiv preprint arXiv:1903.08309*, 2019.

[124] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *arXiv preprint arXiv:1804.02717*, 2018.

[125] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic back-propagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[126] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, et al. Learning by playing-solving sparse reward tasks from scratch. In *ICML*, 2018.

[127] Alberto Rodriguez, Matthew T Mason, and Steve Ferry. From caging to grasping. *The International Journal of Robotics Research*, 31(7):886–900, 2012.

[128] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[129] Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.

[130] Adam N Sanborn, Vikash K Mansinghka, and Thomas L Griffiths. Reconciling intuitive physics and newtonian mechanics for colliding objects. *Psychological review*, 120(2):411, 2013.

[131] Tanner Schmidt, Richard Newcombe, and Dieter Fox. Self-supervised visual descriptor learning for dense correspondence. *IEEE Robotics and Automation Letters*, 2(2):420–427, 2017.

[132] Tanner Schmidt, Richard A Newcombe, and Dieter Fox. Dart: Dense articulated real-time tracking. In *RSS*, 2014.

[133] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[134] Pierre Sermanet, Kelvin Xu, and Sergey Levine. Unsupervised perceptual rewards for imitation learning. *arXiv preprint arXiv:1612.06699*, 2016.

[135] Lin Shao, Parth Shah, Vikranth Dwaracherla, and Jeannette Bohg. Motion-based object segmentation based on dense rgb-d scene flow. *arXiv preprint arXiv:1804.05195*, 2018.

[136] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[137] Kyriacos Shiarlis, Markus Wulfmeier, Sasha Salter, Shimon Whiteson, and Ingmar Posner. Taco: Learning task decomposition via temporal alignment for control. *arXiv preprint arXiv:1803.01840*, 2018.

[138] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2930–2937, 2013.

[139] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, et al. A general reinforcement learning algorithm that masters chess, shogi, & go through self-play. *Science*, 362(6419), 2018.

[140] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[141] Siddhartha S Srinivasa, Dave Ferguson, Casey J Helfrich, Dmitry Berenson, Alvaro Collet, Rosen Diankov, Garratt Gallagher, Geoffrey Hollinger, James Kuffner, and Michael Vande Weghe. Herb: a home exploring robotic butler. *Autonomous Robots*, 28(1):5, 2010.

[142] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 2019.

[143] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[144] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*. Elsevier, 1990.

[145] Jonathan Taylor, Jamie Shotton, Toby Sharp, and Andrew Fitzgibbon. The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 103–110. IEEE, 2012.

[146] George Terzakis, Phil Culverhouse, Guido Bugmann, Sanjay Sharma, and Robert Sutton. A recipe on the parameterization of rotation matrices for non-linear optimization using quaternions. Technical report, Plymouth University, 2012.

[147] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[148] Dhruva Tirumala, Hyeonwoo Noh, Alexandre Galashov, Leonard Hasenclever, Arun Ahuja, Greg Wayne, et al. Exploiting hierarchy for learning and transfer in kl-regularized RL. *CoRR*, abs/1903.07438, 2019.

[149] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, pages 23–30. IEEE, 2017.

[150] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012.

[151] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*. IEEE, 2012.

[152] Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *ACC*, pages 300–306. IEEE, 2005.

[153] Marc Toussaint. Robot trajectory optimization using approximate inference. In *ICML*, pages 1049–1056. ACM, 2009.

[154] Pedro A Tsividis, Thomas Pouncy, Jaqueline L Xu, Joshua B Tenenbaum, and Samuel J Gershman. Human learning in atari. In *2017 AAAI Spring Symposium Series*, 2017.

[155] Sudheendra Vijayanarasimhan, Susanna Ricco, Cordelia Schmid, Rahul Sukthankar, and Katerina Fragkiadaki. Sfm-net: Learning of structure and motion from video. *arXiv preprint arXiv:1704.07804*, 2017.

[156] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M. Czarnecki, et al. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II, 2019.

[157] Niklas Wahlström, Thomas B Schön, and Marc Peter Deisenroth. From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251*, 2015.

[158] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *ICCV*, pages 2794–2802, 2015.

[159] Yu Wang and Matthew T Mason. Two-dimensional rigid-body collisions with friction. *Journal of Applied Mechanics*, 59(3):635–642, 1992.

[160] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed

to control: A locally linear latent dynamics model for control from raw images. In *NIPS*, pages 2728–2736, 2015.

[161] Nicholas Watters, Loic Matthey, Matko Bosnjak, et al. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *CoRR*, abs/1905.09275, 2019.

[162] Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, et al. Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*, 2018.

[163] Theophane Weber, Sébastien Racanière, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, et al. Imagination-augmented agents for deep reinforcement learning. *CoRR*, abs/1707.06203, 2017.

[164] Lingyu Wei, Qixing Huang, Duygu Ceylan, Etienne Vouga, and Hao Li. Dense human body correspondences using convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1544–1553, 2016.

[165] William Whitney. Disentangled representations in neural models. *arXiv preprint arXiv:1602.02383*, 2016.

[166] Wikipedia contributors. Bias–variance tradeoff — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Bias%E2%80%93variance_tradeoff&oldid=909502810, 2019. [Online; accessed 10-August-2019].

[167] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1433–1440. IEEE, 2016.

[168] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic mpc for model-based

reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721. IEEE, 2017.

[169] Paul Wohlhart and Vincent Lepetit. Learning descriptors for object recognition and 3d pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3109–3118, 2015.

[170] Zbigniew Wojna, Vittorio Ferrari, Sergio Guadarrama, Nathan Silberman, Liang-Chieh Chen, Alireza Fathi, and Jasper Uijlings. The devil is in the decoder. *arXiv preprint arXiv:1707.05847*, 2017.

[171] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017.

[172] Zhenjia Xu, Zhijian Liu, Chen Sun, Kevin Murphy, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Unsupervised discovery of parts, structure, and dynamics. *arXiv preprint arXiv:1903.05136*, 2019.

[173] Zhenjia Xu, Jiajun Wu, Andy Zeng, Joshua B Tenenbaum, and Shuran Song. Densephysnet: Learning dense physical object representations via multi-step dynamic interactions. *arXiv preprint arXiv:1906.03853*, 2019.

[174] Tianfan Xue, Jiajun Wu, Katherine Bouman, and Bill Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *NIPS*, pages 91–99, 2016.

[175] Jimei Yang, Scott E Reed, Ming-Hsuan Yang, and Honglak Lee. Weakly-supervised disentangling with recurrent transformations for 3d view synthesis. In *NIPS*, pages 1099–1107, 2015.

[176] Kuan-Ting Yu, Maria Bauza, Nima Fazeli, and Alberto Rodriguez. More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing. In *2016*

IEEE/RSJ international conference on intelligent robots and systems (IROS), pages 30–37. IEEE, 2016.

[177] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[178] Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine. Solar: Deep structured latent representations for model-based reinforcement learning. *CoRR*, abs/1808.09105, 2018.

[179] David Zheng, Vinson Luo, Jiajun Wu, and Joshua B Tenenbaum. Unsupervised learning of latent physical properties using perception-prediction networks. *arXiv preprint arXiv:1807.09244*, 2018.

[180] Jiaji Zhou, Robert Paolini, J Andrew Bagnell, and Matthew T Mason. A convex polynomial force-motion model for planar sliding: Identification and application. In *ICRA*. IEEE, 2016.

[181] Henry Zhu, Abhishek Gupta, Aravind Rajeswaran, Sergey Levine, and Vikash Kumar. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. In *ICRA*, 2019.