Journal of Machine Learning Research 17 (2016) 1-40 Submitted 10/15; Revised 4/16; Published 4/16

End-to-End Training of Deep Visuomotor Policies

Sergey Levine[†] SVLEVINE@EECS.BERKELEY.EDU
Chelsea Finn[†] CBFINN@EECS.BERKELEY.EDU
Trevor Darrell TREVOR@EECS.BERKELEY.EDU
Pieter Abbeel PABBEEL@EECS.BERKELEY.EDU
Division of Computer Science
University of California
Berkeley, CA 94720-1776, USA
[†]These authors contributed equally.

Editor: Jan Peters

Abstract

Policy search methods can allow robots to learn control policies for a wide range of tasks, but practical applications of policy search often require hand-engineered components for perception, state estimation, and low-level control. In this paper, we aim to answer the following question: does training the perception and control systems jointly end-to-end provide better performance than training each component separately? To this end, we develop a method that can be used to learn policies that map raw image observations directly to torques at the robot's motors. The policies are represented by deep convolutional neural networks (CNNs) with 92,000 parameters, and are trained using a guided policy search method, which transforms policy search into supervised learning, with supervision provided by a simple trajectory-centric reinforcement learning method. We evaluate our method on a range of real-world manipulation tasks that require close coordination between vision and control, such as screwing a cap onto a bottle, and present simulated comparisons to a range of prior policy search methods.

Keywords: Reinforcement Learning, Optimal Control, Vision, Neural Networks

1. Introduction

Robots can perform impressive tasks under human control, including surgery (Lanfranco et al., 2004) and household chores (Wyrobek et al., 2008). However, designing the perception and control software for autonomous operation remains a major challenge, even for basic tasks. Policy search methods hold the promise of allowing robots to automatically learn new behaviors through experience (Kober et al., 2010b; Deisenroth et al., 2011; Kalakrishnan et al., 2011; Deisenroth et al., 2013). However, policies learned using such methods often rely on a number of hand-engineered components for perception and control, so as to present the policy with a more manageable and low-dimensional representation of observations and actions. The vision system in particular can be complex and prone to errors, and it is typically not improved during policy training, nor adapted to the goal of the task.

In this article, we aim to answer the following question: can we acquire more effective policies for sensorimotor control if the perception system is trained jointly with the control policy, rather than separately? In order to represent a policy that performs both

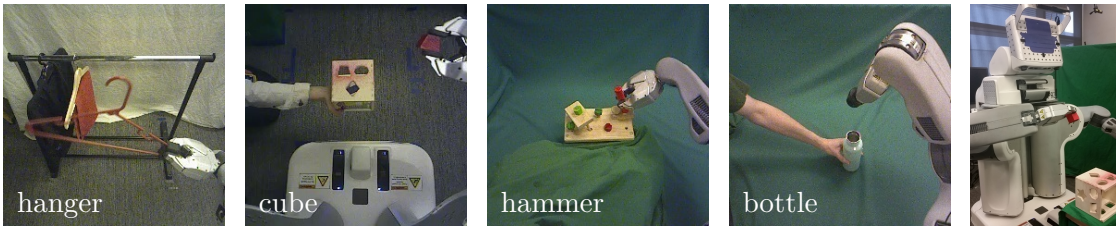


Figure 1: Our method learns visuomotor policies that directly use camera image observations (left) to set motor torques on a PR2 robot (right).

perception and control, we use deep neural networks. Deep neural network representations have recently seen widespread success in a variety of domains, such as computer vision and speech recognition, and even playing video games. However, using deep neural networks for real-world sensorimotor policies, such as robotic controllers that map image pixels and joint angles to motor torques, presents a number of unique challenges. Successful applications of deep neural networks typically rely on large amounts of data and direct supervision of the output, neither of which is available in robotic control. Real-world robot interaction data is scarce, and task completion is defined at a high level by means of a cost function, which means that the learning algorithm must determine on its own which action to take at each point. From the control perspective, a further complication is that observations from the robot's sensors do not provide us with the full state of the system. Instead, important state information, such as the positions of task-relevant objects, must be inferred from inputs such as camera images.

We address these challenges by developing a guided policy search algorithm for sensorimotor deep learning, as well as a novel CNN architecture designed for robotic control. Guided policy search converts policy search into supervised learning, by iteratively constructing the training data using an efficient model-free trajectory optimization procedure. We show that this can be formalized as an instance of Bregman ADMM (BADMM) (Wang and Banerjee, 2014), which can be used to show that the algorithm converges to a locally optimal solution. In our method, the full state of the system is observable at training time, but not at test time. For most tasks, providing the full state simply requires positioning objects in one of several known positions for each trial during training. At test time, the learned CNN policy can handle novel, unknown configurations, and no longer requires full state information. Since the policy is optimized with supervised learning, we can use standard methods like stochastic gradient descent for training. Our CNNs have 92,000 parameters and 7 layers, including a novel spatial feature point transformation that provides accurate spatial reasoning and reduces overfitting. This allows us to train our policies with relatively modest amounts of data and only tens of minutes of real-world interaction time.

We evaluate our method by learning policies for inserting a block into a shape sorting cube, screwing a cap onto a bottle, fitting the claw of a toy hammer under a nail with various grasps, and placing a coat hanger on a rack with a PR2 robot (see Figure 1). These tasks require localization, visual tracking, and handling complex contact dynamics. Our results demonstrate improvements in consistency and generalization from training visuomotor policies end-to-end, when compared to training the vision and control components separately. We also present simulated comparisons that show that guided policy search outperforms a

number of prior methods when training high-dimensional neural network policies. Some of the material in this article has previously appeared in two conference papers (Levine and Abbeel, 2014; Levine et al., 2015), which we extend to introduce visual input into the policy.

2. Related Work

Reinforcement learning and policy search methods (Gullapalli, 1990; Williams, 1992) have been applied in robotics for playing games such as table tennis (Kober et al., 2010b), object manipulation (Gullapalli, 1995; Peters and Schaal, 2008; Kober et al., 2010a; Deisenroth et al., 2011; Kalakrishnan et al., 2011), locomotion (Benbrahim and Franklin, 1997; Kohl and Stone, 2004; Tedrake et al., 2004; Geng et al., 2006; Endo et al., 2008), and flight (Ng et al., 2004). Several recent papers provide surveys of policy search in robotics (Deisenroth et al., 2013; Kober et al., 2013). Such methods are typically applied to one component of the robot control pipeline, which often sits on top of a hand-designed controller, such as a PD controller, and accepts processed input, for example from an existing vision pipeline (Kalakrishnan et al., 2011). Our method learns policies that map visual input and joint encoder readings directly to the torques at the robot's joints. By learning the entire mapping from perception to control, the perception layers can be adapted to optimize task performance, and the motor control layers can be adapted to imperfect perception.

We represent our policies with convolutional neural networks (CNNs). CNNs have a long history in computer vision and deep learning (Fukushima, 1980; LeCun et al., 1989; Schmidhuber, 2015), and have recently gained prominence due to excellent results on a number of vision benchmarks (Ciresan et al., 2011; Krizhevsky et al., 2012; Ciresan et al., 2012; Girshick et al., 2014a; Tompson et al., 2014; LeCun et al., 2015; He et al., 2015). Most applications of CNNs focus on classification, where locational information is discarded by means of successive pooling layers to provide for invariance (Lee et al., 2009). Applications to localization typically either use a sliding window (Girshick et al., 2014a) or object proposals (Endres and Hoiem, 2010; Uijlings et al., 2013; Girshick et al., 2014b) to localize the object, reducing the task to classification, perform regression to a heatmap of manually labeled keypoints (Tompson et al., 2014), requiring precise knowledge of the object position in the image and camera calibration, or use 3D models to localize previously scanned objects (Pepik et al., 2012; Savarese and Fei-Fei, 2007). Many prior robotic applications of CNNs do not directly consider control, but employ CNNs for the perception component of a larger robotic system (Hadsell et al., 2009; Sung et al., 2015; Lenz et al., 2015b; Pinto and Gupta, 2015). We use a novel CNN architecture for our policies that automatically learn feature points that capture spatial information about the scene, without any supervision beyond the information from the robot's encoders and camera.

Applications of deep learning in robotic control have been less prevalent in recent years than in visual recognition. Backpropagation through the dynamics and the image formation process is typically impractical, since they are often non-differentiable, and such long-range backpropagation can lead to extreme numerical instability, since the linearization of a suboptimal policy is likely to be unstable. This issue has also been observed in the related context of recurrent neural networks (Hochreiter et al., 2001; Pascanu and Bengio, 2012). The high dimensionality of the network also makes reinforcement learning difficult (Deisenroth et al., 2013). Pioneering early work on neural network control used

small, simple networks (Pomerleau, 1989; Hunt et al., 1992; Bekey and Goldberg, 1992; Lewis et al., 1998; Bakker et al., 2003; Mayer et al., 2006), and has largely been supplanted by methods that use carefully designed policies that can be learned efficiently with reinforcement learning (Kober et al., 2013). More recent work on sensorimotor deep learning has tackled simple task-space motions (Lenz et al., 2015a; Lampe and Riedmiller, 2013) and used unsupervised learning to obtain low-dimensional state spaces from images (Lange et al., 2012). Such methods have been demonstrated on tasks with a low-dimensional underlying structure: Lenz et al. (2015a) controls the end-effector in 2D space, while Lange et al. (2012) controls a 2-dimensional slot car with 1-dimensional actions. Our experiments include full torque control of 7-DoF robotic arms interacting with objects, with 30-40 state dimensions. In simple synthetic environments, control from images has been addressed with image features (Jodogne and Piater, 2007), nonparametric methods (van Hoof et al., 2015), and unsupervised state-space learning (Böhmer et al., 2013; Jonschkowski and Brock, 2014). CNNs have also been trained to play video games with Q-learning, Monte Carlo tree search, and stochastic search (Mnih et al., 2013; Koutník et al., 2013; Guo et al., 2014), and have been applied to simple simulated control tasks (Watter et al., 2015; Lillicrap et al., 2015). However, such methods have only been demonstrated on synthetic domains that lack the visual complexity of the real world, and require an impractical number of samples for real-world robotic learning. Our method is sample efficient, requiring only minutes of interaction time. To the best of our knowledge, this is the first method that can train deep visuomotor policies for complex, high-dimensional manipulation skills with direct torque control.

Learning visuomotor policies on a real robot requires handling complex observations and high dimensional policy representations. We tackle these challenges using guided policy search. In guided policy search, the policy is optimized using supervised learning, which scales gracefully with the dimensionality of the policy. The training set for supervised learning can be constructed using trajectory optimization under known dynamics (Levine and Koltun, 2013a,b, 2014; Mordatch and Todorov, 2014) and trajectory-centric reinforcement learning methods that operate under unknown dynamics (Levine and Abbeel, 2014; Levine et al., 2015), which is the approach taken in this work. In both cases, the supervision is adapted to the policy, to ensure that the final policy can reproduce the training data. The use of supervised learning in the inner loop of iterative policy search has also been proposed in the context of imitation learning (Ross et al., 2011, 2013). However, such methods typically do not address the question of how the supervision should be adapted to the policy.

The goal of our approach is also similar to visual servoing, which performs feedback control on feature points in a camera image (Espiau et al., 1992; Mohta et al., 2014; Wilson et al., 1996). However, our visuomotor policies are entirely learned from real-world data, and do not require feature points or feedback controllers to be specified by hand. This allows our method much more flexibility in choosing how to use the visual signal. Our approach also does not require any sort of camera calibration, in contrast to many visual servoing methods (though not all – see e.g. Jägersand et al. (1997); Yoshimi and Allen (1994)).

3. Background and Overview

In this section, we define the visuomotor policy learning problem and present an overview of our approach. The core component of our approach is a guided policy search algorithm

that separates the problem of learning visuomotor policies into separate supervised learning and trajectory learning phases, each of which is easier than optimizing the policy directly. We also discuss a policy architecture suitable for end-to-end learning of vision and control, and a training setup that allows our method to be applied to real robotic platforms.

3.1 Definitions and Problem Formulation

In policy search, the goal is to learn a policy $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ that allows an agent to choose actions $\mathbf{u}_t$ in response to observations $\mathbf{o}_t$ to control a dynamical system, such as a robot. The policy comes from some parametric class parameterized by θ, which could be, for example, the weights of a neural network. The system is defined by states $\mathbf{x}_t$, actions $\mathbf{u}_t$, and observations $\mathbf{o}_t$. For example, $\mathbf{x}_t$ might include the joint angles of the robot, the positions of objects in the world, and their time derivatives, $\mathbf{u}_t$ might consist of motor torque commands, and $\mathbf{o}_t$ might include an image from the robot's onboard camera. In this paper, we address finite horizon episodic tasks with $t \in [1, \ldots, T]$. The states evolve in time according to the system dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$, and the observations are, in general, a stochastic consequence of the states, according to $p(\mathbf{o}_t|\mathbf{x}_t)$. Neither the dynamics nor the observation distribution are assumed to be known in general. For notational convenience, we will use $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ to denote the distribution over actions under the policy conditioned on the state. However, since the policy is conditioned on the observation $\mathbf{o}_t$, this distribution is in fact given by $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) = \int \pi_\theta(\mathbf{u}_t|\mathbf{o}_t)p(\mathbf{o}_t|\mathbf{x}_t)d\mathbf{o}_t$. The dynamics and $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ together induce a distribution over trajectories $\tau = \{\mathbf{x}_1, \mathbf{u}_1, \mathbf{x}_2, \mathbf{u}_2, \ldots, \mathbf{x}_T, \mathbf{u}_T\}$:

$$\pi_\theta(\tau) = p(\mathbf{x}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t).$$

The goal of a task is given by a cost function $\ell(\mathbf{x}_t, \mathbf{u}_t)$, and the objective in policy search is to minimize the expectation $E_{\pi_\theta(\tau)}[\sum_{t=1}^{T} \ell(\mathbf{x}_t, \mathbf{u}_t)]$, which we will abbreviate as $E_{\pi_\theta(\tau)}[\ell(\tau)]$. A summary of the notation used in the paper is provided in Table 1.

3.2 Approach Summary

Our methods consists of two main components, which are illustrated in Figure 3. The first is a supervised learning algorithm that trains policies of the form $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t) = \mathcal{N}(\mu^\pi(\mathbf{o}_t), \Sigma^\pi(\mathbf{o}_t))$, where both $\mu^\pi(\mathbf{o}_t)$ and $\Sigma^\pi(\mathbf{o}_t)$ are general nonlinear functions. In our implementation, $\mu^\pi(\mathbf{o}_t)$ is a deep convolutional neural network, while $\Sigma^\pi(\mathbf{o}_t)$ is an observation-independent learned covariance, though other representations are possible. The second component is a trajectory-centric reinforcement learning (RL) algorithm that generates guiding distributions $p_i(\mathbf{u}_t|\mathbf{x}_t)$ that provide the supervision used to train the policy. These two components form a policy search algorithm that can be used to learn complex robotic tasks using only a high-level cost function $\ell(\mathbf{x}_t, \mathbf{u}_t)$. During training, only samples from the guiding distributions $p_i(\mathbf{u}_t|\mathbf{x}_t)$ are generated by running rollouts on the physical system, which avoids the need to execute partially trained neural network policies on physical hardware.

Supervised learning will not, in general, produce a policy with good long-horizon performance, since a small mistake on the part of the policy will place the system into states that are outside the distribution in the training data, causing compounding errors. To

symbol	definition	example/details
$\mathbf{x}_t$	Markovian system state at time step $t \in [1, T]$	joint angles, end-effector pose, object positions, and their velocities; dimensionality: 14 to 32
$\mathbf{u}_t$	control or action at time step $t \in [1, T]$	joint motor torque commands; dimensionality: 7 (for the PR2 robot)
$\mathbf{o}_t$	observation at time step $t \in [1, T]$	RGB camera image, joint encoder readings & velocities, end-effector pose; dimensionality: around 200,000
τ	trajectory: $\tau = \{\mathbf{x}_1, \mathbf{u}_1, \mathbf{x}_2, \mathbf{u}_2, \ldots, \mathbf{x}_T, \mathbf{u}_T\}$	notational shorthand for a sequence of states and actions
$\ell(\mathbf{x}_t, \mathbf{x}_t)$	cost function that defines the goal of the task	distance between an object in the gripper and the target
$p(\mathbf{x}_{t+1}\|\mathbf{x}_t, \mathbf{u}_t)$	unknown system dynamics	physics that govern the robot and any objects it interacts with
$p(\mathbf{o}_t\|\mathbf{x}_t)$	unknown observation distribution	stochastic process that produces camera images from system state
$\pi_\theta(\mathbf{u}_t\|\mathbf{o}_t)$	learned nonlinear global policy parameterized by weights θ	convolutional neural network, such as the one in Figure 2
$\pi_\theta(\mathbf{u}_t\|\mathbf{x}_t)$	$\int \pi_\theta(\mathbf{u}_t\|\mathbf{o}_t)p(\mathbf{o}_t\|\mathbf{x}_t)d\mathbf{o}_t$	notational shorthand for observation-based policy conditioned on state
$p_i(\mathbf{u}_t\|\mathbf{x}_t)$	learned local time-varying linear-Gaussian controller for initial state $\mathbf{x}_1^i$	time-varying linear-Gaussian controller has form $\mathcal{N}(\mathbf{K}_{ti}\mathbf{x}_t + \mathbf{k}_{ti}, \mathbf{C}_{ti})$
$\pi_\theta(\tau)$	trajectory distribution for $\pi_\theta(\mathbf{u}_t\|\mathbf{x}_t)$: $p(\mathbf{x}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{u}_t\|\mathbf{x}_t)p(\mathbf{x}_{t+1}\|\mathbf{x}_t, \mathbf{u}_t)$	notational shorthand for trajectory distribution induced by a policy

Table 1: Summary of the notation frequently used in this article.

avoid this issue, the training data must come from the policy's own state distribution (Ross et al., 2011). We achieve this by alternating between trajectory-centric RL and supervised learning. The RL stage adapts to the current policy $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$, providing supervision at states that are iteratively brought closer to the states visited by the policy. This is formalized as a variant of the BADMM algorithm (Wang and Banerjee, 2014) for constrained optimization, which can be used to show that, at convergence, the policy $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ and the guiding distributions $p_i(\mathbf{u}_t|\mathbf{x}_t)$ will exhibit the same behavior. This algorithm is derived in Section 4. The guiding distributions are substantially easier to optimize than learning the policy parameters directly (e.g., using model-free reinforcement learning), because they use the full state of the system $\mathbf{x}_t$, while the policy $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ only uses the observations. This means that the method requires the full state to be known during training, but not at test time. This makes it possible to efficiently learn complex visuomotor policies, but imposes additional assumptions on the observability of $\mathbf{x}_t$ during training that we discuss in Section 4.

When learning visuomotor tasks, the policy $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ is represented by a novel convolutional neural network (CNN) architecture, which we describe in Section 5.2. CNNs have enjoyed considerable success in computer vision (LeCun et al., 2015), but the most popular

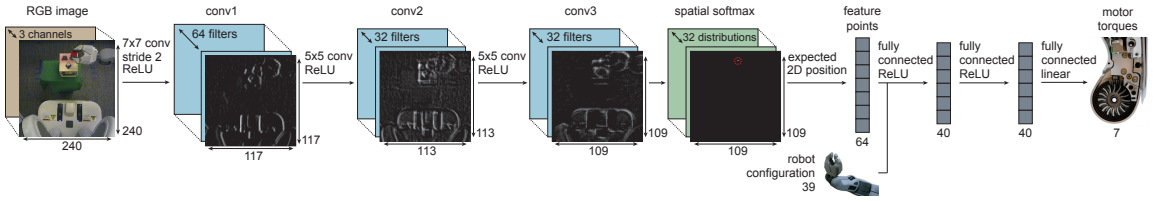


Figure 2: Visuomotor policy architecture. The network contains three convolutional layers, followed by a spatial softmax and an expected position layer that converts pixel-wise features to feature points, which are better suited for spatial computations. The points are concatenated with the robot configuration, then passed through three fully connected layers to produce the torques.

architectures rely on large datasets and focus on semantic tasks such as classification, often intentionally discarding spatial information. Our architecture, illustrated in Figure 2, uses a fixed transformation from the last convolutional layer to a set of spatial feature points, which form a concise representation of the visual scene suitable for feedback control. Our network has 7 layers and around 92,000 parameters, which presents a major challenge for standard policy search methods (Deisenroth et al., 2013).

To reduce the amount of experience needed to train visuomotor policies, we also introduce a pretraining scheme that allows us to train effective policies with a relatively small number of iterations. The pretraining steps are illustrated in Figure 3. The intuition behind our pretraining is that, although we ultimately seek to obtain sensorimotor policies that combine both vision and control, low-level aspects of vision can be initialized independently. To that end, we pretrain the convolutional layers of our network by predicting elements of $\mathbf{x}_t$ that are not provided in the observation $\mathbf{o}_t$, such as the positions of objects in the scene. We also initially train the guiding trajectory distributions $p_i(\mathbf{u}_t|\mathbf{x}_t)$ independently of the convolutional network until the trajectories achieve a basic level of competence at the task, and then switch to full guided policy search with end-to-end training of $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$. In our implementation, we also initialize the first layer filters from the model of Szegedy et al. (2014), which is trained on ImageNet (Deng et al., 2009) classification. The initialization and pretraining scheme is described in Section 5.2.

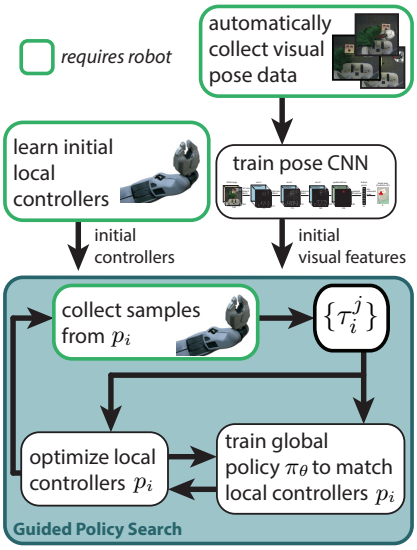
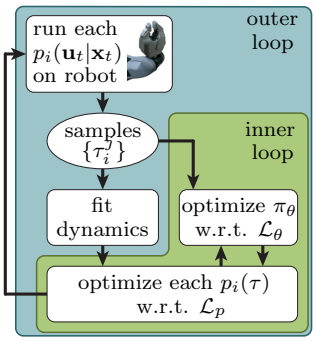


Figure 3: Diagram of our approach, including the main guided policy search phase and initialization phases.

4. Guided Policy Search with BADMM

Guided policy search transforms policy search into a supervised learning problem, where the training set is generated by a simple trajectory-centric RL algorithm. This algorithm

optimizes linear-Gaussian controllers $p_i(\mathbf{u}_t|\mathbf{x}_t)$, and is described in Section 4.2. We refer to the trajectory distribution induced by $p_i(\mathbf{u}_t|\mathbf{x}_t)$ as $p_i(\tau)$. Each $p_i(\mathbf{u}_t|\mathbf{x}_t)$ succeeds from different initial states. For example, in the task of placing a cap on a bottle, these initial states correspond to different positions of the bottle. By training on trajectories for multiple bottle positions, the final CNN policy can succeed from all initial states, and can generalize to other states from the same distribution.



The final policy $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ learned with guided policy search is only provided with observations $\mathbf{o}_t$ of the full state $\mathbf{x}_t$, and the dynamics are assumed to be unknown. A diagram of this method, which corresponds to an expanded version of the guided policy search box in Figure 3, is shown on the right. In the outer loop, we draw sample trajectories $\{\tau_i^j\}$ for each initial state on the physical system by running the corresponding controller $p_i(\mathbf{u}_t|\mathbf{x}_t)$. The samples are used to fit the dynamics $p_i(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ that are used to improve $p_i(\mathbf{u}_t|\mathbf{x}_t)$, and serve as training data for the policy. The inner loop alternates between optimizing each $p_i(\tau)$ and optimizing the policy to match these trajectory distributions. The policy is trained to predict the actions along each trajectory from the observations $\mathbf{o}_t$, rather than the full state $\mathbf{x}_t$. This allows the policy to directly use raw observations at test time. This alternating optimization can be framed as an instance of the BADMM algorithm (Wang and Banerjee, 2014), which converges to a solution where the trajectory distributions and the policy have the same state distribution. This allows greedy supervised training of the policy to produce a policy with good long-horizon performance.

4.1 Algorithm Derivation

Policy search methods minimize the expected cost $E_{\pi_\theta}[\ell(\tau)]$, where $\tau = \{\mathbf{x}_1, \mathbf{u}_1, \ldots, \mathbf{x}_T, \mathbf{u}_T\}$ is a trajectory, and $\ell(\tau) = \sum_{t=1}^{T} \ell(\mathbf{x}_t, \mathbf{u}_t)$ is the cost of an episode. In the fully observed case, the expectation is taken under $\pi_\theta(\tau) = p(\mathbf{x}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{u}_t|\mathbf{x}_t) p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$. The final policy $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ is conditioned on the observations $\mathbf{o}_t$, but $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ can be recovered as $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) = \int \pi_\theta(\mathbf{u}_t|\mathbf{o}_t) p(\mathbf{o}_t|\mathbf{x}_t) d\mathbf{o}_t$. We will present the derivation in this section for $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, but we do not require knowledge of $p(\mathbf{o}_t|\mathbf{x}_t)$ in the final algorithm. As discussed in Section 4.3, the integral will be evaluated with samples from the real system, which include both $\mathbf{x}_t$ and $\mathbf{o}_t$. We begin by rewriting the expected cost minimization as a constrained problem:

$$\min_{p,\pi_\theta} E_p[\ell(\tau)] \text{ s.t. } p(\mathbf{u}_t|\mathbf{x}_t) = \pi_\theta(\mathbf{u}_t|\mathbf{x}_t) \; \forall \mathbf{x}_t, \mathbf{u}_t, t, \tag{1}$$

where we will refer to $p(\tau)$ as a guiding distribution. This formulation is equivalent to the original problem, since the constraint forces the two distributions to be identical. However, if we approximate the initial state distribution $p(\mathbf{x}_1)$ with samples $\mathbf{x}_1^i$, we can choose $p(\tau)$ to be a class of distributions that is much easier to optimize than π_θ, as we will show later. This will allow us to use simple local learning methods for $p(\tau)$, without needing to train the complex neural network policy $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ directly with reinforcement learning, which would require a prohibitive amount of experience on real physical systems.

The constrained problem can be solved by a dual descent method, which alternates between minimizing the Lagrangian with respect to the primal variables, and incrementing

the Lagrange multipliers by their subgradient. Minimization of the Lagrangian with respect to $p(\tau)$ and θ is done in alternating fashion: minimizing with respect to θ corresponds to supervised learning (making π_θ match $p(\tau)$), and minimizing with respect to $p(\tau)$ consists of one or more trajectory optimization problems. The dual descent method we use is based on BADMM (Wang and Banerjee, 2014), a variant of ADMM (Boyd et al., 2011) that augments the Lagrangian with a Bregman divergence between the constrained variables. We use the KL-divergence as the Bregman constraint, which is particularly convenient for working with probability distributions. We will also modify the constraint $p(\mathbf{u}_t|\mathbf{x}_t) = \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ by multiplying both sides by $p(\mathbf{x}_t)$, to get $p(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_t) = \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_t)$. This constraint is equivalent, but has the convenient property that we can express the Lagrangian in terms of expectations. The BADMM augmented Lagrangians for θ and p are therefore given by

$$\mathcal{L}_\theta(\theta, p) = \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[\ell(\mathbf{x}_t, \mathbf{u}_t)] + E_{p(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)}[\lambda_{\mathbf{x}_t,\mathbf{u}_t}] - E_{p(\mathbf{x}_t,\mathbf{u}_t)}[\lambda_{\mathbf{x}_t,\mathbf{u}_t}] + \nu_t \phi_t^\theta(\theta, p)$$
$$\mathcal{L}_p(p, \theta) = \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[\ell(\mathbf{x}_t, \mathbf{u}_t)] + E_{p(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)}[\lambda_{\mathbf{x}_t,\mathbf{u}_t}] - E_{p(\mathbf{x}_t,\mathbf{u}_t)}[\lambda_{\mathbf{x}_t,\mathbf{u}_t}] + \nu_t \phi_t^p(\theta, p),$$

where $\lambda_{\mathbf{x}_t,\mathbf{u}_t}$ is the Lagrange multiplier for state $\mathbf{x}_t$ and action $\mathbf{u}_t$ at time t, and $\phi_t^\theta(\theta, p)$ are $\phi_t^p(\theta, p)$ are expectations of the KL-divergences:

$$\phi_t^p(p, \theta) = E_{p(\mathbf{x}_t)}[D_{\mathrm{KL}}(p(\mathbf{u}_t|\mathbf{x}_t)\|\pi_\theta(\mathbf{u}_t|\mathbf{x}_t))]$$
$$\phi_t^\theta(\theta, p) = E_{p(\mathbf{x}_t)}[D_{\mathrm{KL}}(\pi_\theta(\mathbf{u}_t|\mathbf{x}_t))\|p(\mathbf{u}_t|\mathbf{x}_t)].$$

Dual descent with alternating primal minimization is then described by the following steps:

$$\theta \leftarrow \arg\min_\theta \sum_{t=1}^{T} E_{p(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)}[\lambda_{\mathbf{x}_t,\mathbf{u}_t}] + \nu_t \phi_t^\theta(\theta, p)$$
$$p \leftarrow \arg\min_p \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[\ell(\mathbf{x}_t, \mathbf{u}_t) - \lambda_{\mathbf{x}_t,\mathbf{u}_t}] + \nu_t \phi_t^p(p, \theta)$$
$$\lambda_{\mathbf{x}_t,\mathbf{u}_t} \leftarrow \lambda_{\mathbf{x}_t,\mathbf{u}_t} + \alpha\nu_t(\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_t) - p(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_t)).$$

This procedure is an instance of BADMM, and therefore inherits its convergence guarantees. Note that we drop terms that are independent of the optimization variables on each line. The parameter α is a step size. As with most augmented Lagrangian methods, the weight ν_t is set heuristically, as described in Appendix A.1.

The dynamics only affect the optimization with respect to $p(\tau)$. In order to make this optimization efficient, we choose $p(\tau)$ to be a mixture of N Gaussians $p_i(\tau)$, one for each initial state sample $\mathbf{x}_1^i$. This makes the action conditionals $p_i(\mathbf{u}_t|\mathbf{x}_t)$ and the dynamics $p_i(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ linear-Gaussian, as discussed in Section 4.2. This is a reasonable choice when the system is deterministic, or the noise is Gaussian or small, and we found that this approach is sufficiently tolerant to noise for use on real physical systems. Our choice of p also assumes that the policy $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ is conditionally Gaussian. This is also reasonable, since the mean and covariance of $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ can be any nonlinear function of the observations

$\mathbf{o}_t$, which themselves are a function of the unobserved state $\mathbf{x}_t$. In Section 4.2, we show how these assumptions enable each $p_i(\tau)$ to be optimized very efficiently. We will refer to $p_i(\tau)$ as guiding distributions, since they serve to focus the policy on good, low-cost behaviors.

Aside from learning $p_i(\tau)$, we must choose a tractable way to represent the infinite set of constraints $p(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_t) = \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_t)$. One approximate approach proposed in prior work is to replace the exact constraints with expectations of features (Peters et al., 2010). When the features consist of linear, quadratic, or higher order monomial functions of the random variable, this can be viewed as a constraint on the moments of the distributions. If we only use the first moment, we get a constraint on the expected action: $E_{p(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_t)}[\mathbf{u}_t] = E_{\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_t)}[\mathbf{u}_t]$. If the stochasticity in the dynamics is low, as we assumed previously, the optimal solution for each $p_i(\tau)$ will have low entropy, making this first moment constraint a reasonable approximation. The KL-divergence terms in the augmented Lagrangians will still serve to softly enforce agreement between the higher moments. While this simplification is quite drastic, we found that it was more stable in practice than including higher moments, likely because these higher moments are harder to estimate accurately with a limited number of samples. The alternating optimization is now given by

$$\theta \leftarrow \arg\min_\theta \sum_{t=1}^{T} E_{p(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)}[\mathbf{u}_t^{\mathrm{T}}\lambda_{\mu t}] + \nu_t \phi_t^\theta(\theta, p) \tag{2}$$

$$p \leftarrow \arg\min_p \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[\ell(\mathbf{x}_t, \mathbf{u}_t) - \mathbf{u}_t^{\mathrm{T}}\lambda_{\mu t}] + \nu_t \phi_t^p(p, \theta) \tag{3}$$

$$\lambda_{\mu t} \leftarrow \lambda_{\mu t} + \alpha\nu_t(E_{\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_t)}[\mathbf{u}_t] - E_{p(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_t)}[\mathbf{u}_t]),$$

where $\lambda_{\mu t}$ is the Lagrange multiplier on the expected action at time t. In the rest of the paper, we will use $\mathcal{L}_\theta(\theta, p)$ and $\mathcal{L}_p(p, \theta)$ to denote the two augmented Lagrangians in Equations (2) and (3), respectively. In the next two sections, we will describe how $\mathcal{L}_p(p, \theta)$ can be optimized with respect to p under unknown dynamics, and how $\mathcal{L}_\theta(\theta, p)$ can be optimized for complex, high-dimensional policies. Implementation details of the BADMM optimization are presented in Appendix A.1.

4.2 Trajectory Optimization under Unknown Dynamics

Since the Lagrangian $\mathcal{L}_p(p, \theta)$ in the previous section factorizes over the mixture elements in $p(\tau) = \sum_i p_i(\tau)$, we describe the trajectory optimization method for a single Gaussian $p(\tau)$. When there are multiple mixture elements, this procedure is applied in parallel to each $p_i(\tau)$. Since $p(\tau)$ is Gaussian, the conditionals $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ and $p(\mathbf{u}_t|\mathbf{x}_t)$, which correspond to the dynamics and the controller, are time-varying linear-Gaussian, and given by

$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \mathbf{C}_t) \qquad p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}t}\mathbf{x}_t + f_{\mathbf{u}t}\mathbf{u}_t + f_{ct}, \mathbf{F}_t).$$

This type of controller can be learned efficiently with a small number of real-world samples, making it a good choice for optimizing the guiding distributions. Since a different set of time-varying linear-Gaussian dynamics is fitted for each initial state, this dynamics representation can model any continuous deterministic system that can be locally linearized. Stochastic dynamics can violate the local linearity assumption in principle, but we found that in practice this representation was well suited for a wide variety of noisy real-world tasks.

The dynamics are determined by the environment. If they are known, $p(\mathbf{u}_t|\mathbf{x}_t)$ can be optimized with a variant of the iterative linear-quadratic-Gaussian regulator (iLQG) (Li and Todorov, 2004; Levine and Koltun, 2013a), which is a variant of DDP (Jacobson and Mayne, 1970). In the case of unknown dynamics, we can fit $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ to sample trajectories sampled from the trajectory distribution at the previous iteration, denoted $\hat{p}(\tau)$. If $\hat{p}(\tau)$ is too different from $p(\tau)$, these samples will not give a good estimate of $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$, and the optimization will diverge. To avoid this, we can bound the change from $\hat{p}(\tau)$ to $p(\tau)$ in terms of their KL-divergence by a step size ϵ, producing the following constrained problem:

$$\min_{p(\tau)\in\mathcal{N}(\tau)} \mathcal{L}_p(p, \theta) \text{ s.t. } D_{\mathrm{KL}}(p(\tau)\|\hat{p}(\tau)) \leq \epsilon.$$

This type of policy update has previously been proposed by several authors in the context of policy search (Bagnell and Schneider, 2003; Peters and Schaal, 2008; Peters et al., 2010; Levine and Abbeel, 2014). In the case when $p(\tau)$ is Gaussian, this problem can be solved efficiently using dual gradient descent, while the dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ are fitted to samples gathered by running the previous controller $\hat{p}(\mathbf{u}_t|\mathbf{x}_t)$ on the robot. Fitting a global Gaussian mixture model to tuples $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$ and using it as a prior for fitting the dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ serves to greatly reduce the sample complexity. We describe the dynamics fitting procedure in detail in Appendix A.3.

Note that the trajectory optimization cost function $\mathcal{L}_p(p, \theta)$ also depends on the policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, while we only have access to $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$. In order to compute a local quadratic expansion of the KL-divergence term $D_{\mathrm{KL}}(p(\mathbf{u}_t|\mathbf{x}_t)\|\pi_\theta(\mathbf{u}_t|\mathbf{x}_t))$ inside $\mathcal{L}_p(p, \theta)$ for iLQG, we also estimate a linearization of the mean of the conditionally Gaussian policy $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ with respect to the state $\mathbf{x}_t$, using the same procedure that we use to linearize the dynamics. The data for this estimation consists of tuples $\{\mathbf{x}_t^i, E_{\pi_\theta(\mathbf{u}_t|\mathbf{o}_t^i)}[\mathbf{u}_t]\}$, which we can obtain because both the states $\mathbf{x}_t^i$ and the observations $\mathbf{o}_t^i$ are available for all of the samples evaluated on the real physical system.

This constrained optimization is performed in the "inner loop" of the optimization described in the previous section, and the KL-divergence constraint $D_{\mathrm{KL}}(p(\tau)\|\hat{p}(\tau)) \leq \epsilon$ imposes a step size on the trajectory update. The overall algorithm then becomes an instance of generalized BADMM (Wang and Banerjee, 2014). Note that the augmented Lagrangian $\mathcal{L}_p(p, \theta)$ consists of an expectation under $p(\tau)$ of a quantity that is independent of p. We can locally approximate this quantity with a quadratic by using a quadratic expansion of $\ell(\mathbf{x}_t, \mathbf{u}_t)$, and fitting a linear-Gaussian to $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ with the same method we used for the dynamics. We can then solve the primal optimization in the dual gradient descent procedure with a standard LQR backward pass. This is significantly simpler and much faster than the forward-backward dynamic programming procedure employed in previous work (Levine and Abbeel, 2014; Levine and Koltun, 2014). This improvement is enabled by the use of BADMM, which allows us to always formulate the KL-divergence term in the Lagrangian with the distribution being optimized as the first argument. Since the KL-divergence is convex in its first argument, this makes the corresponding optimization significantly easier. The details of this LQR-based dual gradient descent algorithm are derived in Appendix A.4.

We can further improve the efficiency of the method by allowing samples from multiple trajectories $p_i(\tau)$ to be used to fit a shared dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$, while the controllers $p_i(\mathbf{u}_t|\mathbf{x}_t)$ are allowed to vary. This makes sense when the initial states of these trajectories

are similar, and they therefore visit similar regions. This allows us to draw just a single sample from each $p_i(\tau)$ at each iteration, allowing us to handle many more initial states.

4.3 Supervised Policy Optimization

Since the policy parameters θ participate only in the constraints of the optimization problem in Equation (1), optimizing the policy corresponds to minimizing the KL-divergence between the policy and trajectory distribution, as well as the expectation of $\lambda_{\mu t}^{\mathrm{T}} \mathbf{u}_t$. For a conditional Gaussian policy of the form $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t) = \mathcal{N}(\mu^\pi(\mathbf{o}_t), \Sigma^\pi(\mathbf{o}_t))$, the objective is

$$\mathcal{L}_\theta(\theta, p) = \frac{1}{2N} \sum_{i=1}^{N} \sum_{t=1}^{T} E_{p_i(\mathbf{x}_t, \mathbf{o}_t)} \big[\mathrm{tr}[\mathbf{C}_{ti}^{-1} \Sigma^\pi(\mathbf{o}_t)] - \log |\Sigma^\pi(\mathbf{o}_t)| \\ + (\mu^\pi(\mathbf{o}_t) - \mu_{ti}^p(\mathbf{x}_t)) \mathbf{C}_{ti}^{-1} (\mu^\pi(\mathbf{o}_t) - \mu_{ti}^p(\mathbf{x}_t)) + 2\lambda_{\mu t}^{\mathrm{T}} \mu^\pi(\mathbf{o}_t) \big] ,$$

where $\mu_{ti}^p(\mathbf{x}_t)$ is the mean of $p_i(\mathbf{u}_t|\mathbf{x}_t)$ and $\mathbf{C}_{ti}$ is the covariance, and the expectation is evaluated using samples from each $p_i(\tau)$ with corresponding observations $\mathbf{o}_t$. The observations are sampled from $p(\mathbf{o}_t|\mathbf{x}_t)$ by recording camera images on the real system. Since the input to $\mu^\pi(\mathbf{o}_t)$ and $\Sigma^\pi(\mathbf{o}_t)$ is not the state $\mathbf{x}_t$, but only an observation $\mathbf{o}_t$, we can train the policy to directly use raw observations. Note that $\mathcal{L}_\theta(\theta, p)$ is simply a weighted quadratic loss on the difference between the policy mean and the mean action of the trajectory distribution, offset by the Lagrange multiplier. The weighting is the precision matrix of the conditional in the trajectory distribution, which is equal to the curvature of its cost-to-go function (Levine and Koltun, 2013a). This has an intuitive interpretation: $\mathcal{L}_\theta(\theta, p)$ penalizes deviation from the trajectory distribution, with a penalty that is locally proportional to its cost-to-go. At convergence, when the policy $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ takes the same actions as $p_i(\mathbf{u}_t|\mathbf{x}_t)$, their Q-functions are equal, and the supervised policy objective becomes equivalent to the policy iteration objective (Levine and Koltun, 2014)

In this work, we optimize $\mathcal{L}_\theta(\theta, p)$ with respect to θ using stochastic gradient descent (SGD), a standard method for neural network training. The covariance of the Gaussian policy does not depend on the observation in our implementation, though adding this dependence would be straightforward. Since training complex neural networks requires a substantial number of samples, we found it beneficial to include sampled observations from previous iterations into the policy optimization, evaluating the action $\mu_{ti}^p(\mathbf{x}_t)$ at their corresponding states using the current trajectory distributions. Since these samples come from the wrong state distribution, we use importance sampling and weight them according to the ratio of their probability under the current distribution $p(\mathbf{x}_t)$ and the one they were sampled from, which is straightforward to evaluate under the estimated linear-Gaussian dynamics (Levine and Koltun, 2013b).

4.4 Comparison with Prior Guided Policy Search Methods

We presented a guided policy search method where the policy is trained on observations, while the trajectories are trained on the full state. The BADMM formulation of guided policy search is new to this work, though several prior guided policy search methods based on constrained optimization have been proposed. Levine and Koltun (2014) proposed a formulation similar to Equation (1), but with a constraint on the KL-divergence between

$p(\tau)$ and π_θ. This results in a more complex, non-convex forward-backward trajectory optimization phase. Since the BADMM formulation solves a convex problem during the trajectory optimization phase, it is substantially faster and easier to implement and use, especially when the number of trajectories $p_i(\tau)$ is large.

The use of ADMM for guided policy search was also proposed by Mordatch and Todorov (2014) for deterministic policies under known dynamics. This approach requires known, deterministic dynamics and trains deterministic policies. Furthermore, because this approach uses a simple quadratic augmented Lagrangian term, it further requires penalty terms on the gradient of the policy to account for local feedback. Our approach enforces this feedback behavior due to the higher moments included in the KL-divergence term, but does not require computing the second derivative of the policy.

5. End-to-End Visuomotor Policies

Guided policy search allows us to optimize complex, high-dimensional policies with raw observations, such as when the input to the policy consists of images from a robot's onboard camera. However, leveraging this capability to directly learn policies for visuomotor control requires designing a policy representation that is both data-efficient and capable of learning complex control strategies directly from raw visual inputs. In this section, we describe a deep convolutional neural network (CNN) model that is uniquely suited to this task. Our approach combines a novel spatial soft-argmax layer with a pretraining procedure that provides for flexibility and data-efficiency.

5.1 Visuomotor Policy Architecture

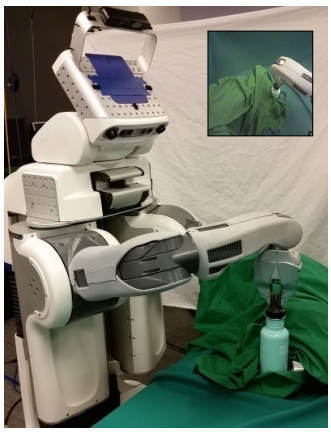
Our visuomotor policy runs at 20 Hz on the robot, mapping monocular RGB images and the robot configurations to joint torques on a 7 DoF arm. The configuration includes the angles of the joints and the pose of the end-effector (defined by 3 points in the space of the end-effector), as well as their velocities, but does not include the position of the target object or goal, which must be determined from the image. CNNs often use pooling to discard the locational information that is necessary to determine positions, since it is an irrelevant distractor for tasks such as object classification (Lee et al., 2009). Because locational information is important for control, our policy does not use pooling. Additionally, CNNs built for spatial tasks such as human pose estimation often also rely on the availability of location labels in image-space, such as hand-labeled keypoints (Tompson et al., 2014). We propose a novel CNN architecture capable of estimating spatial information from an image without direct supervision in image space. Our pose estimation experiments, discussed in Section 5.2, show that this network can learn useful visual features using only 3D position information provided by the robot, and no camera calibration. Further training the network with guided policy search to directly output motor torques causes it to acquire *task-specific* visual features. Our experiments in Section 6.4 show that this improves performance beyond the level achieved with features trained only for pose estimation.

Our network architecture is shown in Figure 2. The visual processing layers of the network consist of three convolutional layers, each of which learns a bank of filters that are applied to patches centered on every pixel of its input. These filters form a hierarchy of local image features. Each convolutional layer is followed by a rectifying nonlinearity of

the form $a_{cij} = \max(0, z_{cij})$ for each channel c and each pixel coordinate (i, j). The third convolutional layer contains 32 response maps with resolution 109×109. These response maps are passed through a spatial softmax function of the form $s_{cij} = e^{a_{cij}} / \sum_{i'j'} e^{a_{ci'j'}}$. Each output channel of the softmax is a probability distribution over the location of a feature in the image. To convert from this distribution to a coordinate representation (f_{cx}, f_{cy}), the network calculates the expected image position of each feature, yielding a 2D coordinate for each channel: $f_{cx} = \sum_{ij} s_{cij} x_{ij}$ and $f_{cy} = \sum_{ij} s_{cij} y_{ij}$, where (x_{ij}, y_{ij}) is the image-space position of the point (i, j) in the response map. Since this is a linear operation, it corresponds to a fixed, sparse fully connected layer with weights $W_{cix} = x_{ij}$ and $W_{cjy} = y_{ij}$. The combination of the spatial softmax and expectation operator implement a kind of soft-argmax. The spatial feature points (f_{cx}, f_{cy}) are concatenated with the robot's configuration and fed into two fully connected layers, each with 40 rectified units, followed by linear connections to the torques. The full network contains about 92,000 parameters, of which 86,000 are in the convolutional layers.

The spatial softmax and the expected position computation serve to convert pixel-wise representations in the convolutional layers to spatial coordinate representations, which can be manipulated by the fully connected layers into 3D positions or motor torques. The softmax also provides lateral inhibition, which suppresses low, erroneous activations, only keeping strong activations that are more likely to be accurate. This makes our policy more robust to distractors, providing generalization to novel visual variation. We compare our architecture with more standard alternatives in Section 6.3 and evaluate robustness to visual distractors in Section 6.4. However, the proposed architecture is also in some sense more specialized for visuomotor control, in contrast to more general standard convolutional networks. For example, not all perception tasks require information that can be coherently summarized by a set of spatial locations.

5.2 Visuomotor Policy Training

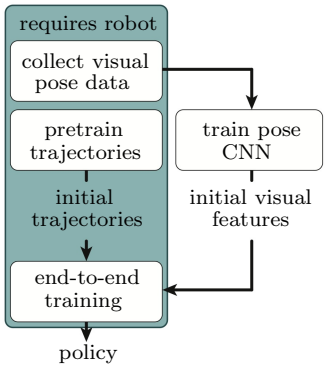


The guided policy search trajectory optimization phase uses the full state of the system, though the final policy only uses the observations. This type of instrumented training is a natural choice for many robotics tasks, where the robot is trained under controlled conditions, but must then act intelligently in uncontrolled, real-world situations. In our tasks, the unobserved variables are the pose of a target object (e.g. the bottle on which a cap must be placed). During training, this target object is typically held in the robot's left gripper, while the robot's right arm performs the task, as shown to the right. This allows the robot to move the target through a range of known positions. The final visuomotor policy does not receive this position as input, but must instead use the camera images. Due to the modest amount of training data, distractors that are correlated with task-relevant variables can hamper generalization. For this reason, the left arm is covered with cloth to prevent the policy from associating its appearance with the object's position.

While we can train the visuomotor policy entirely from scratch, the algorithm would spend a large number of iterations learning basic visual features and arm motions that can more efficiently be learned by themselves, before being incorporated into the policy. To speed up learning, we initialize both the vision layers in the policy and the trajectory distributions for guided policy search by leveraging the fully observed training setup. To initialize the vision layers, the robot moves the target object through a range of random positions, recording camera images and the object's pose, which is computed automatically from the pose of the gripper. This dataset is used to train a pose regression CNN, which consists of the same vision layers as the policy, followed by a fully connected layer that outputs the 3D points that define the target. Since the training set is still small (we use 1000 images collected from random arm motions), we initialize the filters in the first layer with weights from the model of Szegedy et al. (2014), which is trained on ImageNet (Deng et al., 2009) classification. After training on pose regression, the weights in the convolutional layers are transferred to the policy CNN. This enables the robot to learn the appearance of the objects prior to learning the behavior.

To initialize the linear-Gaussian controllers for each of the initial states, we take 15 iterations of guided policy search without optimizing the visuomotor policy. This allows for much faster training in the early iterations, when the trajectories are not yet successful, and optimizing the full visuomotor policy is unnecessarily time consuming. Since we still want the trajectories to arrive at compatible strategies for each target position, we replace the visuomotor policy during these iterations with a small network that receives the full state, which consisted of two layers with 40 rectified linear hidden units in our experiments. This network serves only to constrain the trajectories and avoid divergent behaviors from emerging for similar initial states, which would make subsequent policy learning difficult.

After initialization, we train the full visuomotor policy with guided policy search. During the supervised policy optimization phase, the fully connected motor control layers are first optimized by themselves, since they are not initialized with pretraining. This can be done very quickly because these layers are small. Then, the entire network is further optimized end-to-end. We found that first training the upper layers before end-to-end optimization prevented the convolutional layers from forgetting useful features learning during pretraining, when the error signal due to the untrained upper layers is very large. The entire pretraining scheme is summarized in the diagram on the right. Note that the trajectories can be pretrained in parallel with the vision layer pretraining, which does not require access to the physical system. Furthermore, the entire initialization procedure does not use any additional information that is not already available from the robot.



6. Experimental Evaluation

In this section, we present a series of experiments aimed at evaluating our approach and answering the following questions:

1. How does the guided policy search algorithm compare to other policy search methods for training complex, high-dimensional policies, such as neural networks?
2. Does our trajectory optimization algorithm work on a real robotic platform with unknown dynamics, for a range of different tasks?
3. How does our spatial softmax architecture compare to other, more standard convolutional neural network architectures?
4. Does training the perception and control systems in a visuomotor policy jointly end-to-end provide better performance than training each component separately?

Evaluating a wide range of policy search algorithms on a real robot would be extremely time consuming, particularly for methods that require a large number of samples. We therefore answer question (1) by using a physical simulator and simpler policies that do not use vision. This also allows us to test the generality of guided policy search on tasks that include manipulation, walking, and swimming. To answer question (2), we present a wide range of experiments on a PR2 robot. These experiments allow us to evaluate the sample efficiency of our trajectory optimization algorithm. To address question (3), we compare a range of different policy architectures on the task of localizing a target object (the cube in the shape sorting cube task). Since localizing the target object is a prerequisite for completing the shape sorting cube task, this serves as a good proxy for evaluating different architectures. Finally, we answer the last and most important question (4) by training visuomotor policies for hanging a coat hanger on a clothes rack, inserting a block into a shape sorting cube, fitting the claw of a toy hammer under a nail with various grasps, and screwing on a bottle cap. These tasks are illustrated in Figure 8.

6.1 Simulated Comparisons to Prior Policy Search Methods

In this section, we compare our method against prior policy search techniques on a range of simulated robotic control tasks. These results previously appeared in our conference paper that introduced the trajectory optimization procedure with local linear models (Levine and Abbeel, 2014). In these tasks, the state $\mathbf{x}_t$ consists of the joint angles and velocities of each robot, and the actions $\mathbf{u}_t$ consist of the torques at each joint. The neural network policies used one hidden layer and soft rectifier nonlinearities of the form $a = \log(1 + \exp(z))$. Since these policies use the state as input, they only have a few hundred parameters, far fewer than our visuomotor policies. However, even this number of parameters can pose a major challenge for prior policy search methods (Deisenroth et al., 2013).

Experimental tasks. We simulated 2D and 3D peg insertion, octopus arm control, and planar swimming and walking. The difficulty in the peg insertion tasks stems from the need to align the peg with the slot and the complex contacts between the peg and the walls, which result in discontinuous dynamics. Octopus arm control involves moving the tip of a flexible arm to a goal position (Engel et al., 2005). The challenge in this task stems from its high dimensionality: the arm has 25 degrees of freedom, corresponding to 50 state dimensions. The swimming task requires controlling a three-link snake, and the walking task requires a seven-link biped to maintain a target velocity. The challenge in these tasks comes from underactuation. Details of the simulation and cost for each task are in Appendix B.1.

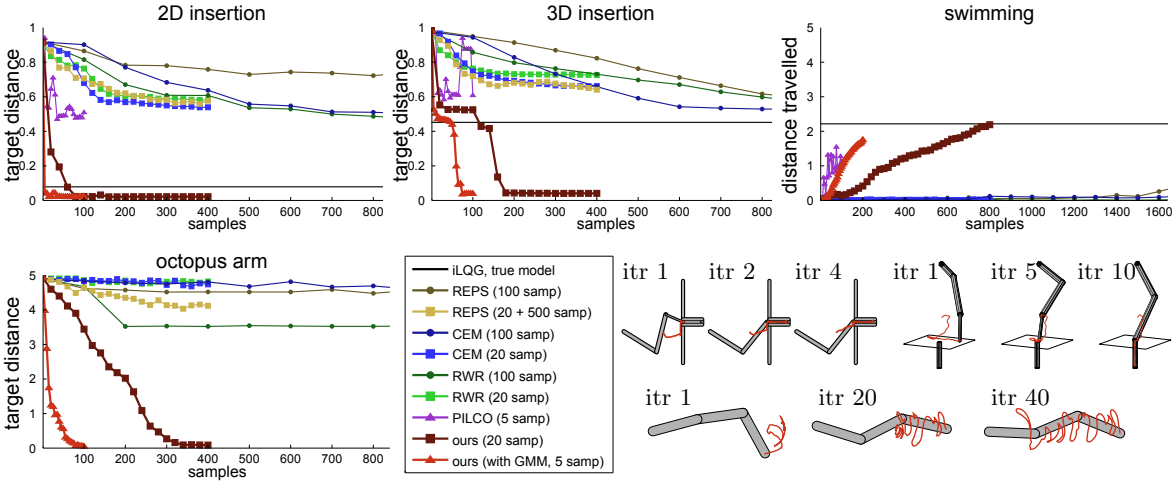


Figure 4: Results for learning linear-Gaussian controllers for 2D and 3D insertion, octopus arm, and swimming. Our approach uses fewer samples and finds better solutions than prior methods, and the GMM further reduces the required sample count. Images in the lower-right show the last time step for each system at several iterations of our method, with red lines indicating end effector trajectories.

Prior methods. We compare to REPS (Peters et al., 2010), reward-weighted regression (RWR) (Peters and Schaal, 2007; Kober and Peters, 2009), the cross-entropy method (CEM) (Rubinstein and Kroese, 2004), and PILCO (Deisenroth and Rasmussen, 2011). We also use iLQG (Li and Todorov, 2004) with a known model as a baseline, shown as a black horizontal line in all plots. REPS is a model-free method that, like our approach, enforces a KL-divergence constraint between the new and old policy. We compare to a variant of REPS that also fits linear dynamics to generate 500 pseudo-samples (Lioutikov et al., 2014), which we label "REPS (20 + 500)." RWR is an EM algorithm that fits the policy to previous samples weighted by the exponential of their reward, and CEM fits the policy to the best samples in each batch. With Gaussian trajectories, CEM and RWR only differ in the weights. These methods represent a class of RL algorithms that fit the policy to weighted samples, including PoWER and PI2 (Kober and Peters, 2009; Theodorou et al., 2010; Stulp and Sigaud, 2012). PILCO is a model-based method that uses a Gaussian process to learn a global dynamics model that is used to optimize the policy. We used the open-source implementation of PILCO provided by the authors. Both REPS and PILCO require solving large nonlinear optimizations at each iteration, while our method does not. Our method used 5 rollouts with the Gaussian mixture model prior, and 20 without. Due to its computational cost, PILCO was provided with 5 rollouts per iteration, while other prior methods used 20 and 100. For all prior methods with free hyperparameters (such as the fraction of elites for CEM), we performed hyperparameter sweeps and chose the most successful settings for the comparison.

Gaussian trajectory distributions. In the first set of comparisons, we evaluate only the trajectory optimization procedure for training linear-Gaussian controllers under unknown dynamics to determine its sample-efficiency and applicability to complex, high-dimensional problems. The results of this comparison for the peg insertion, octopus arm, and swimming

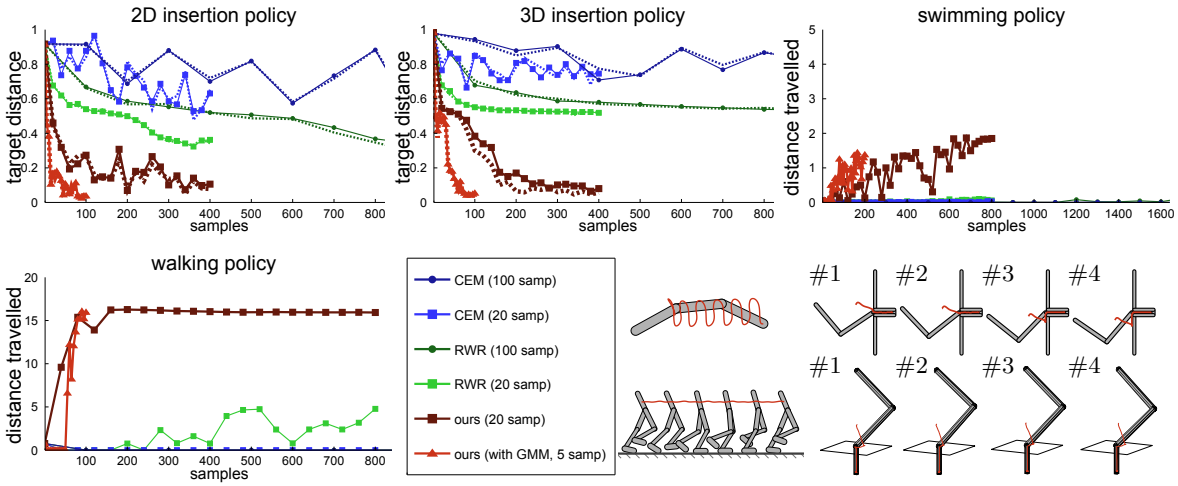


Figure 5: Comparison on neural network policies. For insertion, the policy was trained to search for an unknown slot position on four slot positions (shown above). Generalization to new positions is graphed with dashed lines. Note how the end effector (red) follows the surface to find the slot, and how the swimming gait is smoother due to the stationary policy.

tasks appears in Figure 4. The horizontal axis shows the total number of samples, and the vertical axis shows the minimum distance between the end of the peg and the bottom of the slot, the distance to the target for the octopus arm, or the total distance travelled by the swimmer. Since the peg is 0.5 units long, distances above this amount correspond to controllers that cannot perform an insertion. Our method learned much more effective controllers with fewer samples, especially when using the Gaussian mixture model prior. On 3D insertion, it outperformed the iLQG baseline, which used a known model. Contact discontinuities cause problems for derivative-based methods like iLQG, as well as methods like PILCO that learn a smooth global dynamics model. We use a time-varying local model, which preserves more detail, and fitting the model to samples has a smoothing effect that mitigates discontinuity issues. Prior policy search methods could servo to the hole, but were unable to insert the peg. On the octopus arm, our method succeeded despite the high dimensionality of the state and action spaces.[1] Our method also successfully learned a swimming gait, while prior model-free methods could not initiate forward motion. PILCO also learned an effective gait due to the smooth dynamics of this task, but its GP-based optimization required orders of magnitude more computation time than our method, taking about 50 minutes per iteration. In the case of prior model-free methods, the high dimensionality of the time-varying linear-Gaussian controllers likely caused considerable difficulty (Deisenroth et al., 2013), while our approach exploits the structure of linear-Gaussian controllers for efficient learning.

1. The high dimensionality of the octopus arm made it difficult to run PILCO, though in principle, such methods should perform well on this task given the arm's smooth dynamics.

Neural network policies. In the second set of comparisons, shown in Figure 5, we compare guided policy search to RWR and CEM[2] on the challenging task of training high-dimensional neural network policies for the peg insertion and locomotion tasks. The variant of guided policy search used in this comparison differs somewhat from the version described in Section 4, in that it used a simpler dual gradient descent formulation, rather than BADMM. In practice, we found the performance of these methods to be very similar, though the BADMM variant was substantially faster and easier to implement.

On swimming, our method achieved similar performance to the linear-Gaussian case, but since the neural network policy was stationary, the resulting gait was much smoother. Previous methods could only solve this task with 100 samples per iteration, with RWR eventually obtaining a distance of 0.5m after 4000 samples, and CEM reaching 2.1m after 3000. Our method was able to reach such distances with many fewer samples. Following prior work (Levine and Koltun, 2013a), the walker trajectory was initialized from a demonstration, which was stabilized with simple linear feedback. The RWR and CEM policies were initialized with samples from this controller to provide a fair comparison. The graph shows the average distance travelled on rollouts that did not fall, and shows that only our method was able to learn walking policies that succeeded consistently.

On peg insertion, the neural network was trained to insert the peg without precise knowledge of the position of the hole, resulting in a partially observed problem. The holes were placed in a region of radius 0.2 units in 2D and 0.1 units in 3D. The policies were trained on four different hole positions, and then tested on four new hole positions to evaluate generalization. The hole position was not provided to the neural network, and the policies therefore had to search for the hole, with only joint angles and velocities as input. Only our method could acquire a successful strategy to locate both the training and test holes, although RWR was eventually able to insert the peg into one of the four holes in 2D.

These comparisons show that training even medium-sized neural network policies for continuous control tasks with a limited number of samples is very difficult for many prior policy search algorithms. Indeed, it is generally known that model-free policy search methods struggle with policies that have over 100 parameters (Deisenroth et al., 2013). In subsequent sections, we will evaluate our method on real robotic tasks, showing that it can scale from these simulated tasks all the way up to end-to-end learning of visuomotor control.

6.2 Learning Linear-Gaussian Controllers on a PR2 Robot

In this section, we demonstrate the range of manipulation tasks that can be learned using our trajectory optimization algorithm on a real PR2 robot. These experiments previously appeared in our conference paper on guided policy search (Levine et al., 2015). Since performing trajectory optimization is a prerequisite for guided policy search to learn effective visuomotor policies, it is important to evaluate that our trajectory optimization can learn a wide variety of robotic manipulation tasks under unknown dynamics. The tasks in these experiments are shown in Figure 6, while Figure 7 shows the learning curves for each task. For all robotic experiments in this article, the tasks were learned entirely from scratch,

2. PILCO cannot optimize neural network policies, and we could not obtain reasonable results with REPS. Prior applications of REPS generally focus on simpler, lower-dimensional policy classes (Peters et al., 2010; Lioutikov et al., 2014).

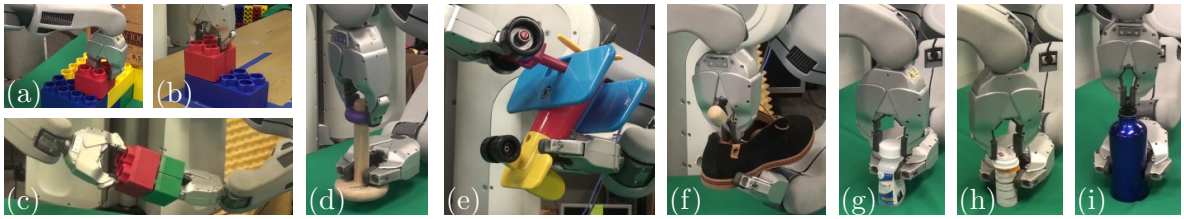


Figure 6: Tasks for linear-Gaussian controller evaluation: (a) stacking lego blocks on a fixed base, (b) onto a free-standing block, (c) held in both gripper; (d) threading wooden rings onto a peg; (e) attaching the wheels to a toy airplane; (f) inserting a shoe tree into a shoe; (g,h) screwing caps onto pill bottles and (i) onto a water bottle.

with the initialization of the controllers $p(\mathbf{u}_t|\mathbf{x}_t)$ described in Appendix B.2. The number of samples required to learn each controller is around 20-25, substantially lower than many prior policy search methods in the literature (Peters and Schaal, 2008; Kober et al., 2010b; Theodorou et al., 2010; Deisenroth et al., 2013). Total learning time was about ten minutes for each task, of which only 3-4 minutes involved system interaction. The rest of the time was spent resetting the robot to the initial state and on computation.

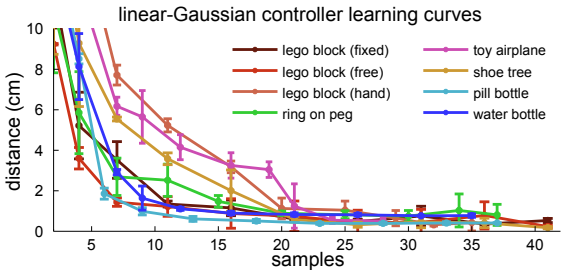


Figure 7: Distance to target point during training of linear-Gaussian controllers. The actual target may differ due to perturbations. Error bars indicate one standard deviation.

The linear-Gaussian controllers are optimized for a specific condition – e.g., a specific position of the target lego block. To evaluate their robustness to errors in the specified target position, we conducted experiments on the lego block and ring tasks where the target object (the lower block and the peg) was perturbed at each trial during training, and then tested with various perturbations. For each task, controllers were trained with Gaussian perturbations with standard deviations of 0, 1, and 2 cm in the position of the target object, and each controller was tested with perturbations of radius 0, 1, 2, and 3 cm. Note that with a radius of 2 cm, the peg would be placed about one ring-width away from the expected position. The results are shown in Table 2. All controllers were robust to perturbations of 1 cm, and would often succeed at 2 cm. Robustness increased slightly when more noise was injected during training, but even controllers trained without noise exhibited considerable robustness, since the linear-Gaussian controllers themselves add noise during sampling. We also evaluated a kinematic baseline for each perturbation level, which planned a straight path from a point 5 cm above the target to the expected (unperturbed) target location. This baseline was only able to place the lego block in the absence of perturbations. The rounded top of the peg provided an easier condition for the baseline, with occasional successes at higher perturbation levels. Our controllers outperformed the baseline by a wide margin.

All of the robotic experiments discussed in this section may be viewed in the corresponding supplementary video, available online: `http://rll.berkeley.edu/icra2015gps`. A video illustration of the visuomotor policies, discussed in the following sections, is also available: `http://sites.google.com/site/visuomotorpolicy`.

		test perturbation							
		lego block				ring on peg			
		0 cm	1 cm	2 cm	3 cm	0 cm	1 cm	2 cm	3 cm
training perturb.	0 cm	5/5	5/5	3/5	2/5	5/5	5/5	0/5	0/5
	1 cm	5/5	5/5	3/5	2/5	5/5	5/5	3/5	0/5
	2 cm	5/5	5/5	5/5	3/5	5/5	5/5	3/5	0/5
	kinematic baseline	5/5	0/5	0/5	0/5	5/5	3/5	0/5	0/5

Table 2: Success rates of linear-Gaussian controllers under target object perturbation.

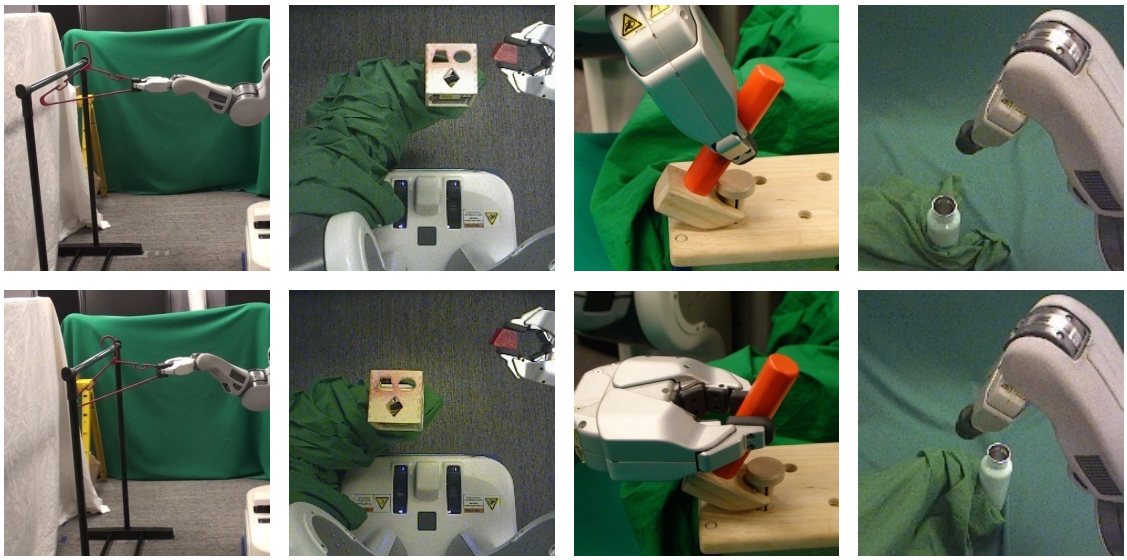
6.3 Spatial Softmax CNN Architecture Evaluation

In this section, we evaluate the neural network architecture that we propose in Section 5.1 in comparison to more standard convolutional networks. To isolate the architectures from other confounding factors, we measure their accuracy on the pose estimation pretraining task described in Section 5.2. This is a reasonable proxy for evaluating how well the network can overcome two major challenges in visuomotor learning: the ability to handle relatively small datasets without overfitting, and the capability to learn tasks that are inherently spatial. We compare to a network where the expectation operator after the softmax is replaced with a learned fully connected layer, as is standard in the literature, a network where both the softmax and the expectation operators are replaced with a fully connected layer, and a version of this network that also uses 3×3 max pooling with stride 2 at the first two layers. These alternative architectures have many more parameters, since the fully connected layer takes the entire bank of response maps from the third convolutional layer as input. Pooling helps to reduce the number of parameters, but not to the same degree as the spatial softmax and expectation operators in our architecture.

The results in Table 3 indicate that using the softmax and expectation operators improves pose estimation accuracy substantially. Our network is able to outperform the more standard architectures because it is forced by the softmax and expectation operators to learn feature points, which provide a concise representation suitable for spatial inference. Since most of the parameters in this architecture are in the convolutional layers, which benefit from extensive weight sharing, overfitting is also greatly reduced. By removing pooling, our network also maintains higher resolution in the convolutional layers, improving spatial accuracy. Although we did attempt to regularize the larger standard architectures with higher weight decay and dropout, we did not observe a significant improvement on this dataset. We also did not extensively optimize the parameters of this network, such as filter size and number of channels, and investigating these design decisions further would be valuable to investigate in future work.

network architecture	test error (cm)
softmax + feature points (**ours**)	**1.30 ± 0.73**
softmax + fully connected layer	2.59 ± 1.19
fully connected layer	4.75 ± 2.29
max-pooling + fully connected	3.71 ± 1.73

Table 3: Average pose estimation accuracy and standard deviation with various architectures, measured as average Euclidean error for the three target points in 3D, with ground truth determined by forward kinematics from the left arm.



(a) hanger (b) cube (c) hammer (d) bottle

Figure 8: Illustration of the tasks in our visuomotor policy experiments, showing the variation in the position of the target for the hanger, cube, and bottle tasks, as well as two of the three grasps for the hammer, which also included variation in position (not shown).

6.4 Deep Visuomotor Policy Evaluation

In this section, we present an evaluation of our full visuomotor policy training algorithm on a PR2 robot. The aim of this evaluation is to answer the following question: does training the perception and control systems in a visuomotor policy jointly end-to-end provide better performance than training each component separately?

Experimental tasks. We trained policies for hanging a coat hanger on a clothes rack, inserting a block into a shape sorting cube, fitting the claw of a toy hammer under a nail with various grasps, and screwing on a bottle cap. The cost function for these tasks encourages low distance between three points on the end-effector and corresponding target points, low torques, and, for the bottle task, spinning the wrist. The equations for these cost functions and the details of each task are presented in Appendix B.2. The tasks are illustrated in Figure 8. Each task involved variation of 10-20 cm in each direction in the position of the target object (the rack, shape sorting cube, nail, and bottle). In addition, the coat hanger and hammer tasks were trained with two and three grasps, respectively. The current angle of the grasp was not provided to the policy, but had to be inferred from observing the robot's gripper in the camera images. All tasks used the same policy architecture and model parameters.

Experimental conditions. We evaluated the visuomotor policies in three conditions: (1) the training target positions and grasps, (2) new target positions not seen during training and, for the hammer, new grasps (spatial test), and (3) training positions with visual distractors (visual test). A selection of these experiments is shown in the supplementary video.[3] For the visual test, the shape sorting cube was placed on a table rather than held in

3. The video can be viewed at `http://sites.google.com/site/visuomotorpolicy`

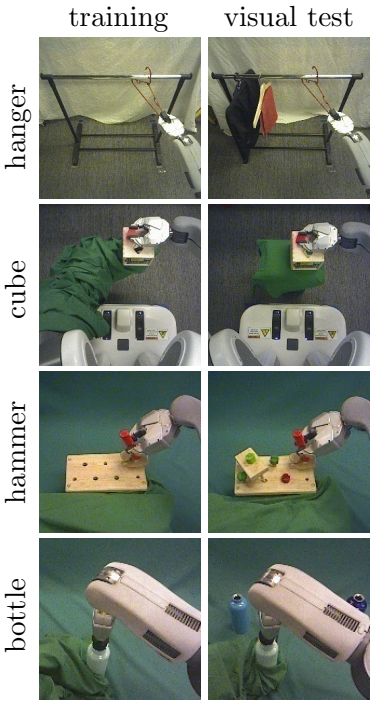
the gripper, the coat hanger was placed on a rack with clothes, and the bottle and hammer tasks were done in the presence of clutter. Illustrations of this test are shown in Figure 9.

Comparison. The success rates for each test are shown in Figure 9. We compared to two baselines, both of which train the vision layers in advance for pose prediction, instead of training the entire policy end-to-end. The features baseline discards the last layer of the pose predictor and uses the feature points, resulting in the same architecture as our policy, while the prediction baseline feeds the predicted pose into the control layers. The pose prediction baseline is analogous to a standard modular approach to policy learning, where the vision system is first trained to localize the target, and the policy is trained on top of it. This variant achieves poor performance. As discussed in Section 6.3, the pose estimate is accurate to about 1 cm. However, unlike the tasks in Section 6.2, where robust controllers could succeed even with inaccurate perception, many of these tasks have tolerances of just a few millimeters. In fact, the pose prediction baseline is only successful on the coat hanger, which requires comparatively little accuracy. Millimeter accuracy is difficult to achieve even with calibrated cameras and checkerboards. Indeed, prior work has reported that the PR2 can maintain a camera to end effector accuracy of about 2 cm during open loop motion (Meeussen et al., 2010). This suggests that the failure of this baseline is not atypical, and that our visuomotor policies are learning visual features and control strategies that improve the robot's accuracy. When provided with pose estimation features, the policy has more freedom in how it uses the visual information, and achieves somewhat higher success rates. However, full end-to-end training performs significantly better, achieving high accuracy even on the challenging bottle task, and successfully adapting to the variety of grasps on the hammer task. This suggests that, although the vision layer pretraining is clearly beneficial for reducing computation time, it is not sufficient by itself for discovering good features for visuomotor policies.

Visual distractors. The policies exhibit moderate tolerance to distractors that are visually separated from the target object. This is enabled in part by the spatial softmax, which has a lateral inhibition effect that suppresses non-maximal activations. Since distractors are unlikely to activate each feature as much as the true object, their activations are therefore suppressed. However, as expected, the learned policies tend to perform poorly under drastic changes to the backdrop, or when the distractors are adjacent to or occluding the manipulated objects, as shown in the supplementary video. A standard solution to this issue to expose the policy to a greater variety of visual situations during training. This issue could also be mitigated by artificially augmenting the image samples with synthetic transformations, as discussed in prior work in computer vision (Simard et al., 2003), or even incorporating ideas from transfer and semi-supervised learning.

6.5 Features Learned with End-to-End Training

The visual processing layers of our architecture automatically learn features points using the spatial softmax and expectation operators. These feature points encapsulate all of the visual information received by the motor layers of the policy. In Figure 10, we show the features points discovered by our visuomotor policy through guided policy search. Each policy learns features on the target object and the robot manipulator, both clearly relevant



coat hanger	training (18)	spatial test (24)	visual test (18)
end-to-end	**100%**	**100%**	**100%**
pose features	88.9%	87.5%	83.3%
pose prediction	55.6%	58.3%	66.7%
shape cube	training (27)	spatial test (36)	visual test (40)
end-to-end	**96.3%**	**91.7%**	**87.5%**
pose features	70.4%	83.3%	40%
pose prediction	0%	0%	n/a
toy hammer	training (45)	spatial test (60)	visual test (60)
end-to-end	**91.1%**	**86.7%**	**78.3%**
pose features	62.2%	75.0%	53.3%
pose prediction	8.9%	18.3%	n/a
bottle cap	training (27)	spatial test (12)	visual test (40)
end-to-end	**88.9%**	**83.3%**	**62.5%**
pose features	55.6%	58.3%	27.5%

Success rates on training positions, on novel test positions, and in the presence of visual distractors. The number of trials per test is shown in parentheses.

Figure 9: Training and visual test scenes as seen by the policy (left), and experimental results (right). The hammer and bottle images were cropped for visualization only.

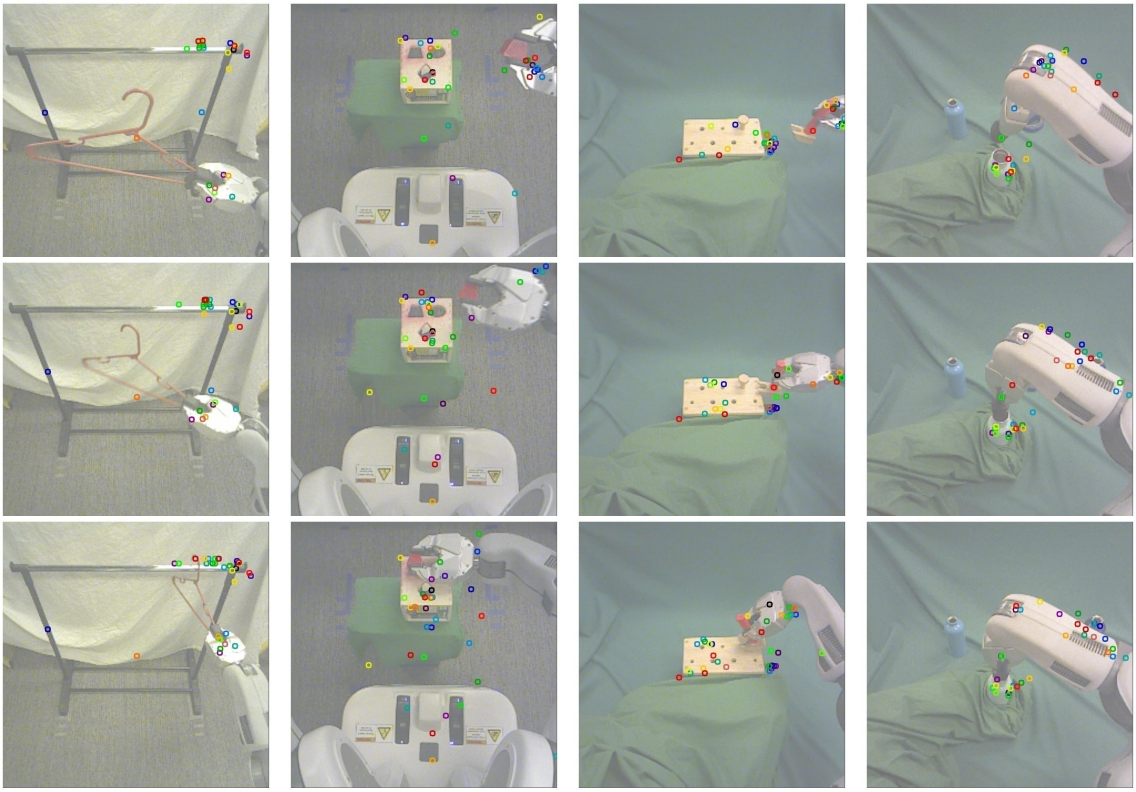
to task execution. The policy tends to pick out robust, distinctive features on the objects, such as the left pole of the clothes rack, the left corners of the shape-sorting cube and the bottom-left corner of the toy tool bench. In the bottle task, the end-to-end trained policy outputs points on both sides of the bottle, including one on the cap, while the pose prediction network only finds points on the right edge of the bottle.

In Figure 11, we compare the feature points learned through guided policy search to those learned by a CNN trained for pose prediction. After end-to-end training, the policy acquired a distinctly different set of feature points compared to the pose prediction CNN used for initialization. The end-to-end trained model finds more feature points on task-relevant objects and fewer points on background objects. This suggests that the policy improves its performance by acquiring *goal-driven* visual features that differ from those learned for object localization.

The feature point representation is very simple, since it assumes that the learned features are present at all times, and only one instance of each feature is ever present in the image. While this is a drastic simplification, both the pose predictor and the policy still achieve good results. A more flexible architecture that still learns a concise feature point representation could further improve policy performance. We hope to explore this in future work.

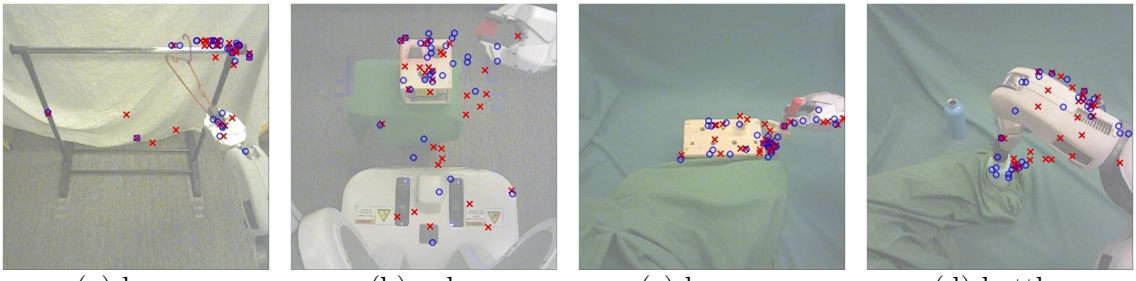
6.6 Computational Performance and Sample Efficiency

We used the Caffe deep learning library (Jia et al., 2014) for CNN training. Each visuomotor policy required a total of 3-4 hours of training time: 20-30 minutes for the pose prediction data collection on the robot, 40-60 minutes for the fully observed trajectory pretraining on



(a) hanger (b) cube (c) hammer (d) bottle

Figure 10: Feature points tracked by the policy during task execution for each of the four tasks. Each feature point is displayed in a different random color, with consistent coloring across images. The policy finds features on the target object and the robot gripper and arm. In the bottle cap task, note that the policy correctly ignores the distractor bottle in the background, even though it was not present during training.



(a) hanger (b) cube (c) hammer (d) bottle

Figure 11: Feature points learned for each task. For each input image, the feature points produced by the policy are shown in blue, while the feature points of the pose prediction network are shown in red. The end-to-end trained policy tends to discover more feature points on the target object and the robot arm than the pose prediction network.

the robot and offline pose pretraining (which can be done in parallel), and between 1.5 and 2.5 hours for end-to-end training with guided policy search. The coat hanger task required two iterations of guided policy search, the shape sorting cube and the hammer required three, and the bottle task required four. Only about 15 minutes of the training time consisted of executing trials on the robot. Since training was dominated by computation, we expect significant speedup from a more efficient implementation. The number of samples for training each policy is shown in Table 4. Each trial was five seconds in length, and the numbers do not include the time needed to collect about 1000 images for pretraining the visual processing layers of the policy.

	number of trials		
task	trajectory pretraining	end-to-end training	total
coat hanger	120	36	156
shape cube	90	81	171
toy hammer	150	90	240
bottle cap	180	108	288

Table 4: Total number of trials used for learning each visuomotor policy.

7. Discussion and Future Work

In this paper, we presented a method for learning robotic control policies that use raw input from a monocular camera. These policies are represented by a novel convolutional neural network architecture, and can be trained end-to-end using our guided policy search algorithm, which decomposes the policy search problem in a trajectory optimization phase that uses full state information and a supervised learning phase that only uses the observations. This decomposition allows us to leverage state-of-the-art tools from supervised learning, making it straightforward to optimize extremely high-dimensional policies. Our experimental results show that our method can execute complex manipulation skills, and that end-to-end training produces significant improvements in policy performance compared to using fixed vision layers trained for pose prediction.

Although we demonstrate moderate generalization over variations in the scene, our current method does not generalize to dramatically different settings, especially when visual distractors occlude the manipulated object or break up its silhouette in ways that differ from the training. The success of CNNs on exceedingly challenging vision tasks suggests that this class of models is capable of learning invariance to irrelevant distractor features (LeCun et al., 2015), and in principle this issue can be addressed by training the policy in a variety of environments, though this poses certain logistical challenges. More practical alternatives that could be explored in future work include simultaneously training the policy on multiple robots, each of which is located in a different environment, developing more sophisticated regularization and pretraining techniques to avoid overfitting, and introducing artificial data augmentation to encourage the policy to be invariant to irrelevant clutter. However, even without these improvements, our method has numerous applications in, for example, an industrial setting where the robot must repeatedly and efficiently perform a task that requires visual feedback under moderate variation in background and clutter conditions.

Our method takes advantage of a known, fully observed state space during training. This is both a weakness and a strength. It allows us to train linear-Gaussian controllers

for guided policy search using a very small number of samples, far more efficiently than standard policy search methods. However, the requirement to observe the full state during training limits the tasks to which the method can be applied. In many cases, this limitation is minor, and the only "instrumentation" required at training is to position the objects in the scene at consistent positions. However, tasks that require, for example, manipulating freely moving objects require more extensive instrumentation, such as motion capture. A promising direction for addressing this limitation is to combine our method with unsupervised state-space learning, as proposed in several recent works, including our own (Lange et al., 2012; Watter et al., 2015; Finn et al., 2015).

In future work, we hope to explore more complex policy architectures, such as recurrent policies that can deal with extensive occlusions by keeping a memory of past observations. We also hope to extend our method to a wider range of tasks that can benefit from visual input, as well as a variety of other rich sensory modalities, including haptic input from pressure sensors and auditory input. With a wider range of sensory modalities, end-to-end training of sensorimotor policies will become increasingly important: while it is often straightforward to imagine how vision might help to localize the position of an object in the scene, it is much less apparent how sound can be integrated into robotic control. A learned sensorimotor policy would be able to naturally integrate a wide range of modalities and utilize them to directly aid in control.

Acknowledgements

This research was funded in part by DARPA through a Young Faculty Award, the Army Research Office through the MAST program, NSF awards IIS-1427425 and IIS-1212798, the Berkeley Vision and Learning Center, and a Berkeley EECS Department Fellowship.

Appendix A. Guided Policy Search Algorithm Details

In this appendix, we describe a number of implementation details of our BADMM-based guided policy search algorithm and our linear-Gaussian controller optimization method.

A.1 BADMM Dual Variables and Weight Adjustment

Recall that the inner loop alternating optimization is given by

$$\theta \leftarrow \arg\min_{\theta} \sum_{t=1}^{T} E_{p(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)}[\mathbf{u}_t^{\mathrm{T}}\lambda_{\mu t}] + \nu_t \phi_t^\theta(\theta, p)$$

$$p \leftarrow \arg\min_{p} \sum_{t=1}^{T} E_{p(\mathbf{x}_t,\mathbf{u}_t)}[\ell(\mathbf{x}_t, \mathbf{u}_t) - \mathbf{u}_t^{\mathrm{T}}\lambda_{\mu t}] + \nu_t \phi_t^p(p, \theta)$$

$$\lambda_{\mu t} \leftarrow \lambda_{\mu t} + \alpha\nu_t(E_{\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_t)}[\mathbf{u}_t] - E_{p(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_t)}[\mathbf{u}_t]).$$

We use a step size of $\alpha = 0.1$ in all of our experiments, which we found to be more stable than $\alpha = 1.0$. The weights ν_t are initialized to 0.01 and incremented based on the following schedule: at every iteration, we compute the average KL-divergence between $p(\mathbf{u}_t|\mathbf{x}_t)$ and $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ at each time step, as well as its standard deviation over time steps.

The weights ν_t corresponding to time steps where the KL-divergence is higher than the average are increased by a factor of 2, and the weights corresponding to time steps where the KL-divergence is two standard deviations or more below the average are decreased by a factor of 2. The rationale behind this schedule is to adjust the KL-divergence penalty to keep the policy and trajectory in agreement by roughly the same amount at all time steps. Increasing ν_t too quickly can lead to the policy and trajectory becoming "locked" together, which makes it difficult for the trajectory to decrease its cost, while leaving it too low requires more iterations for convergence. We found this schedule to work well across all tasks, both during trajectory pretraining and while training the visuomotor policy.

To update the dual variables $\lambda_{\mu t}$, we evaluate the expectations over $p(\mathbf{x}_t)$ by using the latest batch of sampled trajectories. For each state $\{\mathbf{x}_t^i\}$ along these sampled trajectories, we evaluate the expectations over $\mathbf{u}_t$ under $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ and $p(\mathbf{u}_t|\mathbf{x}_t)$, which correspond simply to the means of these conditional Gaussian distributions, in closed form.

A.2 Policy Variance Optimization

As discussed in Section 4, the variance of the Gaussian policy $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ does not depend on the observation, though this dependence would be straightforward to add. Analyzing the objective $\mathcal{L}_\theta(\theta, p)$, we can write out only the terms that depend on Σ^π:

$$\mathcal{L}_\theta(\theta, p) = \frac{1}{2N} \sum_{i=1}^{N} \sum_{t=1}^{T} E_{p_i(\mathbf{x}_t, \mathbf{o}_t)} \left[\mathrm{tr}[\mathbf{C}_{ti}^{-1} \Sigma^\pi] - \log |\Sigma^\pi| \right].$$

Differentiating and setting the derivative to zero, we obtain the following equation for Σ^π:

$$\Sigma^\pi = \left[\frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \mathbf{C}_{ti}^{-1} \right]^{-1},$$

where the expectation under $p_i(\mathbf{x}_t)$ is omitted, since $\mathbf{C}_{ti}$ does not depend on $\mathbf{x}_t$.

A.3 Dynamics Fitting

Optimizing the linear-Gaussian controllers $p_i(\mathbf{u}_t|\mathbf{x}_t)$ that induce the trajectory distributions $p_i(\tau)$ requires fitting the system dynamics $p_i(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ at each iteration to samples generated on the physical system from the previous controller $\hat{p}_i(\mathbf{u}_t|\mathbf{x}_t)$. In this section, we describe how these dynamics are fitted. As in Section 4, we drop the subscript i, since the dynamics are fitted the same way for all of the trajectory distributions.

The linear-Gaussian dynamics are defined as $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}t}\mathbf{x}_t + f_{\mathbf{u}t}\mathbf{u}_t + f_{ct}, \mathbf{F}_t)$, and the data that we obtain from the robot can be viewed as tuples $\{\mathbf{x}_t^i, \mathbf{u}_t^i, \mathbf{x}_{t+1}^i\}$. A simple way to fit these linear-Gaussian dynamics is to use linear regression to determine $f_\mathbf{x}$, $f_\mathbf{u}$, and f_c, and fit $\mathbf{F}_t$ based on the errors. However, the sample complexity of linear regression scales with the dimensionality of $\mathbf{x}_t$. For a high-dimensional robotic system, we might need an impractically large number of samples at each iteration to obtain a good fit. However, we can observe that the dynamics at nearby time steps are strongly correlated, and we can dramatically reduce the sample complexity of the dynamics fitting by bringing in information from other time steps, and even prior iterations. We will bring in this

information by fitting a global model to all of the transitions $\{\mathbf{x}_t^i, \mathbf{u}_t^i, \mathbf{x}_{t+1}^i\}$ for all t and all tuples from several prior iterations (we use three prior iterations in our implementation), and then use this model as a prior for fitting the dynamics at each time step. Note that this global model does not itself need to be a good forward dynamics model – it just needs to serve as a good prior to reduce the sample complexity of linear regression.

To make it more convenient to incorporate a data-driven prior, we will first reformulate this linear regression fit and view it as fitting a Gaussian model to the dataset $\{\mathbf{x}_t^i, \mathbf{u}_t^i, \mathbf{x}_{t+1}^i\}$ at each time step t, and then conditioning this Gaussian to obtain $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$. While this is equivalent to linear regression, it allows us to easily incorporate a normal-inverse-Wishart prior on this Gaussian in order to bring in prior information. Let $\hat{\Sigma}$ be the empirical covariance of our dataset, and let $\hat{\mu}$ be the empirical mean. The normal-inverse-Wishart prior is defined by prior parameters $\mathbf{\Phi}$, μ_0, m, and n_0. Under this prior, the maximum a posteriori estimates for the covariance Σ and mean μ are given by

$$\Sigma = \frac{\mathbf{\Phi} + N\hat{\Sigma} + \frac{Nm}{N+m}(\hat{\mu} - \mu_0)(\hat{\mu} - \mu_0)^{\mathrm{T}}}{N + n_0} \qquad \mu = \frac{m\mu_0 + n_0\hat{\mu}}{m + n_0}.$$

Having obtained Σ and μ, we can obtain an estimate of the dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ by conditioning the distribution $\mathcal{N}(\mu, \Sigma)$ on $[\mathbf{x}_t; \mathbf{u}_t]$, which produces linear-Gaussian dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}t}\mathbf{x}_t + f_{\mathbf{u}t}\mathbf{u}_t + f_{ct}, \mathbf{F}_t)$. The parameters of the normal-inverse-Wishart prior are obtained from the global model of the dynamics which, as described previously, is fitted to all available tuples $\{\mathbf{x}_t^i, \mathbf{u}_t^i, \mathbf{x}_{t+1}^i\}$.

The simplest prior can be obtained by fitting a Gaussian distribution to vectors $[\mathbf{x}; \mathbf{u}; \mathbf{x}']$. If the mean and covariance of this data are given by $\bar{\mu}$ and $\bar{\Sigma}$, the prior is given by $\mathbf{\Phi} = n_0\bar{\Sigma}$ and $\mu_0 = \bar{\mu}$, while n_0 and m should be set to the number of data points in the datasets. In practice, settings n_0 and m to 1 tends to produce better results, since the prior is fitted to many more samples than are available for linear regression at each time step. While this prior is simple, we can obtain a better prior by employing a nonlinear model.

The particular global model we use in this work is a Gaussian mixture model over vectors $[\mathbf{x}; \mathbf{u}; \mathbf{x}']$. Systems of articulated rigid bodies undergoing contact dynamics, such as robots interacting with their environment, can be coarsely modeled as having piecewise linear dynamics. The Gaussian mixture model provides a good approximation for such piecewise linear systems, with each mixture element corresponding to a different linear mode (Khansari-Zadeh and Billard, 2010). Under this model, the state transition tuple is assumed to come from a distribution that depends on some hidden state h, which corresponds to the mixture element identity. In practice, this hidden state might correspond to the type of contact profile experienced by a robotic arm at step i. The prior for the dynamics fit at time step t is then obtained by inferring the hidden state distribution for the transition dataset $\{\mathbf{x}_t^i, \mathbf{u}_t^i, \mathbf{x}_{t+1}^i\}$, and using the mean and covariance of the corresponding mixture elements (weighted by their probabilities) to obtain $\bar{\mu}$ and $\bar{\Sigma}$. The prior parameters can then be obtained as described above.

In our experiments, we set the number of mixture elements for the Gaussian mixture model prior such that there were at least 40 samples per mixture element, or 20 total mixture elements, whichever was lower. In general, we did not find the performance of the method to be sensitive to this parameter, though overfitting did tend to occur in the early iterations when the number of samples is low, if the number of mixtures was too high.

A.4 Trajectory Optimization

In this section, we show how the LQR backward pass can be used to optimize the constrained objective in Section 4.2. The constrained trajectory optimization problem is given by

$$\min_{p(\tau)\in\mathcal{N}(\tau)} \mathcal{L}_p(p,\theta) \text{ s.t. } D_{\mathrm{KL}}(p(\tau)\|\hat{p}(\tau)) \leq \epsilon.$$

The augmented Lagrangian $\mathcal{L}_p(p,\theta)$ consists of an entropy term and an expectation under $p(\tau)$ of a quantity that is independent of p. We can locally approximate this quantity with a quadratic by using a quadratic expansion of $\ell(\mathbf{x}_t,\mathbf{u}_t)$, and fitting a linear Gaussian to $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ with the same method we used for the dynamics. We can then solve the primal optimization in the dual gradient descent procedure with a standard LQR backward pass. As discussed in Section 4, $\mathcal{L}_p(p,\theta)$ can be written as the expectation of some function $c(\tau)$ that is independent of p, such that $\mathcal{L}_p(p,\theta) = E_{p(\tau)}[c(\tau)] - \nu_t\mathcal{H}(p(\tau))$. Specifically,

$$c(\mathbf{x}_t,\mathbf{u}_t) = \ell(\mathbf{x}_t,\mathbf{u}_t) - \mathbf{u}_t^{\mathrm{T}}\lambda_{\mu t} - \nu_t \log\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$$

Writing the Lagrangian of the constrained optimization, we have

$$\mathcal{L}(p) = E_{p(\tau)}[c(\tau) - \eta\log\hat{p}(\tau)] - (\eta+\nu_t)\mathcal{H}(p(\tau)) - \eta\epsilon,$$

where η is the Lagrange multiplier. Note that $\mathcal{L}(p)$ is the Lagrangian of the constrained trajectory optimization, which is not related to the augmented Lagrangian $\mathcal{L}_p(\tau,\theta)$. Grouping the terms in the expectation and omitting constants, we can rewrite the minimization of the Lagrangian with respect to the primal variables as

$$\min_{p(\tau)\in\mathcal{N}(\tau)} E_{p(\tau)}\left[\frac{1}{\eta+\nu_t}c(\tau) - \frac{\eta}{\eta+\nu_t}\log\hat{p}(\tau)\right] - \mathcal{H}(p(\tau)). \tag{4}$$

Let $\tilde{c}(\tau) = \frac{1}{\eta+\nu_t}c(\tau) - \frac{\eta}{\eta+\nu_t}\log\hat{p}(\tau)$. The above optimization corresponds to minimizing $E_{p(\tau)}[\tilde{c}(\tau)] - \mathcal{H}(p(\tau))$. This type of maximum entropy problem can be solved using the LQR algorithm, and the solution is given by

$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t; Q^{-1}_{\mathbf{u},\mathbf{u}t}),$$

where $\mathbf{K}_t$ and $\mathbf{k}_t$ are the feedback and open loop terms of the optimal linear feedback controller corresponding to the cost $\tilde{c}(\mathbf{x}_t,\mathbf{u}_t)$ and the dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t)$, and $Q_{\mathbf{u},\mathbf{u}t}$ is the quadratic term in the Q-function at time step t. All of these terms can be obtained from a standard LQR backward pass (Li and Todorov, 2004), which we summarize below.

Recall that the estimated linear-Gaussian dynamics have the form $p(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}t}\mathbf{x}_t + f_{\mathbf{u}t}\mathbf{u}_t + f_{ct}, \mathbf{F}_t)$. The quadratic cost approximation has the form

$$\tilde{c}(\mathbf{x}_t,\mathbf{u}_t) \approx \frac{1}{2}[\mathbf{x}_t;\mathbf{u}_t]^{\mathrm{T}}\tilde{c}_{\mathbf{xu},\mathbf{xu}t}[\mathbf{x}_t;\mathbf{u}_t] + [\mathbf{x}_t;\mathbf{u}_t]^{\mathrm{T}}\tilde{c}_{\mathbf{xu}t} + \text{const},$$

where subscripts denote derivatives, e.g. $\tilde{c}_{\mathbf{xu}t}$ is the gradient of $\tilde{c}$ with respect to $[\mathbf{x}_t;\mathbf{u}_t]$, while $\tilde{c}_{\mathbf{xu},\mathbf{xu}t}$ is the Hessian.[4] Under this model of the dynamics and cost function, the

4. We assume that all Taylor expansions here are recentered around zero. Otherwise, the point around which the derivatives are computed must be subtracted from $\mathbf{x}_t$ and $\mathbf{u}_t$ in all of these equations.

optimal controller can be computed by recursively computing the quadratic Q-function and value function, starting with the last time step. These functions are given by

$$V(\mathbf{x}_t) = \frac{1}{2}\mathbf{x}_t^{\mathrm{T}} V_{\mathbf{x},\mathbf{x}t}\mathbf{x}_t + \mathbf{x}_t^{\mathrm{T}} V_{\mathbf{x}t} + \text{const}$$
$$Q(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2}[\mathbf{x}_t; \mathbf{u}_t]^{\mathrm{T}} Q_{\mathbf{xu},\mathbf{xu}t}[\mathbf{x}_t; \mathbf{u}_t] + [\mathbf{x}_t; \mathbf{u}_t]^{\mathrm{T}} Q_{\mathbf{xu}t} + \text{const}$$

We can express them with the following recurrence, which is computed starting at the last time step $t = T$ and moving backward through time:

$$\begin{aligned} Q_{\mathbf{xu},\mathbf{xu}t} &= \tilde{c}_{\mathbf{xu},\mathbf{xu}t} + f_{\mathbf{xu}t}^{\mathrm{T}} V_{\mathbf{x},\mathbf{x}t+1} f_{\mathbf{xu}t} \\ Q_{\mathbf{xu}t} &= \tilde{c}_{\mathbf{xu}t} + f_{\mathbf{xu}t}^{\mathrm{T}} V_{\mathbf{x}t+1} + f_{\mathbf{xu}t}^{\mathrm{T}} V_{\mathbf{x},\mathbf{x}t+1} f_{ct} \\ V_{\mathbf{x},\mathbf{x}t} &= Q_{\mathbf{x},\mathbf{x}t} - Q_{\mathbf{u},\mathbf{x}t}^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},\mathbf{x}t} \\ V_{\mathbf{x}t} &= Q_{\mathbf{x}t} - Q_{\mathbf{u},\mathbf{x}t}^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t}, \end{aligned}$$

and the optimal control law is then given by $g(\mathbf{x}_t) = \mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t$, where $\mathbf{K}_t = -Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},\mathbf{x}t}$ and $\mathbf{k}_t = -Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t}$. If, instead of simply minimizing the expected cost, we instead wish to optimize the maximum entropy objective in Equation (4), the optimal controller is instead linear-Gaussian, with the solution given by $p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t; Q_{\mathbf{u},\mathbf{u}t}^{-1})$, as shown in prior work (Levine and Koltun, 2013a).

Appendix B. Experimental Setup Details

In this appendix, we present a detailed summary of the experimental setup for our simulated and real-world experiments.

B.1 Simulated Experiment Details

All of the simulated experiments used the MuJoCo simulation package (Todorov et al., 2012), with simulated frictional contacts and torque motors at the joints used for actuation. Although no control or state noise was added during simulation, noise was injected naturally by the linear-Gaussian controllers. The linear-Gaussian controllers $p_i(\mathbf{u}_t|\mathbf{x}_t)$ were initialized to stay near the initial state $\mathbf{x}_1$ using linear feedback based on a proportional-derivative control law for all tasks, except for the octopus arm, where $p_i(\mathbf{u}_t|\mathbf{x}_t)$ was initialized to be zero mean with a fixed spherical covariance, and the walker, which was initialized to track a demonstration trajectory with proportional-derivative feedback. The walker was the only task that used a demonstration, as described previously. We describe the details of each system below.

Peg insertion: The 2D peg insertion task has 6 state dimensions (joint angles and angular velocities) and 2 action dimensions. The 3D version of the task has 12 state dimensions, since the arm has 3 degrees of freedom at the shoulder, 1 at the elbow, and 2 at the wrist. Trials were 8 seconds in length and simulated at 100 Hz, resulting in 800 time steps per rollout. The cost function is given by

$$\ell(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} w_{\mathbf{u}} \|\mathbf{u}_t\|^2 + w_{\mathbf{p}} \ell_{12}(\mathbf{p}_{\mathbf{x}_t} - \mathbf{p}^\star),$$

where $\mathbf{p}_{\mathbf{x}_t}$ is the position of the end effector for state $\mathbf{x}_t$, $\mathbf{p}^\star$ is the desired end effector position at the bottom of the slot, and the norm $\ell_{12}(z)$ is given by $\frac{1}{2}\|z\|^2 + \sqrt{\alpha + z^2}$, which corresponds to the sum of an ℓ_2 and soft ℓ_1 norm. We use this norm to encourage the peg to precisely reach the target position at the bottom of the hole, but to also receive a larger penalty when far away. The task also works well in 2D with a simple ℓ_2 penalty, though we found that the 3D version of the task takes longer to insert the peg all the way into the hole without the ℓ_1-like square root term. The weights were set to $w_{\mathbf{u}} = 10^{-6}$ and $w_{\mathbf{p}} = 1$. Initial states were chosen by moving the shoulder of the arm relative to the hole, with four equally spaced starting states in a 20 cm region for the 2D arm, and four random starting states in a 10 cm radius for the 3D arm.

Octopus arm: The octopus arm consists of six four-sided chambers. Each edge of each chamber is a simulated muscle, and actions correspond to contracting or relaxing the muscle. The state space consists of the positions and velocities of the chamber vertices. The midpoint of one edge of the first chamber is fixed, resulting in a total of 25 degrees of freedom: the 2D positions of the 12 unconstrained points, and the orientation of the first edge. Including velocities, the total dimensionality of the state space is 50. The cost function depends on the activation of the muscles and distance between the tip of the arm and the target point, in the same way as for peg insertion. The weights are set to $w_{\mathbf{u}} = 10^{-3}$ and $w_{\mathbf{p}} = 1$.

Swimmer: The swimmer consists of 3 links and 5 degrees of freedom, including the global position and orientation which, together with the velocities, produces a 10 dimensional state space. The swimmer has 2 action dimensions corresponding to the torques between joints. The simulation applied drag on each link of the swimmer to roughly simulate a fluid, allowing it to propel itself. The rollouts were 20 seconds in length at 20 Hz, resulting in 400 time steps per rollout. The cost function for the swimmer is given by

$$\ell(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} w_{\mathbf{u}} \|\mathbf{u}_t\|^t + \frac{1}{2} w_v \|v_{x\mathbf{x}_t} - v_x^\star\|^2$$

where $v_{x\mathbf{x}_t}$ is the horizontal velocity, $v_x^\star = 2.0\text{m/s}$, and the weights were $w_{\mathbf{u}} = 2 \cdot 10^{-5}$ and $w_v = 1$.

Walker: The bipedal walker consists of a torso and two legs, each with three links, for a total of 9 degrees of freedom and 18 dimensions, with velocity, and 6 action dimensions. The simulation ran for 5 seconds at 100 Hz, for a total of 500 time steps. The cost function is given by

$$\ell(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} w_{\mathbf{u}} \|\mathbf{u}_t\|^t + \frac{1}{2} w_v \|v_{x\mathbf{x}_t} - v_x^\star\|^2 + \frac{1}{2} w_h \|p_{y_{\mathbf{x}_t}} - p_y^\star\|^2$$

where $v_{x\mathbf{x}_t}$ is again the horizontal velocity, $p_{y_{\mathbf{x}_t}}$ is the vertical position of the root, $v_x^\star = 2.1\text{m/s}$, $p_y^\star = 1.1\text{m}$, and the weights were set to $w_{\mathbf{u}} = 10^{-4}$, $w_v = 1$, and $w_h = 1$.

B.2 Robotic Experiment Details

All of the robotic experiments were conducted on a PR2 robot. The robot was controlled at 20 Hz via direct effort control,[5] and camera images were recorded using the RGB camera on a PrimeSense Carmine sensor. The images were downsampled to $240 \times 240 \times 3$. The learned policies controlled one 7 DoF arm of the robot, while the other arm was used to move objects in the scene to automatically vary the initial conditions. The camera was kept fixed in each experiment. Each episode was 5 seconds in length. For each task, the cost function required placing the object held in the gripper at a particular location (which might require, for example, to insert a shape into a shape sorting cube). The cost was given by the following equation:

$$\ell(\mathbf{x}_t, \mathbf{u}_t) = w_{\ell_2} d_t^2 + w_{\log} \log(d_t^2 + \alpha) + w_{\mathbf{u}} \|\mathbf{u}_t\|^2,$$

where d_t is the distance between three points in the space of the end-effector and their target positions,[6], and the weights are set to $w_{\ell_2} = 10^{-3}$, $w_{\log} = 1.0$, and $w_{\mathbf{u}} = 10^{-2}$. The quadratic term encourages moving the end-effector toward the target when it is far, while the logarithm term encourages placing it precisely at the target location, as discussed in prior work (Levine et al., 2015). The bottle cap task used an additional cost term consisting of a quadratic penalty on the difference between the wrist angular velocity and a target velocity.

For all of the tasks, we initialized all of the linear-Gaussian controllers $p_i(\mathbf{u}_t|\mathbf{x}_t)$ to stay near the initial state $\mathbf{x}_1$, with a diagonal noise covariance. The covariance of the noise was chosen to be proportional to a diagonal approximation of the inverse effective mass at each joint, as provided by the manufacturer of the PR2 robot, and the feedback controller was constructed using LQR, with an approximate linear model obtained from the same diagonal inverse mass matrix. The role of this initial controller was primarily to avoid dangerous actions during the first iteration. We discuss the particular setup for each experiment below:

Coat hanger: The coat hanger task required the robot to hang a coat hanger on a clothes rack. The coat hanger was grasped at one of two angles, about 35° apart, and the rack was positioned at three different distances from the robot during training, with differences of about 10 cm between each position. The rack was moved manually between these positions during training. A trial was considered successful if, when the coat hanger was released, it remained hanging on the rack rather than dropping to the ground.

Shape sorting cube: The shape sorting cube task required the robot to insert a red trapezoid into a trapezoidal hole on a shape sorting cube. During training, the cube was positioned at nine different positions, situated at the corners, edges, and middle of a rectangular region 16 cm × 10 cm in size. During training, the shape sorting cube was moved through the training positions by using the left arm. A trial was considered successful if the bottom face of the trapezoid was completely inside the shape sorting cube, such that if the robot were to release the trapezoid, it would fall inside the cube.

5. The PR2 robot does not provide for closed loop torque control, but instead supports an effort control interface that directly sets feedforward motor voltages. In practice, these voltages are roughly proportional to feedforward torques, but are also affected by friction and damping.
6. Three points fully define the pose of the end-effector. For the bottle cap task, which is radially symmetric, we use only two points.

Toy hammer: The hammer task required the robot to insert the claw of a toy hammer underneath a toy plastic nail, placing the claw around the base of the nail. The hammer was grasped at one of three angles, each 22.5° apart, for a total variation of 45° degrees, and the nail was positioned at five positions, at the corners and center of a rectangular region 10 cm × 7 cm in size. During training, the toy tool bench containing the nail was moved using the left arm. A trial was considered successful if the tip of the claw of the hammer was at least under the centerline of the nail.

Bottle cap: The bottle cap task required the robot to screw a cap onto a bottle at various positions. The bottle was located at nine different positions, situated at the corners, edges, and middle of a rectangular region 16 cm × 10 cm in size, and the left arm was used to move the bottle through the training positions. A trial was considered successful if, after completion, the cap could not be removed from bottle simply by pulling vertically.

References

J. A. Bagnell and J. Schneider. Covariant policy search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

B. Bakker, V. Zhumatiy, G. Gruener, and J. Schmidhuber. A robot that reinforcement-learns to identify and memorize important previous observations. In *International Conference on Intelligent Robots and Systems (IROS)*, 2003.

G. Bekey and K. Goldberg. *Neural Networks in Robotics*. Springer US, 1992.

H. Benbrahim and J. A. Franklin. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*, 22:283–302, 1997.

W. Böhmer, S. Grünewälder, Y. Shen, M. Musial, and K. Obermayer. Construction of approximation spaces for reinforcement learning. *Journal of Machine Learning Research*, 14(1):2067–2118, January 2013.

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1122, 2011.

D. Ciresan, U. Meier, J. Masci, L. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR)*, 2012.

M. Deisenroth and C. Rasmussen. PILCO: a model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, 2011.

M. Deisenroth, C. Rasmussen, and D. Fox. Learning to control a low-cost manipulator using data-efficient reinforcement learning. In *Robotics: Science and Systems (RSS)*, 2011.

M. Deisenroth, G. Neumann, and J. Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013.

J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition (CVPR)*, 2009.

G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng. Learning CPG-based biped locomotion with a policy gradient method: Application to a humanoid robot. *International Journal of Robotic Research*, 27(2):213–228, 2008.

I. Endres and D. Hoiem. Category independent object proposals. In *European Conference on Computer Vision (ECCV)*. 2010.

Y. Engel, P. Szabó, and D. Volkinshtein. Learning to control an octopus arm with Gaussian process temporal difference methods. In *Advances in Neural Information Processing Systems (NIPS)*, 2005.

B. Espiau, F. Chaumette, and P. Rives. A new approach to visual servoing in robotics. *IEEE Transactions on Robotics and Automation*, 8(3), 1992.

C. Finn, X. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel. Learning visual feature spaces for robotic manipulation with deep spatial autoencoders. *arXiv preprint arXiv:1509.06113*, 2015.

K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

T. Geng, B. Porr, and F. Wörgötter. Fast biped walking with a reflexive controller and realtime policy searching. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014a.

R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014b.

V. Gullapalli. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, 3(6):671–692, 1990.

V. Gullapalli. Skillful control under uncertainty via direct reinforcement learning. *Reinforcement Learning and Robotics*, 15(4):237–246, 1995.

X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

R. Hadsell, P. Sermanet, J. B. A. Erkan, and M. Scoffier. Learning long-range vision for autonomous off-road driving. *Journal of Field Robotics*, pages 120–144, 2009.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In *A Field Guide to Dynamic Recurrent Neural Networks*. IEEE Press, 2001.

K. J. Hunt, D. Sbarbaro, R. Żbikowski, and P. J. Gawthrop. Neural networks for control systems: A survey. *Automatica*, 28(6):1083–1112, November 1992.

D. Jacobson and D. Mayne. *Differential Dynamic Programming*. Elsevier, 1970.

M. Jägersand, O. Fuentes, and R. C. Nelson. Experimental evaluation of uncalibrated visual servoing for precision manipulation. In *International Conference on Robotics and Automation (ICRA)*, 1997.

Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

S. Jodogne and J. H. Piater. Closed-loop learning of visual control policies. *Journal of Artificial Intelligence Research*, 28:349–391, 2007.

R. Jonschkowski and O. Brock. State representation learning in robotics: Using prior knowledge about physical interaction. In *Proceedings of Robotics: Science and Systems*, 2014.

M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal. Learning force control policies for compliant manipulation. In *International Conference on Intelligent Robots and Systems (IROS)*, 2011.

S. M. Khansari-Zadeh and A. Billard. BM: An iterative algorithm to learn stable non-linear dynamical systems with Gaussian mixture models. In *International Conference on Robotics and Automation (ICRA)*, 2010.

J. Kober and J. Peters. Learning motor primitives for robotics. In *International Conference on Robotics and Automation (ICRA)*, 2009.

J. Kober, K. Muelling, O. Kroemer, C.H. Lampert, B. Schoelkopf, and J. Peters. Movement templates for learning of hitting and batting. In *International Conference on Robotics and Automation (ICRA)*, 2010a.

J. Kober, E. Oztop, and J. Peters. Reinforcement learning to adjust robot movements to new situations. In *Robotics: Science and Systems (RSS)*, 2010b.

J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotic Research*, 32(11):1238–1274, 2013.

N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *International Conference on Robotics and Automation (IROS)*, 2004.

J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Conference on Genetic and Evolutionary Computation*, GECCO '13, 2013.

A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*. 2012.

T. Lampe and M. Riedmiller. Acquiring visual servoing reaching and grasping skills using neural reinforcement learning. In *International Joint Conference on Neural Networks (IJCNN)*, 2013.

A. Lanfranco, A. Castellanos, J. Desai, and W. Meyers. Robotic surgery: a current perspective. *Annals of surgery*, 239(1):14, 2004.

S. Lange, M. Riedmiller, and A. Voigtlaender. Autonomous reinforcement learning on raw visual input data in a real world application. In *International Joint Conference on Neural Networks*, 2012.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems (NIPS)*, 1989.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, May 2015.

H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *International Conference on Machine Learning (ICML)*, 2009.

Ian Lenz, Ross Knepper, and Ashutosh Saxena. DeepMPC: Learning deep latent features for model predictive control. In *RSS*, 2015a.

Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep learning for detecting robotic grasps. *IJRR*, 2015b.

S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

S. Levine and V. Koltun. Guided policy search. In *International Conference on Machine Learning (ICML)*, 2013a.

S. Levine and V. Koltun. Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2013b.

S. Levine and V. Koltun. Learning complex neural network policies with trajectory optimization. In *International Conference on Machine Learning (ICML)*, 2014.

S. Levine, N. Wagener, and P. Abbeel. Learning contact-rich manipulation skills with guided policy search. In *International Conference on Robotics and Automation (ICRA)*, 2015.

F. L. Lewis, A. Yesildirak, and S. Jagannathan. *Neural Network Control of Robot Manipulators and Nonlinear Systems*. Taylor & Francis, Inc., 1998.

W. Li and E. Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229, 2004.

T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

R. Lioutikov, A. Paraschos, G. Neumann, and J. Peters. Sample-based information-theoretic stochastic optimal control. In *International Conference on Robotics and Automation*, 2014.

H. Mayer, F. Gomez, D. Wierstra, I. Nagy, A. Knoll, and J. Schmidhuber. A system for robotic heart surgery that learns to tie knots using recurrent neural networks. In *International Conference on Intelligent Robots and Systems (IROS)*, 2006.

W. Meeussen, M. Wise, S. Glaser, S. Chitta, C. McGann, P. Mihelich, E. Marder-Eppstein, M. Muja, Victor Eruhimov, T. Foote, J. Hsu, R.B. Rusu, B. Marthi, G. Bradski, K. Konolige, B. Gerkey, and E. Berger. Autonomous door opening and plugging in with a personal robot. In *International Conference on Robotics and Automation (ICRA)*, 2010.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with deep reinforcement learning. *NIPS '13 Workshop on Deep Learning*, 2013.

K. Mohta, V. Kumar, and K. Daniilidis. Vision based control of a quadrotor for perching on planes and lines. In *International Conference on Robotics and Automation (ICRA)*, 2014.

I. Mordatch and E. Todorov. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems (RSS)*, 2014.

A. Y. Ng, H. J. Kim, M. I. Jordan, and S. Sastry. Inverted autonomous helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*, 2004.

R. Pascanu and Y. Bengio. On the difficulty of training recurrent neural networks. Technical Report arXiv:1211.5063, Universite de Montreal, 2012.

B. Pepik, M. Stark, P. Gehler, and B. Schiele. Teaching 3D geometry to deformable part models. In *Computer Vision and Pattern Recognition (CVPR)*, 2012.

J. Peters and S. Schaal. Applying the episodic natural actor-critic architecture to motor primitive learning. In *European Symposium on Artificial Neural Networks (ESANN)*, 2007.

J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008.

J. Peters, K. Mülling, and Y. Altün. Relative entropy policy search. In *AAAI Conference on Artificial Intelligence*, 2010.

Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *CoRR*, abs/1509.06825, 2015.

D. Pomerleau. ALVINN: an autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems (NIPS)*, 1989.

S. Ross, G. Gordon, and A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *Journal of Machine Learning Research*, 15:627–635, 2011.

S. Ross, N. Melik-Barkhudarov, K. Shaurya Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert. Learning monocular reactive UAV control in cluttered natural environments. In *International Conference on Robotics and Automation (ICRA)*, 2013.

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

S. Savarese and L. Fei-Fei. 3D generic object categorization, localization and pose estimation. In *International Conference on Computer Vision (ICCV)*, 2007.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61: 85–117, 2015.

P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition*, 2003.

F. Stulp and O. Sigaud. Path integral policy improvement with covariance matrix adaptation. In *International Conference on Machine Learning (ICML)*, 2012.

Jaeyong Sung, Seok Hyun Jin, and Ashutosh Saxena. Robobarista: Object part based transfer of manipulation trajectories from crowd-sourcing in 3d pointclouds. *CoRR*, abs/1504.03071, 2015.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

R. Tedrake, T. Zhang, and H. Seung. Stochastic policy gradient reinforcement learning on a simple 3d biped. In *International Conference on Intelligent Robots and Systems (IROS)*, 2004.

E. Theodorou, J. Buchli, and S. Schaal. Reinforcement learning of motor skills in high dimensions. In *International Conference on Robotics and Automation (ICRA)*, 2010.

E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 2013.

H. van Hoof, J. Peters, and G. Neumann. Learning of non-parametric control policies with high-dimensional state features. In *International Conference on Artificial Intelligence and Statistics*, 2015.

H. Wang and A. Banerjee. Bregman alternating direction method of multipliers. In *Advances in Neural Information Processing Systems (NIPS)*. 2014.

M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advanced in Neural Information Processing Systems (NIPS)*, 2015.

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

W. J. Wilson, C. W. Williams Hulls, and G. S. Bell. Relative end-effector control using cartesian position based visual servoing. *IEEE Transactions on Robotics and Automation*, 12(5), 1996.

K.A. Wyrobek, E.H. Berger, HF M. Van der Loos, and K. Salisbury. Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot. In *International Conference on Robotics and Automation (ICRA)*, 2008.

B. H. Yoshimi and P. K. Allen. Active, uncalibrated visual servoing. In *International Conference on Robotics and Automation (ICRA)*, 1994.