2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Madrid, Spain, October 1-5, 2018

Deep Reinforcement Learning for Robotic Assembly of Mixed Deformable and Rigid Objects

Jianlan Luo[1], Eugen Solowjow[2], Chengtao Wen[2], Juan Aparicio Ojea[2] and Alice M. Agogino[1]

Abstract— **Reinforcement learning for assembly tasks can yield powerful robot control algorithms for applications that are challenging or even impossible for "conventional" feedback control methods. Insertion of a rigid peg into a deformable hole of smaller diameter is such a task. In this contribution we solve this task with Deep Reinforcement Learning. Force-torque measurements from a robot arm wrist sensor are thereby incorporated two-fold; they are integrated into the policy learning process and they are exploited in an admittance controller that is coupled to the neural network. This enables robot learning of contact-rich assembly tasks without explicit joint torque control or passive mechanical compliance. We demonstrate our approach in experiments with an industrial robot.**

I. INTRODUCTION

Today, industrial robots deployed across various industries are mostly doing repetitive tasks. The overall task performance hinges on the accuracy of their controllers to track pre-defined trajectories. Additionally, the ability of robots to handle unconstructed complex environments is limited in today's manufacturing environments. To this end, endowing these machines with a greater level of intelligence to autonomously acquire skills is desirable. The main challenge is to design adaptable, yet robust, control algorithms in the face of inherent difficulties in modeling all possible system behaviors and the necessity of behavior generalization. Reinforcement learning (RL) methods hold promises for solving such challenges, because they promise agents to learn behaviors through interaction with their surrounding environments and ideally generalize to new unseen scenarios [1, 2, 3]. We tackle a specific task representing the aforementioned challenges in this paper: A peg-in-hole insertion. The peg is rigid and the hole is in a nonlinear deformable work piece. Moreover, the diameter of the hole is smaller than the peg diameter. Such a peg-in-hole setup is very challenging for "conventional" feedback control methods since the contact interaction model is in general unavailable. In manufacturing, it would require carefully hand-engineered heuristics and manual fine tuning, e.g., circling around the hole with the peg and perform insertion, when the force measurement decreases [4]. Furthermore most industrial robots are non-compliant, this poses additional challenges to tasks involving rich contact. However, a force/torque sensor mounted on the robot wrist is often available. These sensors can provide haptic feedback related to contacts of the robot and its environment. For example: a human can do peg-in-hole insertion by feeling the surface and probing the hole with closed eyes. In this work, we leverage such haptic feedback to make purely position and velocity-controlled robots compliant as well as integrating them into a reinforcement learning framework.

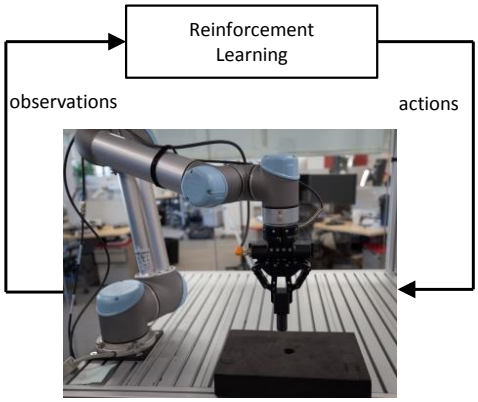


Fig. 1: Reinforcement learning for robotic peg-in-hole insertion.

A. Prior Work

Recent advances in reinforcement learning have gained great success in solving a variety of problems from playing video games [5, 6] to robotic locomotion [7, 8, 9, 10], and manipulation [11, 12, 13, 14, 15]. The core idea of RL is to provide the robot with a high-level specification of what to do instead of how to do it. Reinforcement learning can be distinguished in model-based methods and model-free methods [1, 3]. In this contribution, we focus on model-based policy search, which incorporates a model of the world dynamics that is learned from data [2]. While model-based policy search is computationally more expensive than model-free methods, it requires less data to solve a task. Recent progress in the area of Deep Neural Networks suggests deploying them for parametrizing policies and other functions in RL methods [16, 17, 8]. This is often referred to as Deep Reinforcement Learning (DRL). The idea of combining RL with neural nets to solve assembly tasks has been explored prior to the current serge of DRL. More than 25 years ago, in [18] the combination of RL with neural networks for the peg-in-hole task was studied. However, these approaches required a long training time and were restricted to low dimensional systems.

[1]Authors are with the Department of Mechanical Engineering, University of California, Berkeley, CA, 94720, USA. {jianlanluo, agogino@berkeley.edu}

[2]Authors are with Siemens Corporate Technology, Berkeley, CA, 94704, USA {eugen.solowjow, chengtao.wen, juan.aparicio}@siemens.com

A recently developed model-based reinforcement learning algorithm called guided policy search (GPS) provided new insights into solving the paradoxical complexity-efficiency trade-off problem [9, 17]. GPS is more sample-efficient than previous model-free reinforcement learning methods, because it seeks to find solutions bridging optimal control and deep reinforcement learning, and in contrast to existing policy search algorithms learns local models in the form of linear Gaussian controllers. When provided with roll-out data from these linear local models, a global, nonlinear policy can be learned using an arbitrary parametrization scheme. The method alternates between (local) trajectory optimization and (global) policy search in an iterative fashion [19]. Our work is based on a particular variant of GPS called mirror descent guided policy search (MDGPS), because it is sample-efficient and its use of on-policy sampling makes it a good fit for this task [20]. Inoue et al.[21] use a Long Short-Term Memory network (LSTM) to learn two separate policies for finding and inserting a peg into a hole; however, their methods require several pre-defined heuristics, and also the action space is discrete. Most of the non-DRL methods require a significant amount of engineering and modelling, human demonstration, a prior controller, or carefully designed policies. For example, peg-and-hole use cases have also been successfully tackled with imitation learning with Gaussian Mixture Regression [22, 23].

B. Contributions

The contributions of this paper are several-fold. First, we tackle a novel use case, where a rigid part needs to be inserted into a deformable part. This is highly relevant for a variety of industrial use cases, e.g., assembly of sealing elements. Any contact-rich manipulation, which includes deformable nonlinear material has been very challenging for robot control so far. Traditional feedback control methods either require explicit consideration of the material response as a model or long periods of trial-and-error and fine-tuning. With our second contribution, we introduce a principled way for extending non-compliant robots to learning contact-rich manipulation skills. Currently, most policy search algorithms for contact-rich assemblies are implemented on inherently compliant robot arms such as the PR2, the iiwa or Rethink Robotics' Sawyer [24, 25, 26]. These robot arms have either passive compliance through spring mechanism in motors or have the ability to measure and command joint torques. These properties enable safe physical interaction of the robot with its environment, and joint torque readings can be encoded as features in learning algorithms to describe contact situations. Unfortunately, this is only of limited use for industrial applications because industrial robots are in general not compliant and offer only velocity and position control, but no torque control. However, they can often be equipped with a wrist force-torque sensor. While this still does not provide the ability to command joint torques, it opens the possibility for admittance force-torque control in task space. We show how this can be exploited for GPS usage even if joint torques cannot be directly commanded, but only positions and velocities. Third, we make an extension to MDGPS to directly incorporate the force-torque signals from the wrist sensor. To the best of our knowledge force-torque signals have not been incorporated in variants of the GPS algorithm yet.

The remainder of this paper is structured as follows. Section II introduces the problem of this paper. Section III describes the background and overview of the deployed methods. In Section IV we introduce our method, which is based on MDGPS. Experimental results are presented in Section V. Section VI draws conclusions and presents and future research.

II. PROBLEM STATEMENT

In this contribution we develop a policy search framework, which can be used for a variety of assembly tasks, which include inserting rigid parts into deformable objects. We demonstrate our approach on a modified version of the peg-in-hole task.

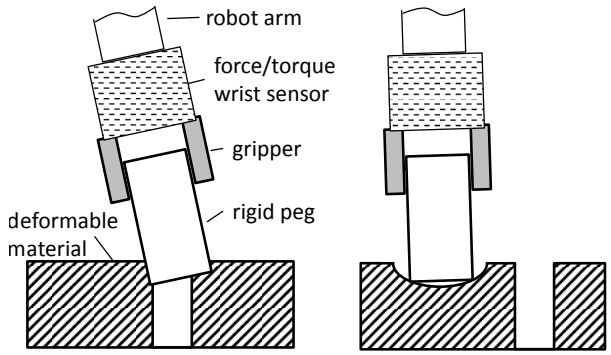
Consider a robot arm without the possibility of joint torque sensing or joint torque control. The robot has no inherent compliance and permits only position and velocity control. However, the robot arm is equipped with a wrist mounted force-torque sensor, whose signals are available. Moreover, the joint angles of the robot are accessible as well. The robot end-effector holds a rigid peg whose position and orientation relative to the end-effector are known with a certain degree of uncertainty. Moreover, the grasp is not tight and the peg exhibits some bounded motion relative to the end-effector whenever the peg is in contact with the environment. A work piece with a hole is placed in the work space of the robot. The work piece consists of a deformable material with nonlinear material properties. The diameter of the hole is smaller than the diameter of the peg. The position of the whole relative to the robot frame is known, but also exhibits uncertainty.

The goal of the peg-in-hole problem is to control the robot in order to insert the rigid peg into the deformable hole. It is challenging to design a feedback control law for achieving this task due to the unknown contact mechanics. Figure 2 illustrates the assembly situation. In Fig. 2 (a) the peg is successfully inserted into the elastic hole. The deformation posses challenges to the insertion tasks, because unlike rigid pairings, material yields if the robot pushes the peg into it as shown in Fig Fig. 2 (b). The key to escape from this local optimum is to properly adjust force/torque on the end-effector promptly.

We solve the peg-in-hole task through DRL without prior knowledge of any system dynamics.

III. REINFORCEMENT LEARNING IN CONTACT-RICH ENVIRONMENT UNDER UNKNOWN DYNAMICS

In this section, we describe the policy learning problem of robotic assembly with deformable material as well as the use of mirror descent guided policy search (MDGPS) algorithm, which is summarized as Algorithm 1) [19]. Furthermore, we introduce several modifications to this method for incorporating Cartesian space force/torque information.



(a) Peg in hole. (b) Peg stuck in material.

Fig. 2: Assembly modes for rigid peg and deformable material.

From a high-level perspective, guided policy search methods first learn local optimal controllers in the form of a time-varying linear-Gaussian $p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \mathbf{C}_t)$, and use these controllers to generate samples for training global policy $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ which are typically parameterized by a deep neural network θ; where $\mathbf{x}_t$ and $\mathbf{u}_t$ are states and actions at time step t, respectively, $\mathbf{o}_t$ are observations recorded during training time that may or may not equal $\mathbf{x}_t$. These time-varying linear-Gaussian controllers are under time-varying linear-Gaussian dynamics in the form $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}_t}\mathbf{x}_t + f_{\mathbf{u}_t}\mathbf{u}_t + f_{c_t}, \mathbf{F}_t)$, the goal is to minimize the expectation $E_{p(\tau)}[\ell(\tau)]$ over trajectory τ by iteratively optimizing linear-Gaussian controllers and re-fitting linear-Gaussian dynamics; where $\tau = \{\mathbf{x}_1, \mathbf{u}_1, \ldots, \mathbf{x}_T, \mathbf{u}_T\}$, $\ell(\tau) = \sum_{t=1}^{T} \ell(\mathbf{x}_t, \mathbf{u}_t)$ denotes the cost along a single trajectory τ, $p(\tau) = p(\mathbf{x}_1)\prod_{t=1}^{T} p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)p(\mathbf{u}_t|\mathbf{x}_t)$. These guided policy search methods also have some mechanism to enforce agreement between state distributions of local policy and global policy [19][17][9][12][27].

In contact-rich robotic assembly problems, getting a fairly accurate dynamical model of the robot itself is feasible. However, modeling physical contact dynamics can be very difficult or even intractable for several reasons: 1) Such contact dynamics are typically highly-nonlinear and discontinuous, deriving an exact physical model is difficult, let alone linearization of such models. 2) While rigid contact is already challenging, the material that the robot interacts with in this paper is deformable, thus making intractable physical dynamical models even more unreliable.

However, one can search for the target hole by "feeling" the surface. This can be done with some simple heuristics based on haptics feedback, for instance by probing the hole before inserting or moving the peg around the surface to search for the insertion point. Such heuristics do not require a precise physical model for contact dynamics. This implies that designing robust strategies by properly processing observations is more desirable in this setting than estimating perfect physical dynamics. Strategies as such may compensate deficiencies of model accuracy, state estimation errors or other components in a common optimal controller design pipeline. An end-to-end policy that directly maps raw observations to actions is a good choice to represent such robust strategies. High-capacity models such as deep neural networks are common choices for these end-to-end policies. The question arises *how do we integrate observed haptics information to local models as well as deep neural networks to effectively optimize the overall cost function?*

Our main motivation to integrate observed haptics information with the GPS framework is that its probabilistic formulation and policy training mechanism provide a convenient interface for processing additional raw sensor data. Note that the distribution for a trajectory is of the form $p(\tau) = p(\mathbf{x}_1)\prod_{t=1}^{T} p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)p(\mathbf{u}_t|\mathbf{x}_t)$. This includes an additional assumption about the initial state distribution $p(\mathbf{x}_1)$. In practice this is approximated by samples $\mathbf{x}_1^i$; hence it is important to pick the right initial states to cover areas of interest. Line 3 of Alg. 1 requires collecting roll-outs by running a local linear-Gaussian controller and deterministic resetting to the same initial state at each iteration. The fitted linear-Gaussian dynamics in the form of $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}_t}\mathbf{x}_t + f_{\mathbf{u}_t}\mathbf{u}_t + f_{c_t}, \mathbf{F}_t)$ do not necessarily reflect the dynamics governed by physics. The model can be a distribution highly over-fitting to a single motion trajectory that the robot iteratively samples from. In other words, if we get two such dynamical models $p_1(\mathbf{u}_t|\mathbf{x}_t)$ and $p_2(\mathbf{u}_t|\mathbf{x}_t)$ of a robot by iteratively sampling from two initial states $\mathbf{x}_1^{(1)}$, $\mathbf{x}_2^{(1)}$, they can be vastly different and non-interchangeable, although they intend to describe the dynamical behavior for the same robot. However, if we get enough of such dynamical models starting from initial states covering the final task's area of interest, the resulting guiding distributions can generate "good enough" data for deep neural networks to yield successful policies in terms of achieving desired goals. Haptic information such as Cartesian-space force/torque readings are generally regarded as external disturbances to the robot system, and not part of the system state space. Also, there is no necessary correlation of such readings between time steps assumed. However, the information occurred during contact interaction described in Sec. II can be highly informative; we can leverage this property by properly arranging force/torque measurements in the linear-Gaussian dynamics and force the neural network to infer underlying patterns from them. Specifically, $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}_t}\mathbf{x}_t + f_{\mathbf{u}_t}\mathbf{u}_t + f_{\tau_t}\tau_t + f_{c_t}, F_t)$, where $\tau_t = [F_x^t, F_y^t, F_z^t, M_x^t, M_y^t, M_z^t]$ denotes Cartesian-space force/torque sensor reading at time step t. The neural network architecture is illustrated in Fig. 3. The force/torque information is low-pass filtered and is concatenated to the second last network layer.

A. Mirror Descent Guided Policy Search (MDGPS)

The derivation in this section follows previous work [19]. For completeness we provide a summary here. Adopting the notations so far, we denote $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ as the global policy, parameterized by θ over actions $\mathbf{u}_t$, and conditioned on the

observations $\mathbf{o}_t$ recorded during training time, $p_i(\mathbf{u}_t|\mathbf{x}_t)$ is the i-th linear-Gaussian local policy. We wish to minimize the expected cost under π_θ evaluated on the current trajectory, i.e., $J(\theta) = \sum_{t=1}^{N} E_{\pi_\theta(\mathbf{x}_t,\mathbf{u}_t)}[\ell(\mathbf{x}_t, \mathbf{u}_t)]$, where $\ell(\mathbf{x}_t, \mathbf{u}_t)$ is the cost function. Thus, our overall optimization problem reads

$$\min_{\theta, p_1, \cdots, p_N} \sum_{i=1}^{N} \sum_{t=1}^{T} E_{p_i(\mathbf{x}_t,\mathbf{u}_t)}[\ell(\mathbf{x}_t, \mathbf{u}_t)]$$
$$s.t. \quad p_i(\mathbf{u}_t|\mathbf{x}_t) = \pi_\theta(\mathbf{u}_t|\mathbf{x}_t) \quad \forall \mathbf{x}_t, \mathbf{u}_t, t, i.$$

Instead of performing optimization on the parameter space by directly computing the gradient of $J(\theta)$, MDGPS is an on-policy sampling algorithm alternating between solving a constrained optimization problem with respect to local policies and training a global policy on samples generated from local policies by supervised learning with a surrogate loss function minimizing the KL-divergence between them. We improve local policies within some trust region on the constraint manifold in policy space, then use supervised learning to project these locally-improved policies back onto the constrained manifold in the parameter space. We then choose a simple representation of our global policy by mixing several state trajectory distributions, where convenient trajectory-centric optimization methods can be applied, e.g., iterative Linear Quadratic Gaussian (iLQG) control. The full algorithm is detailed as Alg. 1, where p_i represents the i-th local policy, o_t is the observation at time step t. Note that the equation includes KL-divergence constraints, which are calculated by linearizing the global policy π_θ, and serve to minimize the difference between the global and local policies. In this implementation, we use the same method to linearize the global policy, which was also used to fit the dynamics, i. e., we query the neural network policy to take an action, record the $\{x_t, u_t, x_{t+1}\}$ tuples, and perform linear regression on them using Gaussian Mixture Models as priors.

Algorithm 1 MDGPS

1: **for** iteration $k \in \{1, ..., K\}$ **do**
2: Generate samples $D_i = \{\tau_{i,j}\}$ by running either p_i or $\pi_{\theta i}$
3: Fit linear-Gaussian dynamics $p_i(\mathbf{u}_t|\mathbf{x}_t)$ using samples in D_i
4: Fit linearized global policy $\bar{\pi}_\theta(\mathbf{u}_t|\mathbf{o}_t)$ using samples in D_i
5: $p_i \leftarrow \arg\min_{p_i} E_{p_i(\tau)}[\sum_{t=1}^{T} l(\mathbf{x}_t, \mathbf{u}_t)]$ such that $D_{KL}(p_i(\tau)||\bar{\pi}_{\theta i}(\tau)) \leq \epsilon$
6: $\pi_\theta \leftarrow \arg\min_{\pi_\theta} \sum_{t,i,j} D_{KL}(\pi_\theta(\mathbf{u}_t|\mathbf{x}_{t,i,j}) ||p_i(\mathbf{u}_t|\mathbf{x}_{t,i,j}))$ (via supervised learning)
7: Adjust ϵ
8: **end for**

During the local trajectory optimization phase, the algorithm iteratively linearizes the dynamics around the current nominal trajectory, constructs a quadratic approximation to the cost, computes the optimal actions with respect to this approximation of the dynamics and cost, and forward runs resulting actions to obtain a new nominal trajectory. We use subscripts, e. g., $\ell_{\mathbf{x}\mathbf{u}t}$ to denote derivatives with respect to vector $[\mathbf{x}_t; \mathbf{u}_t]$. Under the dynamics model and the cost function described in this section, we can write the Q-function and value function as

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

We can solve for V and Q with a recurrence that can be computed backwards through time starting from the last time step $t = T$:

$$\begin{aligned} Q_{\mathbf{x}\mathbf{u},\mathbf{x}\mathbf{u}t} &= \ell_{\mathbf{x}\mathbf{u},\mathbf{x}\mathbf{u}t} + f_{\mathbf{x}\mathbf{u}t}^\intercal V_{\mathbf{x},\mathbf{x}t+1} f_{\mathbf{x}\mathbf{u}t} \\ Q_{\mathbf{x}\mathbf{u}t} &= l_{\mathbf{x}\mathbf{u}t} + f_{\mathbf{x}\mathbf{u}t}^\intercal V_{\mathbf{x}t+1} \\ V_{\mathbf{x},\mathbf{x}t} &= Q_{\mathbf{x},\mathbf{x}t} - Q_{\mathbf{u},\mathbf{x}t}^\intercal Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},\mathbf{x}t} \\ V_{\mathbf{x}t} &= Q_{\mathbf{x}t} - Q_{\mathbf{u},\mathbf{x}t}^\intercal Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t} \end{aligned} \quad (1)$$

This gives us the optimal control law $g(\mathbf{x}_t) = \mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t$, with $\mathbf{K}_t = -Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},\mathbf{x}t}$ and $\mathbf{k}_t = -Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t}$. As shown in previous work [9], we can in fact optimize the MaxEntropy LQR objective by using the time-varying linear-Gaussian controller $p(\mathbf{u}_t|\mathbf{x}_t)$:

$$\min_{p(\mathbf{u}_t|\mathbf{x}_t)} \sum_{t=1}^{T} E_{p(\mathbf{u}_t|\mathbf{x}_t)}[\ell(\mathbf{x}_t, \mathbf{u}_t) - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))]$$

The last term in the objective function is an entropy term encouraging exploration. This objective can be optimized by setting $p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \mathbf{C}_t)$, and $\mathbf{C}_t = Q_{\mathbf{u},\mathbf{u}t}^{-1}$. The intuition behind this is: $Q_{\mathbf{u},\mathbf{u}_t}$ reflects how the controller's action at time step t,$\mathbf{u}_t$ will affect the cost to go $Q(\mathbf{x}_t, \mathbf{u}_t)$; if $Q_{\mathbf{u},\mathbf{u}_t}$ is large, the action will largely affect future return, so we should reduce exploration; if $Q(\mathbf{x}_t, \mathbf{u}_t)$ is small, then we can do a bit more exploration, so setting $\mathbf{C}_t = Q_{\mathbf{u},\mathbf{u}_t}^{-1}$ properly reflects this.

B. Integrating Cartesian-Space Force/Torque Information

Force/torque information is important for contact-rich assembly tasks. It allows robots to feel the surface that is going to be operated on and the reactions from the parts. However, it is difficult to incorporate such information into the dynamic system for several reasons:

- Force/torque information cannot be simply concatenated into the robot state space. By denoting force/torque vector on robot wrist at time step t as $\tau_t = [F_x^t, F_y^t, F_z^t, M_x^t, M_y^t, M_z^t]$, the vector τ_{t+1} can have a relatively weak correlation with τ_t. Two subsequent force/torque vectors are regarded as measurements of external interactions.
- Force/torque sensor signals can be very noisy and therefore it can be difficult to extract useful information.

In our study, we compare several methods to incorporate force/torque information into the policy learning process. First τ_t is concatenated into the robot state space, $\tilde{\mathbf{x}_t} = [\mathbf{x}_t; \tau_t]$, and everything else in Alg. 1 remains unchanged, this

is both, a sanity check and baseline policy learning. Second, we use the formulation mentioned above $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}_t}\mathbf{x}_t + f_{\mathbf{u}_t}\mathbf{u}_t + f_{\tau_t}\tau_t + f_{c_t}, F_t)$ as our local policy and provide force/torque information to the global policy neural network. Figure 3 shows the schematic of the overall control structure. Note that the neural network feeds into an admittance controller (AC), which also incorporates the force/torque signals. The next section provides details on the AC functionality.

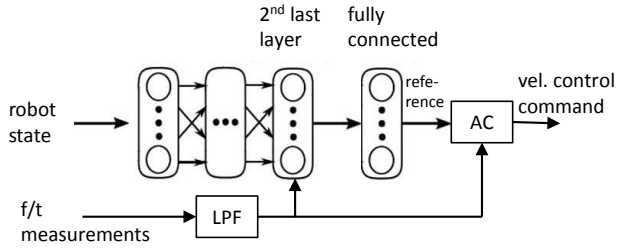


Fig. 3: Neural net architecture with concatenated and low-pass filtered (LPF) force/torque measurement. The neural net outputs a reference to an admittance controller (AC) that computes the velocity control commands.

IV. Policy Search on Non-Compliant Robots Equipped with F/T Wrist Sensor

A. Cartesian-Space Admittance Controller

Most industrial robots only provide interfaces for position and velocity control, and do not have low-level torque interfaces. This raises additional challenges for learning contact-rich assembly tasks, since it is vital that robots can "feel" or "touch" during the contact process, positional information alone is often not enough. Furthermore, non-compliant position-controlled industrial robots can be dangerous in learning regimes; they will stick to pre-calculated trajectories thus yielding unexpected large force or moments when facing contact. We address this issue by feeding Cartesian-space force/torque sensor signals to the robot's Cartesian-space velocity control loop. Specifically, denote τ as the desired force/torque vector in tool space; this includes three forces and three torques in three corresponding Cartesian axes. $\hat{\tau}$ denotes the measured force/torque vector from the robot's wrist sensor. We apply a PD feedback controller to the difference between τ and $\hat{\tau}$, and feed the resulting control to the Cartesian-space velocity command interface of the robot. This involves calculating the inverse Jacobian matrix of the robot and converting Cartesian-space velocity into joint-space velocity. We couple the resulting admittance controller with our reinforcement learning algorithms. Figure 3 presents a diagram of this controller.

While this seems like an obvious step, to the best of our knowledge this has not been explicitly studied in the context of policy search in contact-rich assemblies. Most robots used for policy search research such as Rethink Robotics Sawyer and the popular PR2 are inherently compliant and joint torques can be commanded directly. Our approach allows us to use the same policy search methods on robots, where only joint velocities and positions can be controlled.

B. Optimize Policy with Force/Torque Information

As described in the previous section, we intend to incorporate force/torque information in local policy learning process by adding an additional term to the time-varying linear-Gaussian dynamics in the form of $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}_t}\mathbf{x}_t + f_{\mathbf{u}_t}\mathbf{u}_t + f_{\tau_t}\tau_t + f_{c_t}, F_t)$. The intuition behind this is several fold. First, adding direct measurements of contact loads τ_t can provide better estimates of the state $\mathbf{x}_{t+1}$ if properly processed. Second, although these weak linear dynamical models may not describe contact behavior well, this can be just "good enough" to capture probabilistic transition relations, and policy neural networks might generalize from these distributions. Optimizing this new time-varying linear-Gaussian controllers involves dynamic programming as presented in Eq. (1). For practical implementations, we build upon the open source guided policy search codebase provided in [28]. Hence, we want to reuse the recurrence in Eq. (1). We re-write the dynamics in an equivalent form:

$$\mathbf{x}_{t+1} = f_{\mathbf{x}_t}\mathbf{x}_t + f_{\mathbf{u}_t}\mathbf{u}_t + f_{\tau_t}\tau_t + f_{c_t} + \sigma, \;\; \sigma \in \mathcal{N}(0, F_t)$$

In order to re-use the convenient recurrence in (1), we rearrange the equation as follows:

$$\underbrace{\begin{bmatrix}\mathbf{x}_{t+1}\\ \tau_{t+1}\end{bmatrix}}_{\mathbf{x}'_{t+1}} = \underbrace{\begin{bmatrix}f_{\mathbf{x}_t} & f_{\tau_t}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}}_{f'_{\mathbf{x}_t}} \underbrace{\begin{bmatrix}\mathbf{x}_t\\ \tau_t\end{bmatrix}}_{\mathbf{x}'_t} + f_{\mathbf{u}_t}\mathbf{u}_t + f_{c_t} + \sigma.$$

This is in the form of

$$\mathbf{x}'_{t+1} = f'_{\mathbf{x}_t}\mathbf{x}'_t + f_{\mathbf{u}_t}\mathbf{u}_t + f_{c_t} + \sigma$$
$$\text{or } \; p(\mathbf{x}'_{t+1}|\mathbf{x}'_t, \mathbf{u}_t) = \mathcal{N}(f'_{\mathbf{x}_t}\mathbf{x}'_t + f_{\mathbf{u}_t}\mathbf{u}_t + f_{c_t}, F_t). \quad (2)$$

Equation (2) has the exact form of time-varying linear-Gaussian dynamics used in recurrence of equation (1). In practical implementations, we simply replace $\mathbf{x}_t$ with $\mathbf{x}'_t$, and $f_{\mathbf{x}_t}$ with $f'_{\mathbf{x}_t}$, everything else can be a standard LQR backward pass and forward pass reflected in equation (1). Note that this simple rearrangement is not the same as augmenting the robot state space that we discussed before. This is just for convenience to calculate the optimal control laws. In the case of augmenting the state space, the new "$f'_{\mathbf{x}_t}$" matrix will be entirely fitted using samples, forcing τ_t to relate with every term before time step t in the system, which we should avoid as τ_t, in general, is not Markovian. In our case, we simply write the equation in an equivalent form for conveniently solving trajectory optimization already implemented in the GPS code base [28]. Note that there are two zero blocks in the matrix $f'_{\mathbf{x}_t}$, which will cause $\tau_{t+1} = 0$ for all time steps, in our implementation, we just drop this and use actual sensor readings for τ_t. The aforementioned descriptions in this subsection refer to modifications applied to line 5 of Alg. 1.

We also introduce a novel neural network architecture to process noisy force/torque readings from wrist sensors. The neural net is shown in Fig. 3. Force/torque information is

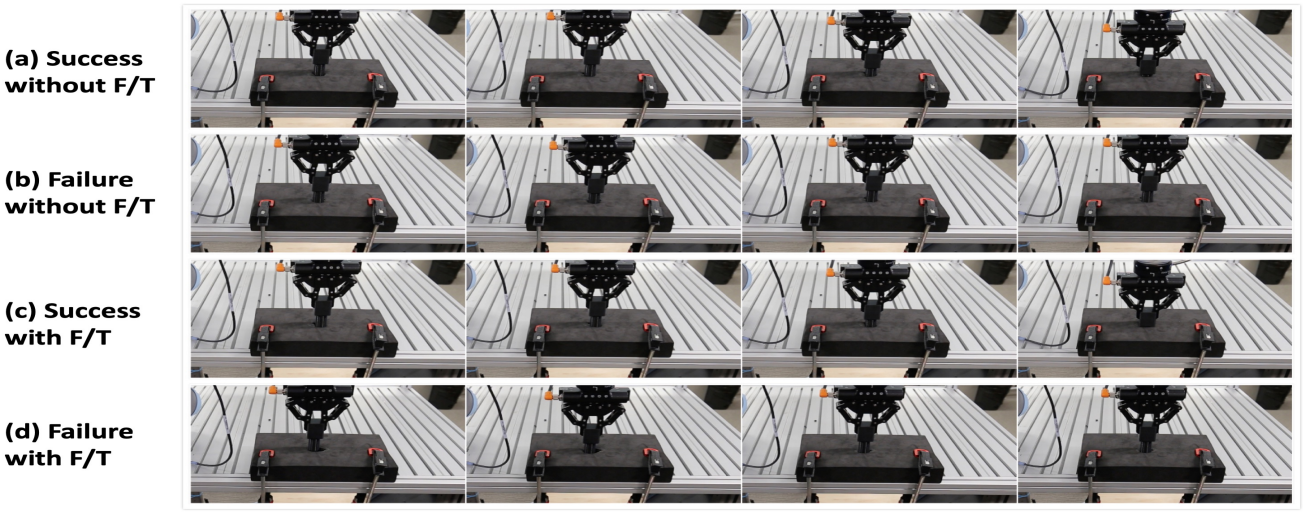


Fig. 4: (a) Baseline method succeeds from an offset of 1 mm to the hole. (b) Baseline method fails from an offset of 10 mm to the hole. (c) Our method succeeds from roughly the same position that (a) fails from. (d) Our method fails when it is too far from its training set.

filtered using a low-pass filter (LFP), then concatenated in the second last layer of the neural network. Intuitively, we would like to provide the most direct haptics information to the neural network as a principle feature; yet preventing neural network to establish unreasonable correspondence between external force/torque readings and robot internal states. This introduces modifications to line 6 of Alg. 1.

.

V. EXPERIMENTS

A. Experimental Setup Details

As shown in Figure 5, we use a UR5 robot manufactured by Universal Robots, Polyscope version 3.5 with the motherboard in the control box having been upgraded to CB3.1. We use the force/torque sensor FT300 from Robotiq with a range of measurements of 300 N for forces and 30 Nm for torques. The noise levels (estimated standard deviations) for F_x, F_y are 1.2 N; F_z is 0.5 N; M_x, M_y are 0.02 Nm; and M_z is 0.03 Nm. The data output frequency is 100 Hz. We use a 85mm adaptive gripper from Robotiq. We remove the fingertips from the gripper, and fixed the peg directly to the gripper. However, the gripper connection allows bounded motions of the peg, which introduces some uncertainty. The hole is part of a work piece, which is made of ethylene-vinyl acetate. Its diameter is 20 mm while the peg's diameter is 25 mm. The cycle time for the UR5 is 8 ms, the robot does not provide access to its real-time control loop, also the force/torque sensor rate is 100 Hz. The robot neither has joint compliance nor does it offer joint torque control. Hence, it is well suited to benchmark our approach.

We vary the robot starting configuration for training local policies. The algorithm will be iteratively executed by deterministic reset to the same configuration after sampling trajectories. We collect five samples for each starting configuration in every iteration. We send control commands at 25 HZ; and each trajectory is 3.5 seconds long.

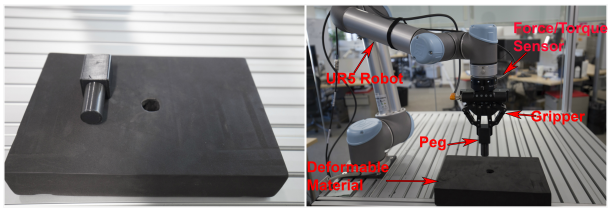


Fig. 5: Experimental setup for peg insertion task.

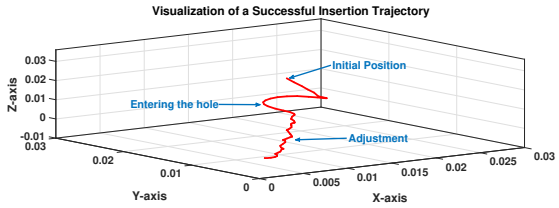


Fig. 6: Visualization of peg trajectory during one successful insertion.

The robot state space consists of joint angles, joint velocities, the end-effector pose, which is represented by three Cartesian points and the velocities of these points. For an initial condition (robot configuration), we specify their target end-effector pose, the cost function follows prior work [11], which is given by $r_\ell(d) = wd^2 + v\log(d^2 + \alpha)$, with $\alpha = 10^{-5}$, $v = 0.01$ and $w = 1.0$. This shaped cost function performs better than an l_2 distance cost function because the second term encourages precise placement near the target position. In order not to damage the material, in all our experiments, we locked rotations in three Cartesian

axes, thus the policy outputs an action $\mathbf{u}_t = [F_x, F_y, F_z]$. We use a four layer neural network to represent our global policy, each layer has 128 neurons with ReLu nonlinearity.

We use a desktop PC with a Xeon E5-2670 CPU, Nvidia Titan GPU, 24 GB of installed memory, the communication from computer to robot is done via the Robot Operating System (ROS). In our experiments, they generally require five iterations of policy training to reach a stable convergence. The total duration accounts to approx. 20 minutes including computation, robot interaction and all intended time-delays.

Our policy search algorithm was implemented on the aforementioned setup. Experiments were carried out under several conditions, which are detailed in the following subsections.

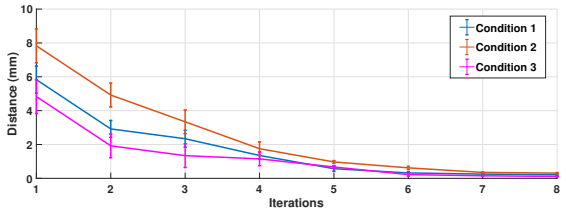


Fig. 7: Learning curve in three experiments starting from different positions.

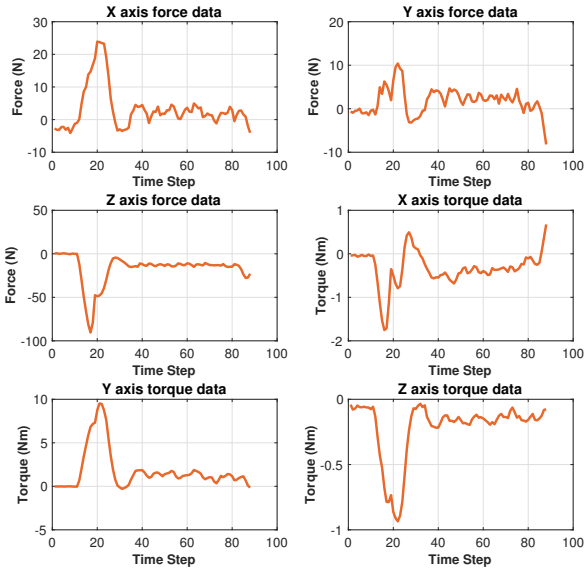


Fig. 8: Force/Torque sensor readings recorded during one successful insertion.

B. LQG without F/T

In this experiment, neither force/torque information is provided to the local policy nor to the global policy. Both policies use only the robot configuration information to generate control actions. We find that the system can learn local policies with some success if the starting position is just above the hole. However, the resulting global policy often gets into local optima and is unable to find the hole if we move the hole position slightly further. Moreover, the system is unable to consistently learn successful local policies, if the starting position exhibits even small position offsets to the hole. Figure 4(a)(b) shows some of those cases.

C. F/T Policy

We benchmark the method of this paper, which exploits the force/torque information. The local policy learning is able to succeed even if the starting position is relatively far away from the hole, when provided with force/torque feedback. The learned global policy is able to generalize to new positions not covered in the training set. These new positions can have relatively large offsets to the hole. The largest offset for which the system reliably succeeds is a 1 cm offset in both X and Y axis. This means that the controller we obtained through RL adjusts itself to a large uncertainty in the hole position. This is of potentially great benefit to manufacturing operations. Figure 4(c)(d) shows some of the successes and failures. Several learned strategies can be observed. In the case that the initial position is slightly off the hole, the robot tends to apply large vertical forces to "slide" into the hole. While for the case that the initial position is further off the hole, the robot often first gets stuck in the material, then slightly lifts its arm to reduce the exerted force from the material, and moves its arm quickly after the lifting to find the hole. Figure 7 shows the learning curve for three separate experiments that start from different positions, we can see the policy training is able to reach a reliable convergence in roughly five iterations. Figure 8 shows force/torque data collected in one successful insertion. From these data, we can see clear indications for some important behaviors such as exploration on the deformable material or probing and entering the hole, this further consolidates our argument of feeding the raw force torque data to the deep neural net without hand-engineered features. We also visualize the trajectory of one successful insertion that is shown in Figure 6. We can observe the learned policy is trying to "feel" the surface before inserting the peg into the hole, and the behavior also shows careful adjustments after the peg enters the hole. Table I summarizes the experimental results described above.

TABLE I: Comparison of success rate, with different offsets to the hole. F/T information is provided to both, local , and global policy. The baseline method uses robot state information only.

Offset to hole	1 mm	5 mm	10 mm
our method	5/5	5/5	3/5
baseline	4/5	3/5	0/5

VI. CONCLUSIONS AND FUTURE RESEARCH

This paper covers robotic assembly of combined rigid and deformable parts. There are various industrial use-cases,

where this is of high relevance, such as installing of sealing elements. In this contribution, we extended the model-based DRL approach MDGPS with haptic feedback for learning the insertion of a peg into a deformable hole, which served as a representative example. Moreover, our approach does not assume that joint torques can be directly commanded. Instead we exploited admittance control and force-torque signals from a wrist sensor, which is the typical setup for industrial robots. Our experimental results showed that the robot is able to rapidly learn the task and we demonstrated robustness against levels of variations. The proposed method worked well when the peg was placed relatively close to the hole. A beneficial extension of our work would be incorporating a vision system so that the robot can be guided into nearby positions. This would add a great deal of flexibility in terms of robot's starting position. One interesting future direction is to add raw vision input to the architecture in Figure 3, thus becoming an end-to-end multi-modal neural network. It would be interesting to see if such learned policy can succeed from arbitrary starting positions in free space.

ACKNOWLEDGEMENT

This work is partially supported by a Siemens' partnership with the UC Berkeley Center for Information Technology in the Interest of Society.

REFERENCES

[1] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

[2] M. P. Deisenroth, G. Neumann, J. Peters, et al. "A survey on policy search for robotics". In: *Foundations and Trends in Robotics* 2.1–2 (2013), pp. 1–142.

[3] J. Kober, J. A. Bagnell, and J. Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[4] D. E. Whitney. "Historical perspective and state of the art in robot force control". In: *The International Journal of Robotics Research* 6.1 (1987), pp. 3–14.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[7] J. Luo, R. Edmunds, F. Rice, and M. Agogino. "Tensegrity Robot Locomotion under Limited Sensory Inputs via Deep Reinforcement Learning". In: *Robotics and Automation (ICRA), 2018 IEEE International Conference on*. IEEE. 2018.

[8] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. "Trust region policy optimization". In: *International Conference on Machine Learning*. 2015, pp. 1889–1897.

[9] 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.

[10] T. Zhang, G. Kahn, S. Levine, and P. Abbeel. "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search". In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 528–535.

[11] 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.

[12] Y. Chebotar, M. Kalakrishnan, A. Yahya, A. Li, S. Schaal, and S. Levine. "Path integral guided policy search". In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE. 2017, pp. 3381–3388.

[13] J. Peters, K. Mülling, and Y. Altun. "Relative Entropy Policy Search." In: *AAAI*. Atlanta. 2010, pp. 1607–1612.

[14] J. Peters and S. Schaal. "Reinforcement learning of motor skills with policy gradients". In: *Neural networks* 21.4 (2008), pp. 682–697.

[15] I. Lenz, R. A. Knepper, and A. Saxena. "DeepMPC: Learning Deep Latent Features for Model Predictive Control". In: *Robotics: Science and Systems*. 2015.

[16] S. Levine, C. Finn, T. Darrell, and P. Abbeel. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[17] S. Levine and V. Koltun. "Guided policy search". In: *International Conference on Machine Learning*. 2013, pp. 1–9.

[18] V. Gullapalli, R. A. Grupen, and A. G. Barto. "Learning reactive admittance control". In: *IEEE International Conference on Robotics and Automation*. 1992, pp. 1475–1480.

[19] W. H. Montgomery and S. Levine. "Guided policy search via approximate mirror descent". In: *Advances in Neural Information Processing Systems*. 2016, pp. 4008–4016.

[20] S. Ross, G. Gordon, and D. Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 627–635.

[21] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana. "Deep reinforcement learning for high precision assembly tasks". In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE. 2017, pp. 819–825.

[22] T. Tang, H.-C. Lin, and M. Tomizuka. "A learning-based framework for robot peg-hole-insertion". In: *ASME 2015 Dynamic Systems and Control Conference*. American Society of Mechanical Engineers. 2015, V002T27A002–V002T27A002.

[23] T. Tang, H. Lin, Y. Zhao, W. Chen, and M. Tomizuka. "Autonomous alignment of peg and hole by force/torque measurement for robotic assembly". In: *Automation Science and Engineering (CASE), 2016 IEEE International Conference on*. IEEE. 2016, pp. 162–167.

[24] Kuka. *Kuka LBR iiwa*. URL: `https://www.kuka.com/en-us/products/robotics-systems/industrial-robots/lbr-iiwa`.

[25] Rethink Robotics. *Sawyer User Guide*. URL: `http://mfg.rethinkrobotics.com/mfg-mediawiki-1.22.2/images/1/1a/Sawyer_User_Guide_3.3.pdf`.

[26] Willow Garage. *PR2 Robot Manual*. URL: `https://www.clearpathrobotics.com/wp-content/uploads/2014/08/pr2_manual_r321.pdf`.

[27] Y. Chebotar, K. Hausman, M. Zhang, G. Sukhatme, S. Schaal, and S. Levine. "Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning". In: *International Conference on Machine Learning (ICML) 2017*. Aug. 2017.

[28] C. Finn, M. Zhang, J. Fu, X. Tan, Z. McCarthy, E. Scharff, and S. Levine. *Guided Policy Search Code Implementation*. Software available from rll.berkeley.edu/gps. 2016. URL: `http://rll.berkeley.edu/gps`.