Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 58-28 (2024) 1019–1024

Guided Policy Search for Stabilizing Contact-rich Motion Plans

Christopher Dagher * Chandika Silva * Aykut C. Satici ** Hasan A. Poonawala ***

* *Boise State University, Boise, ID 83725 USA (e-mail: {christopherdaghe,chandikasilva}@u.boisestate.edu).*
** *Boise State University, Boise, ID 83725 USA (e-mail: aykutsatici@boisestate.edu).*
*** *University of Kentucky, Lexington, KY 40506 USA (e-mail: hasan.poonawala@uky.edu).*

Abstract: Learning policies for contact-rich manipulation is a challenging problem due to the presence of multiple contact modes with different dynamics, which complicates state and action exploration. Contact-rich motion planning uses simplified dynamics to reduce the search space dimension, but the found plans are then difficult to execute under the true object-manipulator dynamics. This paper presents an algorithm for learning controllers based on guided policy search, where motion plans based on simplified dynamics define rewards and sampling distributions for policy gradient-based learning. We demonstrate that our guided policy search method improves the ability to learn manipulation controllers, through a task involving pushing a box over a step.

Keywords: Robotics, machine learning, artificial intelligence, motion planning, robust control.

1. INTRODUCTION

Many human manipulation tasks that we would like robots to perform involve moving objects that are too large to grasp in the robot's gripper. For example, moving objects whose every dimension is larger than the gripper's width, like a large box. Such manipulation tasks are referred to as non-prehensile motion tasks that are not within the hand, from the taxonomy in Bullock and Dollar (2011).

Successful task execution under nonprehensile manipulation requires complex reasoning about how forces applied on the object interact with its natural dynamics and the forces applied by the environment at contact locations. This reasoning is challenging to carry out due to the multiplicity of possible contact modes and the uncertain contact dynamics governing each mode. Recent advances automate this reasoning over multiple modes, like the work by Cheng et al. (2022), resulting in high-quality plans for achieving a task using a sequence of contact modes. However, to achieve computational tractability, these methods simplify the robot-object dynamics by assuming that the robot-and-object are either always in equilibrium (quasi-static), or moving with constant velocity (quasi-dynamic), which constrain the versatility of nonprehensile manipulation. Moreover, the execution of this sequence typically relies on applying low-level position controllers. As pointed out in Cheng et al. (2022), this mismatch between the dynamics used for planning and the true dynamics during execution leads to task failure. However, designing robust feedback controllers with the full system state as input is too challenging using manual methods, due to the size of the state space and the complexity of the dynamics.

The complexity of the dynamics underlying contact-rich manipulation scenarios has led to an increase in computational approaches for synthesizing controllers for these scenarios Hogan and Rodriguez (2020); Tian et al. (2019). One example in this domain is reinforcement learning (RL) (see Sutton and Barto (2018)). The resulting state-based policies act as feedback controllers, providing some robustness to uncertainty during task execution. While RL methods evidently offer more applicability to a wide range of dynamical systems, such as in Heess et al. (2017); Andrychowicz et al. (2020); Lillicrap et al. (2015), many of them discard any potential geometric or physics structures that may be useful in control synthesis. This prohibitively increases the sample complexity of the learning process while risking suboptimal solutions, as pointed out in Levine and Koltun (2013).

In this work, we combine learning-based controller synthesis with contact-aware motion-planning to construct feedback control laws for contact-rich manipulation. The contact-aware motion planning algorithm, namely Contact-Mode Guided Motion Planning (**CMGMP**) proposed in Cheng et al. (2022), identifies contact mode sequences that are valuable for completing the task, despite the actual plan being dynamically infeasible. This identification reduces the challenge of exploration faced by RL algorithms, and is a key insight of our approach. The learning-based algorithm, namely policy search, addresses the challenge of learning controllers for complex high-dimensional dynamics.

⋆ This work was supported by NSF IIS award #2330794.

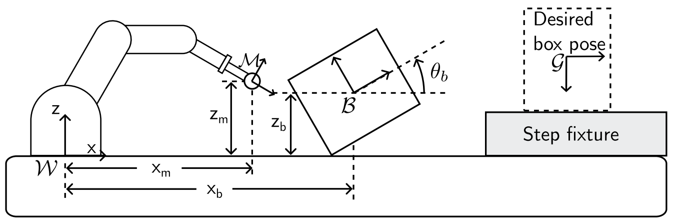

Fig. 1. A 2D representation of the step environment.

Contributions. The first contribution is to define a mechanism for guiding policy search algorithms using motion plans for contact rich manipulation. This work involves defining appropriate loss functions and sampling mechanisms relevant to contact-rich manipulation. The second contribution is demonstrate that our proposed guided policy search algorithm makes the training converge faster wile simultaneously improving task performance. This demonstration uses the task of manipulating a box with a robotic finger in contact with the box, over a step.

2. PROBLEM FORMULATION

This paper addresses the problem of learning a robust controller at the force/torque level for a robotic end-effector that is capable of moving a box from an initial pose $\mathbf{q}_0^o$ to a desired final pose $\mathbf{q}_d^o$, where $\mathbf{q}_0^o, \mathbf{q}_d^o \in \mathrm{SE}(2)$, the 2-dimensional special Euclidean group. The object and end-effector are confined to a vertical plane, where gravity affects the dynamics, in contrast to work in Hogan and Rodriguez (2020) where contact-rich manipulation focuses on pushing an object on horizontal surfaces. The nontrivial nature of this problem is due to the facts that (i) the end-effector is assumed to make a point contact with the object only, and (ii) the goal pose $\mathbf{q}_d^o$ is located on top of a step fixture, which is elevated from the table on which the box initially rests (see Figures 1 or 2). In other words, the manipulator cannot perform a force-closure grasp (see Murray et al. (2017)) of the box; requiring the manipulation plan be informed and take advantage of the natural contact-rich dynamics of the system. Due to use of a single contact, we model the end-effector as an actuated sphere.

Figure 1 sketches the system setup with the relevant configuration variables. The system comprises the manipulator finger and the box, henceforth denoted $\mathcal{M}$, and $\mathcal{B}$, respectively. The configuration space of the system is the product space $\mathcal{Q} = \mathrm{SE}(2) \times \mathrm{SE}(2)$, the set of box and manipulator configurations, $\boldsymbol{q} = (\mathbf{q}^o, \mathbf{q}^m)$. The system's state space $\mathcal{X}$ is the tangent bundle of its configuration space $\mathcal{X} = T\mathcal{Q}$. Please see Table 1 for the definition of all state variables. We assign a frame to the goal state and denote it by $\mathcal{G}$. This controller must map the state of the system to the forces or torques to be produced by the manipulator's actuators. In this work, we parametrize the controller $P = P_\phi$ as a multilayer perceptron.

3. BACKGROUND

3.1 Contact-Mode Guided Motion Planning (CMGMP)

The CMGMP algorithm proposed by Cheng et al. (2022) is a variant of the widely used RRT planning algorithm proposed by Karaman and Frazzoli (2011) that accounts for the differences in dynamics corresponding to different contact modes between the object, manipulator, and environment. The existence of multiple contact modes makes the robot-object-environment a hybrid system instead of a continuous dynamical system, and the CMGMP algorithm modifies the standard steps of the RRT algorithm to account for this difference. An important innovation by Huang et al. (2020) is to be able to enumerate all possible contact modes in a state efficiently, enabling appropriate sampling of the hybrid state space. However, to reduce the complexity of the search, CMGMP uses a Quasidynamic model for the *extend* operation in the standard RRT algorithm, instead of the full nonlinear object-manipulator dynamics. This practical modification results in planned trajectories that may not be dynamically feasible, and for which standard position-based controllers will often fail, as Cheng et al. (2022) reports.

Table 1. Nomenclature for state variables.

	State		
	Configuration		*Velocity*
Rigid Body	*Position*	*Orientation*	
Manipulator	$\mathbf{q}^m$	$\mathbf{R}^m$	$\mathbf{v}^m$
Object	$\mathbf{q}^o$	$\mathbf{R}^o$	$\mathbf{v}^o$
Goal	$\mathbf{q}^g$	$\mathbf{R}^g$	-

3.2 MJX Physics Engine

We use the MJX physics engine which is an implementation of MuJoCo (Todorov et al. (2012)), written in JAX (Bradbury et al. (2018)). This physics engine is differentiable through contact, and can be combined with machine learning techniques in order to develop controllers through analytical, data-driven gradient-based methods.

3.3 Policy Gradient Algorithms

Policy gradients, a form of policy search proposed in Sutton et al. (1999), involve training controllers by performing gradient descent (or ascent) over the parameters of the controller, in this case ϕ, with respect to the expected rewards. A controller synthesis methodology that utilizes policy gradients for updating controller parameters and DAgger (Ross (2011)) for state-space sampling was used in Deits et al. (2019). In this paper, we use a similar approach for state-space sampling. Our main improvement is to use the plan from CMGMP to define a loss function that guides parameter updates, which enables tackling the potentially exponentially many "modes" of the system dynamics.

4. METHODOLOGY

Our proposed algorithm modifies the guided policy search algorithm in Deits et al. (2019), described in Section 3.3, by defining rewards using an expert trajectory found using the CMGMP algorithm.

We design two loss functions, one for unguided (no expert) training, and another for guided policy training, which makes use of the trajectory provided by CMGMP. These loss functions are then differentiated and the gradient backpropagated to update the controller parameters for improved closed-loop performance.

4.1 Neural Network Policy

The neural-network policy, denoted by P is the mapping that takes the system state to the wrenches (forces and torques) exerted on the manipulator finger by the robotic manipulator, i.e.,

$$P : \mathcal{X} \to U \subseteq \mathfrak{se}^*(2), \quad \boldsymbol{x} \mapsto P(\boldsymbol{x};\phi) = (\boldsymbol{f}(\boldsymbol{x};\phi), \tau(\boldsymbol{x};\phi)),$$

where $\boldsymbol{f} : \mathcal{X} \to \mathbb{R}^2$ and $\tau : \mathcal{X} \to \mathfrak{so}^*(2)$ are functions, parametrized by ϕ. These wrenches are then assumed to be generated by the joint torques of the robot using the Jacobian mapping J; for example, if robot manipulator is fully actuated then its joint torques are assigned according to $\boldsymbol{\tau}_{\text{joint}} = J^\top P$. The function $P = P(\boldsymbol{x};\phi)$ is parametrized by using a multilayer perceptron in this work.

4.2 Loss Function

The loss function for a sampled state $\boldsymbol{x}$ has the following form:

$$\begin{aligned}\mathcal{L}(\boldsymbol{x}, \mathrm{x}_d;\phi) =& w_{og}\|\mathbf{q}^o - \mathbf{q}^g\|^2 + w_{oo^*}\|\mathbf{q}^o - \mathbf{q}^o_d\|^2 + \cdots \\ &+ w_{mm^*}\|\mathbf{q}^m - \mathbf{q}^m_d\|^2 + w_{ov}\|\mathbf{v}^o\|^2 + \cdots \\ &+ w_{mv}\|\mathbf{v}^m\|^2 + w_{mo}|g(\boldsymbol{x})|, \end{aligned} \tag{1}$$

where the first term penalizes distance of object (o) to the goal (g), the second term penalizes error in position of the object as defined by the planned path, the third term does the same for the manipulator, the fourth and fifth terms penalize velocities, and the last term penalizes loss of contact between object and manipulator through the gap function $g(\boldsymbol{x})$. When the learning is not guided by the planned motion, the second and third terms are not included.

4.3 Generating Training Data

Similar to Deits et al. (2019), we use DAgger (Ross et al. (2011)) to generate samples for training the policy neural network π_ϕ using the loss function in (1). To use DAgger, we must specify a method to sample *initial conditions* for the trajectories.

Sampling A State: Uniformly sampling over the object-manipulator state space (with configuration in SE(2) × SE(2)) would be inefficient, since states that exhibit contact between the robot, object, and environment correspond to a thin (measure-zero) subset of the object-manipulator configuration space. Similar to Cheng et al. (2022), we use a sequential procedure to generate a sample by sampling an object configuration first and then using contact to guide sampling of the manipulator configuration. This procedure is given in Algorithm 1, where $U(S)$ corresponds to a uniform distribution over the set S, and described below.

The CMGMP algorithm returns a finite sequence of states $\{\boldsymbol{x}_k\}_{1,\dots,N}$ at times t_k. Let $\mathbf{q}^o_k$ be the object configuration at these times, and consider a projection of the object configuration onto the ground plane:

$$\mathcal{P}_z(\mathbf{q}^o) = (x_o, y_o). \tag{2}$$

Algorithm 1. Sampling a state $\boldsymbol{x}$. Point $\boldsymbol{x}_c \in \mathbb{R}^3$ is the contact point between object and manipulator.

```
Input: Maximum contact gap $\varepsilon \geq 0$
Input: Sampling region $G$
Input: Floor height function $h$
Input: Set of admissible contact locations $C_{\partial O}(\mathbf{q}^o)$
Input: Set of admissible velocities $V(\mathbf{q}^o, \mathbf{q}^m)$
Input: Set of preferred object orientations $\mathcal{R}(\mathbf{p}^o)$
Input: Manipulator forward kinematics function $f_m$
$(x_o, y_o) \sim U(G)$
 $z_o \leftarrow h(x_o, y_o) + U([0, \epsilon])$
 $\mathbf{p}^o \leftarrow (x_o, y_o, z_o)$
 $\mathbf{R}^o \leftarrow U(\mathcal{R}(\mathbf{p}^o))$
 $\mathbf{q}^o \leftarrow (\mathbf{p}^o, \mathbf{R}^o)$
 $\boldsymbol{x}_c \sim U(C_{\partial O}(\mathbf{q}^o))$
 $\mathbf{q}^m \leftarrow f_m^{-1}(\boldsymbol{x}_c)$;
 $\mathbf{v}^o, \mathbf{v}^m \sim U(V(\mathbf{q}^o, \mathbf{q}^m))$
 return $\boldsymbol{x} = (\mathbf{q}^o, \mathbf{q}^m, \mathbf{v}^o, \mathbf{v}^m)$
```

We define G to be the convex hull of the points $\mathcal{P}_z(\mathbf{q}^o_k)$ for times t_k, where $k \in \{1, \dots, N\}$. To generate a position for the object, we sample uniformly from G, and then perturb the object along the positive z direction from the ground, through a random distance sampled from $U([0, \epsilon])$. The 'ground' at the sample from G need not be at zero height. The orientation is sampled from a restricted set $\mathcal{R}(\mathbf{p}^o)$, where all elements are such that the bottom of the object is tangent to the ground surface beneath it, but without incurring collision with the surface. We then sample a point p_c on a subset $C_{\partial O}(\mathbf{q}^o)$ of the boundary ∂O of the object. This subset typically excludes inaccessible points on the object boundary, and therefore depends on the object configuration. The manipulator configuration is chosen to ensure contact occurs at p_c within some distance of the object, where this distance is given by a gap function $g\colon \mathrm{SE}(2) \times \mathrm{SE}(2) \to \mathbb{R}_{\geq 0}$.

Once the configurations of the manipulator and object are obtained, velocities of the manipulator and object are sampled uniformly from the set of allowable velocities $V(\mathbf{q}^o, \mathbf{q}^m)$ based on the contact conditions for the manipulator-object configuration, where the set is bounded. Note that the contact conditions are determined by the unperturbed configuration of the object. The fact that the object may not be in actual contact with the environment at the sampled configuration is intentional, and we believe it drives learning of contact stabilization.

Defining a Target State: In the guided case, we determine the distance from each configuration along the CMGMP trajectory to the configuration in the sampled state. Then, we find the state which minimizes these distances to use as the start of $\mathbf{q}^o_d$. In the unguided case, $\mathbf{q}^o_d$ is set to the goal position of the box. This definition can also be thought of as labeling the state.

Generating Data D for a Training Epoch: Within a training epoch, we first generate a mini-batch B of data, of size $|B|$, using one of two methods:

(1) Sample using Algorithm 1 $|B|$ times.
(2) Sample once using Algorithm 1, and then use a modified DAgger algorithm (Sirichotiyakul and Satici

(2023)) to obtain $|B|$ states, discarding the initial state.

The training loop, detailed in Algorithm 2, then simulates the object-manipulator system using the current policy for N time steps. The data D comprises these $|B|$ trajectories, so that it has size $|D|=|B|\times N$.

In a training epoch, we randomly choose between the two methods of generating B, using a Bernoulli distribution with parameter ϵ.

Sampling Using DAgger Throughout the learning process, we utilize the modified DAgger sampling strategy proposed by Sirichotiyakul and Satici (2023) to obtain a batch B of initial states from the state-space $\mathcal{X}$. DAgger is used to sample states along a trajectory from a random initial state given the current control policy. In order to prevent B from being "poisoned," which we define to be B being either partially or entirely consistent of states outside of G, a trimming function is used before including states from the observed trajectory in the dataset D. The trimming function, explicitly provided in Algorithm 3, excludes all samples from a trajectory after the first time at which the projection of the trajectory under $\mathcal{P}_z$ in (2) exits G. This trimming function effectively mitigates the risk of dataset poisoning, and allows the learning process to converge more reliably.

4.4 Training Algorithm

We combine the steps above into Algorithm 2 which describes a single update step (training epoch). Full training involves executing a selected number of epochs.

5. CASE STUDY/RESULTS

We model a simplified manipulation scenario where a manipulator must push a box up a step, as depicted in Figure 1. The simplification consists of restricting the motion to remain in a vertical plane. The motion of the box and manipulator are both restricted to translation and rotation in the plane.

CMGMP Modifications. We use a modified version of the CMGMP algorithm presented in Cheng et al. (2022). There are three differences from the available implementation of CMGMP. First, we define costs based only on position error of the manipulated object; arbitrary orientations incur no cost. The reason is that reaching a desired orientation can be achieved in the controller synthesis stage instead, and removing it from the planning stage reduces the number of samples that CMGMP needs to collect to find a trajectory. Second, the time resolution between points on the CMGMP trajectory was increased to facilitate training of the controller in Section 4.4.

Simulator Modifications. The MJX environment inherently supports 3D environments, but allows for applying these motion constraints by adding degrees of freedom to bodies; if a degree of freedom is not added, motion along that degree of freedom is not allowed. We use this mechanism to achieve constrained motion. Instead of creating a custom CMGMP implementation to match the planar scenario modeled here, we add costs that penalize

Algorithm 2. Training epoch

```
Input: Mini-batch size $|B|$, simulation steps $N$, learning
       rate $\eta$, sampling parameter $\epsilon \in [0,1]$
Input: Loss function $\mathcal{L}(\phi)$, control policy $P(\boldsymbol{x};\phi)$
$B \leftarrow \emptyset$ // Initialize the batch
$y \sim U([0,1])$
if $y > \epsilon$ then
    foreach $i = \{1,2,\dots,|B|\}$ do
        $\boldsymbol{x} \leftarrow$ Algorithm 1;
        $B$.append($\boldsymbol{x}$)
else
    $\boldsymbol{x} \leftarrow$ Algorithm 1
    $\gamma \leftarrow$ sample trajectory starting at $\boldsymbol{x}$ via DAgger, with
      length $|B|$
    $B$.append($\gamma$)
$L \leftarrow \emptyset$ // Initialize the losses
$G \leftarrow \emptyset$ // Initialize the gradients
$i \leftarrow 1$
foreach $s \in B$ do
    $\boldsymbol{x} \leftarrow \boldsymbol{s}$
    $\gamma \leftarrow \emptyset$
    for $1 \le n \le N$ do
        integration-step($\boldsymbol{x}$) with controller $P(\boldsymbol{x};\phi)$
        $\gamma$.append($\boldsymbol{x}$)
    $L_i \leftarrow \mathcal{L}(\gamma)$
    $G_i \leftarrow \frac{\partial L_s}{\partial \phi}$
    $i \leftarrow i+1$
$\phi \leftarrow \phi - \frac{\eta}{|B|}\sum_{i=1}^{|B|} G_i$
```

deviation from a vertical plane, namely a cost of the form wy_o^2 where w is a large weight.

5.1 Algorithmic Details

State. We parametrize the configuration space projected onto the inertial $x-z-$axis by $\boldsymbol{q} = (x_o, z_o, \theta_o, x_m, z_m, \theta_m) \in \mathrm{SE}(2)^2$, which stand for the $x-z-$coordinates of the manipulator and the box with respect to the inertial coordinate system $\mathcal{W}$ and their rotation angles about the inertial $y-$axis.

Neural Network. The neural network has the architecture described in Table 2. The neural network was initialized with random weights and biases, which are then scaled by some user-settable parameter. This was motivated by the need for the network to output relatively large forces in order to move the manipulator and box, and was treated as a hyperparameter for training.

Loss Function. The loss function is also specialized to manipulation in the plane and the weights in equation (1) are selected as $w_{og} = 30$, $w_{ov} = 1/2$, $w_{mv} = 1/2$, and $w_{mo} = 1$ for unguided training and $w_{oo^*} = 25$, $w_{mm^*} = 2$, $w_{ov} = 1/8$, $w_{mv} = 0$, and $w_{mo} = 0$ for the guided training.

Partitioning G. We seed the training using two discrete subsets of G, G_1 and G_2. We partition G such that G_1 is the region on the ground before the step fixture and G_2 is the region on the step fixture. This method samples the state space near the target, and near the start of the step. This enables the training of the controller to more quickly

Algorithm 3. Trimming function used in DAgger

```
Input: Observed trajectory $\gamma$, Sampling region $G$
Input: Gap function $g$ between manipulator and the box
$c_1 = \arg\min x_o \notin G$
$c_2 = \arg\min \dot{x}_o > 4$
$c_3 = \arg\min g((x_o, z_o), (x_m, z_m)) \geq 0.1$
$c = \text{minimum}(\{c_1, c_2, c_3\})$
return $\gamma[1 : c]$
```

converge to a desirable solution than using DAgger alone. These two sampling areas were chosen for the following reasons:

(1) **Near the goal position:** This was chosen so that the controller learns to control the manipulator such that the box slows down due to friction, as the manipulator cannot slow down the box through means such as grasping or pushing to oppose motion beyond the goal.
(2) **Near the start of the ramp or step:** This was chosen so that the controller would learn to push with enough force to start moving the box in order to reach the step.

DAgger sampling is performed for 62.5% of epochs as described in section 4.3.4. For the remaining 37.5% of epochs, we equally split sampling between G_1 and G_2 such that B consists only of states from one set in a given epoch.

Allowable Contacts and Orientations. The set of allowed sample orientations $\mathcal{R}(\mathbf{p}^o)$ is $\{0°, 90°\}$. When the box is not on the step, the possible contacts lie along the vertical side away from the step, at heights in the interval $(z_o - 0.2, z_o + 0.2)$, with a gap from the face uniformly sampled from the interval $[0, 0.1]$. If the box is on the step, then either the manipulator makes contact on the side of the box closer to $x = 0$ using the same procedure above, or it nearly makes contact on the top face of the box at a location x_m along the x-axis uniformly sampled from the interval $(x_o - 0.2, x_o + 0.2)$, with a gap from the face uniformly sampled from the interval $[0, 0.1]$.

Training vs. Validation. For training, we consider states in both G_1 and G_2 as valid initial states in B. For validation, only G_1 is considered for initial states.

5.2 Results

Figure 2 graphically depicts typical final box positions achieved by trained policies under the guided (orange boxes) and unguided trainings (blue boxes). It also depicts the final position of the box when using position control for the end-effector with desired states from the CMGMP motion plan. The guided training is able to achieve the task, manipulating the box to its desired state (green wireframe) from the initial state (red wireframe). On the other hand, the unguided training almost always gets stuck in local minima, preventing convergence to the desired state. This behavior is observed to be independent of learning rate. Table 3 provides the average error in position under the three different strategies. The CMGMP trajectory assumes that the manipulator instantaneously resets its contact location when needed; without manual relocation of the end-effector, this approach fails to achieve the task.

Table 2. The policy's multilayer perceptron architecture (both guided and unguided).

Layer Type	*# Neurons*	*# Parameters*	*Activation*
Input	12	N/A	N/A
Hidden	32	416	Elu
Hidden	16	528	Elu
Hidden	8	136	Elu
Output	3	27	Linear
Tally:	71	1107	

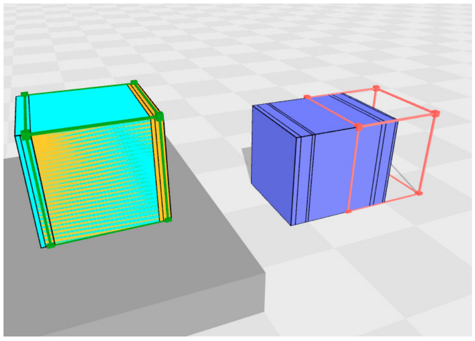

Fig. 2. Comparison of the controllers trained with and without guidance from a motion plan. *Red* wireframe is the initial box position. *Green* wireframe is the goal position. *Blue* boxes are states converged using the controller trained without guidance. *Orange* boxes are states converged using the controller training with guidance. The *cyan* box is the state converged by using position control of the end-effector with targets from the CMGMP trajectory. The end-effector was removed for clarity.

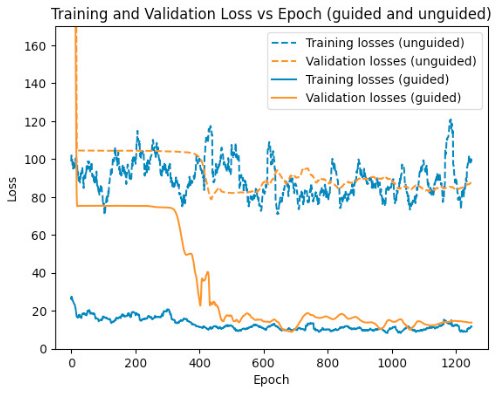

Fig. 3. Training and validation losses for a training with and without guidance from a motion plan. An averaging window of width 25 was used to smooth the plots.

Table 3. The final position error under various policies. (**EE**: End-effector)

	CMGMP		Unguided	Guided
	EE relocation	No relocation		
Loss	0.0707	0.8945	2.5306	0.0305

Figure 3 shows the training and validation loss curves for the unguided and guided cases respectively. The higher final losses for the unguided case reflect the inability to reach the step.

Figure 4 graphically depicts a time-lapse sequence showing the box's transition from its initial position to its goal position using a controller trained with guidance from a motion plan. This visualization demonstrates the controller's effectiveness in executing manipulation to achieve the desired final state.

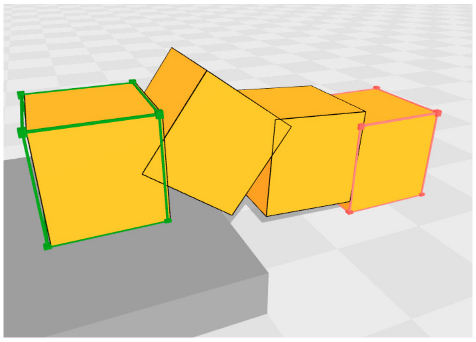

Fig. 4. Depiction of multiple states from the resulting trajectory using a controller trained with guidance. The end-effector was removed for clarity.

6. CONCLUSION

This paper presents a guided policy search algorithm for learning manipulation controllers. The main idea is to learn a controller based on trying to stabilize a contact-rich manipulation plan found using simplified dynamics assumptions.

Through a non-prehensile manipulation task involving pushing a box over a step, we show our guided policy search algorithm outperforms basic policy search. One interpretation is that the motion plan helps solve the issue of state exploration that plagues typical policy gradient algorithms, an issue that is exacerbated by the presence of multiple contact modes.

This work presented a proof-of-concept data-driven manipulation strategy that robustly stabilizes a specific manipulation task. Extending the algorithm to more general classes of manipulation will be the subject of future work.

7. ACKNOWLEDGEMENTS

The authors thank Benton Clark for assistance in generating the CMGMP trajectory used in the case study.

REFERENCES

Andrychowicz, O.M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. (2020). Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1), 3–20.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M.J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs. URL `http://github.com/google/jax`.

Bullock, I.M. and Dollar, A.M. (2011). Classifying human manipulation behavior. In *2011 IEEE international conference on rehabilitation robotics*, 1–6. IEEE.

Cheng, X., Huang, E., Hou, Y., and Mason, M.T. (2022). Contact mode guided motion planning for quasidynamic dexterous manipulation in 3d. In *2022 International Conference on Robotics and Automation (ICRA)*, 2730–2736. IEEE.

Deits, R., Koolen, T., and Tedrake, R. (2019). Lvis: Learning from value function intervals for contact-aware robot controllers. In *2019 International Conference on Robotics and Automation (ICRA)*, 7762–7768. IEEE.

Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S., et al. (2017). Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*.

Hogan, F.R. and Rodriguez, A. (2020). Reactive planar non-prehensile manipulation with hybrid model predictive control. *The International Journal of Robotics Research*, 39(7), 755–773.

Huang, E., Cheng, X., and Mason, M.T. (2020). Efficient contact mode enumeration in 3d.

Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7), 846–894. doi:10.48550/ARXIV.1105.1186. URL `https://arxiv.org/abs/1105.1186`.

Levine, S. and Koltun, V. (2013). Guided policy search. In S. Dasgupta and D. McAllester (eds.), *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, 1–9. PMLR, Atlanta, Georgia, USA.

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

Murray, R.M., Li, Z., and Sastry, S.S. (2017). *A mathematical introduction to robotic manipulation*. CRC press.

Ross, S. (2011). *No-Regret Methods for Learning Sequential Predictions*. Ph.D. thesis, Citeseer.

Ross, S., Gordon, G., and Bagnell, D. (2011). 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*, 627–635. JMLR Workshop and Conference Proceedings.

Sirichotiyakul, W. and Satici, A.C. (2023). Data-driven passivity-based control of underactuated mechanical systems via interconnection and damping assignment. *International Journal of Control*, 96(6), 1448–1456.

Sutton, R.S. and Barto, A.G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R.S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller (eds.), *Advances in Neural Information Processing Systems*, volume 12. MIT Press.

Tian, S., Ebert, F., Jayaraman, D., Mudigonda, M., Finn, C., Calandra, R., and Levine, S. (2019). Manipulation by feel: Touch-based control with deep predictive models. In *2019 International Conference on Robotics and Automation (ICRA)*, 818–824.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. IEEE.