Meta-Policy Learning over Plan Ensembles for Robust Articulated Object Manipulation

Constantinos Chamzas
Rice University
Houston, USA
cchamzas@rice.edu

Caelan Garrett
NVIDIA Research
Seattle, USA
cgarrett@nvidia.com

Balakumar Sundaralingam
NVIDIA Research
Seattle, USA
balakumars@nvidia.com

Lydia E. Kavraki
Rice University
Houston, USA
kavraki@rice.edu

Dieter Fox
NVIDIA Research
Seattle, USA
dieterf@nvidia.com

I. Introduction

Model-based robotic planning techniques, such as inverse kinematics and motion planning, can endow robots with the ability to perform complex manipulation tasks, such as grasping, object manipulation, and precise placement. However, these methods often assume perfect world knowledge and leverage approximate world models. For example, tasks that involve dynamics such as pushing or pouring are difficult to address with model-based techniques [1] as it is difficult to obtain accurate characterizations of these object dynamics. Additionally, uncertainty in perception prevents them populating an accurate world state estimate.

Recent works have shown that integrating learning with model-based planning can address some of these limitations [2, 3, 4]. Following a similar direction, we integrate a geometric model-based motion planner with learning in a mixture-of-experts fashion to solve manipulation problems where the world model and dynamics are only approximately known.

Specifically, we propose using a model-based motion planner to build an *ensemble of plans* under different environment hypotheses. Then, we train a *meta-policy* to decide online which plan to track based on the current history of observations. By leveraging history, this policy is able to switch ensemble plans to circumvent getting “stuck” in order to complete the task. We tested our method on a 7-DOF Franka-Emika robot pushing a cabinet door in simulation, as shown in Fig. 1. We demonstrate that a successful meta-policy can be trained to push a door in settings high environment uncertainty, all while requiring little data ($\leq$ 1000 episodes).

The main contributions of this work are:

1) An ensemble approach for manipulation policies, which outperforms single-trajectory planners.
2) Learning a meta-policy to select among the ensemble.
3) Simulated experiments that show our method results in 40% higher success rate than the non-learning baseline.

II. Related Work

At the core of model-based planning techniques for robot manipulation lies motion planning [5]. Such techniques include constrained motion planning [6], Multi-Modal Motion Planning [7] (MMMP), and Task and Motion Planning (TAMP) [8]. The main advantages of these methods are that they are general, *i.e.* work with any geometries, and have theoretical guarantees with respect to their models, *e.g.* probabilistic completeness [9]. However, they depend critically on the fidelity of their models, limiting their effectiveness when there are state estimation errors and unmodeled constraints, *e.g.* dynamic constraints such as limited pushing force or friction between objects and the environment. Finally, these methods often have long (several seconds) planning times, making it difficult for them to react online in real-time to unexpected events.

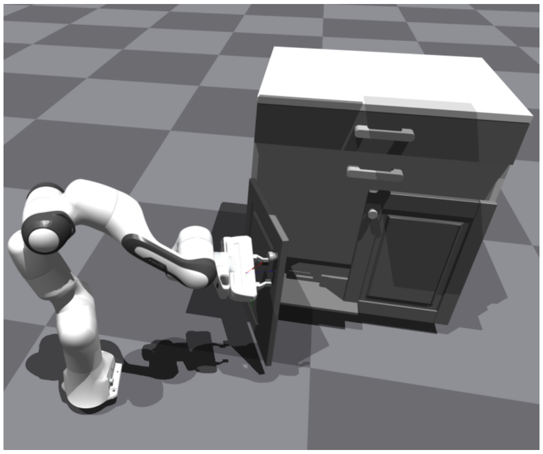

Fig. 1. Example of the panda robot pushing the left door of the cabinet. In this setting, the robot knows its own state and geometry along with the state and geometry of the cabinet. However, the position of the cabinet is only approximately estimated, and the dynamic properties of the objects are unknown. In this work, we attempt to solve this task by combining a model-based planner and learning from experience.

Recent advances in learning have prompted researchers to combine learning with model-based planning to address these limitations. Some techniques try to learn world models [10, 11, 12, 13], warm start policy-learning with model-based techniques [14, 15], or improve planning efficiency [3, 16]. We also leverage learning to improve the downstream performance of a model-based planner. However, we focus on approximate world models with uncertain object poses and geometries and problems that involve dynamics such as pushing.

III. Methodology

In this work, we aim to endow a robotic agent with advanced manipulation skills regarding grasping, pushing, and even throwing objects under realistic assumptions. The realistic assumptions that we consider are that a world model can be engineered but is not entirely accurate. These inaccuracies could be due to pose uncertainty, lack of dynamic modeling, or approximate geometric representations. Our method uses a model-based planner to produce candidate plans using the approximate world model and then uses learning from experience to learn a meta-policy that improves task performance.

We will use the toy problem shown in Fig. 2 **a**) to demonstrate the key concepts in this work. In this toy problem, the robot aims to push the red object to the shaded red goal location. The modeled constraints include collision avoidance and joint limits. Unknown aspects include dynamics, namely how the object moves upon contact with the end-effector. Task failure can occur if the object falls over due to the pushing angle. Uncertainty exists in the object's x-axis position estimation and its exact geometry.

The world state is the prorpioceptive state of the robot along with the potentially uncertain state of the world objects. For example, in the problem shown in Fig. 2 **a**), the state includes the positions of the robot's five joints and an approximate estimate of the position and geometry of the red object along the x-axis. We consider such state estimation realistic as robots usually have accurate joint encoder measurements but perceive the objects in the world around them with some uncertainty.

The proposed approach is composed of two main components. First, a geometric model-based planner creates an ensemble of plans as described in subsection III-A and shown in Fig. 2 **b**). Second, given the currently observed state, we learn a meta-policy that chooses plans from the ensemble as described in subsection IV-C and shown in Fig. 2 **b**) (the large black arrows). The proposed hybrid approach leverages the approximate world model to create plan-ensembles that satisfy known constraints, such as geometric-constraints. Then we use learning to account for unmodeled constraints, *e.g.* dynamics, and be robust to estimation errors.

Algorithm 1 Plan Ensemble Pseudocode

```
1:  procedure PLAN-ENSEMBLE($G$, $\Pi_p$, $K$)
2:      $\Pi_p \leftarrow \emptyset$                                ▷ Plan-Ensembles
3:      while $i \leq N$ do                             ▷ Create Plan-Ensembles
4:          $\hat{s} \leftarrow$ SAMPLE-WORLD()
5:          $p \leftarrow$ PLAN($\hat{s}, G$)                   ▷ Plan Geometric Path
6:          $\pi_p \leftarrow$ MAKE-POLICY($p$)                       ▷ Create Plan
7:          $\Pi_p \leftarrow \Pi_p \cup \pi_p$
8:      while $t \leq T$ do                          ▷ Iteratively Reselect Plan
9:          $o \leftarrow$ OBSERVE()
10:         $\pi_p^{chosen} \leftarrow \operatorname{argmax}_{\pi_p \in \Pi_p}$ BEST($o, \pi_p$)
11:         while PROGRESS($o, p$) do
12:             $o \leftarrow$ OBSERVE()                          ▷ Observe State
13:             if $o \in G$ then
14:                 return True                                  ▷ Success!
15:             $r \leftarrow$ EXECUTE($\pi_p^{chosen}$)
16:     return False                                             ▷ Failure
```

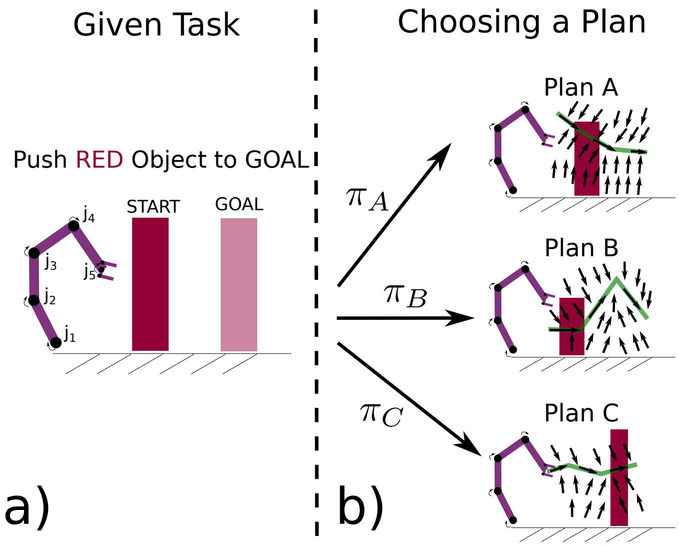

Fig. 2. **a)** An illustrative example of a task that combines geometric, dynamic reasoning and uncertainty. The robot is tasked with pushing the red object to the goal location, without toppling it over. The robot can control its 5 joints and estimate within some error the position and geometric height of the red object. **b)** The proposed method uses model-based planning to create several path policies. The path shown with green is produced with a model-based geometric planner and satisfies the geometric (known) constraints of the problem. The black arrows represent the plan, which is the combination of the path and a position controller. Note that the plan is shown in the end-effector 2D space but its state space is the 5D space of the joints of the robot.

A. *Create Plan-Ensembles*

Creating plan-ensembles is described in Line 3 to Line 7 of Alg. 1. In Line 4, SAMPLE-WORLD() samples a possible world state according to a given uncertainty distribution. In Line 5, given the currently estimated state of the world $\hat{s}$ and a goal G, a model-based planner computes a candidate path p. A path p is as sequence of waypoints where each waypoint specifies the position of the robot and other known movable objects in the world. The first waypoint is the current state, while the last waypoint must satisfy the goal specification. Examples of paths are the green lines shown in Fig. 2 **b**).

In Line 6, the path p is converted into a plan π_p. The path is time-parameterized and a waypoint-tracking trajectory controller is used to follow it. In this context, a plan is converted into an atomic policy that provides an action for any state. Example plan actions are shown as the black arrows that attract towards the green path p in Fig. 2 **b**).

Note that each plan is computed for an estimation of the world state $\hat{s}$. This is illustrated in Fig. 2 **b**), where the objects for each computed path policy are either in different locations (inaccurate pose estimation) or have different geometry (inaccurate geometry estimation). Finally, in Line 7, the plan is added in the plan-ensemble Π_p.

B. *Selecting Plans*

Selecting and executing a plan is described in Line 8 to Line 15 of Alg. 1. Given a plan-ensemble Π_p, we need to choose the most appropriate plan to follow for the current observed state o. In this context, the observed state includes

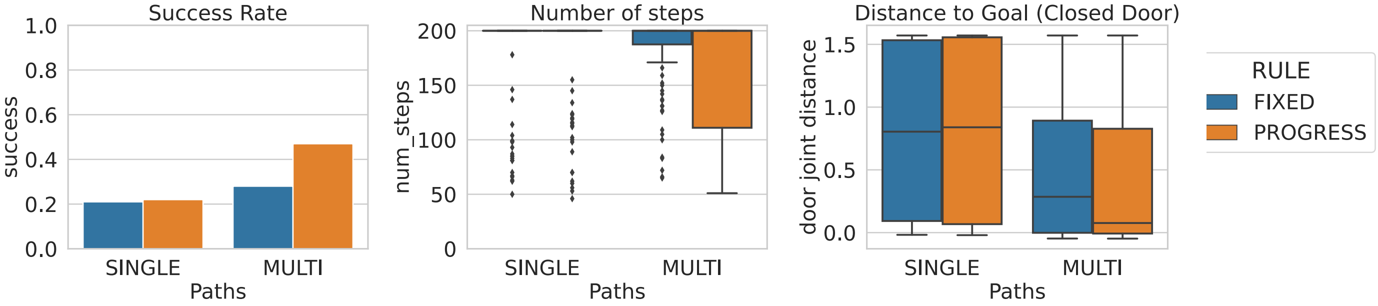

Fig. 3. *Success Rate*, *Number of Steps*, and *Distance to Goal* different configurations of method. All configurations were tested on a 100 different environments, with a maximum of 200 timesteps. SINGLE refers to using just a single path policy while MULTI means that 10 path-policies were used for the path-policy set Π_p respectively. FIXED/PROGRESS denotes whether a fixed interval or the **PROGRESS** rule was used to choose a new path-policy.

the same information as each waypoint of the path. We choose a simple supervised approach to learn how to select a plan.

In Line 9, the known state of the robot and the world is observed, which can be used to determine if we have reached a terminal state. An episode terminates when the goal is reached or when maximum number of steps has passed. An important implementation choice is the decision-frequency at which a new plan is chosen. One end of the spectrum would be choosing a single plan and following it until the end of the episode. The opposing end would be choosing a new plan at every timestep. In this work, we propose using a simple rule that leverages the computed path p to decide when to switch. Taking advantage of this knowledge, in Line 11, the **PROGRESS** function monitors if progress towards the planned path p is made and returns *False* if no progress has been made in the last k timesteps. When this function returns *False*, a new plan is chosen, as shown in Line 10.

To generate training data, we first create the plan-ensemble Π_p and then execute plans π_p randomly. After execution, we collect the state, action, and progress triplet and then store it to later train the network **BEST**. We define the distance to goal that was covered by the execution of the plan as progress. We use the collected data to train the network in supervised manner. The network learns to predict the progress that will be achieved, if we execute the plan π_p^{chosen} from the current observed state o. During testing, we use the trained network **BEST** Line 10 to choose the most promising plan and execute it. Execution continues until the goal is reached or the **PROGRESS** returns *False*.

IV. Experiments

We applied our framework to a pushing a cabinet door problem, as shown in Fig. 1. The robot task is to push the left cabinet door from the open state to the closed state.

A. Experiment Setup

The initial state uncertainty we consider lies in the position of the cabinet relative to the robot. Thus, the SAMPLEWORLD() function samples a random position of the cabinet relative to the robot according to the given uncertainty distribution. Here, we only consider translational uncertainty.

Given the random sample from this distribution $\hat{s}$, the planner produces a geometric path p that maintains contact perpendicular to the door and intends to push it until it closes. The pushing contact location on the cabinet door is chosen randomly; different contact points are sampled for each computed path. We use TRAC-IK [17] combined with a PID controller to convert the geometric path p to a policy which we refer to as a plan π_p. To represent the geometric world and generate the plan-ensembles Π_p, we use PyBullet [18]. We use IsaacGym [19] for physics and control simulation.

We conducted 2 experiments to examine the performance of our proposed method. The first experiment(subsection IV-B), studies the choices of the switching frequency rule and the use of ensembles. Specifically, we investigate the effect of the **PROGRESS** rule, and choosing to use a plan-ensemble instead of a single plan . The second experiment (subsection IV-C), benchmarks the proposed method with respect to a non-learning baseline and ablates different input features to the **BEST** neural network. We evaluated with these metrics:

- **Distance to goal (Progress)**: This simply measures how much the door closed. A distance of 0 means that the door closed (*i.e.* the goal was reached), and a distance of $\pi/2$ means that the door did not move at all.
- **Success rate**: The percentage of tasks solved successfully within the given horizon. The task is considered successful if the door is closed up to some error threshold.
- **Number of steps**: The number of timesteps to complete the episode. The episode terminates if the door is closed or if the maximum of 200 timesteps is reached.

B. Using Plan Ensembles and Progress Rule

Here, the uncertainty was emulated as a uniform distribution ±10 cm on the position of the cabinet and applied across x,y,z dimensions independently. We tested the following:

1) SINGLE/MULTI paths: SINGLE means that only one plan was used, while MULTI means that we had a plan-ensemble of 10 plans. The plan to follow at each timestep was chosen randomly. This comparison motivates why using multiple plans(an ensemble) is better than a single one.
2) FIXED/PROGRESS rule: FIXED means that the decision to switch plan was every 10 fixed timesteps, and PROGRESS means that the rule described in subsection IV-C was used to determine the switching frequency. In both cases,

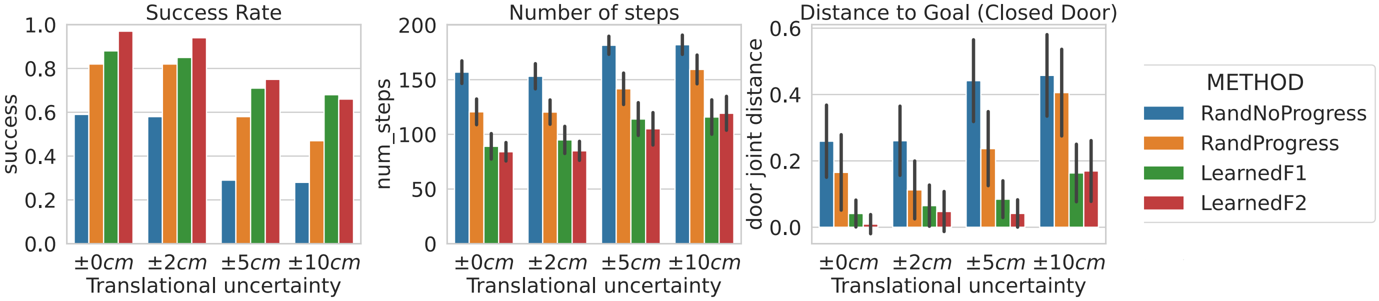

Fig. 4. The 1) Success Rate, 2) Number of Steps, and 3) Distance to Goal for both the two random baselines, with and without the **PROGRESS** (RandNoProgress/RandProgress) rule, as well as the two different feature sets (LearnedF1/LearnedF2) used for training the **BEST** network. The x-axis denotes the translational estimation error on the cabinet position. The error bars indicate 0.25 of the standard deviation.

the new plan π_p is chosen randomly. For SINGLE, FIXED has no effect, while PROGRESS added noise ± 10cm in the end-effector space to help unstuck the robot.

The results demonstrate that the **PROGRESS** rule and using an ensemble significantly improves performance. Intuitively as long as progress is made along the path, the current plan is good and it should not be switched. The advantage of using an ensemble of plans lies in the fact that part of one plan could apply for part of the episode and part of another plan can apply to another part of the episode. Thus an ensemble can complete tasks that a single plan could never complete. The MULTI/PROGRESS combination achieves both the best distance to goal and also requires the least number of steps to complete the task. The main takeaways from these results are first that the **PROGRESS** rule helps significantly in all uncertainty settings.

C. Learning a Meta Policy

In this experiment, we investigated the improvements in task performance of learning the subroutine **BEST**. First, we use a random strategy to choose and execute path-policies for the pushing task in Fig. 1 using the same setting as the experiment setup of subsection IV-B with MULTI paths and the **PROGRESS** rule enabled. We collected data from 1000 episodes.

We trained a regressor neural network **BEST**, which takes as input the plan π_p as well as the current observation o and possibly the history of observations and predicts the progress toward the goal after executing π_p. The neural network architecture has three fully connected layers with batch normalization[20]. Each hidden layer has 64 units, and the network was trained to minimize Mean Squared Error (MSE).

An important decision is how to represent the path and history of observations as the input to the neural network. We use the following values are input features:

- *Current End-Effector(EE) State*: the position, expressed in Cartesian coordinates (3 dim) and the orientation expressed as a quaternion (4 dim).
- *Current Robot State*: joint values of the robot (7 dim).
- *Current Door State*: the joint value of the door (1 dim).
- *Final End-Effector State* (7 dim).
- *Next Immediate End-Effector State* (7 dim).
- *Midpoint End-effector State*: the midpoint between Current and Final End-Effector States (7 dim), using spherical linear orientation interpolation.

The observed state of the world includes the current end effector state, robot joint state, and door state (15 dim). The first set of features $F1$ includes the observation, the next end-effector goal, and the end effector goal (29 dim). The second set of features $F2$ extends $F1$ by adding the midpoint end-effector state, the difference between the current end-effector state and the next end-effector state, and the difference between the current end-effector state and the final end-effector state. Additionally, the last 5 observations are included (175 dim).

We trained the neural network with the same data from ± 10 cm uncertainty and tested it in 100 environments under $\pm 0, \pm 2, \pm 5, \pm 10$ cm of translational uncertainty. We also tested a random strategy similar to the previous section when using and not using the progress rule. In Fig. 4, the success rate, number of steps included, and distance to goal are shown for the four uncertainty settings.

The learned strategies are better than the random strategy, and the F2 set of features seems to help best in the lower uncertainty regime. We hypothesize that the ± 10 cm regime is too uncertain to leverage the observation history or potentially over-fits, while in the lower regime the observation history can help disambiguate the error in position estimation. Overall, incorporating the observation history helps, and it is possible to learn the meta policy over the plan-ensemble. A video demonstration of the experiments is available [1].

V. Conclusion

In this work, we proposed a method for manipulation that learns a meta policy over plan-ensembles using approximate world models. In our preliminary results, we have demonstrated the efficacy of the proposed method over simple baselines in a simulated environment where the robot is tasked with pushing the door of cabinet. The two main assumptions of the proposed method are that an approximate model of the world is given and a quantification of the uncertainty of the model is also available. In the future, we would like to apply the proposed method to a real-world setting and investigate the feasibility of these assumptions. We would also like to investigate extending to more manipulation skills, multi-modal planning, and image-based predictions.

[1] https://www.dropbox.com/s/8gosmt9e838icqg/RSS2023-LTAMP.mp4?dl=0

References

[1] Kejia Ren, Lydia E. Kavraki, and Kaiyu Hang. Rearrangement-based manipulation via kinodynamic planning and dynamic planning horizons. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1145–1152, 2022. doi: 10.1109/IROS47612.2022.9981599.

[2] Joaquim Ortiz-Haro, Jung-Su Ha, Danny Driess, and Marc Toussaint. Structured deep generative models for sampling on constraint manifolds in sequential manipulation. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 213–223. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/ortiz-haro22a.html.

[3] Zhutian Yang, Caelan Reed Garrett, and Dieter Fox. Sequence-based plan feasibility prediction for efficient task and motion planning. *arXiv preprint arXiv:2211.01576*, 2022.

[4] Christopher Agia, Toki Migimatsu, Jiajun Wu, and Jeannette Bohg. Stap: Sequencing task-agnostic policies. *arXiv preprint arXiv:2210.12250*, 2022.

[5] Howie Choset, Kevin M Lynch, Seth Hutchinson, George A Kantor, and Wolfram Burgard. *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[6] Zachary Kingston, Mark Moll, and Lydia E. Kavraki. Sampling-based methods for motion planning with constraints. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):159–185, 2018. doi: 10.1146/annurev-control-060117-105226. URL https://doi.org/10.1146/annurev-control-060117-105226.

[7] Kris Hauser and Jean-Claude Latombe. Multi-modal motion planning in non-expansive spaces. *The International Journal of Robotics Research*, 29(7):897–915, 2010.

[8] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[9] J-C Latombe. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[10] Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomas Lozano-Perez, Leslie Pack Kaelbling, and Joshua Tenenbaum. Inventing relational state and action abstractions for effective and efficient bilevel planning. *arXiv preprint arXiv:2203.09634*, 2022.

[11] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[12] Philipp Wu, Alejandro Escontrela, Danijar Hafner, Pieter Abbeel, and Ken Goldberg. Daydreamer: World models for physical robot learning. In *Conference on Robot Learning*, pages 2226–2240. PMLR, 2023.

[13] Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning compositional models of robot skills for task and motion planning. *The International Journal of Robotics Research*, 40(6-7):866–894, 2021.

[14] Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

[15] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE, 2019.

[16] Constantinos Chamzas, Aedan Cullen, Anshumali Shrivastava, and Lydia E Kavraki. Learning to retrieve relevant experiences for motion planning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 7233–7240. IEEE, 2022.

[17] Patrick Beeson and Barrett Ames. Trac-ik: An open-source library for improved solving of generic inverse kinematics, 2015.

[18] Erwin Coumans. Bullet physics simulation. In *ACM SIGGRAPH 2015 Courses*, page 7. ACM, 2015.

[19] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.