Learning a Value Function Based Heuristic for Physics Based Manipulation Planning in Clutter

Wissam Bejjani, Rafael Papallas, Matteo Leonetti and Mehmet R. Dogar

***Abstract*— In this work, we propose interleaving planning and execution, in a closed-loop setting, using a Receding Horizon Planner (RHP) for pushing manipulation in clutter. In this context, we address the problem of finding a suitable value function based heuristic for planning in near real-time, and for estimating the reward from the horizon to the goal. We estimate such a value function first by using plans generated by an existing sampling-based planner. Then, we further optimize the value function through reinforcement learning.**

I. Introduction

We propose learning a heuristic for planning a manipulation task in clutter. The task is to push an object to a goal region, with little to no repositioning of other objects. Such robotic manipulation skills are needed in a variety of applications such as [1], [2], [3]. This requires pushing certain items out of the way, without dropping them off the shelf, while reaching for a target item.

Impressive planners have been proposed for pushing-based motion planning in clutter [4], [5], [6]. Real-world execution of these trajectories, however, still poses great challenges. The main difficulty is due to the inevitable inaccuracy in the physics model used by the planners. This inaccuracy is emphasized particularly when multiple objects are in contact.

We present an example of the task in Fig. 1, where the green object has to be pushed to a target region (the green region) while keeping the red objects close to their original positions (red regions). The top row of Fig. 1 shows the execution of an open-loop trajectory generated by a sampling-based planner, while the bottom row presents an execution of our system. The overlaid animated figures (on the top-right corner of the images) show the planner's prediction of how the objects should move during interaction. When planned trajectories are executed open-loop, the real motion of the objects can differ significantly from the motion predicted by the planner. For this reason, in the shown example, the open-loop controller fails to accomplish the task.

A solution to this problem is to interleave planning and execution. In this approach, a sequence of actions is planned, but only the first action in this sequence is executed. Then, the current state is updated by observing the environment, after which another sequence of actions is planned, and the routine is repeated. We show an execution of such a control scheme in the bottom row of Fig. 1. Even if objects move differently than predicted, the controller has the opportunity to correct for it.

Authors are with the School of Computing, University of Leeds, United Kingdom {w.bejjani, r.papallas, m.leonetti, m.r.dogar}@leeds.ac.uk

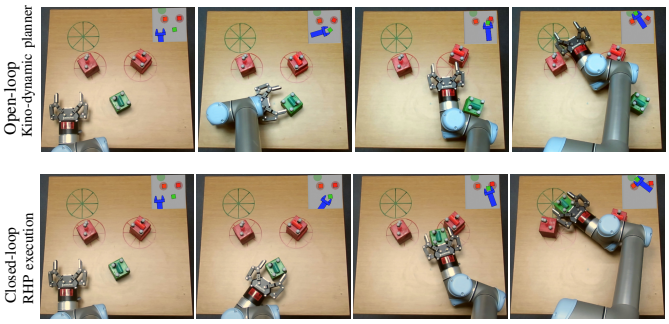

Fig. 1: Top: robot failing to push the green box to the goal region by following a precomputed plan using kino-dynamic planning. Bottom: robot successfully executing the task using closed-loop RHP execution.

One possible implementation is to run one of the aforementioned planners at every step of the execution, to generate the new sequence of actions. The computation time these planners require, however, is prohibitively high, typically taking from tens of seconds to minutes for one plan [7], [4], [5], [6], [2], [8], [9]. In contrast, we are interested in real-time execution, which requires a planner that can quickly suggest an action for the current state of the world.

To generate plans quickly, we propose planning only a small sequence of actions in to the future. We run a *Receding Horizon Planner* (RHP) with a short horizon h, and take advantage of an appropriate proxy function for the *value* of the states beyond the horizon. This value of a state must estimate how rewarding it would be to reach the goal from that state. In this work, we use RHP as the robot control policy. At every step, RHP generates a solution of the form:

$$\langle a_0, \ldots, a_{h-1} \rangle = \underset{\langle a_0, \ldots, a_{h-1} \rangle}{\arg\max} \sum_{k=0}^{h-1} \gamma^k r_{k+1} + \gamma^h \max_a q(s_h, a) \tag{1}$$

where γ is the discount factor, r is the immediate reward, and q is the value of a state-action pair, that is, the expected reward for reaching the goal starting from that state and using that action. We avoid hand-crafted reward functions and set $r = -1$ for every state transition.

In domains where multiple physics-based object-to-object contact is possible, acquiring the value function is a challenge on its own, let alone the optimal one. The horizon can mitigate the inaccuracy of the value function estimate, by ranging from infinity, with the robot planning all the way to the goal, to zero, with the robot acting greedily with the

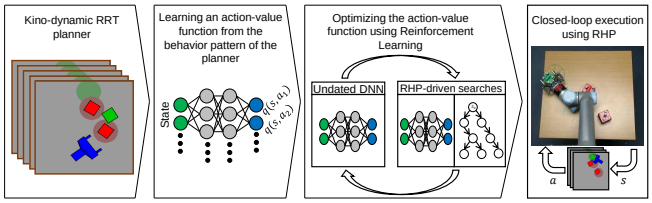

Fig. 2: Overview of the proposed approach.

respect to the value function. With an infinite horizon the value function is ignored, while with $h = 0$ the behavior depends entirely on the value function. In the latter case, if the value function is optimal (that is, $q(s, a) \geq q^{\pi}(s, a)\ \forall s \in S, a \in A$, for any policy π), so is the resulting policy. In practice, a short but non-zero horizon takes advantage of both the planner and the value function without relying on either one entirely, and we experiment with several values for h.

The focus of this work is on learning, with no prior knowledge, a value function that we can use as a heuristic within RHP. In order to do this, we propose a two step data-driven approach: 1) We extract a value function from examples of problem instances[1] solved by existing sampling-based planners, 2) then, we use a reinforcement learning (RL) algorithm to gradually update the value function from estimating the value based on the behavior of the sampling-based planner to estimating the actual optimal value of the manipulation task. Fig. 2 shows an overview of this approach.

II. Learning an Action-Value Function from Sampling-Based Planners

A. Generating example plans

Sampling-based planners treat every new planning instance independently from previously solved instances. Also, they must plan until the goal, and they do not offer useful information on the searched areas of the state-state space from which the goal was not reached. However, sampling-based planners provide a probabilistically complete tool to solve complex planning problems in high-dimensional state spaces, without necessarily requiring a hand-crafted or domain-dependent heuristic. We implement a state-of-the-art kino-dynamic planner [4], used for solving physics-based manipulation in clutter planning problems. We generate P random problem instances. Then, for each instance p, we run the kino-dynamic planner to generate a solution of the form of a sequence of actions. Fig. 3 shows an example of a problem instance solved by the kino-dynamic planner.

B. Learning The Action-Value Function From Observed Trajectories

We use the solution plans to the P problem instances, solved by the kino-dynamic planner, to train a deep neural network (DNN) to predict the action-value estimate for a given state-action pair. The DNN represent the action-value function estimate $\hat{q}(s, a; \theta)$ with parameters θ. To train the DNN, we use every state-action pair encountered along every example plan. For each example plan, and for every state-action pair in that plan, we compute the update target:

$$q(s_l^p, a_l^p) = \sum_{k=0}^{L-l-1} \gamma^k r_{l+k+1} = r(\frac{1-\gamma^{L-l}}{1-\gamma}) \qquad (2)$$

where p stands for the index of the plan generated by the kino-dynamic planner and l is the index of the state-action pair in that plan. The second equality takes advantage of the fact that in our formulation all the immediate rewards, denoted as r, are the same[2].

While the DNN trained as above learns to predict the action-values for the actions executed in states along the trajectory generated by the kino-dynamic planner, the values predicted by the DNN for actions that have **not** been used by the planner along these states can be arbitrary. As a result of function approximation, however, these actions will nonetheless have a value. The value can converge to an arbitrary number, determined by the effect of the target value in the states that the planner did traverse. A possible undesirable effect is that the values of the actions not chosen by the planner can be higher than the chosen one. This can later cause an action that was not favored by the kino-dynamic planner to look more favorable to RHP that uses the action-value function as a heuristic.

In order to avoid this phenomenon, we ground the unchosen actions to a target value that is lower than the target value of the chosen action. Driven by the observation that, in the domain of pushing tasks, a mistake is in most cases not irreparable, but can be overcome through a number of k additional actions. Hence, we use for the action-value of the unchosen actions the following update target:

$$q(s_l^p, a_u^p) = \begin{cases} r(\frac{1-\gamma^{L-l+k}}{1-\gamma}), & \text{if } \hat{q}(s_l^p, a_u^p; \theta) \geq q(s_l^p, a_l^p) \\ \hat{q}(s_l^p, a_u^p; \theta), & \text{otherwise} \end{cases} \qquad (3)$$

where $a_u \in A \setminus \{a_l\}$ is an unchosen action[3]. This imposes that the unchosen actions that have a higher value than the chosen one, have a value equivalent to being k steps further away from the goal than the chosen action.

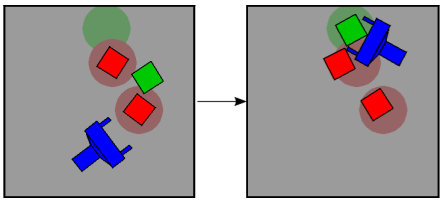

Fig. 3: The initial configuration (left) and the final configuration (right) of an example scene.

[1] A problem instance is defined by the the initial positions of the objects and their corresponding goal regions. Only one of the objects has its goal region placed away from its initial position

[2] If $\gamma = 1$ the equation collapses to $q(s_l^p, a_l^p) = (L - l)r$

[3] if $\gamma = 1$ the first component of the equation collapses to $q(s_l^p, a_u^p) = (L - l + k)r$

III. Heuristic-Guided Deep Reinforcement Learning

The performance of action-value based RHP is bounded by the quality of its heuristic. So far, the knowledge encapsulated in the DNN is based on the average behavior of the kino-dynamic planner. To further optimize the action-value function, we use reinforcement learning to 1) improve the action-value function estimate of the optimal one and to 2) ground the unexperienced state-space transitions to their actual values.

We implement the Deep Q-Learning (DQN) algorithm [10]. We initialize the DNN to the trained DNN from the previous section. Further, We formulate an RL policy, that we call ϵ-RHP, which selects a random action with probability ϵ and with probability $1-\epsilon$ the policy queries RHP for an action. We found that focusing the search towards the goal by augmenting the RL policy with RHP, reduces the chances of the action-value function from diverging which is common problem in RL when used in conjunction with a DNN as a function approximator.

IV. Searching the action-space up to the horizon

We use the learned action-value function $\hat{q}(s,a;\theta)$ as heuristic RHP, i.e. we use it as an approximation in Eq. 1 in-place of the unknown optimal action-value function at s_h.

One naïve way to plan until the horizon is to explore all possible action sequences up to the horizon h. However, an exhaustive search would scale badly with the horizon depth h and the size of the action set A, $\mathcal{O}(|A|^h)$. Instead, we bias the search towards promising actions. We implement RHP such that it simulates n roll-outs of horizon h each. Each of the n roll-outs is started from the current state s_0. At every step t in a roll-out, RHP samples an action using the soft-max of the action-value function:

$$P(a|s_t) = \frac{exp(\hat{q}(s_t,a;\theta)/\tau)}{\Sigma_{a_i \in A} exp(\hat{q}(s_t,a_i;\theta)/\tau)} \tag{4}$$

where τ is the temperature parameter. This would favor exploring actions whose value learned in the previous section is the highest. The return of a roll-out is computed as an h-step return, where the first h rewards are generated by the model, and the action-value function acts as a proxy for the rewards beyond the horizon:

$$R_{0:h} = r_1 + \gamma r_2 + \ldots + \gamma^{h-1} r_h + \gamma^h \hat{q}(s_h, a_h; \theta).$$

RHP then executes the first action in the roll-out that obtains the highest return. This procedure reduces the number of simulated actions per RHP query to $n \times h$. The action-value function, therefore, plays two roles: to inform the search through the soft-max sampling, and as a proxy for the rewards that are not sampled from the model.

V. Experiments

The manipulation scenario follows the same model introduced in Sec. I. We evaluate the applicability of the proposed approach to control a real robot executing a manipulation in clutter task in near real-time. Before implementing the control policy on the real robot, we measure the success rate of a policy in simulation. The success rate is measured over 500 random task instances such that at the end of a run, if any the boxes was out of its corresponding target region, then we considered that run a failure. Otherwise, we considered it a success. As a way of gaging how a policy copes with dynamics that are different then the one it was trained on, we injected different levels of uncertainty in the physics model (shape, friction, and density of the boxes).

TABLE I: The performance results of the kino-dynamic planner (KDP) and the learned value function (GP: greedy policy, RHP-33: n=3,h=3, RHP-66: n=6,h=6) w.r.t. different uncertainty levels

		Planner	**Learned value function**		
		KDP	**GP**	**RHP-33**	**RHP-66**
No uncert.	suc. rate [%]	98.0	51.5	88.8	94.4
	Avg. exec. time [s]	49.4	0.6	7.9	21.2
Low uncert.	suc. rate [%]	24.5	48.6	88.2	94.0
	Avg. exec. time [s]	41.1	0.6	8.0	22.7
Med. uncert.	suc. rate[%]	28.5	47.3	88.0	91.2
	Avg. exec. time [s]	42.5	0.6	8.1	18.4
High uncert.	suc. rate[%]	15.7	45.6	87.3	90.1
	Avg. exec. time [s]	37.6	0.7	8.5	17.3

The DNN is trained over $P = 9000$ plans with $k = 4$. It is further optimized with RHP-guided RL where each RHP query runs $n = 6$ roll-outs of $h = 6$ horizon depth each. The results of our simulated experiments, presented in Table I, confirm that the performance of the kino-dynamic planner degrades when the planning model is different than the execution model. The results also show that increasing the number of roll-outs and the horizon depth helps in mitigating the increased uncertainty. The performance increase comes at a cost of an increased computation time. However, it is still within reasonable limits for near real-time manipulation.

We performed experiments on a UR5 robot. Example task instances are shown in Figures 1 and in the video link https://youtu.be/xwa0fTTuQ1g. In each task, we tested the trained RHP policy and compared it to the open-loop execution of the kino-dynamic planner. During the execution of RHP, closed-loop feedback on object poses was supplied using an OptiTrack system for RHP to run the roll-outs on the model. As expected, the reactive capability of RHP made its reaction robust to the dynamics of the real world.

VI. Conclusions

The key contribution is in showing that a sub-optimal value function can drive RHP to control a robot in the domain of physics-based manipulation in clutter: the robot is able to perform fast closed-loop re-planning to deal with the inherent uncertainty in this domain, which challenges existing planners. We show how a heuristic value function can be learned from sampling-based planners and reinforcement learning.

These findings motivate us to further develop our research on real-time dynamic manipulation. We are currently extending this work to admit visual input for the state representation. This will allow the robot to seamlessly adapt to a changing number of objects in the scene.

REFERENCES

[1] D. Leidner, W. Bejjani, A. Albu-Schäffer, and M. Beetz, "Robotic agents representing, reasoning, and executing wiping tasks for daily household chores," in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2016, pp. 1006–1014.

[2] M. Dogar, K. Hsiao, M. Ciocarlie, and S. Srinivasa, "Physics-based grasp planning through clutter," in *Robotics: Science and Systems*, 2012.

[3] C. Hernandez, M. Bharatheesha, W. Ko, H. Gaiser, J. Tan, K. van Deurzen, M. de Vries, B. Van Mil, J. van Egmond, R. Burger, *et al.*, "Team delfts robot winner of the amazon picking challenge 2016," in *Robot World Cup*. Springer, 2016, pp. 613–624.

[4] J. A. Haustein, J. King, S. S. Srinivasa, and T. Asfour, "Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 3075–3082.

[5] N. Kitaev, I. Mordatch, S. Patil, and P. Abbeel, "Physics-based trajectory optimization for grasping in cluttered environments," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 3102–3109.

[6] J. E. King, J. A. Haustein, S. S. Srinivasa, and T. Asfour, "Nonprehensile whole arm rearrangement planning on physics manifolds," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 2508–2515.

[7] M. R. Dogar and S. S. Srinivasa, "A planning framework for non-prehensile manipulation under clutter and uncertainty," *Autonomous Robots*, vol. 33, no. 3, pp. 217–236, 2012.

[8] A. M. Johnson, J. King, and S. Srinivasa, "Convergent planning," vol. 1, no. 2, pp. 1044–1051, July 2016.

[9] Muhayyuddin, M. Moll, L. Kavraki, and J. Rosell, "Randomized Physics-based Motion Planning for Grasping in Cluttered and Uncertain Environments," *ArXiv e-prints*, Nov. 2017.

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