Learning a Decentralized Multi-arm Motion Planner

1 Justification on Sim2Real Transfer

Our algorithm is tested in the PyBullet simulation environment [1]. We are not able to provide real-world experiments. However, we believe our algorithm is able to generalize to real-world robot setup for the following reasons:

First, our system uses joint state as input instead of estimation from a perception algorithm. In current industry level robot systems, the joint state measurements are often highly accurate and the sim2real difference is negligible.

Second, our benchmark environment takes into account the delay of an inference pass of the motion planning policy. This means by the time the motion planner's actions are received and executed on the robots, the observations from which those actions were computed have been outdated by the amount of time which a forward pass takes, which is the case for the real-world robot setup. However, since our policy has an inference time of 1.09ms on a single CPU thread, our policy is still able to perform well with this delay.

2 Training details

The state of an arm includes its base pose (7), its end-effector pose (7), its link positions ((30 for 10 links), its joint configuration (6), and its target end-effector pose (7). One frame of history for all arm state components except for base pose is stacked on top, giving a final arm state vector of size $(7\times1)+(7\times2)+(30\times2)+(6\times2)+(7\times2)=107$.

The policy consists of an LSTM state encoder and a MLP motion planner. The LSTM state encoder has input dimension 107, hidden dimension 256, 1 layer, is single directional, and uses a zero initial hidden state. The MLP has 3 layers, [256,128],[128,64], and [64,6] where 6 is the action dimension. After each MLP layer is a Hyperbolic Tangent activation function.

Hyperparameter	Value
Actor lr	0.0005
Q function lr	0.001
Discount Factor γ	0.99
Exponential Decay τ	0.001
Batch Size	4096
Warm-up Timesteps	20,000
Replay Buffer Size	50,000

Table 1: Hyperparameters.

The Q function's LSTM shares the same architecture with the policy's LSTM, and its MLP differs only in that its output dimension is 1 and does not have an activation function.

The policy was trained using the curriculum in Tab. 2 on position tolerance ϵ_p and orientation tolerance ϵ_r, and graduates to the next level when it achieves at least 70% success rate on average in the latest 100 episodes.

The hyperparameters used for Soft Actor Critic are shown in Tab. 1.

3 Behavior Cloned Policy to deal with the Sparse Reward Problem

While [2] could use a pretrained behavior cloned policy for RL in their sparse reward setting, their application was in path planning for grounded robots in a 2D configuration space which corresponds to the cartesian space, which is significantly simpler than motion planning for robotic arms in 6 dimensional joint configuration space. We observed that a behaviour cloned multi-arm motion planning policy, despite achieving high success rates initially, quickly collapses to 0% success rate, and is unable to recover, when not provided with expert demonstrations (Fig. 1). We hypothesize that for a task as difficult as

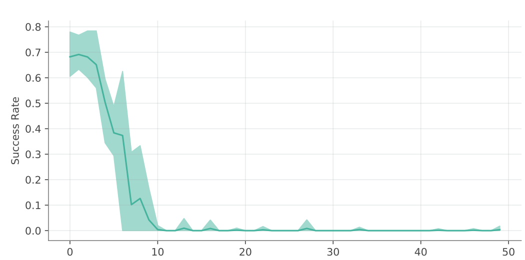

Figure 1: Without expert demonstrations, a behavior cloned policy quickly drops to 0% success rate. Plot is averaged over 5 seeds.

Level	ϵ_p (cm)	ϵ_r (rad)
1	10.0	0.20
2	8.0	0.16
3	6.0	0.14
4	4.0	0.1
5	3.6	0.09
6	3.2	0.08
7	2.8	0.07
8	2.6	0.06
9	2.4	0.05
10	2.2	0.05
11	2.1	0.05
12	2.0	0.05
13	1.9	0.05
14	1.8	0.05
15	1.7	0.05
16	1.6	0.05
17	1.5	0.05
18	1.4	0.05
19	1.3	0.05
20	1.2	0.05
21	1.1	0.05
21	1.0	0.05

Table 2: Training Curriculum

generic multi-arm motion planning for tightly coupled multi-arm systems, a constant supply of expert demonstrations in the context of failure, is much more helpful for the policy than a good initialization.

4 Denser Reward Alternative to side-step the Sparse Reward Problem

Semnani et al. [3] addressed [2]'s drawback of relying on a pretrained behavior cloned policy with a dense delta-position based reward. However, their robots are single-linked, while our robot arms have multiple links, which means the arms can easily get stuck in local optima with a corresponding delta end-effector position reward, especially when arms are close to each other. Thus, such a dense reward scheme would introduce incentives issues, and can not be applied to our problem.

References

[1] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation in robotics, games and machine learning, 2017.

[2] M. Everett, Y. F. Chen, and J. P. How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. *CoRR*, abs/1805.01956, 2018. URL `http://arxiv.org/abs/1805.01956`.

[3] S. H. Semnani, H. Liu, M. Everett, A. de Ruiter, and J. P. How. Multi-agent motion planning for dense and dynamic environments via deep reinforcement learning. *IEEE Robotics and Automation Letters*, 5(2), 2020.