2021 Fifth IEEE International Conference on Robotic Computing (IRC) | 978-1-6654-3416-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/IRC52146.2021.00023

Hierarchical Task and Motion Planning through Deep Reinforcement Learning

Abdullah Al Redwan Newaz[1], Tauhidul Alam[2]
[1]Department of Electrical and Computer Engineering, North Carolina A&T State University, Greensboro, NC, USA
(email: aredwannewaz@ncat.edu)
[2]Department of Computer Science, Louisiana State University Shreveport, Shreveport, LA 71115
(email: talam@lsus.edu)

***Abstract*—Task and motion planning (TAMP) integrates the generation of high-level tasks in a discrete space and the execution of low-level actions in a continuous space. Such planning integration is susceptible to uncertainties and computationally challenging as low-level actions should be verified for high-level tasks. Therefore, this paper presents a hierarchical task and motion planning method under uncertainties. We utilize Markov Decision Processes (MDPs) to model task and motion planning in a stochastic environment. The motion planner handles motion uncertainty and leverages physical constraints to synthesize an optimal low-level control policy for a single robot to generate motions in continuous action and state spaces. Given the optimal control policy for multiple homogeneous robots, the task planner synthesizes an optimal high-level tasking policy in discrete task and state spaces addressing both task and motion uncertainties. Our optimal tasking and control policies are synthesized through deep reinforcement learning algorithms. The performance of our method is validated in realistic physics-based simulations with two quadrotors transporting objects in a warehouse setting.**

I. Introduction

Collaborative and coordinated robot planning is incredibly important in the logistics and object transportation industries. Such planning demands integrated actions to move objects by picking them up and placing them at desired locations in a workspace which is generally known as task and motion planning (TAMP). In a TAMP problem, task planning is needed to find high-level decision strategies, such as how to grasp and where to place an object, and motion planning is required to compute the low-level actions, e.g., how to generate movements to transport objects by a robot. However, integrating task and motion planning is a hard problem since tasks are characterized in a discrete geometric space, and physical actions are executed in a continuous motion system. Therefore, the TAMP problem can also be framed as a hybrid search problem that requires a robot to reason about its actions in both discrete and continuous spaces [1].

There is also an additional action validation challenge to handle the interaction between low-level motion planning and high-level task planning [2], [3]. For instance, when requesting a high-level task, a robot needs to generate sub-tasks as well as motion plans at the execution time to carry out the requested task. However, it is a very challenging issue under uncertainties because, for a proposed sub-task, associated motion plans for numerous action candidates should be verified, which is computationally very expensive. To address this challenge, hierarchical planning can be a computationally efficient solution as it divides the entire task into subtasks and then synthesizes their feasible and scalable motion plans from physical and geometric perspectives.

Most relevant works in this stream of research utilize a symbolic model checker to verify motion plans and improve task planning by learning counterexamples from the symbolic model checker [4], [3]. However, these works consider neither uncertainties nor stochastic environments. To solve sequential decision-making under uncertainties in our TAMP problem, we are interested in leveraging Deep Reinforcement Learning (DRL) which utilizes experiences obtained from interacting with complex, uncertain environments over time to achieve a behavior that maximizes rewards [5]. Furthermore, deep reinforcement learning has had recent successes in addressing this issue for several robotic applications such as manipulation [6], [7], navigation [8], and minimally invasive surgery [9]. Although reinforcement learning-based solutions have been explored in motion planning problems only, it is yet to be investigated in TAMP problems in multi-robot settings.

This paper presents a hierarchical task and motion planning method under uncertainties through DRL. We employ Markov Decision Processes (MDPs) to model task and motion planning in a stochastic environment. To overcome the crux of challenges associated with uncertainties, our hierarchical method first synthesizes an optimal control (motion) policy and then generates an optimal tasking policy utilizing the same given control policy multiple times for multiple homogeneous robots. The motion planner takes into account physical constraints and motion uncertainty to compute an optimal control policy to generate low-level actions for a single robot. However, there are still some unaccounted motion failures (e.g., robot-robot collisions) from the motion planner that are transmitted to the task planner. As such, the task planner considers both task and motion uncertainties to find an optimal high-level tasking policy for multiple robots in an abstract discrete space ignoring their physical constraints. Our proposed task and motion planners utilize DRL algorithms to synthesize the policies so that robots can learn through experience in the case of task and motion uncertainties.

The remainder of the paper is structured as follows. The next section discusses the relevant literature on hierarchical task and motion planning. Section III formulates the problems of our interest. Section IV details our method for solving the formulated problems. We present our experiments with several case studies and performance results in Section V. Section VI concludes the paper with discussions and future directions.

II. Related Work

In the literature, hierarchical task and motion planning has already been studied [10], [11], [12]. In [10], Kaelbling and Lozano-P´erez present a hierarchical method that decomposes the planning problem into subtasks using recursive planning processes until primitive actions are reached and executed. This method solves long-horizon problems with a reasonably short horizon that makes planning feasible. The same authors extend this hierarchical planning in *belief space* for handling domain uncertainty [11]. This extended work handles *future-state* uncertainty by planning in approximate deterministic models and *current-state* uncertainty by planning in the space of a robot's *beliefs* about the state of the world that is necessary for intelligent information gathering. A hierarchical task planning method interleaved with motion planning and geometric reasoning for bi-manual tasks in humanoid robots is presented in [12]. This method divides a bi-manual tasks into small sub-goals and verifies geometric constraints and collisions at the sub-goal level. These hierarchical methods are designed for single robot systems whereas our work is applicable for multi-robot systems.

Reinforcement learning has been integrated with symbolic planning in several methods [13], [14], [15]. Silver et al. [16] study a model-based method to learn symbolic operators for TAMP from data, where operators define an abstract transition model. A task and motion planning framework with the integration of reinforcement learning [17] is presented to solve the TAMP problems in dynamic and uncertain domains. This framework first generates a feasible task-motion plan by iteratively planning in the discrete space and learns to improve its plan by updating relevant action costs evaluated by the motion planner in the continuous space. The scalability of TAMP is improved in a recent work of Wells *et al.* [3] by learning a classifier for feasible motions and use this classifier as a heuristic to search for a task-motion plan. This learned heuristic reduces the total number of motion planning attempts.

III. Problem Formulation

Integrated task and motion planning under uncertainty is modeled as a sequential decision making problem. Fig. 1 shows a TAMP problem in a stochastic environment, where the outer loop represents the task planning problem and the inner loop shows the motion planning problem in a stochastic environment. However, computing a solution for this problem requires simultaneous reasoning about task and motion uncertainties. To make this problem tracktable, our hierarchical planning framework decomposes the task and motion planning problem into two sequential but independent problems.

Problem 1 (Task Planning): We model the task planning problem as a multi-agent Markov decision process (MMDP). An MMDP is a tuple $\mathcal{M} = \langle S, s_0, \alpha, A, P, R\rangle$, where S is

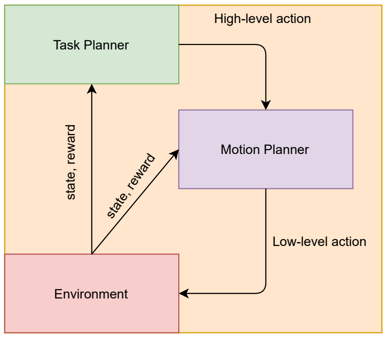

Fig. 1: **Problem Overview:** The TAMP under uncertainties is decomposed into two interconnected planning problems: the high-level task planning and the low-level motion planning. The task planner provides discrete actions that are translated into continuous motions by the motion planner. The environment provides information about states and rewards to both task and motion planners.

a finite set of states, s_0 is a set of initial locations, α is a set of robots, $A = A_1 \times \cdots \times A_n$ is the joint action space where A_i is a set of local actions available to the robot i, $P : S \times A \times S \rightarrow [0, 1]$ is the transition probability function, $R : S \rightarrow \mathbb{R}$ is a real-valued reward function.

In the high-level task planning problem, we are interested in finding an optimal tasking policy in discrete task and state spaces. One way we can view the failure of high-level discrete tasks is because of an invalid low-level control policy since low-level motion uncertainty propagates to the high-level task planner. Thus, the high-level task planning captures the failure of the low-level motion policy using transition probabilities. Additionally, the high-level tasking policy handles task uncertainty. On the other hand, in the low-level motion planning problem, we are interested in synthesizing an optimal control policy in continuous action and state spaces. This low-level optimal control policy addresses low-level motion uncertainty.

Problem 2 (Motion Planning): We consider that the low-level motion planning task is modeled through a Markov Decision Process (MDP). An episodic MDP M is a 4-tuple $\langle X, U, p, r\rangle$, where X is the set of states, U is the set of controls (actions) in the continuous domain, $p(x'|x, u)$ is a transition function that gives the probability of transitioning to state x' after taking an action u in state x, and $r(x, u, x')$ is a reward function that gives the immediate reward for taking an action u in state x and transitioning to state x'.

The goal of our low-level policy is to find an optimal control policy that prescribes motion for a robot based on its current state and high-level task requirements as follows:

$$\begin{aligned}\pi^* &= \underset{a_t \in A, u_t \in U}{\arg\max}\ \mathbb{E}\left\{\int_{t=0}^{\infty} \gamma^t r\left(x_t, u_t, x_{t+1} | a_t, s_t\right)\right\} \\ &= \underset{a_t \in A, u_t \in U}{\arg\max}\ \mathbb{E}\left\{\int_{t=0}^{\infty} \gamma^t \sigma(s_t, a_t) r\left(x_t, u_t, x_{t+1}\right)\right\},\end{aligned} \quad (1)$$

where γ is a discount factor and σ is a high-level tasking policy. Note that, we consider a team of homogeneous cooperative robots for the high-level task planning problem, whereas a single robot is considered for the low-level motion planning problem. Since the robots are homogeneous, we can synthesize an optimal low-level control policy for a single robot and then use the same model multiple times for solving the high-level task planning problem for multiple robots.

IV. Method

In this work, we train deep reinforced learning robots to solve the TAMP problem, where the robots learn an optimal tasking policy through the Deep Q-Network (DQN) algorithm but execute tasks in the physical environment by learning an optimal control policy through the Deep Deterministic Policy Gradient (DDPG) algorithm. In our method, we first synthesize an optimal control policy for a single robot and then computes an optimal tasking policy based on given control policies for multiple homogeneous robots.

A. Deep Deterministic Policy Gradient for learning an optimal control policy

To learn robot controls in the continuous state space, we utilize the DDPG algorithm that combines ideas from DPG (Deterministic Policy Gradient) and DQN algorithms [18]. In particular, the DDPG algorithm uses Experience Replay from the DPG algorithm and slow-learning target networks from the DQN algorithm. The Experience Replay stores a list of experiences (state, control, reward, next state), and instead of learning only from the recent experience, it learns from sampling all of the experiences accumulated so far. Two stable target networks are used and updated in the DDPG algorithm under its Actor-Critic network models as illustrated in Fig. 2. The *Actor* network provides actions for a given state. The *Critic* network predicts if the retrieved controls (actions) from the Actor network is good (positive value) or bad (negative value) given a state and actions.

The proposed actor network structure is composed of two fully connected layers followed by ReLU activation functions. The actor network input is an l-dimensional robot state $x \in X$. Each fully connected layer consists of 64 neurons. The output layer is also a fully connected layer where the number of neurons is equal to the size of the action space followed by tanh activation functions. For example, the size of the action space is 4 that represents the speeds of four rotors of a quadrotor. On the other hand, the critic network also follows the similar structure. The main difference appears in the input and output layers where the critic takes actions as input from the actor and outputs a Q-value for a robot action. Therefore, the critic network input is both actions and a state of a robot. Unlike the actor network, we use a linear activation function for the output layer of our critic network. Finally, we use the Mean Squared Error (MSE) loss function and the Adam optimizer for stochastic gradient descent.

B. Curriculum learning for improving the control policy

To accomplish a complex motion task, we can synthesize a control policy that combines a set of simple motion strategies. The key idea of curriculum learning is to train the motion planner for learning the complex motion task sequentially, where each model tries to learn a complex control policy based on its predecessor.

Decomposing the complex motion task, we define a set of tasks $\mathcal{T}$, where each task is modeled as an MDP M. The goal is to design and choose a full sequence of tasks (i.e., a curriculum) $M_1, M_2, \dots M_t$ (where $t = |\mathcal{T}|$) for a robot to train on, such that the learning speed or performance on the complex motion task is improved. This curriculum learning process of the complex motion task starts with an initial control policy. The robot then chooses an action, where each action corresponds to a different task that can be learned by the robot. The robot interacts with that task, and updates its policy as a result of learning. Learning a task also incurs a cost, which is the amount of time required by the robot to learn the task. This process terminates once the robot learns a control policy for the complex motion task that can achieve a desired performance threshold [19].

C. Deep Q-Network for learning an optimal tasking policy

The goal of our task planner is to prescribe high-level discrete actions to a team of robots by learning an optimal tasking policy $\sigma : S \times A \rightarrow [0, 1]$ that maximizes cumulative future rewards. We utilize a deep neural network [20] to approximate the optimal action-value function $Q^*(s, a_1, .. s, a_n)$ considering actions, $a_1, .., a_n$ of n robots as follows:

$$Q^* = \max_\sigma \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s, a_1, .., a_n, \sigma\right], \tag{2}$$

which is the maximum sum of rewards r_t discounted by γ at each time step t, achievable by a tasking policy $\sigma(a_1, .., a_n|s)$, after making an observation (s) and taking the actions from all robots $(a_1, .., a_n)$. If the optimal value $Q^*(s', a_1^{'}, .., a_n^{'})$ of the sequence s' at the next time-step is known for all possible robot actions $a_1^{'}, .., a_n^{'}$, then the optimal tasking policy is to select the actions $a_1^{'}, .., a_n^{'}$ maximizing the expected value of $r + \gamma Q^*(s', a_1^{'}, .., a_n^{'})$ as follows

$$Q^* = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s', a_1^{'}, .., a_n^{'}) | s, a_1, .., a_n\right]. \tag{3}$$

DQN approximates the state-action value function such that $Q(s, a_1, .., a_n; \theta) \approx Q(s, a_1, .. s, a_n)$, where θ denotes the weights of a neural network which is referred to a Q-network in this case. A deep Q-network can be trained by minimizing a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration,

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(.)}\Big[(y_i - Q(s, a_1, .., a_n; \theta_i))^2\Big], \tag{4}$$

where,

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s, a_1, .., a_n; \theta_{i-1}) | s, a_1, .., a_n\right]$$

is the target for the iteration i.

Fig. 3 shows the proposed DQN network structure which is composed of three fully connected layers followed by ReLU activation functions. The input of our DQN network accepts a k-dimensional robot state $s \in S$. Each fully connected layer consists of 16 neurons. The output layer is also a fully connected layer followed by linear activation functions. The

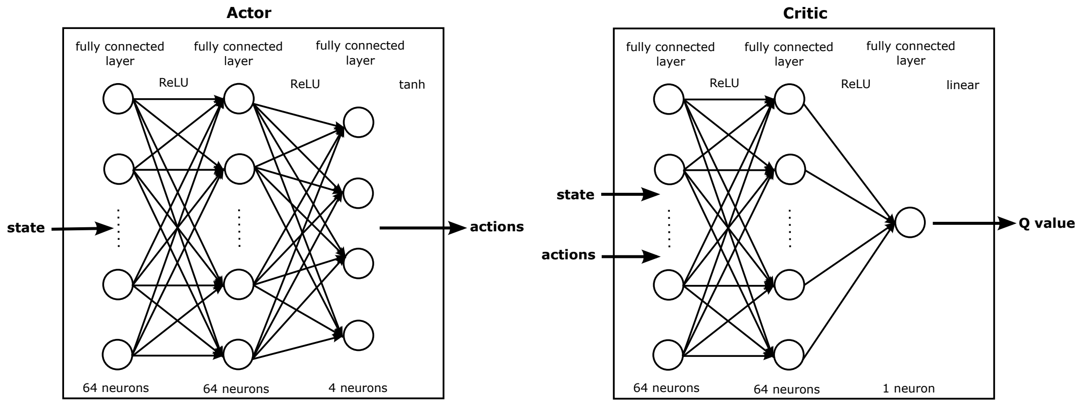

Fig. 2: **Motion Planner:** The low-level control policy is synthesized using the DDPG algorithm that consists of two models to learn the policy in a continuous state space setting: Actor and Critic. The Actor is a policy network that takes the state as input and outputs actions in the continuous space. The Critic is a Q-value network that takes a state and actions as input and outputs the Q-value.

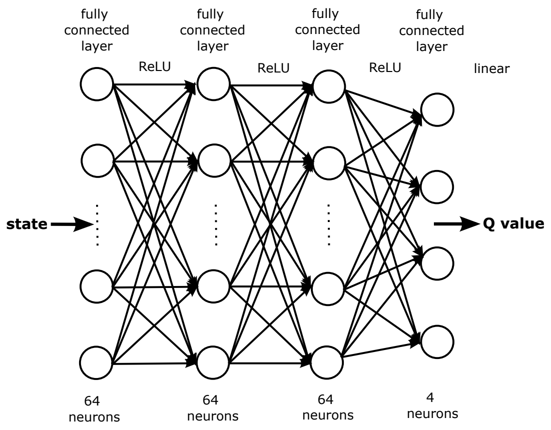

Fig. 3: **Task Planner:** The high-level tasking policy is synthesized using the DQN algorithm. The DQN leverages task planning under uncertainties to learn a policy from abstract and high-level discrete states and actions. The proposed DQN can jointly evaluate actions from multiple robots to compute the high-level tasking policy.

number of neurons in the output layer is equal to the size of the action space which is, for example, 4 for a quadrotor.

V. Experiments

In our experiments, we are controlling two quadrotors equipped with trained neural networks using deep reinforcement learning and curriculum learning. Experiments involve autonomously the objects (e.g., packages) picking and placing tasks in a warehouse environment. All the experiments are conducted on a general purpose laptop with a 2.2GHz quad-core Intel Core i7 processor. We use the Keras-rl [21] library to train the deep neural networks. We used a robotic simulator, CoppeliaSim (V-REP) [22], with built-in Bullet physics engines for training and testing purposes. We design a custom quadrotor model in CoppeliaSim which is comprised of four rotors with four propellers. This quadrotor has two pairs of identical propellers: two propellers run clockwise, and the other two propellers run counterclockwise. We use the independent variation of the speed of each rotor for controlling and generating a wide variety of motions. By changing the

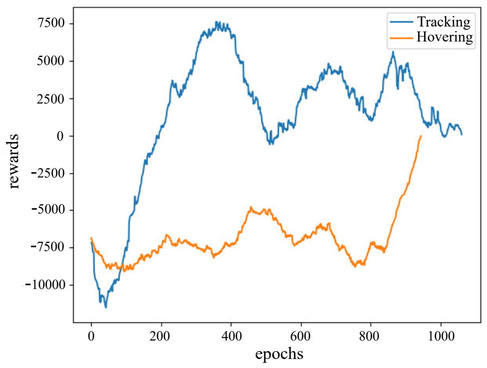

Fig. 4: **Learning curves:** The low-level motion planning problem is divided into two different curriculum learnings: the hovering and tracking behaviors. First, we use the DDPG algorithm to train the robot to learn the hovering behavior. Next, we retrain the previous model to learn the tracking behavior. Cumulative rewards while training for the hovering and tracking behaviors are shown across 1000 and 1100 epochs, respectively.

speed of each rotor, we generate a desired total thrust for lifting the quadrotor in vertical directions, and create a desired total torque for moving the quadrotor in horizontal directions.

A. *Learning the hovering behavior*

Our first goal is to use reinforcement learning to build a quadrotor that can learn the hovering behavior on its own. For this purpose, we define a target location in a collision-free 3D environment and utilize the DDPG algorithm to learn the continuous rotor control to achieve the hovering position at the target location. We design a 12-D state vector to learn the hovering behavior as follows:

$$X = \{e_x, e_y, e_z, e_\phi, e_\theta, e_\psi, \dot{e}_x, \dot{e}_y, \dot{e}_z, \dot{e}_\phi, \dot{e}_\theta, \dot{e}_\psi\}, \quad (5)$$

where the variables e_x, e_y, e_z and e_ϕ, e_θ, e_ψ are positional errors and Euler rotational errors, respectively. Similarly, the variables $\dot{e}_x, \dot{e}_y, \dot{e}_z$ and $\dot{e}_\phi, \dot{e}_\theta, \dot{e}_\psi$ are derivatives of positional errors and derivatives of Euler rotational errors, respectively. The control space is a 4-D continuous variable as follows:

$$U = \{\omega_1, \omega_2, \omega_3, \omega_4\}, \quad (6)$$

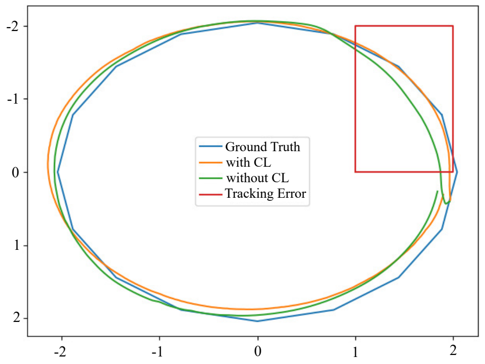

Fig. 5: **Tracking error:** The robot tracking performance improves with curriculum learning (CL). The blue line is the ground truth circular path that the robot needs to track. The green line represents the tracking performance. The yellow line represents the tracking behavior with CL. The red rectangle shows the improvement of the tracking behavior using CL over the hovering behavior.

where $\omega_1, \omega_2, \omega_3, \omega_4$ are the speeds of four rotors whose values are normalized in the range $[-1, 1]$, such that $\forall i : \omega_i \in [-1, 1]$.

We define a $3m \times 3m \times 3m$ cubic workspace to train the robot to learn the hovering behavior. The robot receives a positive reward that is inversely proportional to the distance between the robot's current state and the target state. On the other hand, the robot receives a negative reward if it flips or flies beyond the workspace. Fig. 4 shows the learning curve of the quadrotor during the training phase. The robot successfully learns the hovering behavior after 1000 epochs.

B. Learning to track a trajectory

Once the quadrotor learns the hovering behavior by minimizing the errors between its current state and the target state, we can dynamically change the target position and expect the quadrotor to follow the moving target. However, this introduces an undesirable tracking error because the hovering training does not capture the moving target behavior. To illustrate that, we compare our curriculum learning (CL)-based method to the hovering-based behavior method for the moving target tracking. For this experiment, we design a circular path and then change the target position along that path at a constant speed of 4 m/s as illustrated in Fig. 6. As we can observe from Fig. 5, the CL-based method outperforms the hovering-based method in terms of tracking accuracy. In particular, the robot trajectory followed by curriculum learning significantly deviates less compared to the hovering-based method, resulting in a better tracking accuracy. We also observe in Fig. 4 that curriculum learning has an effect on the speed of convergence of the training process to get a maximum reward. This is because the robot does not learn the tracking behavior from scratch rather utilizes the previous knowledge. Therefore, we notice that after 1000 epochs, the robot obtains notable higher rewards compared to the hovering behavior learning phase.

C. Learning to pick and place objects

We evaluate our method two pick and place tasks in a warehouse environment as depicted in Fig. 7. For this task,

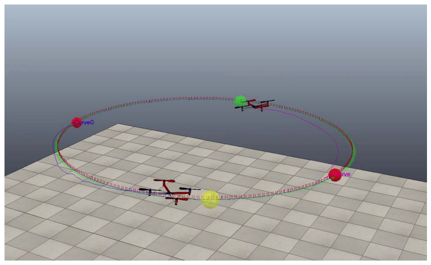

Fig. 6: **Circular path tracking:** Two quadrotors are required to track a circular path by chasing their moving targets, i.e., yellow and green spheres. Quadrotors' trajectories are shown using blue and magenta lines where red dots represent their initial locations.

two flying robots are presented with 6 objects with 2 different colors on two tables. Each robot incorporates a suction manipulator at its body center point to pick and place objects in this environment. The goal is to transfer all the red objects from the first table to the second table and all the yellow objects to the first table. The robot's low-level control policy is trained using the DDPG algorithm in a collision-free environment, the robots must learn the co-operative object transportation behavior using the DQN algorithm. Also, the same low-level control policy is used for two robots. We model the task of picking and placing objects using two robots as a high-level task planning problem where the action space for each robot is comprised of four discrete actions as follows:

$$A = \{\texttt{move}, \texttt{pick}, \texttt{drop}, \texttt{wait}\}\,. \tag{7}$$

Each robot state is comprised of four discrete observations as follows:

$$S = \{\texttt{isCollision}, \texttt{isPicked}, \texttt{isDropped}, \texttt{isFinished}\}\,. \tag{8}$$

Robots get positive rewards if they can transfer objects to the desired locations without any collisions. Otherwise, they receive negative rewards from their task execution. Each robot takes sequential actions, i.e., `pick`, `move`, and `drop` to transfer an object from one location to another. To avoid an inevitable collision, a robot chooses to either `wait` or `move` to a different location. Finally, when all the blocks are transferred to their desired tables, the robots' mission is finished and they receive a higher positive reward collectively.

Fig. 7 shows two different experiments in two different settings. The difference in settings is determined based on the initial positions of red and yellow objects. Results are shown with and without transportation failures. A transportation failure occurs when robots fail to pick an object because of low-level motion uncertainties or collide while moving around. In such cases, the robots learn from their mistakes and iteratively searching for a better tasking policy. Once an optimal safe tasking policy is found by the DQN algorithm, the robots can perform this pick and place task safely and efficiently. The video of experiments can be found at https://youtu.be/rHX4-86TrX8.

VI. Conclusion and Future Directions

We present a hierarchical task and motion planning method under uncertainties. Addressing task and motion uncertainties,

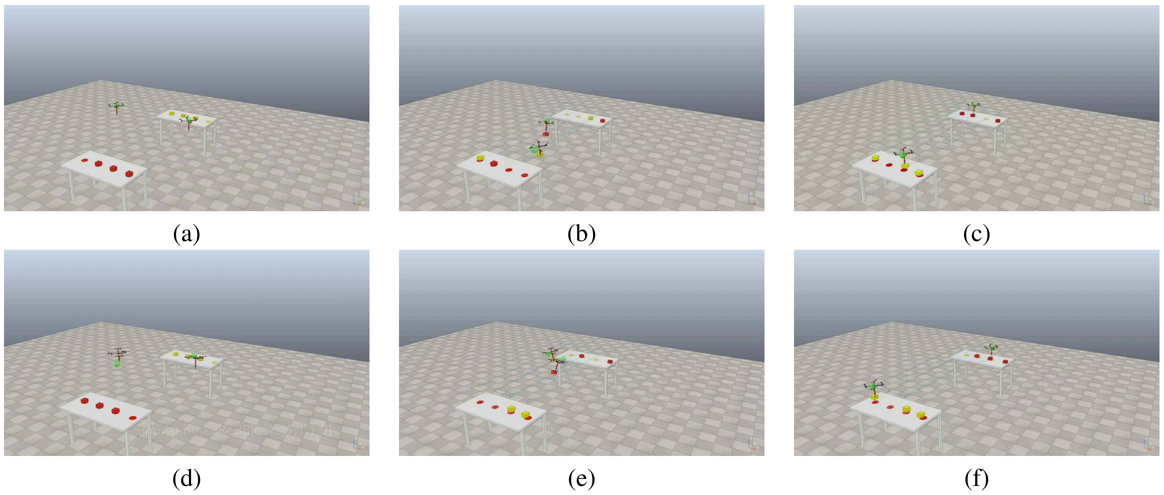

Fig. 7: **Pick and place tasks in a warehouse environment:** Two flying robots pick and place objects from one table to another while avoiding robot-robot collisions. All the red objects from the first table are transferred to the second table and all the yellow objects are transferred from the second table to the first table. Two experiments, i.e., (a)-(c) and (d)-(f), are conducted in two different settings of (red and yellow) objects considering their initial positions.

the task planner finds an optimal high-level tasking policy for multiple robots in a discrete state space. The motion planner considers physical constraints and motion uncertainty to generate an optimal low-level control policy for a single robot to execute motions in a continuous state space. The optimal robot control policy is improved through curriculum learning. We demonstrate that the curriculum learning-based motion planner outperforms the baseline motion planner for the trajectory tracking task. Given this optimal curriculum learning-based control policy, we train the high-level task planner to handle unmodeled motion uncertainty along with task uncertainty for multiple homogeneous robots. To validate our method, we show through realistic physics-based simulations that multiple quadrotors can perform an object transportation task safely and efficiently in a warehouse setting.

Our future plan is to extend the proposed hierarchical task and motion planning method to synthesize policies for Partially Observable Markov Decision Processes (POMDPs).

References

[1] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Ann. Rev. Control, Robot. and Autom. Syst.*, vol. 4, pp. 265–293, 2021.

[2] B. Kim, Z. Wang, L. P. Kaelbling, and T. Lozano-Pérez, "Learning to guide task and motion planning using score-space representation," *Int. J. Robot. Res.*, vol. 38, no. 7, pp. 793–812, 2019.

[3] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1255–1262, 2019.

[4] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach," in *Proc. Robot.: Sci. Syst.*, 2016.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[6] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 3389–3396, 2017.

[7] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine, "Composable deep reinforcement learning for robotic manipulation," in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 6244–6251, 2018.

[8] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 5129–5136, 2018.

[9] X. Tan, C.-B. Chng, Y. Su, K.-B. Lim, and C.-K. Chui, "Robot-assisted training in laparoscopy using deep reinforcement learning," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 485–492, 2019.

[10] L. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 1470–1477, 2011.

[11] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *Int. J. Robot. Res.*, vol. 32, no. 9-10, pp. 1194–1227, 2013.

[12] A. Suárez-Hernández, G. Alenyà, and C. Torras, "Interleaving hierarchical task planning and motion constraint testing for dual-arm manipulation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, pp. 4061–4066, 2018.

[13] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, "FFRob: Leveraging symbolic planning for efficient task and motion planning," *Int. J. Robot. Res.*, vol. 37, no. 1, pp. 104–136, 2018.

[14] C. Hogg, U. Kuter, and H. Munoz-Avila, "Learning methods to generate good plans: Integrating htn learning and reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2010.

[15] Y. Wang, A. A. R. Newaz, J. D. Hernández, S. Chaudhuri, and L. E. Kavraki, "Online partial conditional plan synthesis for POMDPs with safe-reachability objectives: Methods and experiments," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 3, pp. 932 – 945, 2021.

[16] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez, "Learning symbolic operators for task and motion planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021.

[17] Y. Jiang, F. Yang, S. Zhang, and P. Stone, "Task-motion planning with reinforcement learning for adaptable mobile service robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, pp. 7529–7534, 2019.

[18] 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 *Proc. Int. Conf. Learn. Rep.*, 2016.

[19] S. Narvekar and P. Stone, "Learning curriculum policies for reinforcement learning," in *Proc. Int. Conf. Auto. Agents and Multi-Agent Syst.*, pp. 25–33, 2019.

[20] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[21] M. Plappert, "Keras-rl." https://github.com/keras-rl/keras-rl, 2016.

[22] E. Rohmer, S. P. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, pp. 1321–1326, 2013.