Reinforcement Learning for Task and Motion Planning Learning in Mobile Robots

Scott Sikorski
CS6501-3 – Reinforcement Learning
University of Virginia
nqj5ak@virginia.edu

Abstract

Robots are becoming a larger tool to help people complete tasks in their lives. They have been shown to greatly help in structured environmentsm repetitively completing a task. However, there is need for the ability to continuously learn new tasks in any environment that they are deployed in. This requires the robot to become capable of completing sequential actions without external support. These tasks are composed of 2 phases, motion planning and interactable actions. This work proposes an integrated task and motion planning (TAMP) approach which aims to integrate and learn tasks in the spatial and discrete action spaces into one multi-task policy to be used for later use. A Deep Double Q-Network is used to train the agent. The state is encoded as a one-hot vector of the agent and the visible objects rather than an image. A discrete action is outputted in the form of discrete movement or action. Using mobile home robot simulations, this approach can complete a variety of tasks that involve a sequence of motion and interactable actions through one policy. 5 diverse environments and tasks are trained. The agent's ability to learn various new tasks is showcased through an increased reward reception and goal completion. These policies are then validated by completing the same goal task in the optimal number of actions with minimal failure.

1 Introduction

Robots have started to become ubiquitous in a variety of settings to help reduce mundane tasks for humans. These robots have long been limited to factory settings where the task is highly structured and does not need to be adapted. However, we have begun to find use for these robots in highly unstructured and dynamic environments, such as the home. In these settings, the specified task usually requires a sequence of actions that may drastically vary based on the state of the environment. Deciding and planning this exact sequence varies with environment size and shape, where the object is located, what primitives are supported, and more.

The motion planning problem has long used static planners that did not learn about the world. This often caused bottlenecks without parameter tuning or expert domain knowledge. The navigation would be done using methods such as A* or Dynamic Window Approach Fox et al. [1997] to plan optimal motion that could avoid collisions. This can be integrated with the task planning which can be defined statically using Planning Domain Definition Language (PDDL). The preconditions, postconditions, states, and operators are all defined by the developer to achieve a goal.

However, with the introduction of computer vision networks and general navigation learning frameworks, mobile robots can begin to learn about environments they are placed into by exploring and reasoning about the world's properties. This method has been optimized well when presented expert demonstrations or if the robot is contained in the same environment. The motion and interaction tasks are divided into sequences of tasks, $\mathcal{T} = \mathcal{T}_1, \mathcal{T}_2, ...\mathcal{T}_N$. This planning problem is known as Task and

Preprint. Under review.

Motion Planning (TAMP) where the robot must learn the most optimal sequence of actions, or more generally a successful sequence.

Reinforcement learning techniques have been proposed as solutions because of their ability to balance explore and exploit methods that learn from experience. Games such as Mario Bros has been shown to be completed by agents using Proximal Policy Optimization (PPO) Schulman et al. [2017] or Double Deep Q-Networks van Hasselt et al. [2015]. These methods are useful for discrete or continuous state spaces to predict a discrete action space. All of these approaches have been shown to converge to the optimal policy.

I aim to propose an integrated reinforcement learning approach for TAMP to effectively explore and learn a set of tasks in a collection of different environments. I employ a Markov Decision Process (MDP) in the form of Double Deep Q-Networks to plan both motion and task planning of an object of interest. The larger goal task is split up into multiple sub-tasks that are segmented based on moving and object interaction. So even though one model is used, these sub-tasks are fed sequentially to generate a specific approach for each one. Thus, the model learns to complete sub-tasks and transition between sub-tasks to complete a larger goal. Once these tasks are learned, we make sure that the parameter update does not increase the loss on previous environment, showcasing a lack of forgetting.

I aim to make the following contributions. *1:* Develop a hierarchical TAMP learning model using DDQN. *2:* Train the model on a variety of tasks to demonstrate its ability to learn sequences of tasks together *3:* Showcase the task completion with an optimized path and minimal faults.

2 Related Works

2.1 Task and Motion Planning

TAMP was classically solved using a planning domain definition language (PDDL) that is extended to the finite state space Aeronautiques et al. [1998], Helmert [2009]. Given a set of initial and goal states, a policy would attempt to find a sequence of actions that could transition from initial to goal. Each action is specified by a set of constraints on the environment for the action to be successful. This constraint based solution has been useful in developing a skeleton plan that the robot can execute. The optimal skeleton plan was shown to be computed by Mendez-Mendez et al. [2023]. They used previous samples to generate a continuous state-action space visualization. Garrett et al. [2021] provide an integrated approach that solves the constraints while sampling from experience to generate a combined policy.

Other reinforcement learning techniques exist outside of solving this constraint problem Guo et al. [2023]. This includes hierarchical and optimization methods. Hierarchical partitions the tasks into sequences of subtasks to be used with multiple replay buffers as shown by Newaz and Alam [2021]. Optimizations method aim to search for the optimal solution without keeping execution time to a minimum. Kim et al. [2019] propose an actor-critic method using prior experience to solve TAMP and find the optimal solution.

These subtasks can be learned as a part of the entire task Mansouri et al. [2021]. Newaz and Alam [2021] proposed a hierarchical approach to learning low-level subtasks as parts of a larger stochastic TAMP problem using reinforcement learning techniques. The subtasks can be combined to complete the main higher level goal task, or continue building more higher level actions. Shridhar et al. [2020] proposed benchmarks for grounding instructions from natural language commands to subtasks that could be planned sequentially. These subtasks can then be learned from expert demonstrations after decomposition.

Similarly, other groups such as Nemlekar et al. [2023] have used transfer learning as a preliminary offline learning stage using expert demonstrations. These weights can be transferred to the online policy where more active learning can take place in the presence of humans and other stochastic objects.

3 Problem Formulation

The primary goal of the proposed framework is to effectively learn new tasks in an environment without any prior information. Tasks are learned as policies which can be trained in an offline manner.

$$r = \begin{cases} 1 & \text{if } a \text{ completes goal} \\ 0.5 & \text{if goal object is reachable} \\ \frac{0.5}{dis} & \text{if goal object is visible} \\ -1 & \text{if } a \text{ fails} \\ 0 & \text{otherwise} \end{cases}$$

Figure 1: Reward Definition

Once a specific task is required of the robot, this policy can be loaded into the planner and improved upon with online data, similar to Xie and Finn [2021].

These tasks are expressed as a Markov Decision Process $\mathcal{M} = \langle s, a, r, s' \rangle \; s.t. \; s, s' \in \mathcal{S}, a \in \mathcal{A}, r \in [-1, 1]$. This models the transition between states given an action. The state space is produced independent of the environment decisions and is encoded as a one-hot encoding vector as described in Algorithm 1. The state aims to encode the position of agent and its perceived information from that position. The encoding of the visible information can represent the orientation of the robot. This is because it will encode the visible objects and their properties, such as being open, broken, dirty, picked up, etc. The state tensor is determined to be of size $|s| = |O|(|P| + 1) + 3 = 277$.

The movement actions are discretized ($GRIDSIZE = 0.25$). I simplify the motion planning by limiting movement to moving forward by this discretization and rotating left or right. This may cause the model to miss possible motion planning optimizations, especially in relation to collision avoidance and recovery methods in tight spaces. However, this allows us to directly express robot intention, which is deemed important in human environments Andersen et al. [2016]. The interaction actions were also limited to a set of reversible actions, such as open/close, turn on/off, and pickup/drop.

The reward is defined according to Figure 1. This reward penalizes action failures. This can be when the robot collides with an object or attempts an incompatible action on an object. The agent is in turn rewarded for goal completion and finding the object in a preset visibility range. The visibility range is set to 2m (8 cells). I chose this heuristic as a way of guiding the robot to find objects of importance and create a more propagated reward.

Input: Kitchen Objects O, Reachable Objects $R \subset O$, Visible Objects $V \subset O$, Object Properties, P, Agent Position x, y, z
Data: OneHot Vector $s = zeros[|O|][|P| + 1]$
for $v \in V$ **do**
  if $v \in R$ **then** s[v][0] = 1 ;
  for $p \in P$ **do**
    s[v][p] = 1 if v has p
  end
end
`s.flatten() + [x, y, z]`

Algorithm 1: State One Hot Encoding

4 Methodology

The main algorithm, shown in Algorithm 2, illustrates the usage of a Double Deep Q Network strategy to learn the values of the state for a collection of tasks and environments. The policy and target network are both randomly initialized to the same θ parameters with an empty replay buffer B. The environments with their goal tasks are given to the robot to learn. So for 2000 episodes, the DDQN are trained in each unique environment and its task. Actions are selected with an ε-greedy approach and attempted. Fails may cause no environmental change ($s = s'$); however, the replay buffer still stores this with $r = -1$ to learn from these failures. This transition is stored in B, and we draw a random minibatch sample[1]. Following suit, the expected state values are calculated according to Line

[1]At the time of the presentation, this replay buffer and batch sample was not implemented. This addition ultimately helped stabilize the results. I believe this is due to the robot learning information about the state compared to other states to aid in a higher Q value

Data: $\gamma = 0.99$, $\tau = .002$, $\epsilon_0 = .9$, $\epsilon_{end} = .003$, $N = 2000$, $M = 100$, Batch Size b=64
Input: Policy network θ, target network θ_{tar}, goals G, environments E, replay buffer B

for $environment_i \in E$ **do**
  for *episode = 1, 2, ... N* **do**
    $s = s_1$
    for *epoch = 1, 2, ... M* **do**
      $a = \begin{cases} max_{a \in A} Q_\theta(s, a) & \text{if } prob > \epsilon \\ Uniform(\mathcal{A}) & \text{if } prob > 1 - \epsilon \end{cases}$
      $s' = step(s)$
      Store transition $t = (s, a, r, s')$ in replay buffer B
      Draw random minibatch sample $\{(s, a, r, s')\}_j^b$ with size b from B
      y = $\begin{cases} r & \text{if } g_i \text{ completed} \\ r + \gamma max_{a'} Q_{\theta_{tar}}(s', a') & \text{otherwise} \end{cases}$
      Calculate $MSE(Q_\theta(s, a), y)$
      $\theta_{tar} = \tau\theta + (1 - \tau)\theta_{tar}$
      if g_i *completed* **then**
        break
      end
      $s = s'$
    end
    Reset $environment_i$
  end
  Reset θ, θ_{tar}
end

Algorithm 2: Overview of main TAMP learning algorithm using DDQN

	Env 1	Env 2	Env 3	Env 4	Env 5
Size	399	690	532	360	399
Objective	Grab Apple	Turn Off Lights	Heat up Coffee	Open the Fridge	Grab Apple
Primitives	Move Pickup	Move Toggle Off	Move Toggle On	Move Open	Move Open Pickup
Distance b/t Start & Goal	Close	Far	Close	Far	Medium

Table 1: Environment and Task Descriptions

9. I used an MSE loss over the drawn minibatch as the back propagation error for the networks. The episode is considered to be terminated if the goal task is finished or truncated if more than M actions are attempted in an episode. If the state action pair results in either, the environment is completely reset for a new episode to proceed through.

To test the algorithm's ability to learn a variety of tasks, 5 tasks were set up, each in their own unique environment.

1. Grabbing an apple acts as a simple baseline for this approach by moving a small distance and picking up the apple. This is considered the easiest task.
2. The ability to cover a large amount of distance without seeing the goal object was tested. The robot starts in the crowded section but quickly learns to navigate away, explore a more open space, and find the important section.
3. This task aims to learn the right object given numerous unimportant surrounding objects. The counter top is covered with cups and glasses which the robot learns to disregard.
4. This environment was a particularly skinny one where the robot needed to approach the fridge from the correct direction to complete the task.
5. This tested the multiple sequential tasks ability of the model. It was necessary to move, open, and pickup the correct object to complete this task.

5 Results

Experiments were conducted on my local machine[2] and sequentially[3]. Each policy would be saved after training and both network θ parameters reset.

5.1 Reward

Figure 2 illustrates the reward over episodes for each environment. This was calculated each episode as an average of the rewards given during that episode with an average over the past 100 episodes. As seen, the reward is growing in a linearly positive manner demonstrating that the robot is learning the task and completing the goal. Some of the environments begin to slightly plateau indicating that there is no more learning to be done. The reward peak reaches an absolute max that is generally seen to be plateauing at larger episodes for the first and fourth environment. The reward peak is limited by the distance of the goal object from the start and number of actions that can be taken.

Generally past episode 1000, there are hardly any dips below 0. Failures were defined by collisions or incompatible action choices resulting in a reward $r = -1 < 0$. Most episodes learn how to avoid the typical beginning failures. These are mostly navigating tight areas where it learns the movements not to take in certain states. Environment 1 and 3 were simple tasks that had the agent start between the center table and counters. This caused an early amount of collisions which were avoided. Environments 2 and 4 required the robot to navigate to an open area to find the goal object. This didn't cause a lot of early failures since moving without fault wasn't considered a poor action. However, this caused some later failures as it would try exploring the crowded section where there was no previous guidance. These late episode reward trends illustrate the lack of failing actions. And seen in section 5.4, only 1 failure was committed across all environments.

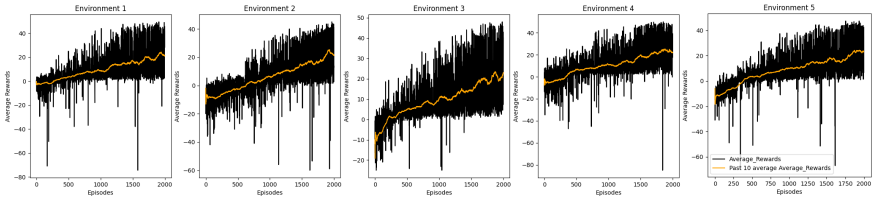

Figure 2: Average and Average Over Last 100 Episodes Reward Across Environments and Tasks Over Episodes

5.2 Goals Completed

In conjunction with the increased reward during training, the ability for the robot to complete the goal within an episode increased, as seen in Figure 3. Cumulative goal completion across episodes was tracked to indicate the progress in reaching the goal state. Like most of the environments, the goal starts out not progressing much. Eventually, it should have a logarithmic growth which transitions into linear growth, which is exactly seen in all environments. This indicates that the agent fully learns the task and can complete it every time. The exploit portion is used to solidify the optimal policy.

The logarithmic growth is most apparent in environment 5 because of the multi sequential nature of the problem. This supports the theory that sequential tasks are supported and can be learned over time. The longer the sequence, the more episodes and exploration is required of the agent.

In general, there is a distinct correlation between the reward reaching above 0 and the goal being completed. There appears to be a large jump in the average reward at point where the logarithmic growth begins to turn into linear growth. This can be confirmed by environment 1, an easy task that's learned early on, does not exhibit a low starting reward or any large jumps.

[2]GPU server access was limited due to the AI2THOR opening a Unity window to execute the simulation.

[3]Environment 5 had to be run separately because the computation time to run all 5 environments together again took too long.

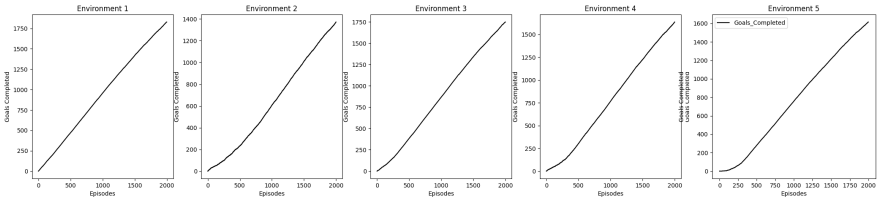

Figure 3: Number of Goals Completed Across Environments During Training

5.3 Actions Taken

To demonstrate the learned optimal policy, Figure 4 shows the actions taken by the agent across episodes[4]. As seen, the number of actions tends to decrease over the episodes. This is primarily because of the decaying ε-greedy policy that is used which prioritizes exploration at the beginning of training. This is a beneficial approach, and along with our goal completion results, illustrates that the goal is found in a sufficient amount of time. Once found, the planned path can begin to be optimized to achieve the minimum number of actions to finish the goal.

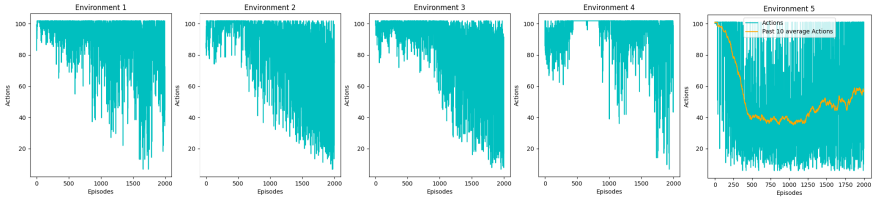

Figure 4: Number and Average Over Last 100 Episodes of Actions Taken Across Environments Over Each Episode

5.4 Testing the Learned Optimal Policy

The learned optimal policies were then tested in each environment to determine the efficiency and failure rate. Figure 5 shows that each policy executed the task in the least amount of actions necessary. Environment 1 and 3 consisted of 2 movement actions: forward, rotate and rotate, rotate, respectively. This was followed up with their respective goal completing action. These tasks didn't exhibit any failures during testing showing that easy, close tasks have no problems. Environment 2 and 4 also found the optimal movement path to their object of interest. The optimal movement was unconventional consisting of moving backward instead of rotating then moving to the object. While optimal, this does not express intention to other humans or agents in the environment. Environment 2's failure came from choosing to toggle on the already on light switch.

Lastly, environment 5 showcased that the agent could efficiently execute the multi sequential task. A motion path was taken with a series of movements such that the agent approaches the fridge from the front. This was necessary as the fridge can be opened from the side that was initially approached, but then the agent would need to traverse around the table. After accurately being front on, the open action was taken and then the apple was picked out of the available items. This task was the one that stressed the motion and sequential task planning the most. The various environments indicate that the optimal sequence of discrete motion and interactable actions can be found to complete longer sequences of tasks.

6 Limitations and Future Work

Currently, this approach only saves the policy that's generated from the offline learning to be used for later. Other work Xie and Finn [2021] have shown that this can be supplemented with online learning to better adapt to the current scenario. A proposed extension would be to look into experimentation

[4]Only environment 5 has the average due to runtime issues

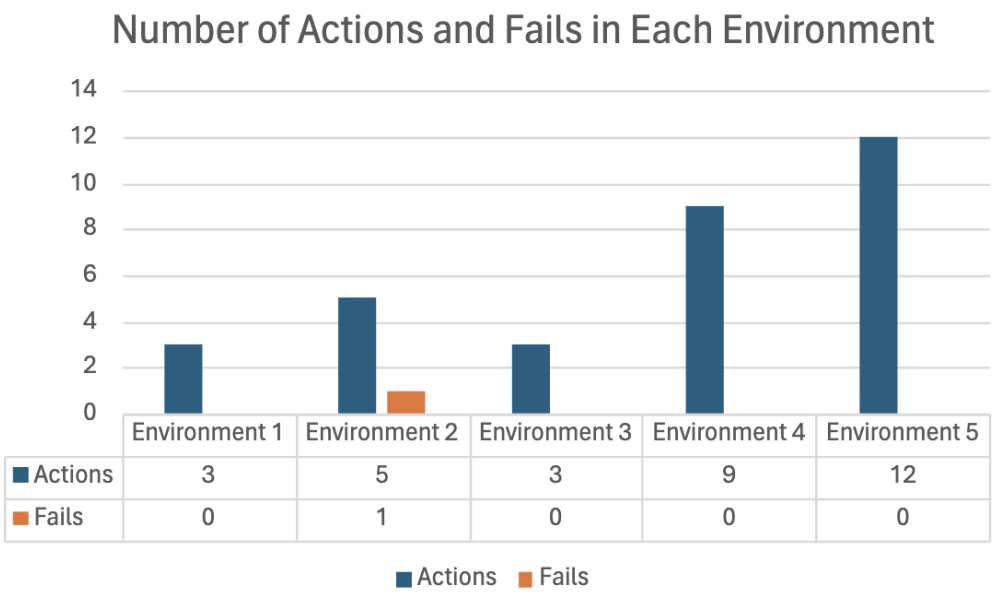

Figure 5: Number of Actions and Fails To Complete Goal During Testing

methods that train the policy for a certain task and store all the experience in separate buffers which can be loaded and sampled from. When the agent is online, more data can be collected to update the policy with new environment information.

As mentioned in section 3, the state space is discretized according to the grid size where the agent steps by moving or rotating preset values. However, real world robots move through a continuous space by applying torque to the motors. Jerky start and stopping to move forward and rotate would be a viable option. Thus, the framework can be changed to consider this continuous state and action space where movement and rotation values are determined. The primary method would be to use an actor-critic approach such as PPO or Twin-delayed DDPG Dankwa and Zheng [2019]. This way the model and framework could more accurately mimic true motion planning.

The action space was limited to a few actions that can't easily be put together for large sequences of tasks. Multi-sequence task was done with environment 5, but there was no mix of more planning beyond that. I would like to extend the experimentation and framework to include more actions and tasks that could exhibit the ability to learn deeper sequence of tasks. This was ultimately limited to computation time that required a larger truncation value to explore more deeply. Some small framework changes would be need to be changed to support the completion of other goals and run on them.

7 Conclusion

I developed an integrated reinforcement learning agent to solve the task and motion planning problem. The proposed framework utilizes DDQNs to learn a sequence of discrete state action transitions. To reduce dimensionality, a state is encoded as a one-hot encoding of the environment and its interactable objects. A discrete action with ε-greedy probability is predicted to either move or interact with an object.

Results showed that the agent is able to optimally learn a policy for a variety of diverse environments. The average reward increased linearly, and the agent was able to find and execute the goal. The optimal policy was shown to be generated through a decreasing number of actions taken during training and test results. The test results demonstrated that the minimal number of actions to complete a task was achieved, while no failure actions were taken (besides one in environment 2). This work shows promise in using RL techniques to generate integrated offline policies of diverse environments.

These policies couple sequential motion and interactable actions together to achieve a task. It was shown that deeper tasks consisting of previous action dependencies can be resolved and learned.

8 Supplementary Material

Source code for this project can be found at `https://github.com/sgsikorski/LLTAMP`.

This project was a continuation of work done for the final project for Dr. Yen-Ling Kuo's course, Interactive Learning for Mobile Robots Kuo [2023]. That work focused on lifelong learning to prevent forgetting across environments. However, it used a simple DNN to predict an action based on the agent's position. The DNN was used for all environments and updated such that the loss did not increase Lopez-Paz and Ranzato [2017]. This approach performed poorly in fully exploring the state space and could not complete tasks that required a lot of movement. This reinforcement learning approach proved to be a lot more effective in completing all tasks. That source code can be found at `https://github.com/sgsikorski/LLfTC`.

References

Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins Sri, Anthony Barrett, Dave Christianson, et al. Pddl| the planning domain definition language. *Technical Report, Tech. Rep.*, 1998.

Rasmus S. Andersen, Ole Madsen, Thomas B. Moeslund, and Heni Ben Amor. Projecting robot intentions into human environments. In *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 294–301, 2016. doi: 10.1109/ROMAN.2016.7745145.

Stephen Dankwa and Wenfeng Zheng. Twin-delayed ddpg: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent. In *Proceedings of the 3rd international conference on vision, image and signal processing*, pages 1–5, 2019.

D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33, 1997. doi: 10.1109/100.580977.

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.

Huihui Guo, Fan Wu, Yunchuan Qin, Ruihui Li, Keqin Li, and Kenli Li. Recent trends in task and motion planning for robotics: A survey. *ACM Computing Surveys*, 55(13s):1–36, 2023.

Malte Helmert. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence*, 173(5):503–535, 2009. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2008.10.013. URL `https://www.sciencedirect.com/science/article/pii/S0004370208001926`. Advances in Automated Plan Generation.

Beomjoon Kim, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Adversarial actor-critic method for task and motion planning problems using planning experience. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 8017–8024, 2019.

Yen-Ling Kuo. Learning for interactive robots, 2023. URL `https://ylkuo.notion.site/Learning-for-Interactive-Robots-Fall-2023-e42e76cdb5f441c59ecc0c4800c1de97`.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continuum learning. *CoRR*, abs/1706.08840, 2017. URL `http://arxiv.org/abs/1706.08840`.

Masoumeh Mansouri, Federico Pecora, and Peter Schüller. Combining task and motion planning: Challenges and guidelines. *Frontiers in Robotics and AI*, 8:637888, 2021.

Jorge Mendez-Mendez, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Embodied lifelong learning for task and motion planning, 2023.

Heramb Nemlekar, Neel Dhanaraj, Angelos Guan, Satyandra K Gupta, and Stefanos Nikolaidis. Transfer learning of human preferences for proactive robot assistance in assembly tasks. In *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, pages 575–583, 2023.

Abdullah Al Redwan Newaz and Tauhidul Alam. Hierarchical task and motion planning through deep reinforcement learning. In *2021 Fifth IEEE International Conference on Robotic Computing (IRC)*, pages 100–105. IEEE, 2021.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10737–10746, 2020. doi: 10.1109/CVPR42600.2020.01075.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

Annie Xie and Chelsea Finn. Lifelong robotic reinforcement learning by retaining experiences. *CoRR*, abs/2109.09180, 2021. URL `https://arxiv.org/abs/2109.09180`.