DS 551 Final Report - Group 4: High-to-Low-Level Task Planning and execution for Mobile Manipulators using RL

Farhan Seliya, Manav Bichu, Sarvesh Nair, Sai Hitesh Viswasam
Department of Robotics Engineering
Worcester Polytechnic Institute

***Abstract*—This report presents a framework for high-to-low-level task planning and execution for mobile manipulators, leveraging Hierarchial Reinforcement Learning (HRL) and Reward Shaping. HRL decomposes complex tasks into manageable subtasks for efficient policy learning, while reward shaping addresses sparse or delayed rewards, accelerating convergence. With Intrinsic Curiosity, we can ensure the agent to explore by itself during training by assigning rewards internally. Using Unity ML Agents as an interactive platform, we develop a proof-of-concept model where a mobile manipulator performs sequential navigation, pick and place operations in an unexplored environment. The approach seamlessly integrates high-level decision-making with low-level control, addressing challenges in continuous action spaces and multi-stage task execution. Future work will incorporate large language models (LLMs) to interpret high-level commands, decompose tasks, and design reward functions, combining them with RL for optimal policy derivation. Experimental results mentioned in this report highlight the framework's effectiveness, robustness, and scalability, laying the foundation for advanced robotic task automation.**

I. Introduction

Reinforcement learning (RL) applied to robots for performing household tasks presents an interesting problem with significant potential for real-world applications. RL models can initially be developed in small environments, scaled to larger ones, and transferred across multiple tasks to create robust agents capable of generalizing their learned behaviors. To make robot interactions even more accessible and human-friendly, recent work has focused on translating human language into low-level robotic commands that serve as inputs to RL agents. Our work focuses on this framework within the context of a simple household application, where high-level commands are transformed into executable actions for a robot using RL-based task planning and execution.

We hence present our project **High-to-Low-Level Task Planning and execution for Mobile Manipulators using RL**, within the context of a household application. We describe the existing work, problem setting. methodology, experimental challenges and results in the further sections of this report.

II. Literature Review

Significant amount of work has been published in the context of aiding RL models with Foundation Models to have the model to be able to develop human-like reasoning for high level tasks. One similar work is the ExploRLLM [1], which

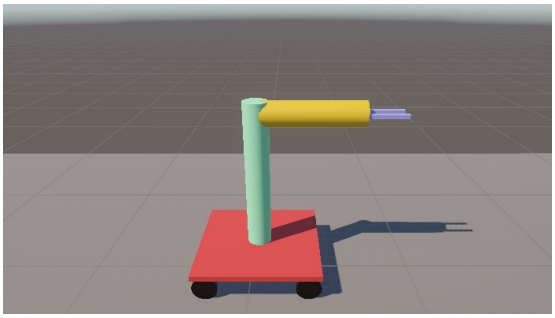

Fig. 1. This is a simple Mobile Manipulator model we considered for training. The base can move in the x and z directions, the links are connected by hinge joints. The dimensions of the base are 0.5*0.1*05 units, the arms are cylinders with radius 0.05 units and a height of 0.3 units

describes the work of a foundation model enhancing the RL agent for a pick and place task. The authors note that the main advantage of pairing an LLM with an RL model is it improves the sample efficiency for training. A similar framework was implemented in the paper Plan-Seq-Learn [2]. In this framework, the long-horizon tasks are solved through three modules, which are the Planning, Sequencing and the Learning modules. From these papers, we take the idea of the entire framework and the modules and wish to implement a similar structure in our project.

The paper 'Do As I Can, Not As I Say: Grounding Language in Robotic Affordances' [3] presents a similar framework using LLMs and RL to perform low-level tasks from a text input. Here the RL is specifically used to refine the actions generated by the LLM based on the context setting. With similar framework in place, another similar work 'Language to Rewards for Robotic Skill Synthesis' [4] approached the solving the low level control problem through designing rewards directly. This work bridges the connection between the LLM module and the RL module by generating the reward functions for the RL task through the LLM module itself.

Since our work involves deploying the framework on a robot

in a household environment, we have created our custom environment by defining the observation and action spaces and being able to code our own reward functions into the environment. We created our environment using ML-Agents in Unity.

To design the reward model, we went through several existing papers. The approach mentioned in 'Reward Shaping for Deep Reinforcement Learning in VizDoom' [7] integrates extrinsic rewards from the environment with intrinsic rewards based on state novelty and agent curiosity. We intent to design our reward models on similar lines by introducing Reward Shaping functions like the distance to the Target and Stabilizing Joints. This method provides motivation for the agent for exploration while preserving task-specific extrinsic rewards like in our case reaching the target.

Mathematically,

Novelty Computation -

$$\text{Nov}(tr) = 1 - \max_i p_i(tr)$$

Where:

- $p_i(tr)$: Probability estimated by the i-th classifier that a given transition tr has been observed recently.
- $\max_i p_i(tr)$: The maximum probability among all classifiers.
- $\text{Nov}(tr)$: The novelty score, with higher values indicating more novel transitions.

Intrinsic Reward -

$$R_{\text{int}}(tr) = \alpha \cdot \text{Nov}(tr)$$

Where:

- $R_{\text{int}}(tr)$: Intrinsic reward for the transition.
- α: A scaling parameter that ensures the intrinsic reward is comparable to extrinsic rewards.
- $\text{Nov}(tr)$: The novelty score as defined above.

Total Reward

$$R_{\text{total}} = R_{\text{ext}} + R_{\text{int}}$$

Where:

- R_{total}: Total reward used for learning.
- R_{ext}: Extrinsic reward provided by the environment.
- R_{int}: Intrinsic reward generated from the novelty score.

These equations form the core mathematical framework of the reward-shaping approach. We intend to utilize these in designing our reward model.

The paper on 'Curiosity-driven Exploration by Self-supervised Prediction' [8] introduces a curiosity-driven intrinsic reward mechanism to encourage exploration in reinforcement learning (RL) environments where extrinsic rewards are sparse or absent. We aim to utilize the similar mechanism in designing our reward model. Curiosity incentivizes the agent to seek out new or uncertain experiences, in the absence of extrinsic rewards (e.g., rewards provided explicitly by the environment), enabling it to learn and acquire knowledge that is useful in solving the problems, in our case that of navigation and pick&place.

Following are equations related to policy learning objective, intrinsic reward, and total reward which we will be using.

A. Curiosity using Unity ML Agents

ML-Agents enables a modular approach to defining reward signals, offering three types that can be combined to shape your agent's behavior. The extrinsic reward signal, which is activated by default, reflects the rewards specified in your environment. Additionally, the curiosity reward signal encourages exploration, especially when extrinsic rewards are limited.

The curiosity Reward Signal activates the Intrinsic Curiosity Module. This implementation is based on the approach detailed in "Curiosity-driven Exploration by Self-supervised Prediction" [8]. It involves training two neural networks:

- **Inverse Model:** This network takes the agent's current and next observations, encodes them, and predicts the action taken between the two observations using the encoded representations.
- **Forward Model:** This network uses the encoded current observation and action to predict the encoded next observation.

The forward model's loss—calculated as the difference between the predicted and actual encoded observations—serves as the intrinsic reward. A larger reward is given when the model encounters greater prediction error, reflecting its surprise.

Unity ML agents allows us to tune the performance of the curiosity module using the following set of hyperparameters:

- **Strength:** Represents the magnitude of intrinsic module generated by the curiosity module. It has value that ranges from (0.001 - 0.1) .
- **Gamma:** Represents the discount factor for future rewards. It has value that ranges from (0.8 - 0.995).
- **Encoding Size:** Defines the dimensionality of the ICM's compressed observation, balancing efficiency in compression with the ability to differentiate behaviors.

III. Our Approach

In this section, we present the approach we incorporated towards completing the objectives of the project. Our objective remains to develop a framework to train a mobile robot with a manipulator for household tasks, such as garbage collection. We made significant progress towards completing the navigation , pick and place operations including the implementation of essential components such as environment setup, agent scripting, reward designing and extensive policy testing. The steps are outlined as follows:

1) **Reinforcement Learning Framework for Task Optimization:**
   Our RL model forms the foundation of the robot's

decision-making process. It is designed to maximize positive rewards associated with successful task completion actions. Positive rewards are assigned for navigating to target locations, identifying target object, and performing pick-and-place actions.

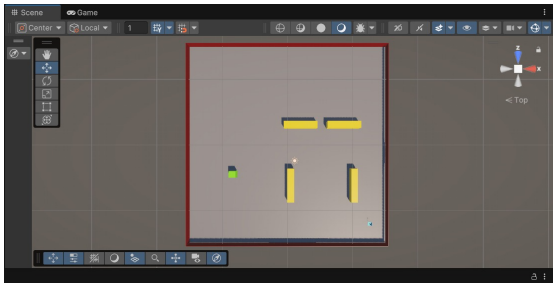

Fig. 2. The simplest environment we tested with only the target and the agent for training for the Navigation Task. The room environment is assumed to be 15x15 units. Obstacles dimensions (5x1x1 units

2) **Development of Simplified Training Environment in Unity:**

For efficient training, we simplified the environment to first focus on the navigation task. The environment includes:

- Static obstacles to simulate real-world challenges.
- Defined paths and randomized target locations to ensure adaptability and generalization.
- A reset mechanism to reconfigure the environment at the start of each training episode.

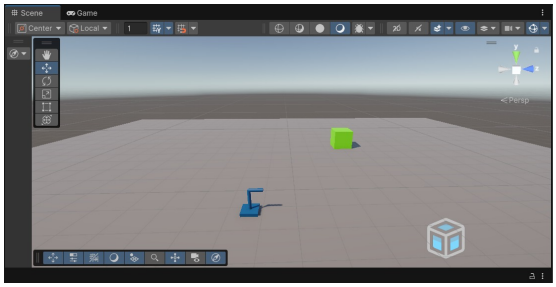

Fig. 3. The simplest environment we tested with only the target and the agent for training for the Navigation Task. The room environment is assumed to be 15x15 units. Obstacles dimensions (5x1x1 units)

3) **Custom Observation and Action Spaces:**

- **Observation Space:**
  The observation space for navigation task includes:
  - The agent's position within the environment- x, y, and z directions (3 values).
  - Sensor-based distances to nearby obstacles for collision avoidance.
  - The target's position within the environment - x, y, and z directions (3 values)
  - Linear velocity parameters of the agent - x and z directions (2 values).

  The observation space for pick task includes:
  - The agent manipulator's joint angles position (2 values).
  - The target's position within the environment - x, y, and z directions (3 values)

  The custom environment developed using the Unity 3D platform features an observation space of size 8 for navigation and of size 5 for manipulation operation,collectively providing comprehensive state representation for the environment.
- **Action Space:**
  The action space is designed for precise control of the robot, including:
  - Commands for planar translational movements.
  - Controlling the joint angles of the 2-Dof manipulator using motor force.
  - Discrete actions for halting or evasive maneuvers in response to obstacles, environments and task completion.

4) **Unity ML Agents**

This project utilizes the Unity ML-Agents framework, an open-source toolkit for developing and training intelligent agents via reinforcement learning in Unity simulations. Our agent, a mobile manipulator, combines navigation using a mobile base with manipulation via an arm. The environment includes a room bounded by walls, obstacles of varying sizes and orientations, and a target table for object placement.Below is an overview of the implemented functionality:

- **Agent & Environment Definition:** We interact with the environment through a script that defines the agent, obstacles, walls, and target as rigid bodies, specifying their physical properties such as collision, mass, friction, and inertia. The script implements the problem logic by defining the observation space, continuous action space, reward function, and episode conditions.

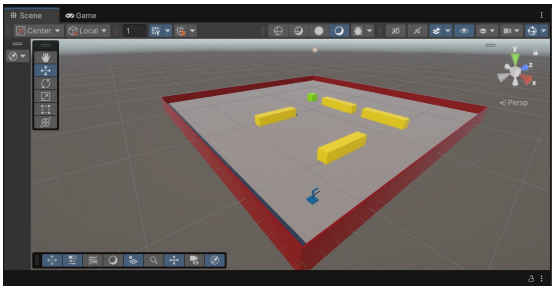

Fig. 4. Environment with our Agent (BLUE)

- **Collision Detection:** The script uses bounding box intersection logic to check for collisions between the robot's base and obstacles. This ensures real-time feedback on the robot's proximity to obstacles and triggers negative rewards for collision events.

- **Reward Assignment:** Positive rewards are assigned for reaching the target successfully, while negative rewards are given for collisions or moving away from the target. To address sparse rewards, we employ reward shaping, providing positive rewards for reaching the goal and intermediate rewards for reducing the distance to the target and stabilizing joint angles, while penalizing interactions with walls or obstacles.
- **Environment Reset:** A reset function was implemented to reinitialize the environment at the end of each episode. This includes repositioning obstacles, randomizing target locations, and resetting the robot's position for better generalization and adaptability to real-world settings.
- **End of Episode:** For efficient agent learning, we establish safety and robustness criteria to terminate an episode if the agent collides with a wall or obstacle, tips over, or exceeds a predefined time limit within a single episode.

5) **Implementing RL with Proximal Policy Optimization (PPO):**
The task involves addressing a continuous action space scenario, for which the Unity ML Agents Toolkit provides support for algorithms like Soft Actor Critic (SAC) and Proximal Policy Optimization (PPO). In this project, we focus on utilizing PPO for sequential task planning due to its advantages, which align with our objectives:

- Stability achieved through a clipped objective function that limits drastic policy updates.
- Well-suited for the structured action and observation spaces required for the navigation task.

The details of the algorithm's implementation are provided in the following section.

IV. METHODOLOGY

The robot's high-to-low-level task execution is divided into three phases using three different agents: a navigation task, a pick-and-place task, and the integration of these tasks using Hierarchical RL. First, the navigation agent is trained separately to reach the target position for picking an object and then navigate to the drop location after picking it. Similarly, another agent is trained to perform the pick operation using the robot manipulator. Finally, the agents are combined to complete the sequence-to-sequence task. The split up is further discussed in the Future Plan section.

A. Phase 1: Navigation

For the navigation task, we previously implemented the training for the navigation of the robot in the big room environment. The agent was first trained in the room with a sparse reward of obtaining $+1$ value upon finishing the task and a value of -1 on hitting any obstacles or walls.

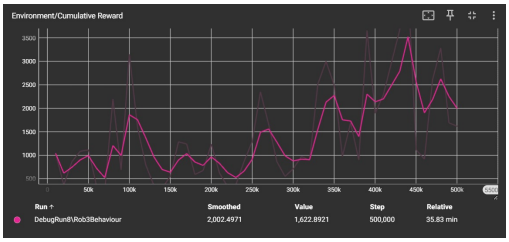

Fig. 5. Reward vs Episode graph for the case of testing with the updated reward function. The rewards range is as follows because of the unscaled positive reward the agent obtained during every step it makes towards the target in the environment

This made the agent learn a sub optimal strategy of not doing anything or making small movements just to avoid the negative reward. In the case of changing the reward function to a dense reward based on distance, the training curve did improve 5 but the agent was still failing to generalize better and was failing when the agent goes into certain states. This can be attributed to the large state space of the room and the complexity of the navigation with obstacles present.

The agent failing to adapt to the complex environment despite an apparent increase in reward means that the reward shaping needs to be revised and the training approach needs to be changed. Hence we used the approach of **Curriculum Learning** [10] by training the agent in sequential environments of increasing difficulty starting with the simplest case. This ensures that the agent can master the complex task of navigating the room with obstacles by learning simple behaviors along the way. Each new environment uses the previously trained model of the agent as a warm-start so that the agent can 'remember' the previous behaviors it learnt in preceeding ennvironments.

We decided the curriculum schedule of our agent to be as depicted in 6. The agent was first trained in the simple environment with a fixed target and without any obstacles. Additionally, a virtual bounding box has also been created around the target which acts as a virtual boundary for the agent, further constraining it to roam within a confined space, reducing the observation space. The reward function for this case was chosen to be the simplest sparse reward of $+1$ on reaching goal and -1 if the agent falls off the plane.

The agent is trained by PPO algorithm with the choice of hyperparameters as shown in I. The first agent trained for 140,000 episodes and was able to implement the task, by collecting an average mean reward of 0.997. The reward curve for the agent is shown in fig.7.

Now that the agent has learnt to move towards the target, in the next training step, we trained the agent in the environment by randomizing the position of the target. However, while training with random locations of the target, the agent struggled to find a strategy to move towards it, as it has only

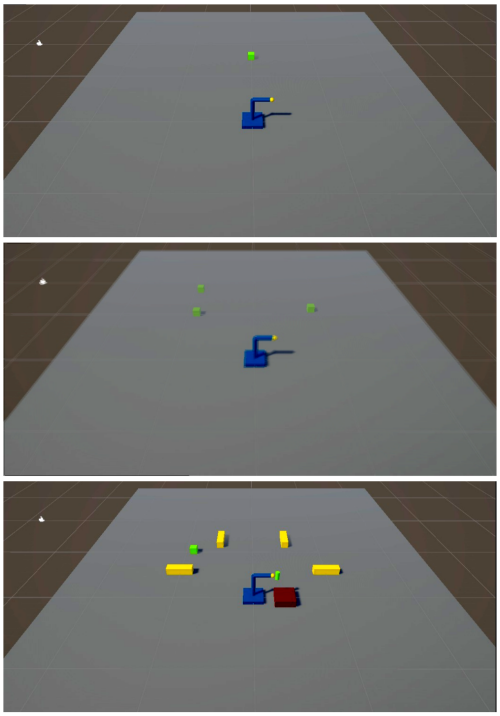

Fig. 6. The process of Curriculum Learning for the navigation task. The agent was first trained to navigate to a fixed target, then to a target with random locations and finally to navigate through the scene with obstacles.

Hyperparameter	Value
Batch Size	512
Buffer Size	1024
Discount Factor (γ)	0.99
ϵ	0.2
β	5e-4
Learning Rate	3e-4
Learning Rate Schedule	Linear
β Schedule	Constant
ϵ Schedule	Linear
No. of Layers in Policy Network	2
No. of Hidden layers	128

TABLE I
HYPERPARAMETERS FOR PPO FOR THE BASIC ENVIRONMENT

learnt to move in a straight line so far. This behavior can be shown in the training curve8.

Hence, to fix this, we made the environment even simpler for the agent to learn and only randomized the target's position in the z direction while keeping its x position fixed, so that the agent would learn to move horizontally as well. Now in this case, the agent was able to figure out to learn the action of moving horizontally as well to reach the target. This agent trained for 160,000 episodes and accumulated an average mean reward of 0.8847 as shown in fig.9.

Similarly, in the next training case, the agent could now learn to reach the target even when the target is randomized in different positions along both x and z directions. Now with the agent able to reach to a random target position without

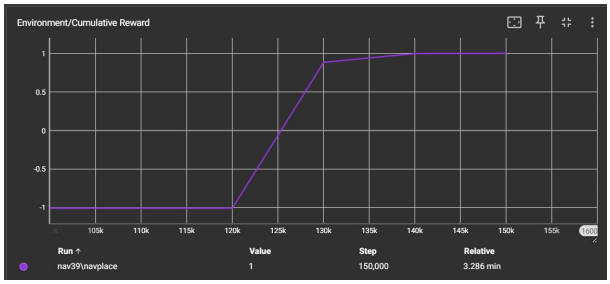

Fig. 7. Reward vs Episode graph for the case of simple navigation with a fixed target.

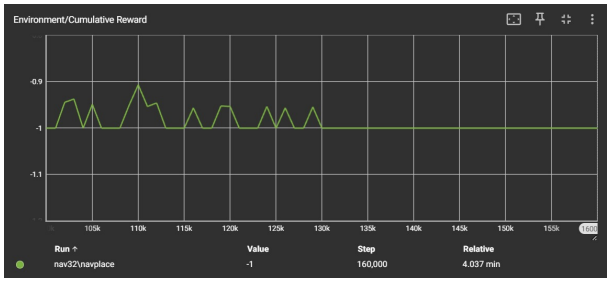

Fig. 8. Reward vs Episode graph for the case of random position of the target, the agent was unable to learn a good strategy as it could not adjust to the randomness of the target quickly. It kept falling off the platform most of the times hence collecting a reward of -1

the obstacles, the next step is to introduce obstacles to the environment. Along with adding the obstacles to the scene, we also changed the reward function by setting a reward of -1 whenever the agent encounters an obstacle and by terminating the episode. The agent started learning slowly by hitting obstacles in its initial runs and collecting negative rewards, but as the episodes progressed, it could figure out a path that avoids obstacles and reach the target. The learning curve for this training run is shown in 10.

The model that was trained with the above environment was able to avoid obstacles every episode, but there was a major room for improvement in the model. The agent only learnt a rigid strategy of always going left around the obstacles irrespective of the position of the target. This behavior is depicted further in the figure11.

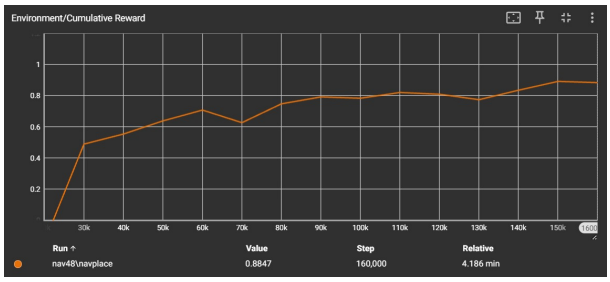

Fig. 9. Reward vs Episode graph for the case of target randomized only in z direction.

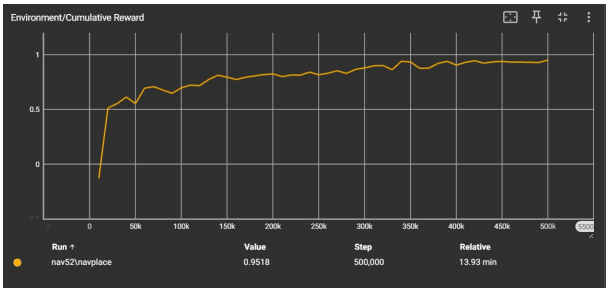

Fig. 10. Reward vs Episode graph for the case of environment with obstacles.

Hence to improve upon this behavior, we made a few changes to the environment and the reward structure so as to teach the agent to learn an optimal path toward the target and avoid learning a rigid strategy. For the agent to learn an optimal path, we changed the reward function to introduce a negative reward for every action step taken by the agent in the environment so as to encourage the agent to take lesser steps to reach the target. While for the environment, we randomized the positions of the obstacles within a threshold region so as to avoid the RL agent learning any rigid strategy. The agent was trained in this environment for 500,000 episodes and obtained a mean reward of -0.196. The negative reward simply indicates the reward obtained due to the penalty, but overall reward of the agent improved as training progressed, as shown in 12

B. Phase 2: Manipulation(Pick Operation)

For the pick task, the agent, a 2-DOF manipulator, is trained to move its end-effector to touch the target. The setup involves placing the target object on a tabletop within the robot arm's reach. The agent learns an optimal policy to manipulate the target, effectively devising the best strategy within the constraints of its training environment.

Initially, training was conducted using sparse rewards, where the agent received a reward only upon successfully reaching the target. The reward function used for this training was:

$$Reward = \begin{cases} +1 & \text{if } \texttt{touchedtarget} \\ 0 & \texttt{otherwise} \end{cases}$$

However, this approach faced significant challenges. The agent struggled to complete the task due to the vast state space, which made it difficult to explore effectively, and the sparse reward structure, which provided limited feedback on intermediate actions. As a result, the agent was unable to consistently learn a reliable policy for solving the task within the given environment.

We then iterated by integrating reward shaping and retrained the previously trained model adding the following rewards.[14]

$$Reward = \begin{cases} +1 & \text{if } \texttt{targetreached} \\ -1 & \texttt{tablehit} \\ -0.01 & \texttt{ArmxTarget} \\ -0.01 & \texttt{ArmxTable} \end{cases}$$

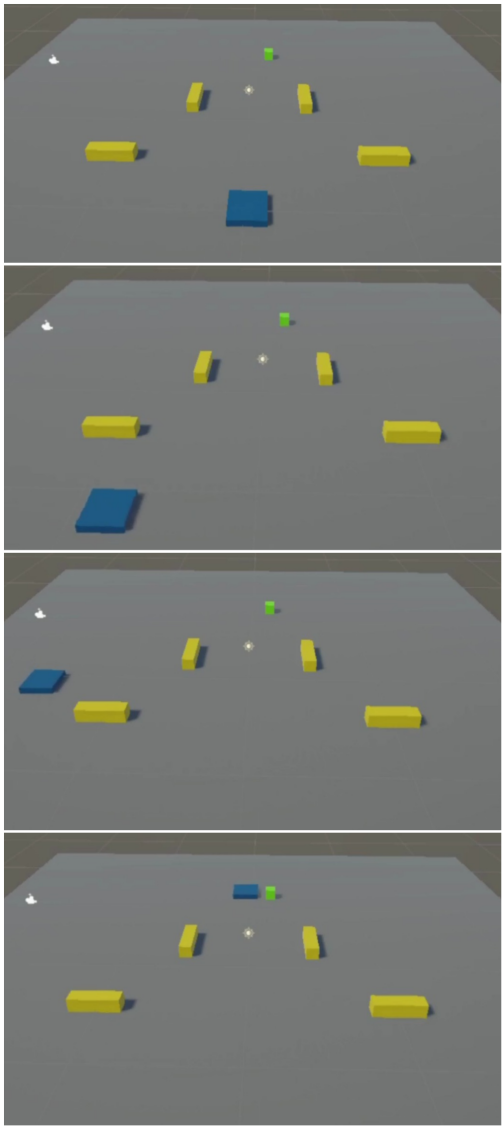

Fig. 11. The behavior learnt by the agent in the environment with fixed obstacles. As seen from top to bottom, the agent only learnt to avoid the obstacles by going around them even when the target position is on the right side of the room.

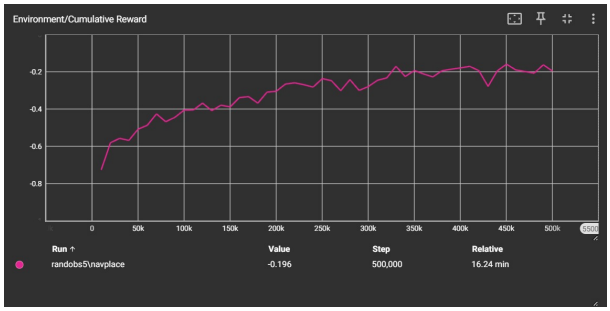

Fig. 12. Reward vs Episodes graph for the agent in the environment with random obstacles. The reward obtained due to the penalty, but overall reward of the agent improved as training progressed

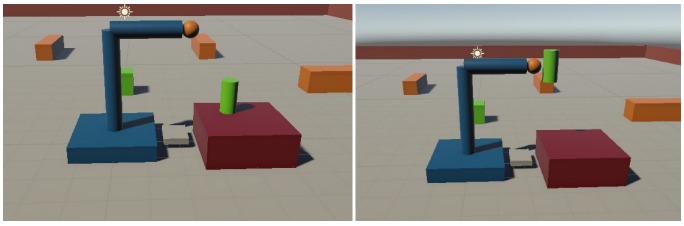

Fig. 13. Pick Agent (a): Before picking (b): After Picking

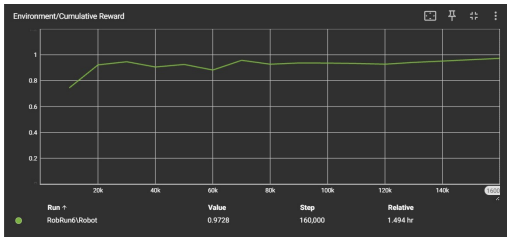

Fig. 14. Reward vs Episode graph for the case of picking the target with curiosity-driven reward added to previously trained model.

The agent gets a high negative reward when it hits the table or the floor. It is further penalized with a negative reward if the any of the robot arms hit the target or the table. This enabled the agent to improve its learning process. However, the policy the agent learned was still suboptimal , thereby performing an inefficient operation. In the final iteration, we implemented the Intrinsic Curiosity Module (ICM) to the pre-trained model to address the challenges posed by the sparse reward structure and the large state space. The ICM introduced curiosity-driven exploration, enabling the agent to seek out and learn from unexplored states while providing intermediate rewards based on the agent's prediction error. This mechanism encouraged the agent to actively explore its environment, leading to more effective training by balancing extrinsic rewards with intrinsic motivation to discover new strategies and refine its policy.

Hyperparameter	**Value**
Batch Size	10
Buffer Size	100
Discount Factor (γ)	0.99
ϵ	0.2
β	5e-3
Learning Rate	3e-4
Learning Rate Schedule	Linear
β Schedule	Constant
ϵ Schedule	Linear
No. of Layers in Policy Network	2
No. of Hidden layers	128
Curiosity Reward	0.1
Curiosity Encoding Size	128
Curiosity learning rate	1e-4
Curiosity Discount Factor (γ)	0.99

TABLE II
HYPERPARAMETERS FOR PICK OPERATION

C. Phase 3 : End-to-End Integration

After independently training agents for each of the three tasks, the final phase integrates them into a unified sequence-based workflow. The task sequence proceeds as follows: the navigation agent is initialized at its starting position and completes its goal of reaching the target location. Upon successful completion of the first task, the second task activates, enabling the agent to perform the pick operation. Once the pick task is completed, the final navigation task begins, where the agent navigates from the pick position to the designated place position. These tasks are executed sequentially, culminating in the successful completion of the overall objective. After approximately 75 diverse iterations, we achieved the desired trained behavior as planned. 15

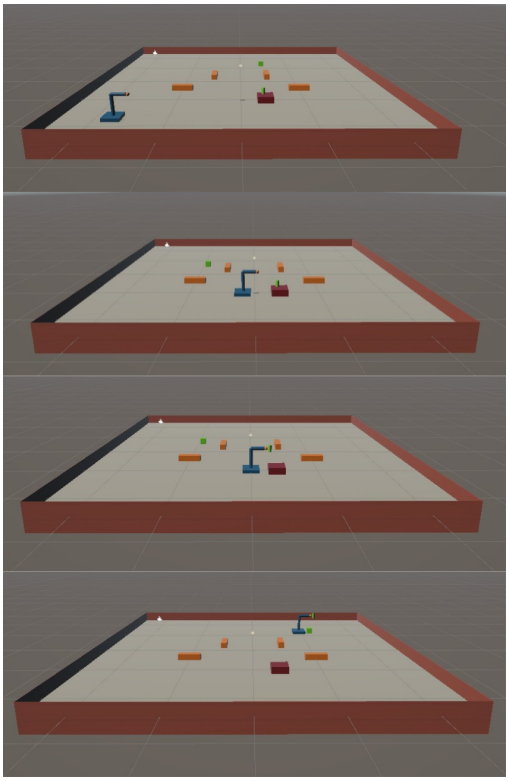

Fig. 15. Top to Bottom: Spawn Position of Agent , Agent Executing navigation towards target object (Green Cylinder), Agent Picking the object , Agent Navigating navigation towards drop location

V. FUTURE SCOPE

Future work on this framework includes integrating Large Language Models (LLMs) to interpret high-level human commands, decompose them into actionable subtasks, and dynamically design reward functions, enabling robots to execute complex, multi-step tasks with enhanced adaptability. Additionally, real-world deployment of the system on physical robots in household environments will test its scalability, robustness, and ability to handle real-world challenges such as hardware constraints and sensor inaccuracies. Furthermore, incorporating Augmented Reality (AR) interfaces could provide intuitive mechanisms for users to command and monitor robotic actions, offering a transparent and interactive way to guide task execution while enhancing user experience.

VI. Conclusion

Through this project , we implemented our idea of implementing **High-to-Low-Level Sequential Task Planning and execution for Mobile Manipulators using RL** with a solid understanding of **Curriculum learning, Reward Shaping, Intrinsic Curiosity Module and Proximal Policy Optimization(PPO)**. In future, we plan to implement a combined framework of a language model and an RL agent to solve the problem of converting natural language into reward functions and training the agent to optimize a policy in a given environment.

VII. Acknowledgments

We would like to express our sincere gratitude to **Prof. Yanhua Li** for his invaluable guidance, insights, and support throughout the course of this project. His mentorship played a pivotal role in shaping our understanding and execution of this work. We are also deeply thankful to our Teaching Assistant (TA), **Mingzhi Hu**, for her constructive feedback and assistance during the course, which greatly enhanced the quality of our work. Lastly, we extend our heartfelt thanks to our friend, **Mr. Mohit Amrutlal Patel**, for his encouragement and helpful discussions, which significantly contributed to the success of this project.

References

[1] Runyu Ma, Jelle Luijkx, Zlatan Ajanovic, and Jens Kober. "ExploRLLM: Guiding Exploration in Reinforcement Learning with Large Language Models." *arXiv preprint* arXiv:2403.09583, 2024. Available at: https://arxiv.org/abs/2403.09583.

[2] Murtaza Dalal, Tarun Chiruvolu, Devendra Chaplot, and Ruslan Salakhutdinov. "Plan-Seq-Learn: Language Model Guided RL for Solving Long Horizon Robotics Tasks." *arXiv preprint* arXiv:2405.01534, 2024. Available at: https://arxiv.org/abs/2405.01534.

[3] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances." *arXiv preprint* arXiv:2204.01691, 2022. Available at: https://arxiv.org/abs/2204.01691.

[4] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. "Language to Rewards for Robotic Skill Synthesis." *arXiv preprint* arXiv:2306.08647, 2023. Available at: https://arxiv.org/abs/2306.08647.

[5] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control." In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026-5033, 2012. doi: 10.1109/IROS.2012.6386109.

[6] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. "Unity: A General Platform for Intelligent Agents." *arXiv preprint* arXiv:1809.02627, 2020. Available at: https://arxiv.org/abs/1809.02627.

[7] Vitalii Sopov, Ilya Makarov. "Reward Shaping for Deep Reinforcement Learning in VizDoom".
Available at: https://ceur-ws.org/Vol-3094/paper_17.pdf.

[8] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, Trevor Darrell. "Curiosity-driven Exploration by Self-supervised Prediction." *arXiv preprint* arXiv:1705.05363, 2017. Available at: https://arxiv.org/pdf/1705.05363

[9] Andrew Levy, Robert Platt, Kate Saenko. "Hierarchical Reinforcement Learning with Hindsight."
Available at: https://openreview.net/references/pdf?id=ryQPmm9AX

[10] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. "Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey." *arXiv preprint* arXiv:2003.04960, 2020. Available at: https://arxiv.org/abs/2003.04960.