Journal of Artificial Intelligence Practice (2025)
Clausius Scientific Press, Canada

DOI: 10.23977/jaip.2025.080313
ISSN 2371-8412 Vol. 8 Num. 3

Prioritized Reward of Deep Reinforcement Learning Applied Mobile Manipulation Reaching Tasks

Zunchao Zheng[1,a,*]

[1]*Intelligent Process Automation and Robotics Lab, Karlsruhe Institute of Technology, Karlsruhe, Germany*
[a]*zheng@ira.uka.de*
**Corresponding author*

Keywords: Mobile Manipulation, Deep Reinforcement Learning, Reward Engineering

Abstract: In this paper, we apply deep reinforcement learning (DRL) for reaching target positions with a mobile manipulator while coordinating the mobile base and the manipulator and study the performance of different reward functions to get a higher success rate and more efficient movement. The reward is basically defined by the function of distance between the robot and the goal. We propose principles to build reward functions based on geometric series theory and discuss possible reward forms combined with different elements. We also present a prioritized reward function for mobile manipulation to weight movements of different parts and further provide a method to define the weights. Experiments are carried out in both two-dimensional and three-dimensional collision-free environments, and a further investigation into a relative task of going through an opening doorway is evaluated in the end.

1. Introduction

A mobile manipulation system consists of a mobile platform, manipulators, and grippers. The robot has the advantages of both mobility and dexterity. The mobile part extends the task space for the manipulator. However, the planning and control of the mobile manipulator becomes extremely difficult due to the high degrees of freedom. In this work, the robot prototype contains models of Neobotix MP500, Schunk LWA 4D arm and Schunk PG70 gripper.

Traditional model-based methods require precise models for the robot. [1] releases an open-source implementation of WBC framework which is successfully applied on the PR2 mobile manipulator for motion control and simple human robot interaction tasks. [2] builds a controller architecture for dynamic whole-body mobile manipulation of DLR's Justin.

The framework provides robust task execution that can be defined in the intuitive, low-dimensional Cartesian space, and saves planning time. The research and implementations of whole-body control models for Justin are deeply studied in later works. [3] formulates the Model Predictive Control (MPC) problem for tracking tasks in mobile manipulation. An MPC module generates control inputs for the robot to follow an end-effector trajectory while respecting equality and inequality constraints. The problem is solved online, allowing it to deal with dynamic scenarios and unforeseen events. Constraints such as acceleration, velocity, position, collision avoidance, and

Field-of-View are enforced while maintaining the linear-quadratic formulation of the problem. [4] presents a whole-body optimal control framework to jointly solve the problems for a dynamically balancing mobile manipulator. The optimization is performed using an SLQ-MPC approach, and the controller is formulated by incorporating a variety of tasks. The method actively corrects for dynamic instabilities, making the system dynamically stable while manipulating. The approach requires torque controllable actuators, yet these are unavailable in most high-payload robots. [5] presents an MPC method which plans the whole-body motion for a mobile manipulator while avoiding collisions. Collision avoidance uses visual information. The method is faster and produces shorter whole-body paths than some sampling-based planners RRTX and BiFMT.

For purposes of stability and accuracy, robot dynamics require careful calibration and precise formulations. The models are generally simplified, but the modelling process is still very complex. Once the model is applied to another similar but different robot, or when wear and tear occurs after a long time running, the parameters need to be well tuned, or even the dynamics are rebuilt. Considering the physical properties such as friction between wheels and the ground, it is hard to represent the changes due to the physical complexity and imprecise measurement.

For an instance of MPC based implementation, we apply the mobile manipulator with OCS2 (Optimal Control for Switched Systems) [6] toolbox which provides solvers of various optimal control problems, such as Stochastic Linear Quadratic, Iterative Linear Quadratic Regulator, Sequential Quadratic Programming, Stochastic Linear Programming and Integrated Pest Management. It is developed by the ADRL team at ETH Zurich and maintained by RSL at ETH Zurich.

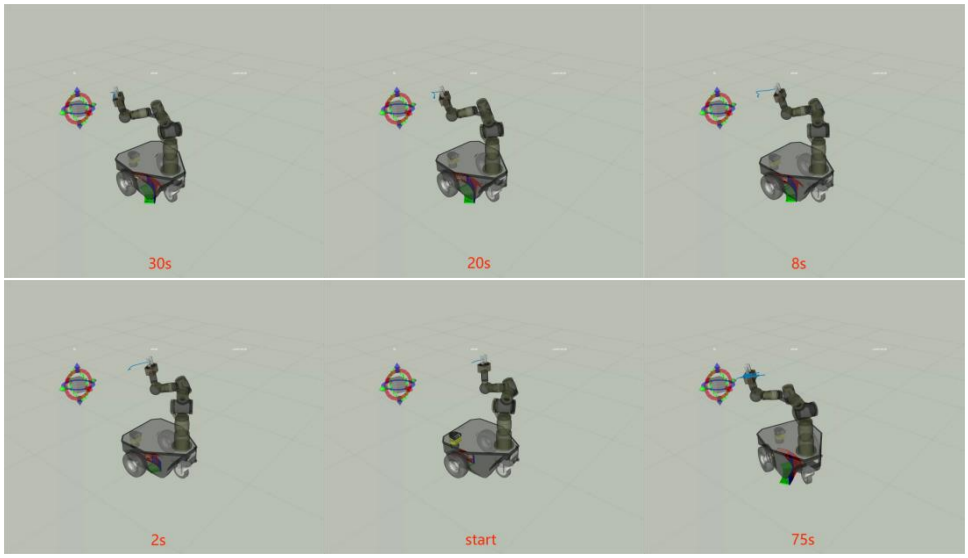


Figure 1: Path steps of the robot moving to the target pose, problem is solved by OCS2.

We sample some target poses for the robot. The robot can find solutions if the goal is given properly, but in most cases, the robot moves fine at the beginning, but the solution is given slowly after a particular path point. Sometimes it takes minutes, and the goal is still not reached, and the robot is wandering at some points near the goal in the end. As shown in Figure 1, in the sub-figures, the blue line is the end-effector's trajectory, and the red arrows are the base trajectory optimized by the MPC method.

Since traditional planners struggle in high dimensional space under motion constraints, a general solution is difficult to yield. We try to plan and optimize robot motions directly from robot behaviors without knowledge of robot kinematics, dynamics, or control models, so learning-based methods are developed. In recent years, DRL has been studied to solve complicated robotics

problems, such as reaching, pushing, pick-placing, grasping, manipulation, etc. Combining powerful deep learning to represent raw data and policy, DRL outputs optimal policies and generates a sequential trajectory of states and actions in high dimensional and continuous space. At the beginning, DRL is applied to deal with raw data from sensors. In our work, the environment is simulated and known. Instead of perceptions, we learn directly from low dimensional feature vector observations of different physical units such as robot poses and joint states which are easily recorded from the robot. One of the core parts in reinforcement learning is a reward function which guides the agent to learn more efficiently. As the reward function represents the performance of different behaviors from one state to another, a proper reward function scales up the potential to solve larger and more complex problems.

2. Motivation

The motivation in this work is to design reasonable reward functions to solve mobile manipulator reaching by DRL agents which are trained with state-of-the-art DRL algorithms of Deep Deterministic Policy Gradient (DDPG) [7], Twin Delayed DDPG (TD3) [8] and Proximal Policy Optimization (PPO) [9]. It doesn't matter which agent we use, as the reward function is the key point, any agent should work.

We start by analyzing basic reward functions used widely, where the rewards are based on the Euclidean distance of robot and goal. Based on geometric series theory, we evaluate the performance of different reward functions by analyzing the success rate and conclude principles of defining reward function. Considering the special structure of a mobile manipulator, we propose a prioritized reward function with which the movement of mobile base is prioritized when the robot is far from the goal, and the manipulator dominates gradually the movement when the robot moves close to the goal. In this way the mobile manipulator reaches the goal faster, and the results are evaluated in simulation. The experiments are carried out on simulation as the training of real robot costs a lot both in time and safety.

3. Related Work

Due to slow convergence in most reinforcement learning tasks, the reward function plays an important role in improving the learning. Progress estimators [10] provide a partial, goalspecific metric. In the experiments, three mobile robots try to find and grasp all the pucks into 'home' region while avoiding colliding with each other. The method designs different rewards for different behaviors, and evaluates the weighted rewards, but the results have no obvious acceleration in the learning process. [11] uses different reward structures when representing a reinforcement learning or exploration problem. [12] discusses the binary and continuous reward function with initial Q-values to solve goal-directed tasks. [13] investigates positive and negative potential functions with different discount factors on potential-based reward shaping. An end-to-end RL approach to whole-body control of manipulator is applied in [14]. The learned policies steer a mobile manipulator to target poses while avoiding obstacles. The input consists of data which are processed by neural networks from raw 2D LiDAR, robot states and the target position. A simulated randomized corridor environment is used for training models. The action space is discretized, and some arm joints are deactivated. The learning is simplified into two-dimensional levels. [15] uses a whole-body learning policy to control the robot and train the policy in a simulator. With the learned policy, the mobile manipulator can solve mobile picking tasks autonomously only based on the onboard sensors. The learned policy is tested with different objects in various scenarios. Unlike end-to-end learning, the visual perception part is separated from the DRL training.

The robot reaches a random goal from a random state in an environment, and the problem is

generalized by DRL agents. The main attribute in the task is the distance between the robot and the goal. The distance is an efficient heuristic to guide the robot reaching the goal, and we can make full use of this information. The reward of Euclidean distance between the arm and the goal is mainly considered in the above applications of mobile manipulator, and few study on the reward function itself. As main contribution of this paper we propose a prioritized reward function for mobile manipulator, where both the influences from the mobile base and arm are weighted. Furthermore, we present a method to define the weights.

4. Methodology

4.1. Problem Formulation

(1) **Motion planning**: Motion planning is a problem of planning collision-free motions from start state to goal state while satisfying constraints on motions or control inputs. The planning is in configuration space or $\mathcal{C}-$space. Every configuration of the robot is represented as q in $\mathcal{C}-$space. $\mathcal{C}_{\text{free}}$ consists of the configurations where collisions never happen. In this work we consider the planning only in $\mathcal{C}_{\text{free}}$ space. The definition of motion planning is:

Given start state s_{start} and goal state s_{goal} , find a time T and a set of controls $u:[0,T]\in\mathcal{U}$ such that $s_0=s_{start}$, $s_T=s_{goal}$ and $\mathrm{q}(\mathrm{s_t})\in\mathcal{C}_{\text{free}}$ for $\mathrm{t}\in[0,T]$.

(2) **Reinforcement Learning**: RL solves a sequential decision-making problem. It is generally modeled as Markov decision process (MDP) which is a tuple $\langle\mathcal{S},\mathcal{A},\mathcal{P},r,\gamma\rangle$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P}$ is transition probability function, r is reward function, $\gamma\in(0,1)$ is discount factor. At each time step t , an agent receives observation s_t and reward r_t from environment and performs action a_t to the environment by policy $\pi(s_t,a_t)$. A policy $\pi:\mathcal{S}\rightarrow\mathcal{P}(\mathcal{A})$ maps state to a probability distribution over the action. The goal of RL is to find the optimal policy to maximize the expected future return for each state, where the return is the discounted accumulated reward with γ, as shown in (1). The discount determines how rewards perform in the long run.

$$R_t=\sum_{k=0}^{\infty}\gamma^k r_{t+k} \tag{1}$$

Value function, Q-value function, and advantage function are defined by:

$$\mathrm{V(s)}=\mathbb{E}[\mathrm{R_t}|\mathrm{s_t}=\mathrm{s}] \tag{2}$$

$$Q(a,s)=\mathbb{E}[R_t|s_t=s,a_t=a] \tag{3}$$

$$\mathrm{A(s,a)}=\mathrm{Q(s,a)}-\mathrm{V(s)} \tag{4}$$

(3) **Robot Reaching**: Robot reaching aims to find motion planning solutions for the robot to reach any goal from any initial state in a workspace. The DRL agent yields general solutions without giving precise robot models of dynamics or control, the solution is a trajectory of states and actions which is also a path from start to goal state.

$$s_{start}\overset{a_0}{\rightarrow}s_1\overset{a_1}{\rightarrow}s_2\overset{a_0}{\rightarrow}\cdots s_{goal} \tag{5}$$

4.2. Agents

(1) **DDPG**: DDPG is a model-free off-policy actor-critic algorithm, which could be applied to continuous action space by parameterizing the actor function which deterministically maps the state to a particular action. The critic network estimates Q-values function using recursive Bellman equation as follows.

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma Q(s_{t+1}, \pi_\theta(s_{t+1})) \quad (6)$$

Where π_θ is actor which is represented as a parameterized network. The actor is trained to maximize the critic's Q-values. A noise is added to the action for exploration. With techniques of experience replay and target network, large neural networks are trained while avoiding divergence. DDPG performs well in many continuous control tasks.

(2) **TD3**: TD3 is presented since DDPG sometimes overestimate the Q values which lead to policy breaking. TD3 learns two Q-functions instead of one and updates policy network less frequently than Q network. Two Q-functions are shared with a single target, the smaller value of both Q values is chosen as the target value.

$$\begin{aligned} L(\theta_1) &= \mathbb{E}_{s,a\sim p}[(y - Q(s,a;\theta_1))^2] \\ L(\theta_2) &= \mathbb{E}_{s,a\sim p}[(y - Q(s,a;\theta_2))^2] \\ y &= \mathbb{E}_{s'}\left[r + \gamma(1-d)min(Q_{\theta_1}, Q_{\theta_2})\right] \end{aligned} \quad (7)$$

The policy is learned by maximizing one Q-function Q_{θ_1}.

(3) **PPO**: PPO is a model-free on-policy actor-critic algorithm, which improves the training stability by updating policy at each step and minimizes the cost function while ensuring a small deviation from the previous policy. PPO uses a clipped surrogate objective which takes the minimum one between the original value and the clipped version in order to lose the motivation for increasing the policy update to extremes.

$$J^{CLIP}(\theta) = \mathbb{E}\left[min(\frac{\pi(a_t|s_t)}{\pi_{old}(a_t|s_t)}\hat{A}_{\theta_{old}}(s_t, a_t)), clip(\frac{\pi(a_t|s_t)}{\pi_{old}(a_t|s_t)}, 1-\varepsilon, \epsilon)\hat{A}_{\theta_{old}}(s_t, a_t))\right] \quad (8)$$

During the training, the network architecture runs with shared parameters for both actor and critic. PPO has been tested on a set of benchmark tasks and proved to produce better results with much greater simplicity and becomes the popular reinforcement learning algorithm.

In this work, the three algorithms are used in the experiments. We only care about the reward functions but not the agents, the reward functions should always work no matter which agent we use as long as the training is under the same experimental settings, so the hyper-parameter tuning is not discussed.

4.3. States and Actions

Since the mobile base has a differential drive, the observation space S is defined by relative position of base and goal p_b^g, relative position of end-effector and goal p_e^g, base orientation o_b, end-effector orientation o_e, base velocity v_b, joint angles of arms q. Since the robot is fully observed, the state is also the observation. The definition is as follows.

$$\mathrm{S} = \{\mathrm{p_b^g}, \mathrm{o_b}, \mathrm{v_b}, \mathrm{p_e^g}, \mathrm{o_e}, \mathrm{q}\} \quad (9)$$

The action space A is represented by the differences of joint angles $\dot{q}$, linear velocity of base, angular velocity of base, as (8), where v_b = (v_x, v_y, w_z).

$$\mathrm{A} = \{\dot{\mathrm{q}}, \mathrm{v_b}\} \quad (10)$$

4.4. Reward Functions

Reward function rt returns a real value at time step t where the robot moves from current state s_t to next state s_{t+1}. It is represented as sparse or dense value. Sparse reward is the basic form in

reinforcement learning, the robot gets a positive value if reaching the goal, otherwise gets no reward or negative reward. The sparse reward gives little useful information during the training, so it is suitable for small discrete tabular problems. For high-dimensional problems, the dense reward is chosen instead of sparse one, normally it is defined by a continuous function of distance from current state to goal state. A continuous reward function is formulated in positive or negative forms. The agent with positive reward function accumulates rewards as much as possible to avoid fast termination, in this way, the agent may be trapped in local optima where it keeps repeating the actions to accumulate total positive rewards. Negative reward stops the episodes quickly to avoid getting more penalties. Motion planning requires the shortest path. As a result, a negative reward is chosen to define reward function, or a positive reward is used by giving penalties of time steps.

(1) **Base Reward Function**: The reward function in reaching task could be formulated basically by $r = f(d)$. OpenAI gym [16] provides classic DRL environments, where the reaching tasks(e.g., FetchReach-v1) define the reward as negative Euclidean distance. Similar definitions show in [17-19]. As shown in Figure 2a, a monotonic decreasing function of distance between the end-effector and the goal, such as $r = -d^2$, $r = -ln(d)$, $r = 1/d$, $r = e^{-d}$, etc. The principle to build a reward function is, the agent gets more rewards by moving to the goal. The reward increases faster in the state near the goal than that in the distant area, so that the agent explores more in the remote space in case of traps caused by self-collision or boundary limits.

As derived in (11), the return is a normal geometric series formula given the maximum reward r_{max} in episodes and the discount factor $\gamma \in (0, 1)$. In order to maximize the expectation of return, r_{max} is given at the terminating state where the robot reaches the goal successfully. r gets maximum when $d_{arm} = 0$, that is, $r_{max} = f(0)$. As a result, if r is always non-positive, then $r_{max} = 0$ satisfies the above objective; if r has positive values, then $r_{max} = \infty$ satisfies.

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \leq \sum_{k=0}^{\infty} \gamma^k r_{max} = \frac{r_{max}}{1-\gamma}, \gamma \in (0,1) \tag{11}$$

(2) **Hybrid Reward Function**: A single reward function is not enough to describe how good the current state and the corresponding action are, as different actions on s_t may lead to a same s_{t+1}. Furthermore, the reward may change little in some areas due to small gradients, as shown in Figure 2a. Besides the distance, there are other elements to consider, for example, the penalty of time steps and energy consumption or negative squared Euclidean norm of the action. They could be expressed in different magnitudes and weighted to show relative importance. The base reward functions can also be hybrid with coefficients at different distances, as small gradients of some functions may result in slow learning. Practically it is difficult to choose the best coefficients for the reward function. The choice is empirical. We generally scale the coefficient up if one parameter plays an important role in learning, and vice versa.

$$\mathrm{r_{hybrid}} = \sum_{\mathrm{i}=0}^{\mathrm{N}} \mathrm{c_i r_i} + \sum_{\mathrm{j}=0}^{\mathrm{M}} \mathrm{w_j p_j} \tag{12}$$

Where r_i represents different base reward functions of distance, p_j means different penalties.

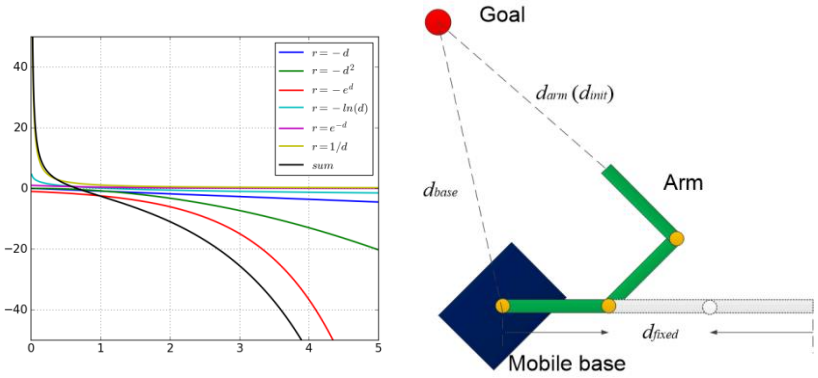
(3) **Prioritized Reward Function**: Prioritized reward function is presented due to the special structure of mobile manipulator. When mobile manipulator moves to a goal, the mobile base moves simultaneously with the arm, so the evaluation of base movement is also considered. The idea is the two parts of the robot dominate the movement at different moving stages by applying weights. An approach of setting $f(\tau)$ is proposed to define the weights, as formulation (13).

$$\tau = \frac{d_{arm}}{d_{init}}, if\ d_{base} > d_{fixed} \tag{13}$$

Take d_{fixed} and d_{init} as thresholds, where d_{fixed} is a threshold which is smaller than maximum distance the arm can reach in base frame and d_{init} is the initial distance between the end-effector

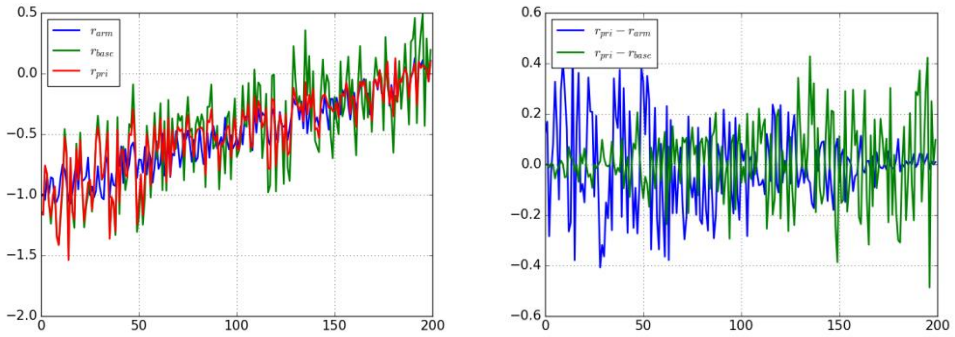
and the goal. As shown in Figure 2, d_{arm} is the spatial distance between the end-effector and the goal, d_{base} is the planar distance between the base and the goal, so prioritized reward function is formulated as (14). In the equation, r_{arm} is function of d_{arm}, r_{base} is function of d_{base}, they share a same base reward function. Since d_{arm} distributes in the region around d_{base}, the weights w_1, w_2 or w_3 scales and guarantees the monotonicity of reward functions in different levels, the basic relations are $w_1 \geq w_2 \geq w_3 > 0$ if r $>$ 0 and $w_3 \geq w_2 \geq w_1 > 0$ if r $\leq$ 0. There are three stages weighted by above w in the function, which shows clearly the movement relations of the base and the arm. From the function, the arm's behavior guides the final reward if d_{base} is within $(0, d_{fixed}]$. The base movement affects the final reward more and the arm's movement affects less while the arm moves far from d_{fixed} gradually. The base's behavior dominates the final reward if the arm runs beyond d_{init}.

$$r_{pri} = \begin{cases} w_1 * r_{arm}, if\ d_{base} \leq d_{fixed} \\ w_2 * [(1 - f(\tau)) * r_{arm} + f(\tau) * r_{base}], if\ d_{base} > d_{fixed}\ and\ \tau < 1 \\ w_3 * r_{base}, if\ d_{base} > d_{fixed}\ and\ \tau \geq 1 \end{cases} \quad (14)$$



(a) (b)

Figure 2: (a): The base reward functions. (b): Distance definitions.



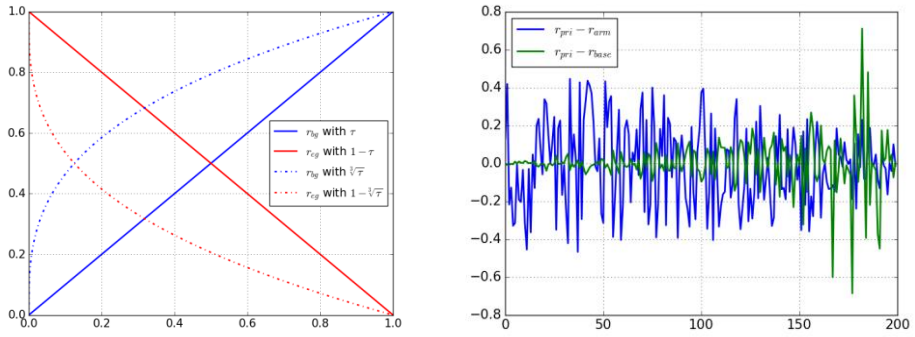
(a) (b)

Figure 3: (a): Prioritized reward changes with r_{arm} and r_{base}. (b): Prioritized reward errors with r_{arm} and r_{base}.

We make the linear function of Bezier curve as a reference, so in prioritized reward function, the core formulation is $(1 - f(\tau)) * r_{arm} + f(\tau) * r_{base}$, which describes the relations between base and arm movements.

To visualize the formulation and compare the prioritized reward and corresponding separate reward, the reward at each time step is calculated by the distance changing from 1 to 0 with

Gaussian noise so that the positions of the robot do not change smoothly, which means the arm has multiple positions when the base stays in one state. The rewards are computed overall 200-time steps. Figure 3a shows the changes of r_{pri}, r_{arm} and r_{base}, where $r_{arm} = -d_{arm}$ and $r_{base} = -d_{base}$. r_{pri} in red line follows the changes of r_{base} in green line at the beginning. With the time steps increase, r_{pri} changes along as the blue line of r_{arm} changes. In Figure 3b, the blue line shows the error between r_{pri} and r_{arm}, and it increases gradually. The green line shows the error between r_{pri} and r_{base}, and it deceases along the time steps increase. In other words, r_{pri} left follows the changes of r_{base} in green line at the beginning, with time steps increase, r_{pri} changes along as the blue line of r_{arm} changes. It means, with the prioritized reward, at the beginning the mobile base has the priority of the movement, and the arm dominates the robot action gradually while the robot is getting closer to the goal.



(a) (b)

Figure 4: (a): f(τ) in prioritized reward acts on r_{arm} and r_{base}. (b): Reward errors with weights of $f(\tau) = \sqrt[3]{\tau}$.

We assign the priorities by applying different functions of τ as weights, then (15) is the basic form of weight function. The solid lines in Figure 4a shows how τ influences the base reward and arm reward. When r_{base} increases, r_{arm} decreases. Take $f(\tau) = \sqrt[3]{\tau}$ as an example, as the dot lines shown in Figure 4a, 87.5% of the case the base reward has higher weight, so the base dominates the movement most time, vice versa. The errors between r_{pri} and r_{base} in Figure 4b keep small for a long time, which means the base dominates the movement in most time of the episode.

$$f(\tau) = \tau^n, n = N \; or \; n = 1/N, N = 1,2,3,\cdots \tag{15}$$

It is not deterministic which n is used in the application. A basic principle is, the larger the space is, the smaller the parameter n is.

5. Experimental Results and Discussion

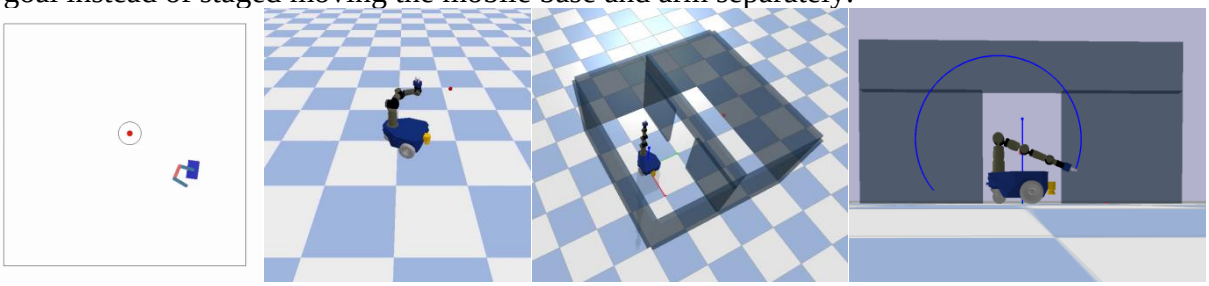
5.1. Environment Setup

We build two environments in both two-dimensional and three-dimensional space for DRL agents to learn the optimal reaching trajectory. Figure 5a is a robot running in two-dimensional planar based on OpenAI gym rendering. In this environment, the red dot is the goal, the black circle is the reaching boundary, the blue box is mobile robot which moves by translation and rotation, the 3-link arm rotates around the joints. The three-dimensional robot in Figure 5b is setup in Pybullet [20]. In the environment, the red dot is the goal which is reset randomly in a space height of $[0.5, 1.5]$. In reaching tasks, the goal and robot state are randomly initialized. In these collision-free

environments, the space range is $[-1, 1]$. For equation (15), $n = 1$ is used.

A scenario of two separated rooms connected with an opening doorway is built. In the 3D scenario with walls, the robot tries to run from one room to another while going through an opening doorway.

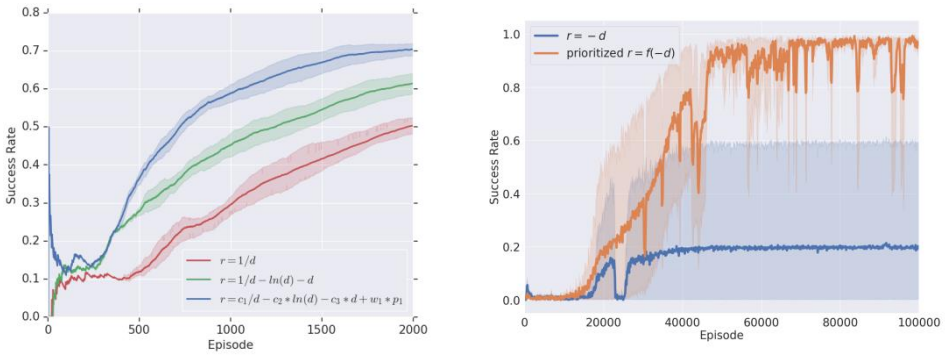
Figure 5c is the full view of the 3D scenario with walls. The size of outer wall is $[4, 0.1, 2]$ in $[length, width, height]$, the size of inner doorway is $[1, 0.1, 2]$ where the length is a little wider than the length of the robot base. For equation (15), $n = 1/3$ is used since the space range is bigger, the base moves mainly in distant areas. Figure 5d is the front view of the robot and inner wall, where the robot base is at the original point and the blue line is the projection on the inner wall of the maximum distance the arm can reach when the second joint is actuated. Figure 5d shows that the robot cannot go through the doorway when the arm is initialized in some positions. For example, if the arm is straight up, it is impossible for the robot to go through the doorway without moving the arm, the arm must change the joint positions in order to avoid colliding with inner walls. The 3D scenario with walls tells that in some tasks the mobile base and arm must move together to achieve the goal instead of staged moving the mobile base and arm separately.



(a) (b) (c) (d)

Figure 5: Simulated environments and scenario with walls and opening doorway. (a) 3-link planar robot. (b) 3D robot. (c) 3D scenario with walls. (d) Front view of 3D scenario with walls.

5.2. Success Rates of Hybrid Reward Function



(a) (b)

Figure 6: (a) Mean success rates of base and hybrid reward. (b) Mean training success rates with base and prioritized reward functions.

We train robot reaching two-dimensional space with different forms of hybrid reward functions. As shown in Figure 6a, with particular coefficients, the success rate of hybrid reward $r = c_1/d - c_2 * \ln(d) - c_3 * d + w_1 * p_1$ is higher than that of single r. The coefficients ci here is randomly set between (0, 1) during the training. In conclusion, a hybrid function with well-tuned coefficients outperforms a single reward function. In this work, we focus on the conclusions of different reward

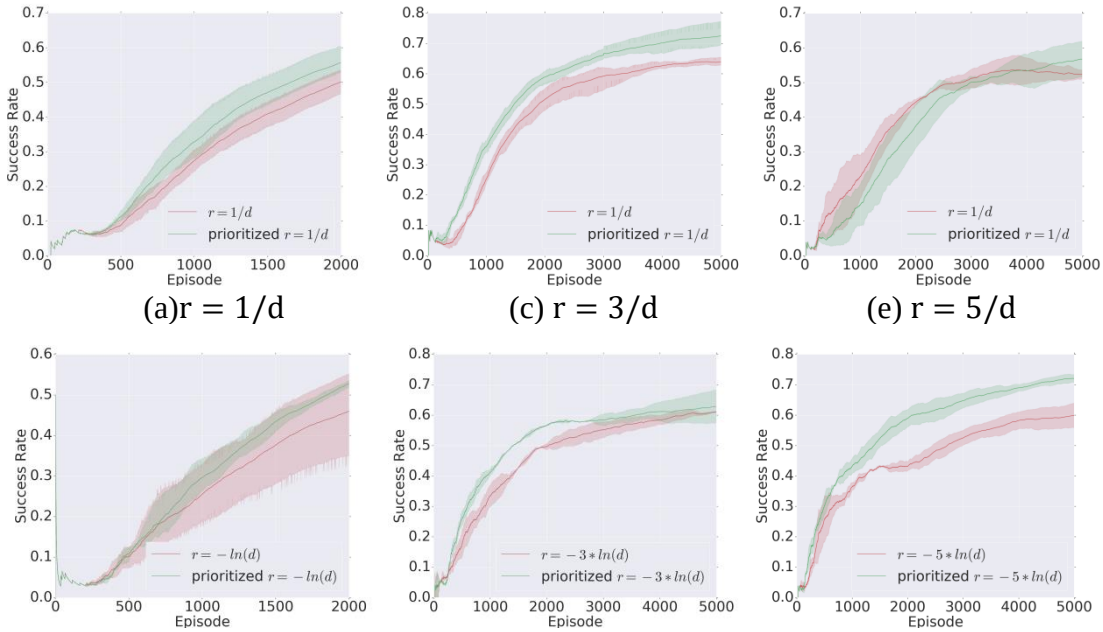
functions under the same experimental conditions, as a result, no best coefficients are given formally which also depend on experimental settings.

In these experiments, the success rate is not high enough in some tasks as the success rate is caught during the training, the failures in the training are accumulated, it can never reach 100%. In this case, we mark the experiments feasible if the success rate keeps going up.

5.3. Success Rates of Prioritized Reward Function

We train DDPG agent for two-dimensional robot reaching with base reward function and prioritized reward function. Figure 7 is comparison with the base and prioritized reward function of $r = -ln(d)$ and $r = 1/d$ with different coefficients in different workspaces.

Figure 7a and Figure 7b compare the success rates of base and prioritized ones within 2000 episodes in a workspace of $[-2, 2]$. Figure 7c and Figure 7d perform in a workspace of $[-3, 3]$. Figure 7e and Figure 7f show the results in a workspace of $[-5, 5]$. For each environment setup, we run three trials. For each comparison group, we use the same random seeds under the same DRL hyper parameter setting. The figures show that prioritized reward functions in green lines yield higher success rates than corresponding base ones in red lines.



(a)r = 1/d (c) r = 3/d (e) r = 5/d

(b)r = −ln(d) (d) r = −3ln(d) (f) r = −5ln(d)

Figure 7: Mean training success rates of base and prioritized reward functions in 2D environment.

We also train TD3 agents by Stable Baselines library [21] with base reward function and prioritized reward function in three-dimensional environment. The base reward function is r = −d, as the negative Euclidean distance is mostly used in the reward formulation for reaching tasks. The weights of prioritized function are $w_1 = 1, w_2 = 1, w_3 = 1$, and the workspace in planar space is $[-1.5, 1.5]$. Each group of experiments runs in five different seeds of $1, 2, 3, 4, 5$. The success rate is calculated every 100 episodes in the training, so sometimes it can reach 100%. In the figure, the agent with base reward gets success rate up to 60%, the other with prioritized reward has much higher success rate.

5.4. Mean Steps of Prioritized Reward Function

Table 1 evaluate from a view of mean steps per episode for all success reaching episodes in three trials of 3D robot reaching. Trial 1 has a total training time step of 5M, and the mean steps per episode with prioritized reward function is 101 steps compared to 109 steps with base reward function. In trial 2 and 3, the total training time steps is 10M, the mean steps per episode for all successful reaching episodes with prioritized reward are smaller than those with base reward. Table 2 shows mean steps per episode for all success reaching episodes in three trials of 3D scenario with walls. During the training, the robot base tries to go through the doorway reaching the goal of [0, 2.3, 0]. There is no target position for the arm, but the arm must move to avoid the walls. The mean steps are 518 in total 900 success episodes for base reward, and the mean steps are 453 in total 900 success episodes for prioritized reward. From the table, the prioritized reward makes the robot reaching successfully with fewer steps, which means the reaching is more efficient. From the tables, the prioritized reward makes the robot reach successfully with fewer steps, which means the reaching is more efficient.

Table 1: Mean steps per episode for all the success reaching episodes during the training of three trials of 3D robot reaching tasks.

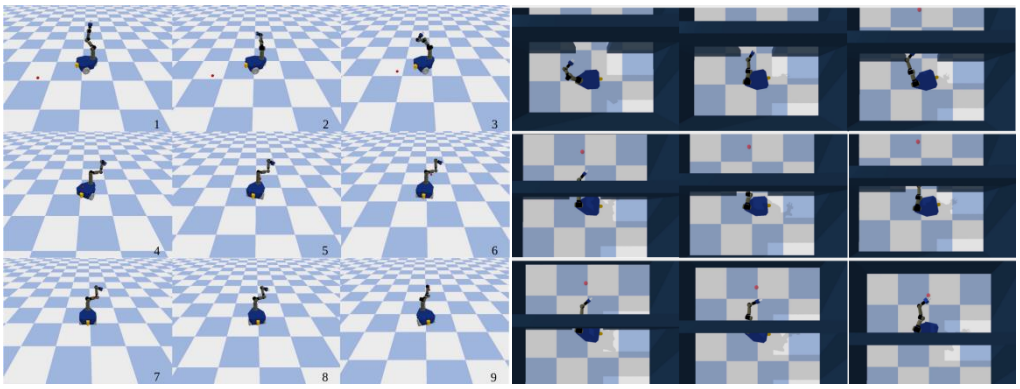
trial	base reward	prioritized reward
1	109	101
2	205	122
3	166	144

Table 2: Mean steps per episode for 900 success reaching episodes during the training of 3D scenario with walls.

	base reward	prioritized reward
mean steps	518	453

5.5. Trajectory Rollout

Figure 8a shows a trajectory trained by PPO in Pybullet, where $r = 1/d_{arm}$. Figure 8b is a trajectory for the robot reaching through the opening doorway, the rollout is from the same training result as that in Table 2.



(a) (b)

Figure 8: (a) Trajectory rollout for 3D robot reaching. (b) Trajectory rollout for 3D robot going through opening doorway.

6. Conclusions

In the experiments of deep learning, generally the return is evaluated by presenting whether it is converged smoothly or not, yet in our work the reward functions are different, so the success rate is evaluated instead. The success rate is not high enough in some tasks as the success rate is caught during the training, the failures in the training are accumulated. The reward function is the only difference under the same configurations of the comparing group, so it doesn't matter which agent we use. In this case, we mark the experiments feasible if the success rate keeps going up. Another point is about the coefficients and weights in the reward function, we may present a proper configuration for specific tasks after tuning, but it is still an unwanted tuning work since the hyper-parameters of the agents are already too many. In this work, we only give out the formulations and evaluate some examples to show the comparisons.

The contribution of this paper is, we have presented different reward functions for mobile manipulators in the task of reaching in different environments. According to the geometric series formula, we analyze the return distribution and proposed the principles to choose reward function. For mobile manipulators, we present prioritized rewards to optimize learning, with which the robot movement is weighted into two parts to move more efficiently. We also propose an approach to define the weights. The experiment is extended to a complex reaching task of going through a doorway. The evaluation shows that reaching with prioritized rewards gets higher success rates with fewer steps.

In the future, the experiments will be evaluated in more aspects, such as the relations between space range and time steps or the parameter n in equation (15), and more random seeds and trials for a task which might hardly have an end. And the coefficients and weights in the reward functions need to be evaluated in order to find the rules for better results. Furthermore, the workspace will be expanded so that the agent explores more space. The larger the space is, the higher the computing time is. In this case, the efficiency requires to be guaranteed. Finally, we would extend the study to more complex environments or tasks.

References

[1] R. Philippsen, L. Sentis, O. Khatib, "An open source extensible software package to create wholebody compliant skills in personal mobile manipulators," in 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011, pp. 1036–1041.

[2] A. Dietrich, T. Wimbock, A. Albu-Schaffer, G. Hirzinger, "Reactive whole-body control: Dynamic mobile manipulation using a large number of actuated degrees of freedom," IEEE Robotics & Automation Magazine, vol. 19, no. 2, pp. 20–33, 2012.

[3] G. B. Avanzini, A. M. Zanchettin, P. Rocco, "Constraint-based model predictive control for holonomic mobile manipulators," in 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1473–1479.

[4] M. V. Minniti, F. Farshidian, R. Grandia, M. Hutter, "Whole-body mpc for a dynamically stable mobile manipulator," IEEE Robotics and Automation Letters, vol. 4, no. 4, pp. 3687–3694, 2019.

[5] J. Pankert, M. Hutter, "Perceptive model predictive control for continuous mobile manipulation," IEEE Robotics and Automation Letters, vol. 5, no. 4, pp. 6177–6184, 2020.

[6] F. Farshidian et al., OCS2: An open source library for optimal control of switched systems, [Online]. Available: https://github.com/leggedrobotics/ocs2.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, et al., "Continuous control with deep reinforcement learning," arXiv preprint arXiv:1509.02971, 2015.

[8] S. Fujimoto, H. van Hoof, D. Meger, "Addressing function approximation error in actor-critic methods," CoRR, vol. abs/1802.09477, 2018. arXiv:1802.09477. [Online]. Available: http://arxiv.org/abs/1802.09477.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, "Proximal policy optimization algorithms," CoRR, vol. abs/1707.06347, 2017.

[10] M. J. Mataric, "Reward functions for accelerated learning," in Machine Learning Proceedings 1994, Elsevier, 1994, pp. 181–189.

[11] S. Koenig, R. G. Simmons, "The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms," Machine Learning, vol. 22, no. 1-3, pp. 227–250, 1996.
[12] L. Matignon, G. J. Laurent, N. Le Fort-Piat, "Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning," in International Conference on Artificial Neural Networks, Springer, 2006, pp. 840–849.
[13] M. Grzes and D. Kudenko, "Theoretical and empirical analysis of reward shaping in reinforcement learning," in 2009 International Conference on Machine Learning and Applications, IEEE, 2009, pp. 337–344.
[14] J. Kindle, F. Furrer, T. Novkovic, J. J. Chung, R. Siegwart, and J. Nieto, "Whole-body control of a mobile manipulator using end-to-end reinforcement learning," arXiv preprint arXiv:2003.02637, 2020.
[15] C. Wang, Q. Zhang, Q. Tian, et al., "Learning mobile manipulation through deep reinforcement learning," Sensors, vol. 20, no. 3, p. 939, 2020.
[16] G. Brockman, V. Cheung, L. Pettersson, et al., Openai gym, eprint: arXiv:1606.01540. 2016.
[17] S. Jauhri, J. Peters, G. Chalvatzaki, "Robot learning of mobile manipulation with reachability behavior priors," IEEE Robotics and Automation Letters, vol. 7, no. 3, pp. 8399–8406, 2022.
[18] F. Schmalstieg, D. Honerkamp, T. Welschehold, A. Valada, Learning hierarchical interactive multi-object search for mobile manipulation, IEEE Robotics and Automation Letters, 8(12):8549–8556, 2023
[19] T. Ni, K. Ehsani, L. Weihs, and J. Salvador. Towards disturbance-free visual mobile manipulation. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pages 5219–5231, 2023.
[20] E. Coumans, Y. Bai, Pybullet, a python module for physics simulation for games, robotics and machine learning, http://pybullet.org, 2016–2019.
[21] A. Hill, A. Raffin, M. Ernestus, et al., Stable baselines, https://github.com/hill-a/stable-baselines, 2018.