Assistive Dual-Robot System for Patient Care: Target Following and Object Retrieval

Zhengyang Wang, Chuyi Zhao

Abstract

In this research paper, we investigate the fundamentals of Deep Q-Networks (DQN) and Kalman Filters that we've discussed during our class. We apply the knowledge and focus the implementation of DQN algorithm for robotic grasping and using Kalman Filter for improved robotic navigation. We've also tried to alter the PyBullet Kuka environment to see the impact on the learning process of our agent and learn to adopt a Model Predictive Control for better path planning.

1 Introduction

1.1 Overview

We start our paper with presenting some studies of reinforcement learning applied to robotic grasping tasks, with a focus on the Deep Q-Networks (DQN) algorithms, and also include research that discuss robotic navigation. We then discuss the motivation of choosing Robotic Grasping through Reinforcement Learning and Robotic Navigation as the direction of our project. Following this, we first describe the PyBullet Kuka robot environment used in our study, highlighting its various parameters and the rationale behind altering them to analyze their impact on the learning process. Subsequently, we provide a brief overview of the DQN reinforcement learning algorithms, explaining their core concepts. Following this, we also detail the process of building the "Target Follower" Robot in the PyBullet simulation environment by using a TurtleBot.

Lastly, we present the findings of our experiments, showing the learning process of DQN in the context of robotic grasping tasks in the Kuka robot environment, and the performance of the target-following robot. We further discuss the implications of our results and identify limitations of our current research and possible areas for further improvement.

1.2 Motivation

Reinforcement learning and robotic grasping have gained significant attention in recent years, both in academia and industry, due to their potential to revolutionize various applications such as manufacturing, logistics, and healthcare.

In our class, we have been introduced to many lectures and videos that emphasize the importance of reinforcement learning in robot grasping and navigation tasks. As someone with little prior knowledge of robotics and reinforcement learning, these resources have sparked our curiosity and motivated us to dive deeper into the subject.

The objective of this research paper is twofold: to contribute to the existing body of knowledge and to document our learning journey within this field. Although some aspects of this study may appear fundamental to experts, it serves as a valuable learning experience for us as we explore the nuances of reinforcement learning and its applications in robotic grasping and robotic navigation.

As mentioned in the overview section, our research aims to examine the Deep Q-Networks (DQN) algorithms in robotic grasping tasks and Kalman Filter in Robotic Navigation, seeking a deeper understanding of these functionalities that can be applied to real-world robotic systems. Through systematic experimentation and analysis, we endeavor to foster further exploration of reinforcement learning and robotic navigation, both within our class and beyond.

2 Background and Related Work

The inspiration for our research on creating an assistant robot primarily stems from the work presented in the paper ***"Assistive Gym: A Physics Simulation Framework for Assistive Robotics"***[6]. by Zackory Erickson, et al. This paper has significantly contributed to the field of assistive robotics by introducing an open-source physics simulation framework built on top of the OpenAI Gym [3] and the PyBullet [5]physics engine. The primary objective of Assistive Gym is to provide a standardized, accessible platform for researchers to develop, train, and evaluate robotic agents capable of assisting humans with physical tasks.

Erickson and his team have developed a series of simulated environments representing various assistance tasks, such as feeding, drinking, dressing, and hygiene assistance. These environments include human and robot models, as well as the necessary tools and objects for performing the tasks. The authors propose a set of benchmark tasks to evaluate the performance of different robotic control algorithms in the context of assistive robotics. The key contribution of this paper lies in facilitating research and development in assistive robotics, ultimately encouraging the creation of robotic systems that can improve the quality of life for individuals with disabilities and the elderly.

Two main resources that we used for learning the fundamentals of DQN and applying DQN to the Kuka Robot environment are the Reinforcement Learning (DQN) Tutorial [2] written by Adam Paszke and the Post written by Mahyar Abdeetedal [1]. In the tutorial, Adam discusses how to use PyTorch to train a Deep Q learning agent by providing an in-depth explanation of all the packages required for the training, the definition of Replay Memory, the DQN algorithm, and the hyperparameters and utilities for the training. Example codes are also provided for both the DQN algorithm itself and the training loop. Our work

on the DQN algorithm section and training mainly refers to this code. The post written by Mahyar provides us with the fundamentals of setting up and publishing our own environment and teaches us how to integrate the DQN algorithm into the PyBullet environment with example code.

In terms of the Target Follower Robot, we also referred to other related sources, such as literature reviews [7][8] of various notable healthcare robots in recent years, as well as some specific techniques, including a vision-based target-following robot [9], which combines Computer Vision with Robotics to make full use of the sensor information.

3 Methodology

3.1 Tasks

The original problem could be split into two parts. The task of "fetching something for the patient" could be defined as **Object Retrieval**, while "staying around the patient" could be achieved by developing a **Target Follower** robot, which chases the patient and keeps within a certain distance. In that case, we split the primitive attempt into two tasks and ended up accomplishing both separately.

3.2 Object Retrieval - Kuka Environment

We implement the Kuka Environment from the Bullet Physics Development Team [4]. This environment includes functions that can rest the environment at the beginning of an episode, randomly places the objects in the bin with object fall to the tray individual to prevent object intersection, return observation as an image for DQN training, control for the Kuka robot to attempt grasping behavior, and reward function where we reward our agent with 1 if the robot arm is above a height of 0.2 at the end of the episode. Also, based on the instruction provided by Mahyar Abdeetedal in his post "OpenAI Gym Environments with PyBullet [1], we've successfully uploaded our altered environment to the PyPI website so that we can use "pip" to directly install our desired package. The following figure shows an example of installing the package.

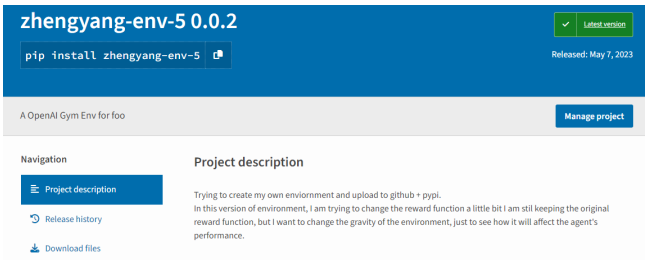


Figure 1: Instruction on installing the environment

3.3 Object Retrieval - DQN algorithm and Training Process

In terms of DQN algorithm and the training process, by following the instruction provided by the Reinforcement Learning (DQN) Tutorial [2], we explore a task where the agent is responsible for controlling a robotic manipulator in order to grasp an object. The agent has to decide between seven actions: moving the manipulator in the x or y directions, adjusting the vertical angle offset, or not moving at all. We assume that the velocity for each direction is equal and the gripper automatically moves downward for each action.

The agent observes the current state of the environment and chooses an action, causing the environment to transition to a new state while returning a reward that indicates the consequences of the action. In this task, the reward is set to 1 if one of the objects is above a height of 0.2 at the end of the episode (corresponding to the reward function in the Kuka environment). The PyBullet Kuka environment is designed to provide (48, 48, 3) images representing the environment state as inputs to the agent. To solve the task, we utilize neural networks that can infer the required actions purely by analyzing the scene. We employ a stack of consecutive screens as input to capture the dynamics of the environment, ultimately aiming to develop an efficient and versatile robotic system that can effectively perform grasping tasks.

In order to train our Deep Q-Network (DQN) for the robotic grasping task, we employ experience replay memory, a technique that has proven to be effective in stabilizing and improving the DQN training procedure. Experience replay memory stores the transitions observed by the agent, allowing us to reuse this data for subsequent training. By random sampling from the stored transitions, we can create decorrelated batches, which contribute to the stability of the training process. To implement experience replay memory, we introduce

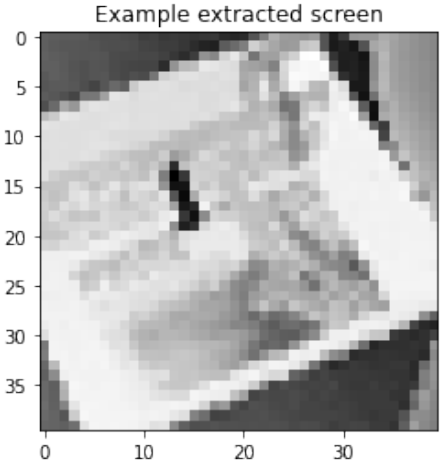


Figure 2: Example Extracted Image

two classes: The first one is Transition - A named tuple representing a single transition in our environment. Transition essentially maps (state, action) pairs to their corresponding (nextstate, reward) results. In our case, the state is represented by the consecutive stacked screens image, as described earlier. This class allows us to store and manage the transitions observed by the agent during the training process. Figure 2 gives us an example of the extracted image.

The other one is ReplayMemory - A cyclic buffer of the bounded size that holds recently observed transitions. ReplayMemory ensures that the buffer retains the most recent data while discarding older information once the buffer reaches its capacity. Additionally, the ReplayMemory class implements a .sample() method, which enables us to select a random batch of transitions for training purposes.

For our training process, the tutorial offers predefined hyperparameter values, such as setting the learning rate to 1e-4. Future explorations could involve modifying these hyperparameters to examine the differences in learning efficiency for our agent. However, as beginners in the DQN algorithm and reinforcement learning, our primary objective is to execute at least one successful training session and make adjustments to the environment to observe their impact on the agent's learning process.

We perform several steps to update the model during the training process. First, we ensure the memory size is sufficient for training by checking if the length of the memory is smaller than the batch size. If it is, we return from the function. If the memory size is adequate, we sample a batch of transitions from the replay memory and separate the individual components of the batch, such as states, actions, rewards, and next states. We then create a mask of non-final states to identify which next states are not terminal and compute the Q-values for the current state-action pairs using the policy network.

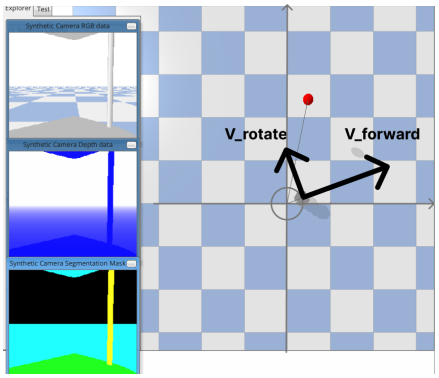
Next, we calculate the expected Q-values for the next states (V(st+1)) using the target network. For non-final next states, we select the maximum Q-value across all possible actions. We then multiply these next state values by the discount factor and add the reward batch to obtain the expected state-action values. Following this, we compute the Huber loss between the state-action values obtained from the policy network and the expected state-action values.

3.4 Target Following

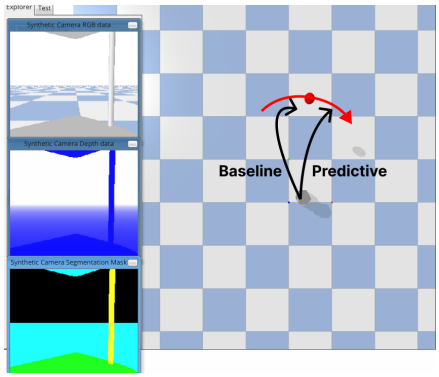
In terms of the target following part, we define the problem as a robot chasing and following a moving patient, through which manner stays by their side like a personal assistant.

In practice, we developed in the **PyBullet** simulation environment for the experiment, used a **TurtleBot** equipped with an **RGB camera** and a **depth camera** as the robot, and a moving target object, to simulate a moving human patient or some sensory object which the patient holds.

The **Baseline** motion control function for the TurtleBot is to obtain the real-time position information of the moving target object from sensors, and go over directly, which is shown in Fig 3a. In detail, the TurtleBot requires both



(a) Kinatics of the baseline motion control, combining the forward velocity and rotation velocity together to directly heading for the real-time target position.



(b) The predictive motion control, ideally going through a short path compared to the direct baseline control.

a velocity to move forward and an orthogonal velocity for rotation in order to reach the target position. That could be achieved by assigning the subtraction and summation of them respectively to the two wheels, which works totally fine for targets that are standing still or moving slowly.

However, the TurtleBot becomes inefficient as the target moves faster. Specifically, chances are that when the TurtleBot finally reaches a previous target position, the target object has already moved on to the next destination. In that case, the TurtleBot has to detour, turn around and head for the next position. Especially when the target speed is way higher than the TurtleBot, the TurtleBot has to turn around all the time, wasting a lot of time and still failing to keep close to the patient. It is impossible to merely raise the TurtleBot's speed or motor force since that makes the TurtleBot unstable and possible to collide or fall down. Therefore, we came up with the idea to improve the motion control function by adopting a **Model Predictive Control (MPC)** for better path planning, as shown in Fig 3b. Here the predictive control uses *Kalman Filter* to predict a possible future target position based on previous movements and lets the TurtleBot head for the predicted direction directly. As shown in Fig 3b, while the baseline path might be twisted, the predictive control tends to make a shortcut, thus taking more effect.

4 Experimental Results

4.1 Results for Object Retrieval

First, we would like to show that our model is actually running and learning to grab the object. We aimed to demonstrate the successful training of our agent using the Deep Q-Network (DQN) reinforcement learning algorithm for per-

forming robotic tasks. As discussed in the previous sections, we were primarily focused on obtaining a well-trained agent before making any modifications to the environment. One method for evaluating the performance of our model is to track the best mean reward achieved during the training process. Mean reward refers to the average reward obtained during a specific number of episodes in a reinforcement learning problem. It provides an indication of how well the agent is performing in the environment. Our mean reward is calculated as the average of the rewards obtained in the last 100 episodes, which is then multiplied by 100. It helps track the agent's performance over time, allowing us to understand if the agent is learning and improving its decision-making in the environment. Our environment is considered to be solved once the mean reward is above 50 and at least in 200 episodes.

Best Mean Reward Previous State	Best Mean Reward Current State
44	45
45	46
46	47
47	48
48	49
49	50
50	51

Table 1: Proof of Learning Process

Again, the table is purely for the purpose of showing that our model (our robot arm agent) is using the DQN Reinforcement learning algorithms and is improving from time to time.

Then to evaluate the model, we will use the rolling average (as shown in Figure 3). The rolling average of rewards is a way to smooth the reward data over a fixed window of episodes It is a simple evaluation metric to observe the performance of the agent during training. In the context of DQN reinforcement learning, the rolling average of rewards can be used to evaluate how well the agent is learning to maximize its cumulative rewards over a sequence of episodes. It can help in identifying if the agent is converging to a better policy. Figure 3 is a bit hard to see the trend of improvement, but we can still see that the overall trend is increasing. The default environment does take longer (3 hours) and more episodes (9412 episodes) to resolve compared to our altered environment.

After we've reached the goal of proving that we have an agent that has learned to grab objects through DQN learning and provided the related evaluation of the model, then we tried to alter the parameter in our environment. For example, we tried to change the gravity in the environment to see how that will impact the learning process of our agent. The Rolling Average Reward over 100 episodes looks like Figure 4.

This graph shows an overall trend of increase, it indicates that the agent is learning to improve its policy and achieve higher rewards over time. The change in the rolling average reward from 0.1 to 0.4 before episode 600 and the

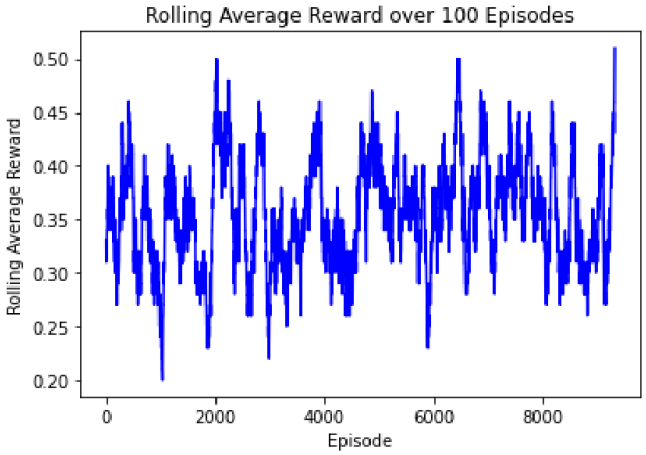


Figure 3: The Rolling Average Reward over 100 Episodes-Default Environment

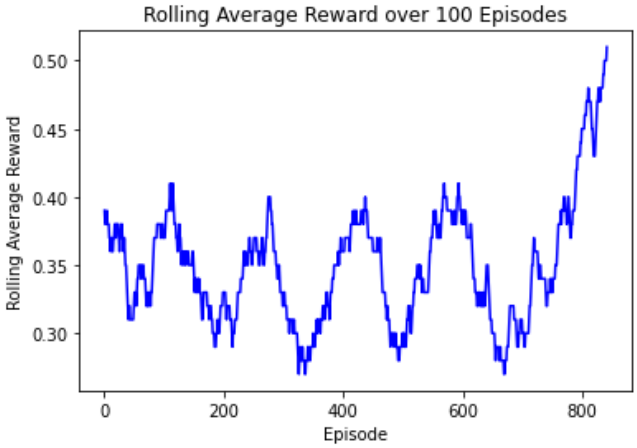


Figure 4: The Rolling Average Reward over 100 Episodes-Altered Environment

continuous increase after episode 600 suggests that the agent's performance is improving as it learns from more episodes. Reaching 0.5, in the end, signifies that the agent has achieved a higher average reward over the last 100 episodes compared to the initial episodes. Compare to the default environment, it takes less time and episodes to resolve.

4.2 Results for Target Following

In order to evaluate the performance of the target-following robot, we define the terminating condition of the experiment as the case that the robot manages to stay within `DISTANCE_THRESHOLD=0.5` to the observed object for

`TIME_THRESHOLD=32` amount of time steps. Based on that, we come up with an evaluation metric *effectiveness*, as the number of time steps it takes for the robot to achieve the terminating condition.

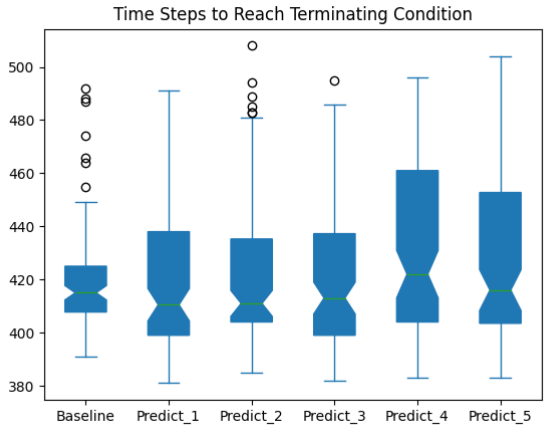


Figure 5: The box plot of the effectiveness results from 100 experiment runs, predictive controls tend to have a better or equal performance lower bound, with some unstably large outliers though.

After running 100 experiments of each control function, including using the baseline control and the predictive path planning control with a predicted time step number ranging from 1 to 5, we generated a box plot of all their *effectiveness* as shown in Fig 5. A more detailed statistical analysis of the results is shown in Table 2, the unit is the number of time steps.

	Baseline	Predictive-1	Predictive-2	Predictive-3	Predictive-4	Predictive-5
mean	419.79	419.45	422.92	420.65	431.53	428.56
std	20.40	27.10	28.41	27.85	32.12	31.90
min	391.00	381.00	385.00	382.00	383.00	383.00
25%	407.75	399.00	404.00	399.00	404.00	403.50
50%	415.00	410.00	411.00	413.00	422.00	416.00
75%	425.00	438.00	435.25	437.25	461.00	452.75
max	492.00	491.00	508.00	495.00	496.00	504.00

Table 2: Statistical analysis of the experiment results.

From the data, we can see that, from the lower bound to the 50th percentile of all predictive control results appear to have a better or equal performance compared to the baseline metric. However, there are many outliers existing and causing the mean and max values to be large, which reveals the instability of the predictive control methods. In conclusion, predictive motion controls tend

to surpass the baseline control slightly, though there is great space for further optimization.

5 Conclusion and Future Work

5.1 Conclusion

5.1.1 Object Retrieval

Furthermore, we have gained valuable insights into the PyBullet Kuka Environment and its various functionalities, which have proven instrumental in understanding and simulating robotic grasping tasks. We have also dived into the fundamental concepts of Deep Q-Network (DQN) learning and successfully applied them to our simulated environment. This has allowed us to train an agent capable of effectively grasping objects under various conditions. (we have explored the effects of altering parameters within the Kuka environment, such as changing gravity, and altering the reward function by adding a penalty to the agent, to observe the impact on the learning performance of our agent.) Our study demonstrates the potential of integrating advanced algorithms like DQN with robotic platforms in Kuka-simulated environments for the development of efficient robotic systems

5.1.2 Target Following

As discussed before in Section 4.2, our project demonstrates that predictive motion controls can potentially improve the target following task, outperforming the baseline control up to the 50th percentile. However, the existence of numerous outliers signifies the instability of these methods, necessitating further optimization. Future efforts will be geared towards refining control strategies to enhance their reliability and performance, contributing to the advancement of predictive motion controls in target following applications.

5.2 Future Work

In terms of Robotic Navigation, we aim to explore and implement a wider range of optimization techniques to enhance the performance of the target follower robot, ensuring its adaptability to various environments and scenarios. This may include but is not limited to, obstacle avoidance, path planning, and navigation through crowded or dynamic spaces. By experimenting with different optimization methods, we can develop a more robust robotic system that can efficiently navigate and interact with its surroundings.

Furthermore, we propose a future direction that aims to "merge" the functionalities of two robots into a systematic workflow, creating a more efficient and versatile system. Robot A will be responsible for recognizing and grasping objects for the patient, such as a medicine pill bottle, and subsequently placing

it onto Robot B. Robot B, in turn, will follow and predict the patient's path, delivering the medicine pill bottle to the patient promptly.

However, there are potential limitations to this approach that must be addressed, including challenges related to object recognition, grasping precision, and reliable path prediction for seamless interaction with patients. Additionally, our current research has not delved deeply into the optimization of hyperparameters in the Reinforcement Learning algorithm, such as batch size, learning rate, and gamma. Therefore, future work should focus on experimenting with these hyperparameters to enhance the performance of the DQN algorithm and explore its potential in our altered environment.

References

[1] Mahyar Abdeetedal. Openai gym environments with pybullet. `https://www.etedal.net/2020/04/pybullet-panda_2.html`, 2020–2023.

[2] Mark Towers Adam Paszke. Reinforcement learning (dqn) tutorial. `https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html`, 2016–2021.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[4] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.

[5] Benjamin Ellenberger. Pybullet gymperium. `https://github.com/benelot/pybullet-gym`, 2018–2019.

[6] Zackory Erickson, Vamsee Gangaram, Ariel Kapusta, C. Karen Liu, and Charles C. Kemp. Assistive gym: A physics simulation framework for assistive robotics. *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[7] Rong Huang, Hongxiu Li, Reima Suomi, Chenglong Li, and Teijo Peltoniemi. Intelligent physical robots in health care: Systematic literature review. *J Med Internet Res*, 25:e39786, Jan 2023.

[8] Rae Yule Kim. Robots in healthcare. *Interactions*, 30(1):48–51, jan 2023.

[9] Mingyi Zhang, Xilong Liu, De Xu, Zhiqiang Cao, and Junzhi Yu. Vision-based target-following guider for mobile robot. *IEEE Transactions on Industrial Electronics*, 66(12):9360–9371, 2019.