



DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

**Branch Dueling Deep Q-Networks for  
Robotics Applications**

**Baris Yazici**







DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

**Branch Dueling Deep Q-Networks for  
Robotics Applications**

**Branchenduell tiefe Q-Netzwerke für  
Robotikanwendungen**

Author: Baris Yazici  
Supervisor: Prof. Dr. Alois Knoll  
Advisor: Msc. Mahmoud Akl  
Submission Date: 15.07.2020





I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.07.2020

Baris Yazici



## Acknowledgments



# **Abstract**



# **Kurzfassung**



# Contents

<b>Acknowledgments</b>	<b>v</b>
<b>Abstract</b>	<b>vii</b>
<b>Kurzfassung</b>	<b>ix</b>
<b>1. Introduction</b>	<b>1</b>
1.1. Objectives . . . . .	1
1.2. Contribution . . . . .	1
1.3. Challenges . . . . .	2
1.4. Report Layout . . . . .	2
<b>2. Background</b>	<b>3</b>
2.1. Related Works . . . . .	3
2.1.1. Analytical Grasping Approach . . . . .	3
2.1.2. Empirical Grasping Approach . . . . .	3
2.2. Learning applications on grasping . . . . .	4
2.2.1. Deep Learning for Detecting Grasps candidates . . . . .	4
2.2.2. Dex-net . . . . .	4
2.2.3. Deep Reinforcement Learning for Vision-Based Robotic Grasping: A Simulated Comparative Evaluation of Off-Policy Methods . . . . .	5
2.2.4. Comparing Task Simplifications to Learn Closed-Loop Object Pick- ing Using Deep Reinforcement Learning . . . . .	6
2.2.5. Solving Rubik's Cube With a Robot Hand . . . . .	7
2.3. Reinforcement Learning . . . . .	7
2.3.1. Markov Chain . . . . .	9
2.3.2. Markov Decision Process . . . . .	10
2.3.3. Reward and Value function . . . . .	11
2.3.4. Q-Learning . . . . .	11
2.4. Function Approximation . . . . .	12
2.4.1. Neural Networks . . . . .	13
2.5. Value-Based Reinforcement Learning . . . . .	13
2.5.1. DQN . . . . .	14
2.5.2. Extension of Q-networks . . . . .	14
2.5.3. Branch Dueeling Q-networks . . . . .	14
2.6. Policy-Based RL . . . . .	14
2.6.1. Policy Gradient . . . . .	15

2.6.2. Entropy-Based Approach . . . . .	15
2.6.3. Soft Actor Critic . . . . .	15
2.7. Curriculum Learning . . . . .	15
2.8. State of Art . . . . .	15
2.8.1. Dexnet . . . . .	15
2.8.2. DQL Google . . . . .	15
<b>3. Simulator Choice</b>	<b>17</b>
3.0.1. Mujoco . . . . .	17
3.0.2. Gazebo . . . . .	18
3.0.3. PyBullet . . . . .	20
<b>4. Reinforcement Learning</b>	<b>23</b>
4.1. Policy-Based RL . . . . .	23
4.1.1. Policy Gradient . . . . .	23
4.2. Value-Based Reinforcement Learning . . . . .	23
4.2.1. DQN . . . . .	23
<b>5. Experimental Setup</b>	<b>25</b>
5.1. Implementation . . . . .	25
5.1.1. BDQ Algorighm Implementation . . . . .	25
5.1.2. Network Architecture . . . . .	25
5.1.3. External Libraries . . . . .	25
5.1.4. Testing Structure . . . . .	25
5.1.5. Run the Code . . . . .	25
5.2. Curriculum Integration . . . . .	25
5.2.1. Curriculm Parameters . . . . .	25
5.3. Hyperparameter Search . . . . .	25
5.4. GPU vs. CPU . . . . .	25
<b>6. Evaluation</b>	<b>27</b>
<b>7. Conclusion &amp; Future Work</b>	<b>29</b>
7.1. Conclusion . . . . .	29
7.2. Future Work . . . . .	29
<b>A. General Addenda</b>	<b>31</b>
A.1. Detailed Addition . . . . .	31
<b>B. Figures</b>	<b>33</b>
B.1. Example 1 . . . . .	33
B.2. Example 2 . . . . .	33
<b>List of Figures</b>	<b>35</b>

<b>List of Tables</b>	<b>37</b>
<b>Bibliography</b>	<b>39</b>



# 1. Introduction

## 1.1. Objectives

Manipulation of objects is one the most inherent action of the humankind. Humans never stood and plan about how to manipulate an object. It is a natural instinct for humans to grab objects in certain ways. Even an infant human can easily manipulate different shapes and colors of objects. Manipulation helps us to use tools, gadget to achieve tasks and provide service. Therefore, manipulation skills will be central for robots of any kind. From rehabilitation to service robots, tool usage is vital to enable them to achieve their objective. Our daily manipulation tasks are shown in figure 1.1. For a complicated manipulation task first of all grasping the object is essential. We need to first firmly grasp a water bottle to drink it. Without a firm grasp the safety of the manipulation process is projected to risk.

Nowadays, every kinds of robots are helping us produce in the industry. However, those robots at the manufacture facilities are only capable of doing same repetitive task. The task they are responsible of expands the car manufacturing to high precision microchip producing. On all those tasks robots are extremely efficient and precise. On the other hand, if a the car company decides to manufacture a new model of car with a slightly different chasis the whole process needs to be changed and program of the robots needs to be hardcoded from zero. Hardcoding every move of the robot in a slight change of a task is fundamentally opposes the very nature of the robots. Main objective of this master's thesis is to enable robotic manipulators to incrementally learn to grasp tools and generalize the learned policy to unseen objects. Rather than overfitting to a certain set of objects, we measure the success rate on unseen dataset. This preassumption promises a robust actor that can adapt well on unseen object and environments.

## 1.2. Contribution

This projects provides two main boiler plate for developers. Firstly well documented and tested robot gripper environment based on Open AI Gym interface[[openai gym](#)]. Gym Environment provides easy to use interface to interact. Moreover, gripper environment complies with the most standard and well documented libraries such as python3, pybullet and numpy. Secondly, we present a novel implementation of BDQ reinforcement algorithm algorihtm adopted to most used Stable Baselines project[2].

## 1. Introduction

---

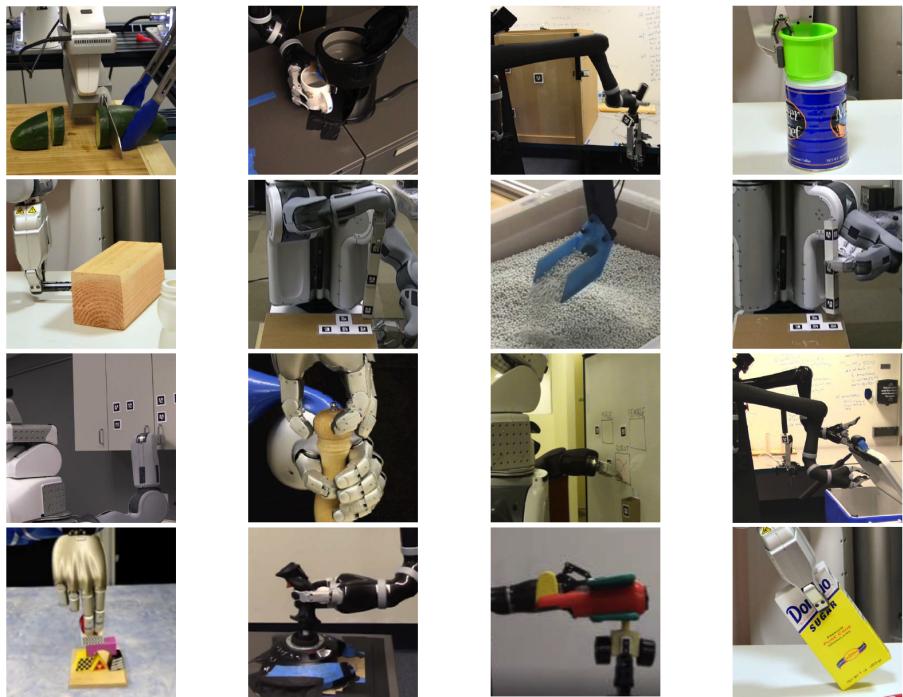


Figure 1.1.: Different manipulation skill adopted to robotic manipulators [1]

### 1.3. Challenges

### 1.4. Report Layout

## 2. Background

### 2.1. Related Works

Robot manipulation and grasping is an intensively researched area. The wide span of the subject makes it hard to categorize all different approaches under one umbrella. Nevertheless, analytical and empirical grasping approaches tend to be the most used categorization in the literature [3].

#### 2.1.1. Analytical Grasping Approach

The analytical approach has been introduced first by Nguyen. He tries to solve the grasping problem by defining analytical objectives. Such as a successful grasp should achieve force closure and satisfy the stability conditions by closing the object from each direction [4]. Nguyen models the hand and the object, to be grasped, to compute the stable force closure grasp analytically. This kind of analytical techniques comes with an exhaustive computation burden. Although the proposed algorithm works fast and correct when the object is planar and flat with a small number of faces, they don't scale to household objects such as mugs or bottles [3]. Moreover, in most robotics setups full geometrical models of the objects are not available [5].

#### 2.1.2. Empirical Grasping Approach

The empirical grasping approach introduced to overcome the difficulties faced with the analytical grasping approach. Empirical methods include learning, which is based on sampling and then training. Some examples of empirical grasping approaches are imitating a human teacher [6], learning from the handcrafted features [7], and deep reinforcement learning(RL) technique to learn close-loop dynamic visual grasping strategies [8]. Imitating a human teacher can easily learn the demonstrated training data. However, it has difficulties to scale to the novel objects, which were not seen during the training [3]. Saxena et al.'s technique to learn from the feature effectively generalize to unseen objects but it fails to choose the best grasp for a particular task [3]. RL approaches learn a particular grasp of an object based on a definition of a goal. Thus, the learned grasp of an object already linked to the task's goal. The flexibility of RL made itself attractive among grasping researchers. One disadvantage of the RL approach is that, it requires a considerable amount of data to train. Although recent applications introduced randomization for sim-to-real transfer of the models [9], it still has difficulties to perform reliably in the real-world [10].

## 2. Background

---

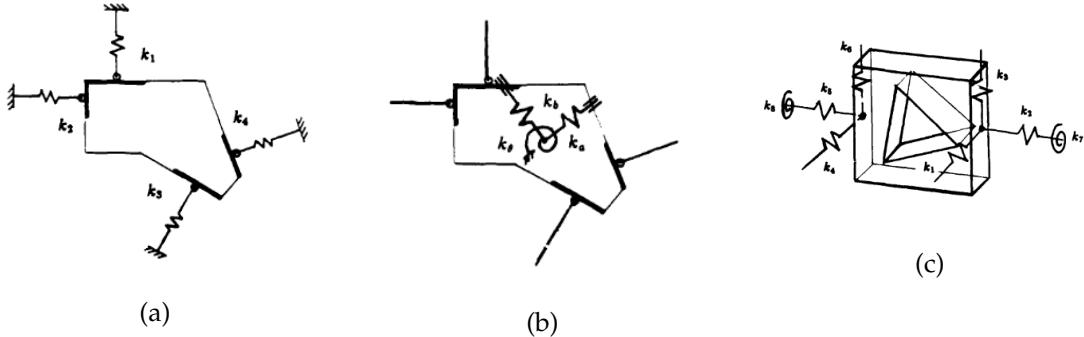


Figure 2.1.: Stable force closure examples fingers are represented as virtual springs. Nguyen proved that all force closures can modified to be a stable grasp candidate[4].

## 2.2. Learning applications on grasping

Experts implement handcrafted features and hand-coded controllers. Thus, it is time-consuming and incapable of generalizing to new objects and scenery. Among all other approaches, machine learning applications on grasping has proven to give a high success rate on novel objects.

### 2.2.1. Deep Learning for Detecting Grasps candidates

Lenz et al. propose a five parameters model trained by supervised learning from the candidate grasps database. They draw a rectangle box representing the best grasp location with the five-dimensional parameter model, x, y coordinates, width, height, and orientation.

Their cascaded network architecture bypasses the need for a handcrafted features. After training, the first layer network learns low-level features and delivers possible naïve grasping candidates, and the second layer chooses the top-ranked grasping candidate 2.2.

Although they achieved up to 90% success rate on grasping Lab tools, this method is limited to only parallel plate gripper. When one wants to use another gripper type, the dataset should be updated entirely for that gripper. Besides data collection process is time-consuming and biased

### 2.2.2. Dex-net

**[TODO: Architectur'u anlat. Robustness factor nasıl parametrize edildi?]**

Another approach for grasping is to learn the grasp robustness factor. Mahlet etal trained convolutional neural networks to learn the grasps robustness function from 6.7 million point cloud data. They collect the data similarly to Lenz et al. with an analytical

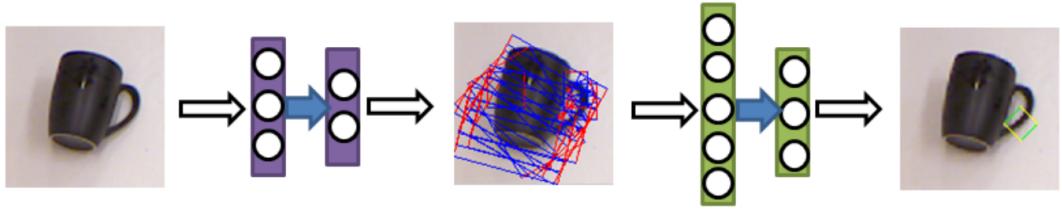


Figure 2.2.: Red-blue rectangles represent possible grasp candidate. Green rectangle is the top-ranked grasp rectangle [11]

grasp metric to evaluate the quality of a grasp.

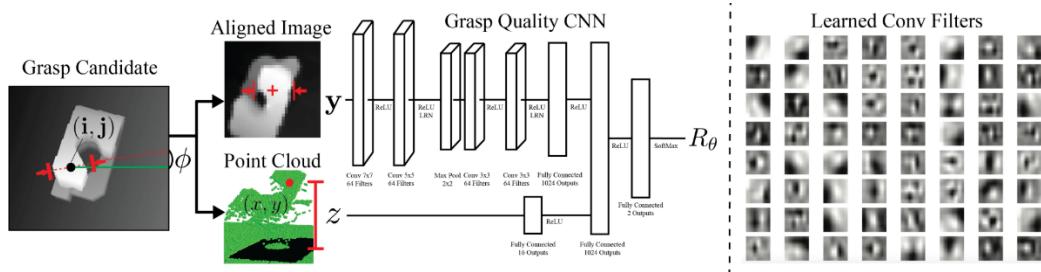


Figure 2.3.: Dex-net architecture [11]

With 98% success rate, their approach indeed proved to be robust. However, the designed network only outputs the grasp location such that a grasp planning algorithm is needed to complete the grasping process. Moreover, data collection can be tedious due to a large number of a machine-labeled dataset.

### 2.2.3. Deep Reinforcement Learning for Vision-Based Robotic Grasping: A Simulated Comparative Evaluation of Off-Policy Methods

The sequential decision-making process is inherent in all grasp tasks. This property of grasping helps us to model it with a reinforcement learning framework. Unlike other works, this approach trains end-to-end policies to automatically learn to grasp without any prior knowledge about the environment or the gripper model. Moreover, the end-to-end nature of their system eliminates the need for additional grasp planner, which was a default in previous works. Quillen et al. show that off-policy RL can increase the robustness of grasping models. Based on their investigation, corrected Monte Carlo and Deep Q Learning outperform the earlier supervised learning approach. They observe that model-free RL approaches allow pre-grasp manipulation to increase the chances of grasp in future actions.

They suit one complete network to concatenate perception input and actions from two different branches 2.6. Although the number of parameters increases drastically with convolutional layers from the perception branch, it makes the system more compact and

## 2. Background

---

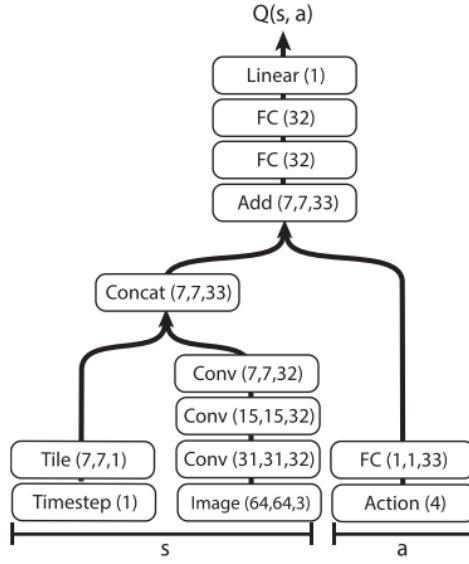


Figure 2.4.: Deep Learning network architecture of [12]

easier to understand.

### 2.2.4. Comparing Task Simplifications to Learn Closed-Loop Object Picking Using Deep Reinforcement Learning

The learning setup is similar to the Quillen et al. with one robot gripper and randomly spawned objects in the simulation. One significant difference regarding the learning setup is Breyer et al. generates the robot gripper only until the wrist without the rest of the arm and, therefore, doesn't need to calculate the inverse kinematics equation.

Breyer et al. enhanced the RL based data-driven grasp approach with Curriculum Learning and the introduction of auto-encoder for perception. Thanks to the curriculum learning setup, which increases the task difficulty based on the agent's current performance, they shortened the time for the agent to explore the environment. In other words, curriculum learning guides the agent throughout its learning phase. Besides, with the help of autoencoder for the perception layer, their network tends to spend less time to comprehend the observation compare to the Quillen et al.'s approach.

Unlike Quillen et al., they experimented with policy-based on-policy RL algorithm, TRPO. Policy -based algorithms tend to **[TODO: Policy based vs. value based here not exploration ]** explore better than epsilon greedy based exploration. This property may also contribute to the higher success rate and faster convergence. After training grasping models on simulation, they tested them on a real robot. Although they didn't use domain randomization to minimize the reality gap, they achieved up to 78% success rate.

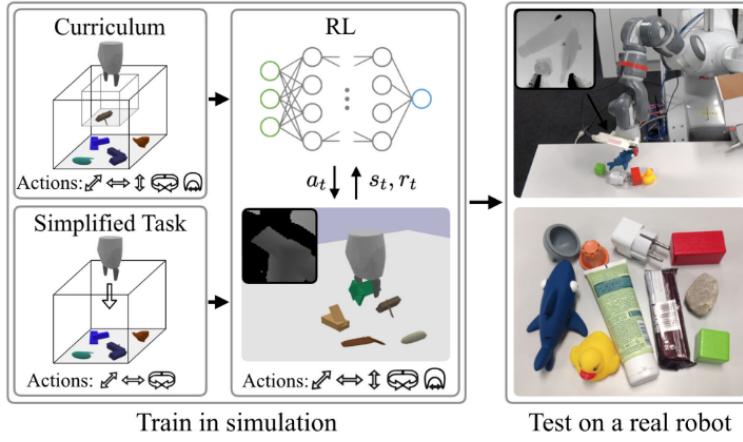


Figure 2.5.: Breyer et al. [13]

### 2.2.5. Solving Rubik's Cube With a Robot Hand

OpenAI et al. present a novel approach for in-hand manipulation problems. Their system attacks Sim2Real discrepancy between simulated models performing on real robots.

Training the Deep RL models on simulation is becoming more and more common. The popularity of simulation increases the demand for enhanced Sim2Real model transfer algorithms. Domain randomization has shown great success in bridging the gap between reality and simulation [14]. More researchers implement machine learning methods that randomize the environment and gripper material parameters. Through this approach, trained models can achieve higher generalization property with robustness to increasing noise on the sensor or environment settings. OpenAI proposes a Meta-Learning method with policies that involve memory like recurrent neural networks that can learn the underlying dynamics of the environment—combining meta-learning with automated domain randomization algorithm results on an enhanced adaptation of models on real-world robots.

## 2.3. Reinforcement Learning

The simplest examples of learning come from our own life; we learn to walk, speak the language, or to cook. All those activities span our entire life, it influences who we are and the decisions we take in life. We know that living animals such as mammals learn from their social and asocial interactions with the environment [16]. Although we have not yet developed a full-scale theory of animal learning, we have developed computational objectives for machines to learn [17]. This computational approach falls into three categories as Supervised, Unsupervised, or Reinforcement Learning.

In this chapter, we will consider the Reinforcement Learning objectives and problem formulation. Reinforcement Learning provides a systematic approach to maximize the

## 2. Background

---

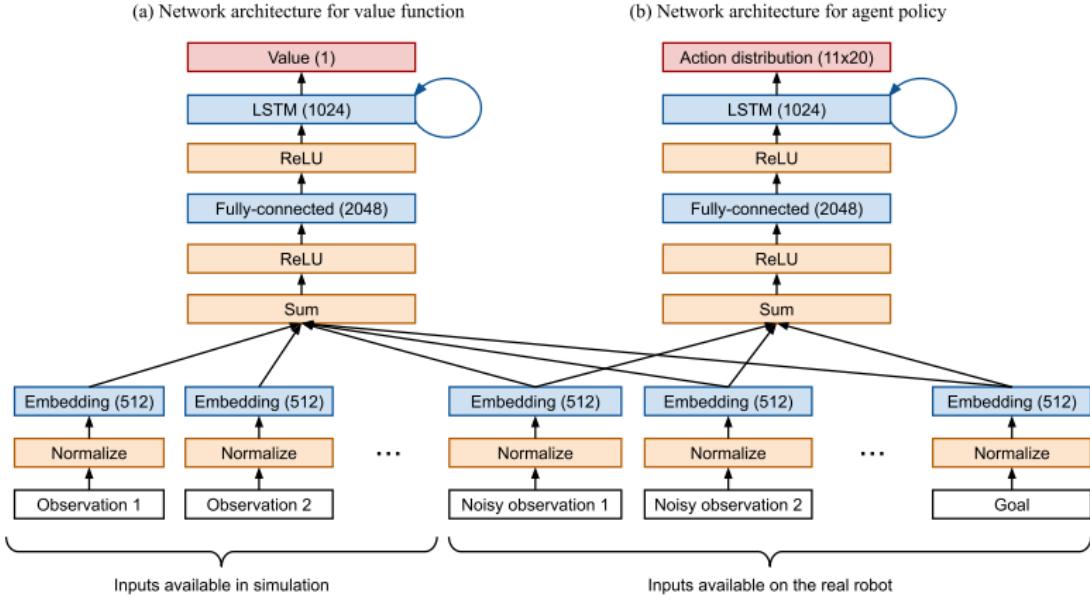


Figure 2.6.: OpenAI policy and value network architecture [15]

reward by linking observations to actions. A reinforcement learning agent creates its data by interacting with the environment. Therefore, it is fundamentally different from supervised and unsupervised learning, where the data is already provided [17]. Another important difference is the inherent goal-oriented approach. Reinforcement learning agent maximizes the rewards for its inherent general goal. Other machine learning approaches lack this goal [17]. For instance, supervised learned software recognizing faces can be used for security reasons to detect criminals or can be easily used to unlock phones. However, a reinforcement learning agent trained to drive a car autonomously can only drive a car. In a sense, reinforcement learning provides us end-to-end learning.

A core feature of reinforcement learning is that it acts on uncertain environments and, in return, receives the observation and reward. Fundamentally, a learning agent collects this experience and tunes its action to increase the expected reward. The expected reward term refers to the end of the horizon. For example, a chess-playing agent can choose to sacrifice the queen in the next move for a checkmate in the move after. In this case, the reward would decrease when the agent loses a queen, but the goal of the agent will be satisfied by terminating the game. For a well-defined reinforcement learning system, we can speak of four main components: Policy to decide the actions, a reward to maximize the expected reward in the horizon, and a model of the environment, telling which directions the chess pieces can move. The components of reinforcement learning are formulated based on Markov Decision Process. In the MDP chapter, we will detailly explain RL components. In the next chapters first Markov Chains, the simple version of MDP, then MDP, the slightly advanced version of Markov Chains, will be explained by some simple modifications [18].

### 2.3.1. Markov Chain

The weak law of large numbers has a tremendous significance on stochastic modeling. This law states that the average of a large number of experimentations converges to the real value of the probability of a particular task. As an example, if one tosses infinite amounts of the coin, the average number of heads should converge to 0.5 [19]. Bernoulli's weak law of large numbers only covers independent events. Markov proved that Bernoulli's law also holds on dependent cases [19]. As the law of large numbers suggests, if one conducts a large number of iterations on this problem, one can deduce the transition matrix. This matrix proves to be the only information one needs to compute the next state. This characteristic defines the famous Markov property; the current state captures all the necessary information one needs to predict the next state. If we extrapolate this example to slightly complex systems, for example, weather forecast, we just need today's weather report to predict tomorrow's forecast, assuming that we know the transition matrix. Naturally, one can formulate other events with Markov Chain lawn mower, and random walk are the straightforward ones in the literature[19].

It is also possible to attach rewards to Markov Chain's formulation. In figure 2.7, the transition between states is represented with the arcs. And each transition has a probability similar to the transition matrix; we defined before. Each state has an immediate reward and a value function. The immediate reward is received directly when the actor moves to that state. The value of a state represents how likely the future actor will end up collecting high rewards.

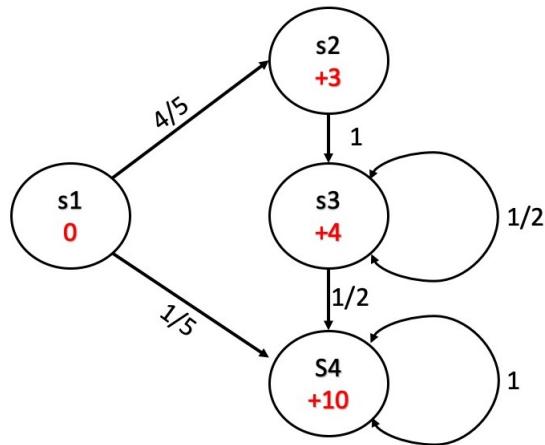


Figure 2.7.: Markov chain with transition probabilities and rewards

The value of a state will be later significant to solve Reinforcement Learning problems through Value Iteration methods. They are one of the essential algorithms that led to the initial success of Reinforcement Learning research.

## 2. Background

---

### 2.3.2. Markov Decision Process

Markov decision process is a slightly advanced version of the Markov Chain. It includes action on top of the Markov Chain. Reinforcement learning problems are formalized as a Markov Decision Process rather than Markov Chain because RL agents are free to choose from different actions.

MDPs first came into play as part of optimal control problem by Bellman [20]. Bellman applied dynamic programming methods to solve the MDP problem optimally. However, this methodology was not scalable to larger problems stem from the curse of dimensionality problem [Sutton].

The fundamental elements of MDP are as follows:

- Agent: The actor takes action on the environment to learn.
- Environment: The agent interacts with the Environment(Plant).
- Rewards: Environment returns rewards based on the interaction made by the actor.
- State: View of the environment from the eyes of the actor.

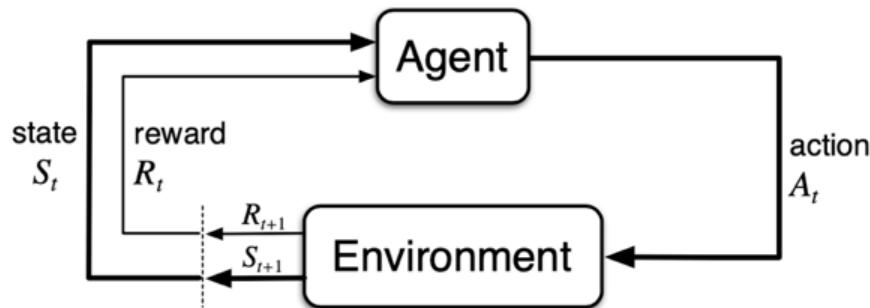


Figure 2.8.: MDP structure

The process of action follows; the agent acts on the environment at time  $t$  and environment return the reward and state of the action at time  $t + 1$ . Based on the state of the environment at the  $t + 1$  agent makes another action  $A_{t+1}$ , which results in  $R_{t+2}$  and  $S_{t+2}$ .

In every MDP system, agents should be reachable to every state through a sequence of actions. The transition between states is of great value for the MDP framework. One can compute the transition probability in MDP, given the inner dynamics of the environment [17]. The below equation defines the internal dynamics probability. It tells, how probable it is to end up in state  $s'$  with reward  $r$  by taking action  $a$  in state  $s$ .

$$p(s', r | s, a) = \Pr\{S_t = s', R_t = r | S_{t-1} = d, A_{t-1} = a\} \quad (2.1)$$

One can calculate the transition probability function from the inner dynamics' probability function.

$$p(s'|s, a) = \Pr\{S_t = s' | S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in R} p(s', r | s, a) \quad (2.2)$$

Next chapter, we will dive into the definition of the reward and value function. Based on those concepts we will build the logic on how to solve MDPs optimally

### 2.3.3. Reward and Value function

#### [TODO: Ilk paragraf]

As defined in the Markov Chain section, rewards and value functions are the essence of value iteration algorithms. In this section, we will describe the objective of the RL problem formally. As we mentioned in previous chapters, we want the increase the number of rewards we achieve when we reach the terminal state. If we define the reward at final state  $T$  as  $R_T$  and the reward at the initial state  $t$  is  $R_t$ , returns we receive is the sum of rewards in every state.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T \quad (2.3)$$

$G_t$  is the notation of expected return. We will mostly use the discounted version of the expected return calculation. We introduce a discount factor ( $\gamma$ , to value the rewards that the agent receives now, compare to the rewards in the future.

$$G_t = R_{t+1} + \gamma^1 * R_{t+2} + \gamma^2 * R_{t+3} + \dots + \gamma^{T-1} * R_T = \sum_{k=0} \gamma^k R_{t+k+1} \quad (2.4)$$

For instance, if a rational human offered 1m Euros now, versus 1m Euros in 50 years, would usually choose 1m Euros now. Therefore, our agent also weighs the rewards it receives now, over 50 steps from the current state. In the meantime, we do not want the agent to undervalue the importance of reaching the terminal state through the highest reward sequence. Given the below example, if we introduce a discount factor of 0, the agent will always try to maximize the immediate reward and take a sequence of s1-a2-s3-a4-s4 and end up with non-optimal greedy algorithm with  $G_t = 20$ . But with a discount factor, in this example everything between  $0 < \gamma < 1$  works, would find the optimal sequence(s1-a1-s2-a3-s4) with  $G_t = 25$ .

The value function is the expectation of returns while an agent is following the policy ( $\pi$ ). Policy namely represents the probability distribution of an agent taking action  $a$  in state  $s$ . The policy is similar to the transition probability matrix in the Markov Chain section. Using policy, we can define the value function of an agent following the policy  $\pi$  as below.

### 2.3.4. Q-Learning

Q-learning algorithms have been the core of the RL research for almost 20 years. It combines the idea of TD learning in approximating action-value function. Through this

approach, the computation converges faster than state-value approximation algorithms. Another strength of the Q-learning algorithm is its off-policy nature. The Q-learning agent can learn from the results of different policies. Off-policy nature makes the agent more data-efficient. An agent is not obliged to learn from the outcome of one policy; instead, many policies can contribute to learning.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))] \quad (2.5)$$

## 2.4. Function Approximation

As mentioned before, it is often not possible to find either the optimal policy or the optimal value function due to computational resources. It is usually enough to find approximate solutions. Various function approximators are available for us to use. The main categorization follows as; linear and nonlinear function approximators. Their working principle is similar; they both follow a specific policy and sample experiences. Based on the experiences they approximate either policy or the value function.

The application of neural networks provided RL methods a boost.(REFERENCE) Since the introduction of function approximators, the success of RL applications has increased significantly. Although function approximation tools bring convergency issues in some corner cases, it shows excellent success practically.

If one defines the value function under some weight parameterization, then the  $V(s, w)$  notation replaces the  $V(s)$ . To find suitable weight parameters that represent the value function as general as possible, one needs to update parameters  $w$  at each step towards the smallest loss region based on a particular objective description. Firstly, we need to define an objective function and then find a method to tweak our weight parameters towards the objective gradually. There are a couple of possible objectives to learn in the literature; the state-value function, action-value function, or the directly the policy. In the case of the state value function, one can choose the objective function as the mean squared value loss between the estimated state-value function and the target state-value function (2.6). While for policy approximation, expected return represents the objective function 2.9.

(Silebilirim) Target state-value function can be the monte-Carlo or TD target. Monte Carlo's target is the expected return ( $G_t$ ), and TD target is the bootstrapped value. The main difference between those targets is that Monte Carlo ensured to converge to a global optimum. In contrast, the bootstrapped TD target converges only to a local optimum near the global optimum. (Silebilirim)

$$VE(w) = \sum_{s \in S} \mu(s)[v_\pi(s) - \hat{v}(s, w)]^2 \quad (2.6)$$

Secondly, one needs a way to optimize the objective function. The stochastic gradient descent method is the most prominent solution to improve the weight parameters based

on a defined loss function. 2.7 shows one gradient update step of the stochastic gradient descent method on the weight vector.

$$w_{t+1} = w_t - \frac{1}{2}\alpha \nabla \left[ v_\pi(s) - \hat{v}(s, w) \right]^2 \quad (2.7)$$

The optimal weight vector needs to find the right balance between strongly representing one state and generalizing to similar unseen states. As a result, our approximator needs to avoid either overfitting or underfitting.

One can also approximate action-value function in the same way as state-value function. The gradient update step of the action-value function is similar to the state-value function as in 2.8

$$w_{t+1} = w_t - \alpha \left[ U_t - \hat{q}(S_t, A_t, w_t) \right] \nabla \hat{q}(S_t, A_t, w_t) \quad (2.8)$$

$U_t$  being the target of the approximation can be either TD or Monte Carlo target.

So far, we have only considered the function approximation on the action and state functions. Another promising and popular technique is to approximate the policy function directly. Policy function can point us in a more direct way than action and state values. Since it directly aims to solve the problem at hand, to find the optimal policy [21]. The objective function of policy gradient methods is simply the expectation of total return under policy  $\pi$ .

$$J(\theta) = E \left[ \sum_{t=0}^H R(s_t, u_t) \right] \quad (2.9)$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \quad (2.10)$$

#### 2.4.1. Neural Networks

Neural networks are the unique case of function approximators. Although they follow the same procedure as SGD methods, their weight matrix is nonlinearly related to the approximated  $V(S_t, w_{at})$ . Nonlinear relation brings convergency issues and problems with instabilities. However, the practical success of these kinds of methods shadows the weak theoretical basis.

A generic neural network(2.9) comprises layers of artificial neurons connected with weight, bias vectors, and activation functions to each other. Each activation function feeds nonlinearity to the general equation.

## 2.5. Value-Based Reinforcement Learning

Approximation of action-value or state-value function methods falls into the category of Value-based RL. These methods are model-free, meaning that they use sampling to solve the optimal solution for the problem [22]. At the same time, they are off-policy

## 2. Background

---

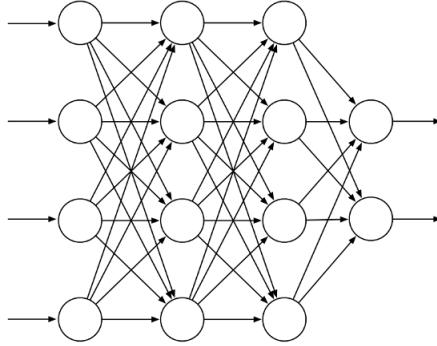


Figure 2.9.: Generic neural network with two hidden layers

algorithms, which makes them data efficient but high variance algorithms [21]. The fact that these methods only indirectly optimize the policy make them unstable during the learning phase [17]. Value function-based approaches can overestimate the selected actions from the policy, which is the typical problem with Q-learning based algorithms.

### 2.5.1. DQN

DQN is the neural network extension of the Q-learning algorithm. It has shown RL can handle high dimensional nonlinear control problems. DQN optimizes Bellman error by parameterizing the action-value function. Some extensions of DQN introduced experience replay buffer to cope with the high variance problem [23]. [22] inputs randomly collected mini batches of experiences to the neural network consisting of multiple layers. Thanks to their neural network structure, they overcome the convergency issues.

$$L(\theta) = E_{(s,a,r,s') \sim U(D)} \left[ \left( r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right] \quad (2.11)$$

### 2.5.2. Extension of Q-networks

### 2.5.3. Branch Dueeling Q-networks

## 2.6. Policy-Based RL

Policy-based solutions directly parameterize the policy to find the optimal policy which returns the highest reward. These approaches proved to be more stable than Value-based counterparts. One reason behind stability is that actions are sampled from the continuous policy parameterization function, which outputs smooth action probabilities [17]. Similar to Value-based methods, policy-based RL also aims to solve the RL problem only with sampling in model-free fashion.

### 2.6.1. Policy Gradient

Policy gradient methods incorporate stochastic gradient ascent on policy objective function. This update optimizes the theta parameters of the policy. Unlike DQN, policy gradient methods perform each update in on-policy fashion. It uses only the experiences taken under the particular policy to update the parameters of this policy [21].

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a|s, \theta) \quad (2.12)$$

$$= E_\pi \left[ \sum_a q_\pi(S_t, a) \nabla \pi(a|S_t, \theta) \right] \quad (2.13)$$

### 2.6.2. Entropy-Based Approach

### 2.6.3. Soft Actor Critic

## 2.7. Curriculum Learning

One can gradually increase the level of the difficulty of a specific task to be learned to guide the training process and speed up the convergence time. Based on phycological studies, humans learn faster when they subject to the information in the form of a unique curriculum. That is because we are subjected to curriculum learning since they were toddlers.

## 2.8. State of Art

### 2.8.1. Dexnet

### 2.8.2. DQL Google



## 3. Simulator Choice

Simulator choice was the first important architecture decision we made, because it influences the whole software architecture. Simulator lays the groundwork of the robotics environment implementation. Especially for continuous control optimization tasks, accurate simulation plays a significant role in the performance of the controller. As Todorov et al. indicated, if the simulation is not accurate enough, an optimization algorithm will find a way to exploit it [24]. We compared three states of art simulators: Mujoco, Gazebo, and Pybullet. Each simulator comes with advantages and drawbacks. Our decision is based on the factors: active support, community, stability, scalability, and ease of use. It is important to note that our analysis includes subjective factors such as ease of use, documentation coverage, and tutorials quality. Although we tried our best to ground our points on facts, subjective criteria include a certain amount of personal preferences.

### 3.0.1. Mujoco

Mujoco is released in 2015, making it the newest physics engine in our comparison. It stands for Multi-Joint Control. Hence the name, its primary purpose is Robot simulation.

We first considered Mujoco as our main simulator. Our consideration was based on; Roboticist mainly using Mujoco at OpenAI and DeepMind [25]. In addition, simulation engine reviews featured Mujoco as the best performing and stable engine [26]. Erez et al. conducted tests that compared the performance of physics engines on grasping tasks, where Mujoco by far performed better than the other engines 3.1. Additionally, Mujoco provides an API that lets users develop in C language. C language API is a significant advantage for researchers aim to get as close to the hardware as possible to save time. Another plus is mujoco-py offers a Python API to control Mujoco simulation easily. OpenAI product Mujoco-py is entirely open-source with MIT license [27].

On the other hand, MuJoCo requires a license to use for research purposes. Personal non-commercial license costs 500\$ as of 2020 July. License cost is the main drawback because we prefer complete open-source software. Another factor is the training time since our models require over 10M time-steps to train, we need to make use of every millisecond time gain in every step. So, GPU support is highly crucial. Mujoco only supports CPU simulation, which has the potential to be slower than GPU. As a side-note, we have not compared the MuJoCo CPU version speed with Bullet GPU version speed. MuJoCo developers prepared a performance page where they explain the GPU vs. CPU comparison. They pointed out that if decoupled systems are to be simulated like particle simulation, GPU has an advantage. Whereas, one simulates

### 3. Simulator Choice

---

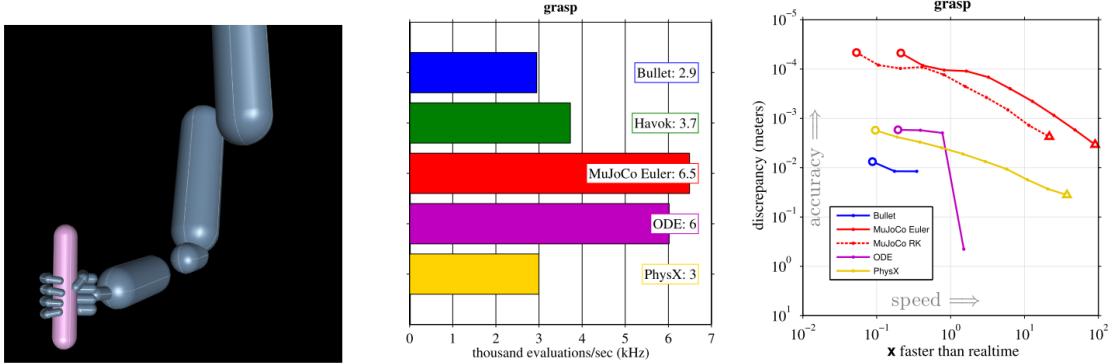


Figure 3.1.: physics engine performance on grasping task. Mujoco performs the best [26]

coupled system such as a humanoid robot CPU gain a slight edge 3.2. They also noted that there is room for research in the areas besides coupled or decoupled systems. [<http://mujoco.org/performance.html>]

Finally, observing large open-source projects like OpenAI-Roboschool, which initially used MuJoCo, deprecating, and suggesting PyBullet, made us reconsider. We believe that the reason community migrating towards PyBullet is completely open-source codebase.

#### 3.0.2. Gazebo

The Gazebo is the oldest simulator among Bullet and Mujoco. The development of Gazebo dates back to the 2002 University of Southern California. Later, Willow Garage took over action and extended Gazebo to ROS and PR2. Gazebo became the primary simulation engine of the ROS community. Eventually, in 2012 Gazebo became part of OSRF (Open Source Robotics Foundation), a spin-off from Willow Garage.

Technically, Gazebo is a simulation platform, which inherits different physics engines, such as Bullet, ODE, Simbody, and DART. According to the documentation of Gazebo 11.0, it supports ODE engine default, and other engines can be used, if developers compile Gazebo from the source. That means the performance of the overall Gazebo simulation highly depends on those individual physics engine's performances. Another dependency of Gazebo is ROS(Robotics Operating System). ROS infrastructure handles all communication. Thus, users need to rely on ROS to interact with Gazebo. We believe this assumption is too large. Even if relying on ROS can bring a lot of well-structured tool with it, learning ROS has a steep learning curve and has the potential to cause a large overhead for simple projects. Nonetheless, developers invested highly on neat and clean documentation to reduce the overhead for new users. We consider the documentation and tutorials as a merit of the open-source project. Correspondingly, being open source contributes hugely to a large community willing to support, answering questions on forums, and submitting pull requests for possible bug fixes.

According to Pitonakova et al., Gazebo has usability issues due to not having a 3D mesh editing option and difficulties of installing dependencies for 3rd party models.

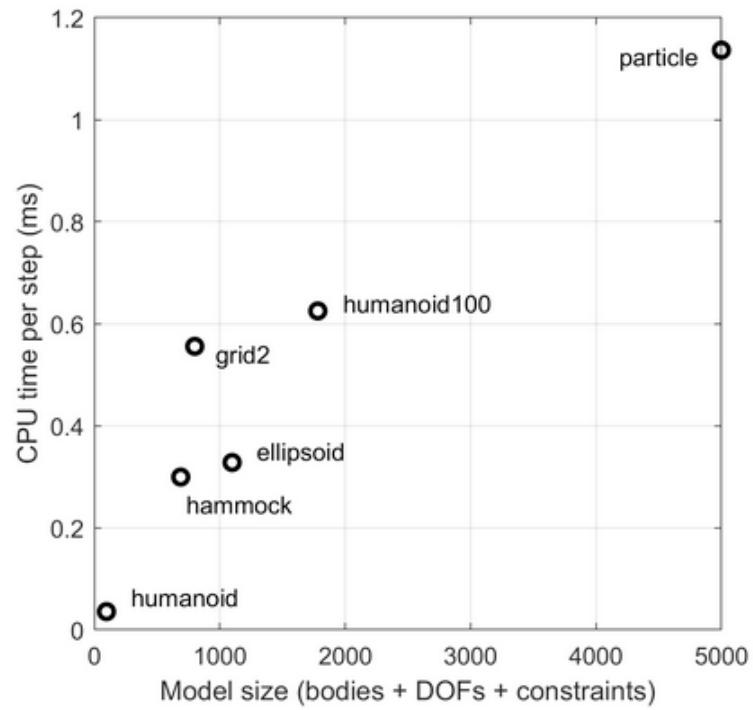


Figure 3.2.: CPU time per step comparison of different tasks [26]

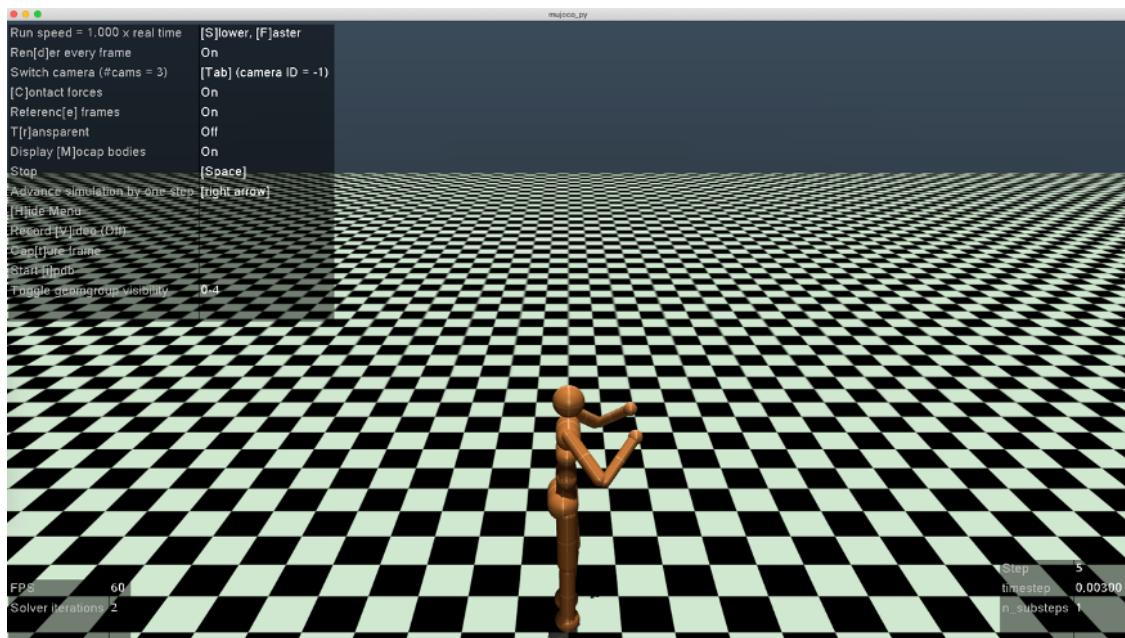


Figure 3.3.: Mujoco simulation of a humanoid model. Rendered with MJViewer

### 3. Simulator Choice

---

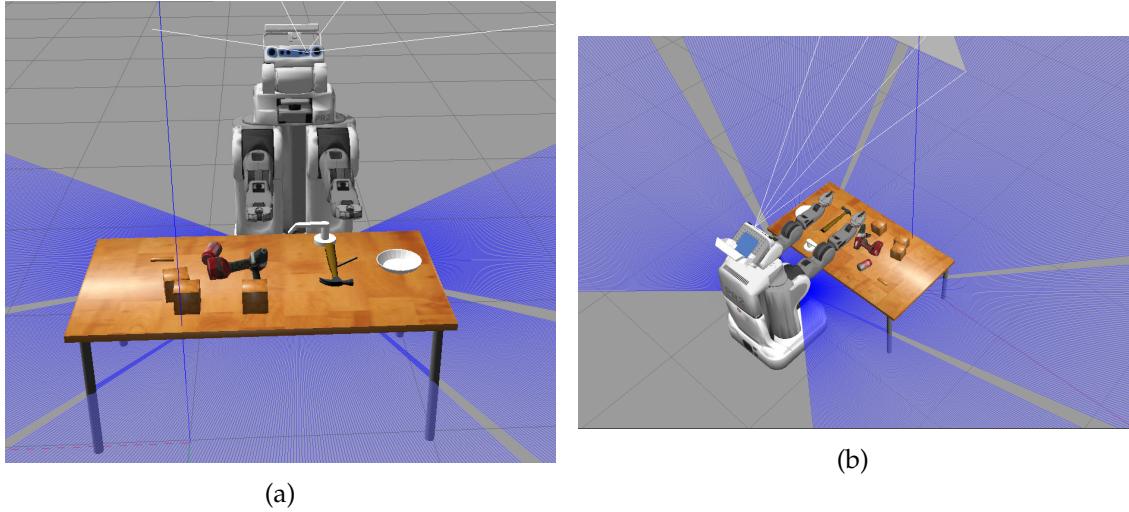


Figure 3.4.: Table-top environment in Gazebo simulation with variety of objects and PR2 robot

They also noted that Gazebo performs fairly well in large simulation environments, so it could be more convenient to conduct extensive swarm robotics experiments on Gazebo [28].

Based on our experiments in 3.4, we found that Gazebo provides useful models to easily setup a table-top environment for robotics picking applications. Although we have not performed any grasping experiments on Gazebo, being able to edit the size, position, and orientation of models directly on simulation GUI is a useful feature, which lacks both on Pybullet and Mujoco.

#### 3.0.3. PyBullet

PyBullet is a simulator built on the Bullet physics engine. Bullet's initial release dates back to 2006. Bullet engine was initially game, and graphics focused, but lately, with PyBullet, it has been increasingly popular among roboticists. We decided on using PyBullet for several strong reasons and overall satisfied by the performance and the features it offers. Firstly, PyBullet recently published their RL resources on Github [[RLBullet](#)]. These resources provide everything necessary to quickly start experimenting with any choice of robot, environment, and algorithm. Secondly, our reference papers, in the area of RL robotics research, are using PyBullet for their experiments [12] [13]. Therefore, it is more convenient to compare results by using the same physical engine to avoid simulation differences. Thirdly, it has sizeable open-source support and free to use.

On the other hand, based on Erez et al. performance tasks, which compares Bullet, Mujoco, ODE, and PhysX engines, Bullet is either at the last place or the one above the last position 3.5 figure. Similarly, they mention the incompatibility of Bullet Engine's

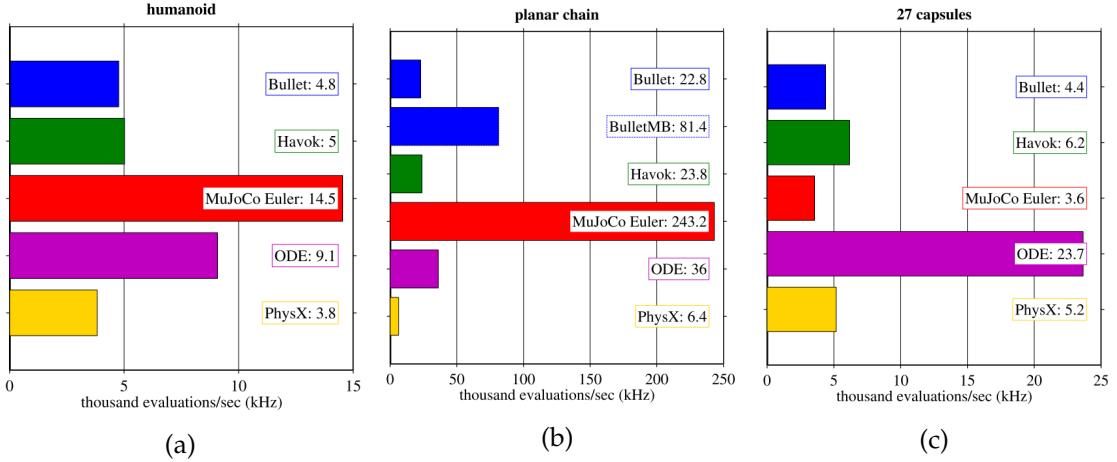


Figure 3.5.: Comparing physics engine performances. Bullet is either at the last place or the one above the last position [26]

spring-damper system with the standard PD controller design [26].

Many open-source robotics learning resources migrating to PyBullet. This migration brings more open-source contributors who are more focused on Reinforcement Learning. For example, open-source contributors recently developed multi-threading GPU support to PyBullet. Also, stable-baselines creators implemented a RL training package under Bullet's Github repository [29] [2].

In conclusion, PyBullet is the most actively developed physics engine among three contenders. Thanks to the open-source community, any small bug is reported and solved in a short time, which is missing in Gazebo and Mujoco. We assume more roboticist will migrate to PyBullet in the future. Hereby, it will become primary physics engine for robotics research. Despite research papers represent Pybullet engine performance as the worst among three, in practice we have not noticed any significant failure related to engine performance.

Simulator	Gazebo	MuJoCo	Pybullet
<b>Designed For</b>	Robotics	Robotics	Graphics/Games/Robotics
<b>License</b>	Open-Source	Closed-Source	Open-Source
<b>API</b>	C++/Python	C	Python
<b>Documentation</b>	4/5	3/5	5/5
<b>Ease of Use</b>	2/5	3/5	5/5

Table 3.1.: Comparison of simulators. Overall PyBullet is decided based on given criteria

### 3. Simulator Choice

---

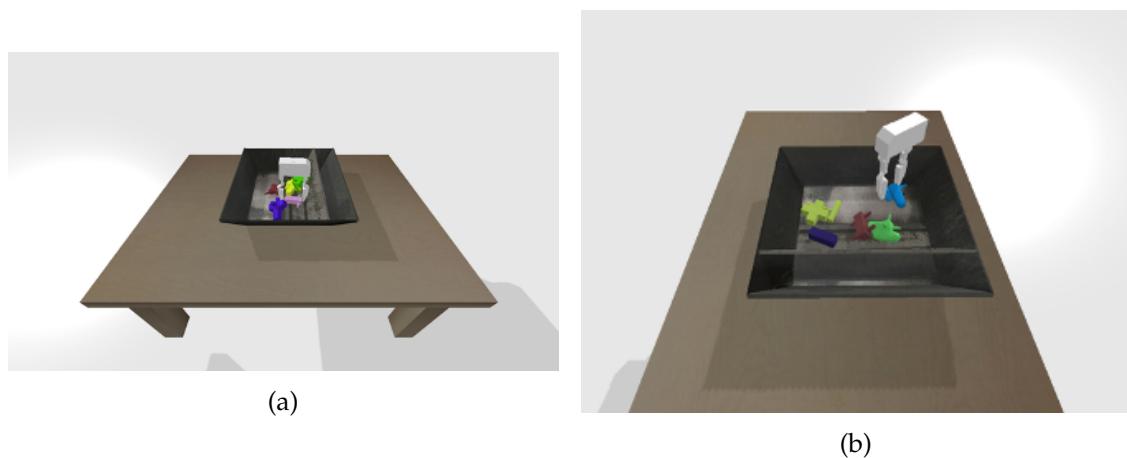


Figure 3.6.: Screenshot from our trained hand model on Pybullet

## 4. Reinforcement Learning

### 4.1. Policy-Based RL

Policy-based solutions directly parameterize the policy to find the optimal policy which returns the highest reward. These approaches proved to be more stable than Value-based counterparts. One reason behind stability is that actions are sampled from the continuous policy parameterization function, which outputs smooth action probabilities [17]. Similar to Value-based methods, policy-based RL also aims to solve the RL problem only with sampling in model-free fashion.

#### 4.1.1. Policy Gradient

Policy gradient methods incorporate stochastic gradient ascent on policy objective function. This update optimizes the theta parameters of the policy. Unlike DQN, policy gradient methods perform each update in on-policy fashion. It uses only the experiences taken under the particular policy to update the parameters of this policy [21].

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a|s, \theta) \quad (4.1)$$

$$= E_\pi \left[ \sum_a q_\pi(S_t, a) \nabla \pi(a|S_t, \theta) \right] \quad (4.2)$$

### 4.2. Value-Based Reinforcement Learning

Approximation of action-value or state-value function methods falls into the category of Value-based RL. These methods are model-free, meaning that they use sampling to solve the optimal solution for the problem [22]. At the same time, they are off-policy algorithms, which makes them data efficient but high variance algorithms [21]. The fact that these methods only indirectly optimize the policy make them unstable during the learning phase [17]. Value function-based approaches can overestimate the selected actions from the policy, which is the typical problem with Q-learning based algorithms.

#### 4.2.1. DQN

DQN is the neural network extension of the Q-learning algorithm. It has shown RL can handle high dimensional nonlinear control problems. DQN optimizes Bellman error by parameterizing the action-value function. Some extensions of DQN introduced

#### 4. Reinforcement Learning

---

experience replay buffer to cope with the high variance problem [23]. [22] inputs randomly collected mini batches of experiences to the neural network consisting of multiple layers. Thanks to their neural network structure, they overcome the convergency issues.

$$L(\theta) = E_{(s,a,r,s') \sim U(D)} \left[ \left( r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right] \quad (4.3)$$

# **5. Experimental Setup**

## **5.1. Implementation**

### **5.1.1. BDQ Algorighm Implementation**

**Object Recognition**

### **5.1.2. Network Architecture**

### **5.1.3. External Libraries**

### **5.1.4. Testing Structure**

### **5.1.5. Run the Code**

## **5.2. Curriculum Integration**

### **5.2.1. Curriculum Parameters**

## **5.3. Hyperparameter Search**

## **5.4. GPU vs. CPU**



## **6. Evaluation**



## **7. Conclusion & Future Work**

Use with pdfLaTeX and Biber.

### **7.1. Conclusion**

### **7.2. Future Work**



# **A. General Addenda**

If there are several additions you want to add, but they do not fit into the thesis itself, they belong here.

## **A.1. Detailed Addition**

Even sections are possible, but usually only used for several elements in, e.g. tables, images, etc.



## B. Figures

### B.1. Example 1

✓

### B.2. Example 2

✗



# List of Figures

1.1.	Different manipulation skill adopted to robotic manipulators [1] . . . . .	2
2.1.	Stable force closure examples fingers are represented as virtual springs. Nguyen proved that all force closures can modified to be a stable grasp candidate[4]. . . . .	4
2.2.	Red-blue rectangles represent possible grasp candidate. Green rectangle is the top-ranked grasp rectangle [11] . . . . .	5
2.3.	Dex-net architecture [11] . . . . .	5
2.4.	Deep Learning network architecture of [12] . . . . .	6
2.5.	Breyer et al. [13] . . . . .	7
2.6.	OpenAI policy and value network architecture [15] . . . . .	8
2.7.	Markov chain with transition probabilties and rewards . . . . .	9
2.8.	MDP structure . . . . .	10
2.9.	Generic neural network with two hidden layers . . . . .	14
3.1.	physics engine performance on grasping task. Mujoco performs the best [26] . . . . .	18
3.2.	CPU time per step comparison of different tasks [26] . . . . .	19
3.3.	Mujoco simulation of a humanoid model. Rendered with MJViewer . . . . .	19
3.4.	Table-top environment in Gazebo simulation with variety of objects and PR2 robot . . . . .	20
3.5.	Comparing physics engine performances. Bullet is either at the last place or the one above the last position [26] . . . . .	21
3.6.	Screenshot from our trained hand model on Pybullet . . . . .	22



# List of Tables

3.1. Comparison of simulators. Overall PyBullet is decided based on given criteria . . . . .	21
--	----



# Bibliography

- [1] O. Kroemer, S. Niekum, and G. Konidaris. "A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms". In: (2019). arXiv: 1907.03146. URL: <http://arxiv.org/abs/1907.03146>.
- [2] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. *Stable Baselines*. <https://github.com/hill-a/stable-baselines>. 2018.
- [3] A. Sahbani, S. El-Khoury, and P. Bidaud. "An overview of 3D object grasp synthesis algorithms". In: *Robotics and Autonomous Systems* 60.3 (2012), pp. 326–336. ISSN: 09218890. doi: 10.1016/j.robot.2011.07.016. URL: <http://dx.doi.org/10.1016/j.robot.2011.07.016>.
- [4] V.-d. Nguyen. "Constructing Stable Grasps in 3D". In: *Fortune* (1987), pp. 234–239.
- [5] P. Schmidt, N. Vahrenkamp, M. Wachter, and T. Asfour. "Grasping of Unknown Objects Using Deep Convolutional Neural Networks Based on Depth Images". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2018), pp. 6831–6838. ISSN: 10504729. doi: 10.1109/ICRA.2018.8463204.
- [6] S. Ekvall and D. Kragic. "Interactive grasp learning based on human demonstration". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2004.4 (2004), pp. 3519–3524. ISSN: 10504729. doi: 10.1109/robot.2004.1308798.
- [7] A. Saxena, J. Driemeyer, and A. Y. Ng. "Robotic grasping of novel objects using vision". In: *International Journal of Robotics Research* 27.2 (2008), pp. 157–173. ISSN: 02783649. doi: 10.1177/0278364907087172.
- [8] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine. "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: CoRL (2018), pp. 1–23. arXiv: 1806.10293. URL: <http://arxiv.org/abs/1806.10293>.
- [9] O. A. M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba. "Learning dexterous in-hand manipulation". In: *International Journal of Robotics Research* 39.1 (2020), pp. 3–20. ISSN: 17413176. doi: 10.1177/0278364919887447. arXiv: 1808.00177.
- [10] S. Caldera, A. Rassau, and D. Chai. "Review of deep learning methods in robotic grasp detection". In: *Multimodal Technologies and Interaction* 2.3 (2018). ISSN: 24144088. doi: 10.3390/mti2030057.

## Bibliography

---

- [11] I. Lenz, H. Lee, and A. Saxena. "Deep Learning for Detecting Robotic Grasps". In: (Jan. 2013). arXiv: 1301.3592. URL: <http://arxiv.org/abs/1301.3592>.
- [12] D. Quillen, E. Jang, O. Nachum, C. Finn, J. Ibarz, and S. Levine. "Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2018), pp. 6284–6291. ISSN: 10504729. DOI: 10.1109/ICRA.2018.8461039. arXiv: 1802.10264.
- [13] M. Breyer, F. Furrer, T. Novkovic, R. Siegwart, and J. Nieto. "Comparing Task Simplifications to Learn Closed-Loop Object Picking Using Deep Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 4.2 (Mar. 2018), pp. 1549–1556. ISSN: 23773766. DOI: 10.1109/LRA.2019.2896467. arXiv: 1803.04996. URL: <http://arxiv.org/abs/1803.04996>.
- [14] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *IEEE International Conference on Intelligent Robots and Systems 2017-Septe* (2017), pp. 23–30. ISSN: 21530866. DOI: 10.1109/IROS.2017.8202133. arXiv: 1703.06907.
- [15] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang. "Solving Rubik's Cube with a Robot Hand". In: *arXiv preprint* (2019).
- [16] E. L. Thorndike. "Animal Intelligence: Experimental Studies." In: (1911).
- [17] R. S. Sutton and A. G. Barto. *Reinforcement Learning, Second Edition: An Introduction - Complete Draft*. 2018, pp. 1–3. ISBN: 9780262039246.
- [18] T. Lozano-Pérez and L. Kaelbling. "6.825 Techniques in Artificial Intelligence (SMA 5504)". Massachusetts Institute of Technology: MIT OpenCourseWare, <https://ocw.mit.edu>. Fall 2002. URL: <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-825-techniques-in-artificial-intelligence-sma-5504-fall-2002/index.html#>.
- [19] P. A. Gagniuc. *From Theory to Implementation and Experimentation*. Wiley, 2017. ISBN: 9781119387572.
- [20] R. Bellman. "Dynamic programming and stochastic control processes". In: *Information and Control* 1.3 (1958), pp. 228–239. ISSN: 00199958. DOI: 10.1016/S0019-9958(58)80003-0.
- [21] J. Achiam. "Spinning Up in Deep Reinforcement Learning". In: (2018).
- [22] V. Mnih, D. Silver, and M. Riedmiller. "Deep Q Network (Google)". In: (), pp. 1–9.

- [23] L.-J. Lin. "Reinforcement learning for robots using neural networks". In: *PhD Thesis* (1993), p. 160. URL: <https://search.proquest.com/docview/303995826?accountid=12063%7B%5C%7D0Ahttp://fg2fy8yh7d.search.serialssolutions.com/directLink?%7B%5C&%7Dauthor=Lin%7B%5C%7D2C+Long-Ji%7B%5C%7Dissn=%7B%5C&%7Dtitle=Reinforcement+learning+for+robots+using+neural+networks%7B%5C&%7Dauthor=Lin%7B%5C%7D2C+Long-Ji%7B%5C%7Dissn=%7B%5C&%7Dtitle=Reinforcement+learning+for+robots+us>.
- [24] E. Todorov, T. Erez, and Y. Tassa. "MuJoCo: A physics engine for model-based control". In: *IEEE International Conference on Intelligent Robots and Systems* (2012), pp. 5026–5033. ISSN: 21530858. doi: 10.1109/IROS.2012.6386109.
- [25] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.
- [26] T. Erez, Y. Tassa, and E. Todorov. "Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX". In: *Proceedings - IEEE International Conference on Robotics and Automation 2015-June.June* (2015), pp. 4397–4404. ISSN: 10504729. doi: 10.1109/ICRA.2015.7139807.
- [27] OpenAI. *mujoco-py*. <https://github.com/openai/mujoco-py>. 2017.
- [28] L. Pitonakova, M. Giuliani, A. Pipe, and A. Winfield. "Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10965 LNAI (2018), pp. 357–368. ISSN: 16113349. doi: 10.1007/978-3-319-96728-8\_30.
- [29] E. Coumans and Y. Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. <https://bit.ly/3jqU7Kd>. 2016–2020.