Tampere University

Noora Raitanen

EXPLORATION PROBLEM IN DEEP REINFORCEMENT LEARNING

Faculty of Information Technology and Communication Sciences
Bachelor’s thesis
April 2020

ABSTRACT

Noora Raitanen: Exploration problem in deep reinforcement learning
Bachelor's thesis
Tampere University
Information Technology
April 2020

Deep reinforcement learning (DRL) has shown great promise on problem solving complex learning problems and has enabled reinforcement learning scale to problems having high-dimensional state-action spaces. Applied to robotics DRL allows robots to solve problems directly based on low-level inputs, such as pixels or sensor data. To be able to perform tasks, the robot must know how different actions affect the surrounding environment. The process of gathering this information is called exploration.

Exploration plays a crucial role in deep reinforcement learning applications, especially in robotics, but it still has some persistent issues. Exploration aims to always gain knowledge of the unknown and if not balanced with exploitation the robot may never learn to maximize the short term reward. Exploitation refers to action selection where the selected action are known to produce good rewards. The balance of these known as the *dilemma of exploration and exploitation* is one of the greatest challenges of reinforcement learning. Other issues discussed in this thesis include the exploration in areas where the movement is restricted for reasons based safety or cost, and exploration in environments where rewards are rare, such as in object moving tasks. Another significant issue is the simulation-reality gap. Many reinforcement learning applications can be effectively produced in simulated environments, but the simulated model often fails in real world setting due to differences in action distribution.

This thesis studies different ways to perform exploration in robotics. Based on research the most used are undirected exploration methods, such as ϵ-greedy or optimal policy with added noise. These methods rarely reach the best possible result, but they are simple to use and commonly reach an adequate level of knowledge. More advanced methods, such as frontier exploration and Gonzales-Banos and Latombe's exploration, require carefully tuned parameters to success in exploration and require more calculations, but outperform the simpler methods in knowledge gain.

Keywords: exploration, deep reinforcement learning, robotics

TIIVISTELMÄ

Noora Raitanen: Eksploraatio-onglema syvässä koneoppimisessa
Kandidaatin tutkinto
Tampereen yliopisto
Tietotekniikan tutkinto-ohjelma
Huhtikuu 2020

Syväoppiminen (engl. Deep reinforcement learning, DRL) on mahdollistanut kompleksisten koneoppimisongelmien ratkaisemisen. Sovellettuna robotiikkaan syväoppiminen on mahdollistanut ongelmien ratkaisun suoraan matalan tason prosessoimattomasta datasta, kuten valokuvapikseleistä ja sensoridatasta. Syväoppiminen perustuu algoritmin itsenäiseen kokeiluun löytää toimintatapa, jossa palkintosignaali maksimoidaan. Ympäröivällä tilalla ja ympäristöllä on suuri vaikutus algoritmin toiminnan onnistumiseen ja robotin tulee olla tietoinen ympäristöstä, jossa se toimii. Tämä työ esittelee erilaisia eksploraatiomenetelmiä, joita voidaan hyödyntää robotiikassa tilan dynamiikan hahmottamiseen ja tilan rajoitteisiin tutustumiseen. Lisäksi työssä pohditaan eri menetelmien sopivuutta robotiikan tarpeisiin ja esitellään keskeisimmät haasteet.

Eksploraatio tarkoittaa informaation hankkimista robotin toimintaympäristöstä ja eri tapahtumien vaikutuksesta ympäristön tilaan. Nämä tiedot ovat robotille välttämättömiä toimivan toimintamallin luomiseksi. Eksploraation tavoitteena on kerätä riittävästi informaatiota ympäristöstä, jotta robotti pystyy toimimaan siinä itsenäisesti. Eksploraatio voidaan toteuttaa usealla eri strategialla ympäristöstä ja tarpeesta riippuen. Eksploraation suurin haaste robotiikassa on sen toteuttaminen simulaatioiden ulkopuolella, sillä eksploraatio vaatii useita toistoja, joiden tekeminen voi olla vaarallista, kallista ja aikaa vievää. Lisäksi on huomioitava ympäristön mahdollinen muuttuminen eksploraation aikana sekä eksploraatiota ohjaavien palkintosignaalinen tasainen saatavuus.

Tutkimus osoittaa, että yleisimmät käytetyt eksploraatiomenetelmät ovat satunnaisuuteen pohjautuvia menetelmiä. Nämä menetelmät eivät usein saavuta parasta mahdollista tulosta, sillä ne eivät huomioi aiemmin esiteltyjä ongelmia. Niiden käyttöä kuitenkin puoltaa menetelmien käytön helppous ja usein riittävän ympäristön dynamiikan tuntemuksen saavuttaminen. Edistyneemmät menetelmät vaativat tarkemmin määriteltyjä parametreja toimiakseen, mutta ovat kykeneviä määrittelemään myös haastavien ympäristöjen dynamiikkaa. Lisäksi edistyneemmät menetelmät mahdollistavat eksploraation tehokkuuden huomioimisen ja kohdentamisen, jolloin ympäristöstä syntyvä kuva on kattavampi ja täyttää halutut tarpeet paremmin.

Avainsanat: koneoppiminen, robotiikka, eksploraatio, syväoppiminen

PREFACE

This thesis concludes my bachelor's degree studies. Firstly, I would like to thank my supervisor Nataliya Strokina. Without her guidance this thesis would not have been possible. I would also like to thank Wenyan Yang and Nikolay Serbenyuk for helping me with this thesis and answering my questions when needed.

Lastly, I want to say my thanks my family and friends, who tirelessly supported me through the process of writing this thesis.

Tampere, 29th April 2020

Noora Raitanen

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

π	Policy
a	Action
DDPG	Deep Deterministic Policy Gradient
DRL	Deep Reinforcement Learning
E	Environment in which an agent operates
GB-L	Gonzales-Banos and Latombe's exploration method
MDP	Markov Decision Process
R,r	Reward
RH-NBVP	Receding Horizon Next-Best-View planning exploration algorithm
s	State
safeDAgger	Safe Data Aggregation
UCB-1	Upper Confidence Bound exploration algorithm
V	Value function estimating expected return of an action

1 INTRODUCTION

Nowadays robots easily outperform humans in many tasks that require complex planning or precision. Artificial intelligence has successfully played games of Go against the grandmaster Lee Sedol with a win rate of 4 to 1 [7]. Go is a game that cannot be calculated by a computer due to the high number of available legal moves. Similarly, robots outperform humans in high precision tasks such as welding with ability to control voltage, current and welding speed constantly without getting distracted [24]. However one skill remains unobtainable for robots. Humans can easily observe their surrounding and adjust their behavior according to the events. For example, driving a car is a rather simple task that is easily managed by many adults. However, in the world of robotics driving a vehicle is seen as a complex task. It requires a constant, updated knowledge of the surroundings and suitable actions. The driver must be able to determine if the path in front of that vehicle is suitable for driving, if the vehicle is on the pavement and if the speed is suitable for the situation. These are aspects we humans manage naturally, robots on the other hand, must be thought individually.

One way to allow the robot to learn is to use demonstrations of the task to be performed. The robot observes the given demonstrations and as the simplest form of learning, aims to mimic the actions to solve a task. However, simply mimicking demonstrated motions rarely produces adequate movements to solve the task. This may be due to the inability of the robot controller to perfectly execute the movements in demonstrations or slight differences between the demonstrated and real task [4]. Another issue is the inability of robot controllers to adapt to changes in the surrounding world. This is an issue especially in real world setting where a slight change in lighting or weather may cause the need to fine-tune the controller. Learning from demonstrations is a useful tool for many robotics applications but the lack of adaptability causes the need for solutions with active learning and adaptability. Reinforcement learning is a learning paradigm that aims to provide adaptability to the environment based on trial and error. The robot is not provided with examples or demonstrations of wanted movements but a reward function, which determines the success of a taken action. The system receives a reward indicating the success of behavior after each consecutive action. The aim of the agent is to maximize that reward by improving the action selection. [13]

To perform adequate actions and solve tasks, the agent must be aware of the surrounding environment, i.e. how an action affects the current state the robot is at [13]. To reach a satisfactory knowledge of the environment, the agent must explore it or have a model providing knowledge of the environment. Exploration is the process of gathering information and knowledge of the surrounding, unknown environment. The exploration problem of robotics aims to allow robots to be able to succeed in this. [30]

Using pure exploration will restrain the agent from maximizing reward on short-term. This is due to selection of actions that ensure exploration but may cause the rewards to be negative. The amount of exploration has also a great effect on required learning time and the quality of learning. Exploitation is used to adjust these issues. Exploitation is the opposite of exploration and aims for selection of actions that have proven to be successful in the past. Pure exploration may lead to a situation where the selected action are not optimal. This will restrain the agent from maximizing long-term rewards. This issue of balance is known as the *dilemma of exploration and exploitation*. [30]

The purpose of this thesis is to provide an overview of different exploration strategies used in robotics both in real-world and in simulated environments. The thesis begins with introduction of reinforcement learning and deep reinforcement learning in Chapter 2. In Chapter 3 exploration is discussed in more detail and different methods of exploration are discussed in detail. The discussed methods are compared in Chapter 4, where also simulated and real-world use cases are presented. To conclude this thesis, Chapter 5 presents a conclusion of this thesis and discusses possible future work regarding the topic.

2 REINFORCEMENT LEARNING

Reinforcement learning is a type of machine learning that bases on trial and error. It assumes the world can be described as set of stages, S, with a fixed number of actions, A, that an agent performs [26]. A reinforcement learning algorithm learns a model by interacting with the environment aiming to perform the best possible solution for the current problem [5]. The model includes a set of the states *S* in environment *E* the agent operates in, the possible actions *A* in each state and set of scalar reinforcement rewards [14].

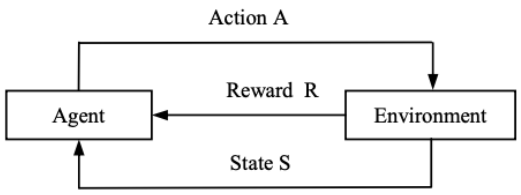


Figure 2.1. *Reinforcement learning model. [17]*

Reinforcement learning can be separated to model-free or model-based approaches. Model-free applications aim to learn policies directly with value function. Value function measures the probability of selecting each state in the environment. This type of learning does not require explicit estimates of the surrounding environment. Model-based approaches aim to estimate the surrounding environments dynamics and use that estimate to compute the expected value of actions in the environment. The benefit of model-based learning is that the model can avoid extraneous steps at the environment, which may be costly or time consuming. This makes learning faster and more efficient. [2]

Reinforcement learning problems can also be divided into policy based and value based methods. Value based methods estimate the expected reward of current state s. The value-state function V estimates the reward R received when starting on state s with pursue of a policy π [2]:

$$V^{\pi}(s) = \mathbb{E}[R|s, \pi]. \tag{2.1}$$

The optimal value function V^* always picks the action a that maximizes the expected reward. Policy based methods try to obtain the optimal policy π^*. At the beginning a parametrized policy π_θ is created. The parameters of the policy are updated to maximize the reward estimate $\mathbb{E}[R|\theta]$. The aim is to find the optimal value of θ by optimization. Value and policy based methods can be combined to reach a hybrid solution, having the benefits of both methods. This hybrid method called an actor-critic approach, is created by combining the value function with policy search. The actor is responsible for learning a policy using feedback provided by the critic as value function. Figure 2.2 describes the method. The environment gives the current state to the actor, which selects an action to perform according to the policy. The critic obtains the state and the received reward from selected action to calculate the error. This information is used to update the the actor and the critic to minimize the error. [2]

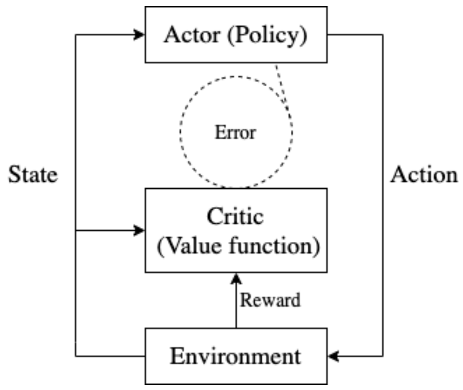


Figure 2.2. *Actor-critic model. Recreated from [2].*

2.1 Markov Decision Process

In reinforcement learning the interaction between an agent and the environment is typically described as Markov Decision Process (MDP). Markov Decision Process is a framework for solving the path finding problems [25]. The agent's interaction with the environment can be described as Markov Decision Process, MDP. MDP is described as a discrete set of values, (S, A, p_M, r, γ), where S is state space, A denotes the action space and p_M defines the transition dynamics. At each time step the agent performs an action A and receives a reward $r \in R$. The aim is to maximize the reward weighted by the discount factor γ [10]:

$$R_t = \sum_{i=t+1}^{\infty} \gamma^i r(s_i, a_i, s_{i+1}). \tag{2.2}$$

MDP models the probabilities of the agents transitioning from state s to state s' with an action a and receiving a reward r [25]. The actions are selected in respect to a policy π: $S \rightarrow A$. The policy is a set of rules that define the strategy the agent uses [18]. The action is chosen from a distribution states $s \in S$ that the policy has visited prior. The policy has a value function that indicates the expected return of an action a taken on a stage s:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[R_t|s,a]. \tag{2.3}$$

The value function for a policy π is computed with Bellman operator T^{π} [10]:

$$T^{\pi}Q(s,a) = \mathbb{E}'_{\sim}[r + \gamma Q(s', \pi(s'))]. \tag{2.4}$$

Bellman equation is commonly used to iteratively update the value function, $Q_{i+1}(s,a)$. In theory, this type of value iteration algorithms converge to the optimal value $Q_i \to Q^*$ as $i \to \infty$. In practice this is difficult since the value function is estimated separately at each iteration due to lack of generalization. Instead, a function approximator can be used to estimate the value function [20]:

$$Q(s,a;\theta) \approx Q^*(s,a). \tag{2.5}$$

In reinforcement learning the function approximator is commonly a linear function approximator, but it can be also non-linear, such as a neural network [20].

2.2 Deep Reinforcement Learning

Deep reinforcement learning (DRL) refers to reinforcement learning algorithms that use a neural networks for value function approximations [8]. It is a step towards building an autonomous system which is able to comprehend the world on a higher level. One of the most important properties of DRL is the usage deep neural networks which allows the systems to find low-dimensional features from high-dimensional data, such as from image pixels or text. This allows DRL applications to perform tasks that have previously been impossible, such as play video games. An experiment introducing an algorithm able to play Atari 2600 game directly from pixels is considered to be the first real success of deep reinforcement learning techniques [20]. The algorithm successfully outperforms human experts in three of six games [20]. This experiment also introduces end-to-end reinforcement learning which allows all controls to be based directly on low-level inputs, in this case pixel data. This allows systems to solve more complex issues. The same technique can be applied to robotics as well allowing them to learn directly from sensor data. [20][2]

3 EXPLORATION

The essence of reinforcement learning is learning with interaction. This means the agent tests different actions in the environment and observes the effects. The agent must acquire experience of the environment to behave in a presumed way and to be able to obtain the highest possible rewards [3]. The agent optimizes the actions based on the rewards it gets [2]. The process of getting to know the environment by trying new actions and learning to choose the best action to take is called exploration. It aims for better approximation of a policy to gain higher rewards in the future states. The less time and steps in the environment the agent requires to reach the desired level of exploration, the less time is needed for learning a policy or a model. This makes the application more time and performance efficient. Simultaneously reducing the learning time usually increases the cost of learning, for example the performance requirements during the learning. [3]

Exploration strategies can be divided into two categories: directed and undirected. Directed strategies consider the agents history in the environment to influence the future selection of exploration space. This knowledge is used to guide the exploration process to make it more efficient. Undirected strategies explore the environment with randomly generated actions without consideration of the agents history in environment. Considering the exploration-exploitation trade-off, the undirected strategies often modify the probability distribution from which the actions are selected. [9]

3.1 Dilemma of exploration and exploitation

Exploration prevents efficient short term reward maximization, because it aims always to select an action that has not been selected prior. The rewards received from these actions are often non-optimal. The exploration process is balanced with exploitation. Exploitation is the strategy where an action the model or policy knows to provide good reward is repeated with the aim of maximizing the reward. However, exploitation prevents maximizing the long term reward, because known actions may produce good rewards, but ignore the long-term optimal actions. This issue is referred as *the dilemma of exploration and exploitation*. Finding the balance of exploration and exploitation is a significant issue in reinforcement learning. [29]

To balance exploration-exploitation ratio, some methods consider how often a state-action pair is visited. This restricts the policy always selecting the most optimal or similar actions and enforces explorations. Similarly, if all visited states are unknown, the algorithm is forced to select known actions enforcing exploitation. If the model is efficient enough in learning, it can be guaranteed to have a rather correct results after a sufficient number of state-action visitations have been performed [19]. Other used methods to balance exploration-exploitation ratio in reinforcement learning are to utilize counters to control action selection, use model learning, or use reward compensation for non-optimal rewards. [29]

3.2 Strategies

Exploration in an unknown environment can be executed following multiple different techniques. The goal of exploration is always to reach an adequate understanding of the environment to execute tasks successfully. A single exploration strategy is often suitable for only in a limited number of environments making the correct selection of strategy essential. This section introduces some commonly used strategies.

3.2.1 Directed

Directed exploration strategies use different measures to assess the success of exploration. *Counter based exploration* counts the state-action pair visitations and evaluates the success based on exploration-exploitation ratio. *Error based exploration* relies on exploring the environment on areas where the errors were large on last update and *recency based exploration* bases on exploring actions that have not been explored for the longest time. [28]

A-C-G

A-C-G is an information based exploration strategy, basing on the concept of relative entropy. It uses an evaluation function to rate possible observation points. The evaluation function aims to provide the best observation spot to gather as much new information as possible with consideration of travel distance and overlap with previous known area. The evaluation function for observation point p is described as:

$$f(p) = \frac{1}{N+A} \sum_{i \in A \cup U} ln(\frac{\sigma_{\text{unc,i}}}{\sigma}) + N \cdot ln(\frac{\sigma}{P}) + \sum_{i \in A} ln(\frac{\sigma}{\sigma_{\text{p,i}}}) + N \cdot ln(\frac{2\pi c}{\sigma}), \quad (3.1)$$

where
N is the number of points expected to be visible from point p,
A is the number of known points observed again from point p,
$\sigma_{\text{unc,i}}$ defines the standard deviation of error in measurements
σ is the standard deviation of sensor accuracy,
P is the expected area mapped,
$\sigma_{\text{p,i}}$ is the standard deviation of already sensed point i, and
c is the distance between the current position of robot and point p. [1]

Frontier exploration planning

This method is based on *frontiers*, the regions on the border between known open space and unexplored space. The method aims to maximize the information gain during exploration by moving to the boundary between known space and unexplored space. The space is assumed to be contiguous and the path to the robots original position must exist at all times. The border of frontiers and unknown move until all the space has been explored or only inaccessible frontiers are available. [31]

Frontier exploration planning works well in environments consisting of separate regions but often reaches full state exploration slowly due to back and forth movement between regions [23]. Figure 3.1 shows the steps frontier exploration. The agent detects three frontiers (0,1,2), moves to the closest one (Frontier 0) and is able to expand the known area marked white [31].

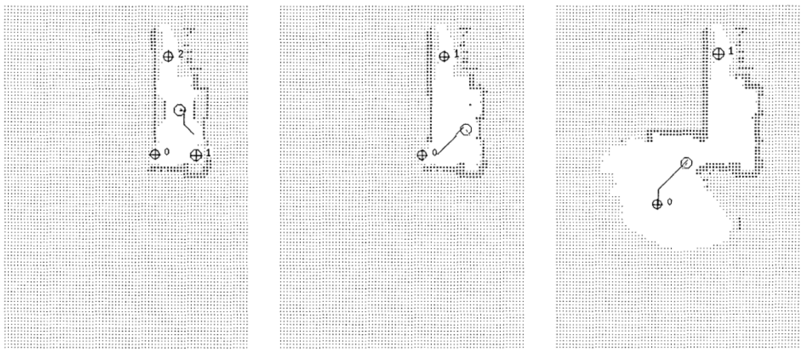


Figure 3.1. *Frontier exploration. [31]*

Gonzales-Banos and Latombe's

The Gonzales-Banos and Latombe's (GB-L) exploration method aims to find the next best position to expand the known area. It evaluates the next candidate observation position p based on the new information that can be obtained and the cost of travel to reach the position:

$$f(p) = A(p) \cdot exp(-\lambda \cdot L(p)), \tag{3.2}$$

where $A(p)$ is the estimate of unexplored area visible from position p, $L(p)$ defines the length of path to reach position p and γ defines a weight parameter based on obtainable information and cost of travel. The point p providing the greatest value of $f(p)$ is selected as next point to explore. [1]

Receding Horizon Next-Best-View planning

RH-NBVP is a sampling based approach that picks the best next viewpoint available in combination with Rapidly-exploring Random Trees (RRT). RTT scores each point p based on new information obtainable, which is the weighted with the cost of moving there:

$$s(p) = g(p) - e^{-\lambda c(p)}, \tag{3.3}$$

where $g(p)$ marks the potential information gain, $c(p)$ defines the cost of moving to the point p and λ is a penalty coefficient to control the moving distance. Figure 3.2 visualizes the sampled points and possible information gain on each. [23]

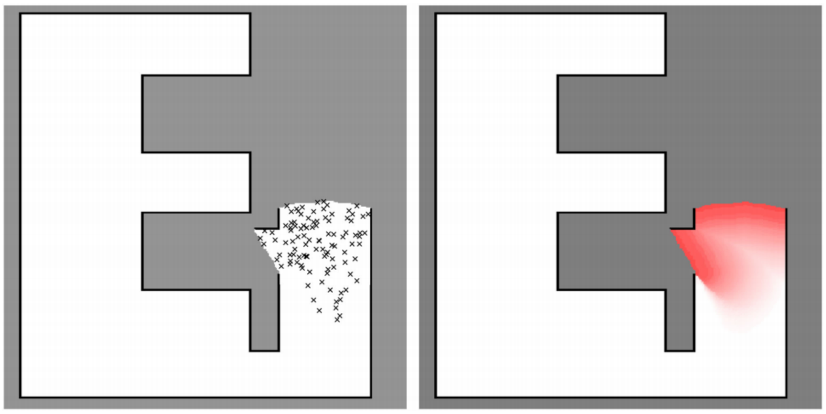


Figure 3.2. *Left: Sampled points. Right: Possible information gain (marked red) of unknown area (marked gray) at each point. [23]*

RH-NBVP assumes the world as a bounded 3D space V $\subset \mathbb{R}^{⊮}$. The exploration problem referred to be completed when $V_{free} \cup V_{occ}$ = V / V_{res}, where V_{free} refers to space that is free and V_{occ} refers to a space with an obstacle.The exploration is done by a robot and sensors, and some spaces will not be explored, such as hollow spaces and narrow pockets. This residual space is denoted as V_{res}. The paths for exploration are only planned through free space V_{free} to allow collision-free navigation in space. After reaching a sufficient V / V_{res} ratio, the exploration ends and the space has been explored fully. [6] Receding Horizon Next-Best-View planning excels in exploring individual regions, but may not be able to cover all regions in case multiple are available. It also may get stuck at a local optimum and be unable to cover all space available. [23]

Upper Confidence Bound

Upper Confidence Bound (UCB-1) algorithm keeps track of the selected actions and counts how many times an action a has been executed on a state s. The strategy is to select first all actions once when a state is visited. After selecting each action, the algorithm begins to select the action a_t with the highest expected return in state s_t:

$$a_t = argmax_a(Q_t(s_t, a) + bonus(s_t, a)), \tag{3.4}$$

where bonus refers to exploration bonus calculated after each state has been visited at least once:

$$bonus(s_t, a^i) = 100 \times C \times \sqrt{(2} \times \frac{logN(s_t)}{N(s_t, a^i)}). \tag{3.5}$$

In bonus calculation N defines the selection rate of an action on state s_t. N(s,a) defines how often the action a has been selected on state s. C defines the amount of exploration and ranges from C = 0 upwards. When C = 0 there is no exploration. [28]

3.2.2 Undirected

Undirected strategies are driven by random action selection. The simplest form is to always select a random action regardless of the reward received or expected to be received. Methods selecting the action based on utility of an action are also considered undirected. [28] Alongside strategies introduced in this section two often used undirected methods are random exploration and optimal policy exploration with added noise. Random exploration strategy selects a random action each time and optimal policy exploration with added noise selects the optimal action, which randomly gets noise added to ensure exploration.

ϵ -greedy

ϵ -greedy method is one of the most commonly selected methods of exploration [28]. This is due the method not having the requirement to memorize exploration specific data and the method is known to reach near optimal results in many applications [29].

The agent chooses a random action with probability of $\epsilon \in$ [0,1] and otherwise the best possible action available [29]:

$$\pi(s) = \begin{cases} \text{random action from A(s),} & \text{if } \xi < \epsilon \\ \text{argmax}_{a \in A(s)} Q(s, a), & \text{otherwise.} \end{cases}$$

The action selection is shown in Figure 3.3. A random number $0 \leq \xi \leq 1$ is selected at each time [29]. The larger value of ϵ, the more exploration is performed by the agent [28]. When ϵ is decreased over time, the agent processes towards more exploitation [2].

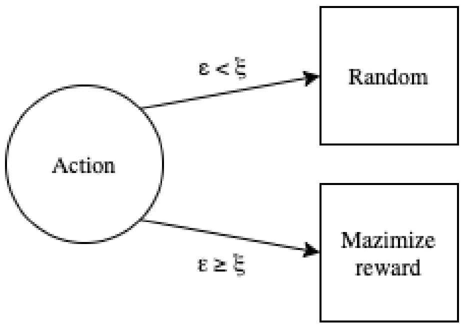


Figure 3.3. *ϵ-greedy action selection.*

Boltzmann exploration

Boltzmann exploration (softmax exploration) uses a Boltzmann distribution function to assign a probability $\pi(s_t, a)$ to selected actions. The actions are selected based on $\pi(s_t, a)$, which denotes the probability of selecting an action a in state s:

$$\pi(s_t, a) = \frac{e^{Q_t(s_t,a)/T}}{\sum_{i=1}^{M} e^{Q_t(s_t,a^i)/T}}. \tag{3.6}$$

Temperature parameter T is a positive number that restricts or encourages exploration. When T = 0 no exploration is performed and when $T \to \infty$ agent select a random value each time. With moderate values of T the agent will likely select optimal actions. Other actions are rated based on optimality. Compared to total random selection, where the worst possible action and second-best action are equally likely to be selected, this helps the agent to select better actions and create a successful policy faster. [28]

Pursuit

Pursuit is a method that maintains estimates of action-value pairs and preferred actions of current state. At each time step an action is selected with greedy selection. For a policy the probability of selecting the selected action is incremented towards one with a fraction β:

$$\pi_{t+1}(s_t, a^*_{t+1}) = \pi_t(s_t, a^*_{t+1}) + \beta[1 - \pi_t(s_t, a^*_{t+1})]. \tag{3.7}$$

To avoid all actions having high probabilities, the probabilities of selecting other actions are decreased simultaneously towards zero with the same fraction β:

$$\pi_{t+1}(s_t, a) = \pi_t(s_t, a) + \beta[0 - \pi_t(s_t, a^*_{t+1})]. \tag{3.8}$$

The value of β in both equations describes the leaning rate for action preferences and is always β > 0. [28]

3.3 Practical challenges in DRL and exploration

Deep reinforcement learning is growing rapidly, but it still has some persistent issues especially in real-life oriented applications. Previously explained techniques of exploration assume unrestricted access to the environment and possibility test all possible actions in each state. All taken actions are assumed to return a result defining the success of taken action. These conditions are not always fulfilled and other measures must me taken into account.

3.3.1 Restricted movement in environment

Practical reinforcement learning applications must often learn from a fixed dataset without a possibility to have further data collection or without interaction with the environment due to the process of information gathering being for example costly or difficult to obtain. This means learning from a data collected in prior is a crucial requirement when scaling reinforcement leaning applications to real-life applications, such as self-driving cars. [10]

Batch-constrained deep reinforcement learning is one solution to a situation where exploration is not possible. [10] To allow the agents to learn off-policy from a dataset, growing batch learning is used. It is done by collecting and storing data in an experience replay dataset which the agent uses for training before further data collection can occur. However, the agent fails to learn if the value distribution within the dataset does not match with the distribution of used policy. This error is denoted as extrapolation error. Extrapolation error is the mismatch between state-action pairs in a dataset and the actual state-action visitations under the policy. This leads unseen state-action pairs to have an unrealistic values. Without proper correction this will cause the algorithm to perform poorly due to impossibility of learning the value function for a policy selecting actions outside the provided dataset. The error is commonly has several reasons: Absent data, training mismatch or model bias.

All state-action pairs may not be available for selection. When a state-action pair (s, a) cannot be selected, the value must be approximated using similar data and near by state-action pairs. When there is no sufficient data near, the approximation $Q_\theta(s'$,$\pi(s'))$ will be arbitrarily bad. In some cases the data distribution in the dataset does not always match with the expected distribution with current policy causing training mismatch. This leads the value function making inadequate estimates of actions the current policy selects. Training mismatch may occur even with sufficient data since the loss is given with respect to the likelihood and the value function may provide poor estimates. The loss is defined as:

$$L \approx \frac{1}{|B|} \sum_{(s,a,r,s') \in B} ||r + \gamma Q_{\theta'}(s', \pi(s')) + Q_\theta(s, a)||^2, \tag{3.9}$$

where $Q_{\theta'}$(s',π(s')) is estimate by the value function. The value function for a policy is calculated with a Bellman operator Tπ. During Q-learning an approximation of the Bellman operator is created by sampling values from batch B. The batch has a finite number of samples and the sampling may produce a biased estimate of the transition dynamics, where the estimate is calculated with based on values in batch instead of MDP. This causes the model to be biased. [10]

3.3.2 Environments with sparse rewards

In real-life robotics rewards are commonly sparse due to the common aim to achieve a binary result, such as moving an object to a desired location. This makes exploration is space difficult because the agent receives rewards sparsely and has difficulties to tell if a movement was successful or not. Commonly robotic tasks require multiple steps to success and tasks must be generalized due to varying instances. This may result in a condition where the agent sees rewards so rarely, that it fails to find successful actions. In cases like these demonstrations from previous experiences can be used to solve the issue of exploration. Demonstrations are incorporated into reinforcement learning allowing the random exploration phase to be removed and non-zero rewards and a successful policy are reached within reasonable time of training. [21]

3.3.3 Simulation-reality gap

DRL has proven to be efficient in many applications in simulated environments, but the problem of applying same techniques to real-life environments persists [27]. Training deep learning networks requires a lot of data to be collected and available for use. Data collection is often costly and time consuming process, but can be eased by using simulated environments for data collection. However, the simulate data will have a different data distribution as the true environment would provide. This leads to situations where models trained in simulated environments commonly fail to generalize solutions in real-life situations. [27]

Closing this gap is an essential requirement for wider use of simulation based methods and solving it would ease the development of real world applications that require exploration in dangerous or costly manner. One method is to improve the simulations to make policies more transferable from simulation to real world solutions. Adding noise to the simulation generates more robust behavior that tolerates the change of environment and co-evolving the robot and the simulator narrows the differences between reality and simulation while the robot learns [16]. The simulation is adapted based on the robot's interactions with the environment allowing the robot to create a policy that does not distinguish the difference between simulation and real-world [32]. Other methods combine simulated learning with limited exploration in real environment to allow the policy to adjust on real-world setting. Techniques modeling the gap itself have also been proposed. These methods aim to create a deep learning model of the reality gap and apply that directly to the controller allowing it to adjust to the change of environment. These methods successfully close the gap in complex, high-dimensional problems but require a rather high volume of computing and number of real world experiments. [16]

3.3.4 Safety measures

In real-world reinforcement applications the note of safety is crucial. Allowing a robot or an autonomous machine train uncontrollably can lead to dangerous situations and hence the training must be controlled. [27]

The safety of exploration is commonly assured by using safety-based controls. Safe Data Aggregation (SafeDAgger) uses a safety policy which aims to predict errors the primary policy would make. The primary policy is trained as usual with supervised policy. SafeDAgger can indicate if the primary policy is probable to depart from the reference policy by taking partial observations of a state and policy inputs. [27]

In [15], an experiment of autonomous driving the system is trained in real-world environment. The aim was to have a car to stay on a road using deep reinforcement learning. In this case a safety method was implemented to stop the car every time it reached an unrecoverable position. This allowed the driver to return the car back to the center of the road and continue the training. The vehicle was also stopped in a case the speed exceeded 10 km/h or the driver intervened in any way. Every time the car would be reset and next training episode would continue from the center of the lane. [15]

4 APPLICATIONS AND PERFORMANCE EVALUATION

Different applications are used for robotics exploration. Exploration methods and their performance are evaluated in this section. The selection of a suitable strategy for the environment has an extensive role on the success of exploration. This section introduces different methods used. All methods are explained in detail in Chapter 3.

4.1 Applications

Robotics applications utilizing deep reinforcement learning can be divided into groups based on task aim: navigation and manipulation. Navigation tasks aim to allow the robot to be able to successfully perform movements within that area without need for further supervision. Manipulation tasks cover all tasks related to changing the surrounding world, such as moving or sorting objecting.

4.1.1 Navigation

One of the most common navigation task is autonomous driving. The selected exploration strategy in [17] on simulated autonomous driving is the Boltzmann method. During the learning phase, the agent will have limited knowledge of the state-action pairs making efficient exploration necessary in order to obtain sufficient knowledge. Furthermore, this is needed to allow the Q values to converge. The used temperature value T decreases towards the end to decrease exploration and to ensure learning. The system uses a neural network based Q-learning to solve the task. The simulated vehicle succeeds on avoiding obstacles during simulation and reaches the target successfully after the training period. [17]

A real-life autonomous driving is presented in [15]. This method uses Deep Deterministic Policy Gradient (DDPG) [18] as a reinforcement learning algorithm and exploration is created by adding Ornstein-Uhlenbeck process noise to the optimal policy. The noise x_t is added to each action:

$$x_{t+1} = x_t + \Theta(\mu - x_t) + \sigma\epsilon_t, \tag{4.1}$$

where Θ, μ, σ are hyperparameters used in tuning and $_{tt}$ are independently and identically distributed random variables sampled from a normal distribution $N(0, 1)$. Figure 4.1

shows the different parts of the used actor-critic algorithm and used inputs. [15]

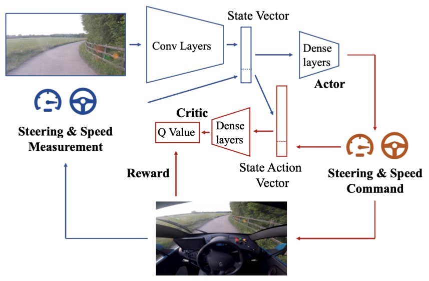


Figure 4.1. *Reinforcement algorithm used for autonomous driving. [15]*

The algorithm is first tested in a simulated environment, where the learning parameters are also tuned to create a successful policy. After reaching successful results in simulation the same algorithm is applied to a real car. The vehicle is equipped with monocular front-facing camera and aims to drive for 250 meters. The simulated environment and real-world view are shown in Figure 4.2. The reinforcement algorithm is able to solve the problem and successfully drive on it's own after 30 minutes of training. [15]

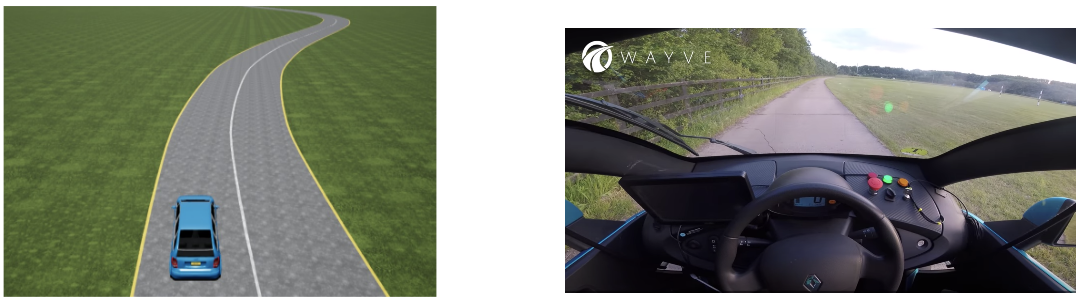


Figure 4.2. *Simulated training environment and real-word route view. [15]*

Another example of navigation is provided in [22], in which robots can be used for urban search and rescue (USAR) applications. In USAR applications the ability to explore a unknown, cluttered environment efficiently is essential. In [22], a frontier based-exploration with deep reinforcement learning is presented. This method allows robots autonomously explore environments of varying size and layout by finding appropriate frontier locations. [22] In this method the robot creates a 2D grid-like model of the surrounding world. The areas are in the grid are describes ad open, occupied or unknown cells. The created grid is shown in Fig. 4.3. The exploration of the area is done using A^* algorithm to generate a route from robot's current location to the most suitable frontier location. The robot successfully explores unknown, cluttered environment in both simulation and in real-life setting. [22]

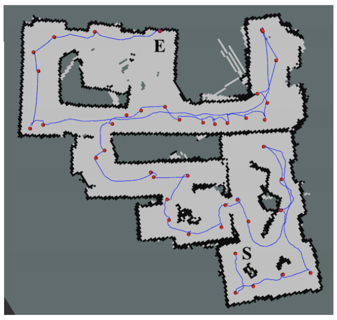


Figure 4.3. *Generated occupancy grid. Occupied areas are marked black, open area is light gray and unknown area is dark gray. The line describes the robots movement from start S to end E with dots as visited frontier locations. [22]*

4.1.2 Manipulation

A study introduced in [11] uses a real world settings to teach a robot arm to block Lego's while avoiding an obstacle. The task is shown in Figure 4.4 The training of the robotic arm is done in two sections: first aiming to learn to avoid the obstacle and second learning to block the Lego's. Afterward both policies are combined to allow the robot perform both actions simultaneously. [11]

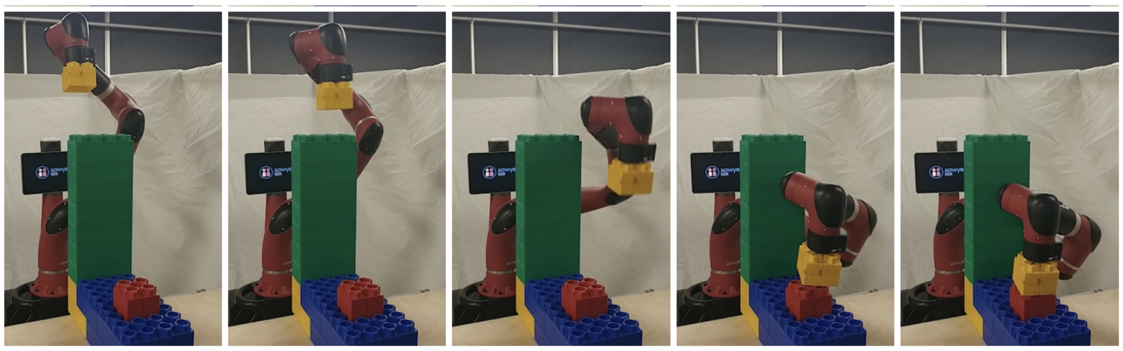


Figure 4.4. *Task visualization of Lego stacking. [11]*

The experiment is concluded with three different learning algorithms: Soft Q-learning [12], Deep Deterministic Policy Gradient [18] and Normalized Advantage functions [11]. The selected exploration strategy is Boltzmann exploration. This allows all actions to have a non-zero probability and actions with higher probability are more likely to be selected. This allows the policy directly explore regions with higher rewards. The exploration is sufficient enough and after a training period the robot is able to stack Lego's. [11]

Robots have traditionally been used for simple repetitive tasks, such as assembly and robot welding has existed for decades, but until recently these application have not been able to utilize deep reinforcement learning. An end-to-end robotic welding system is introduced in [33]. The system aims to use deep reinforcement for calibrating the arm movement based on visual input. For each image the system model predicts weld point pixel coordinates and assesses the 3D coordinates for the arm to reach the points. The system uses actor-critic method to obtain models and consists of two models: pixel-to-point model estimates the pixel coordinates of weld points and the point-to-wrist model estimates the 3D coordinates for the wrist by maintaining geometric model of the environment. The models are trained using reinforcement learning. The learning process is guided by the critic calculating a policy gradient bade on feedback from the environment. [33]

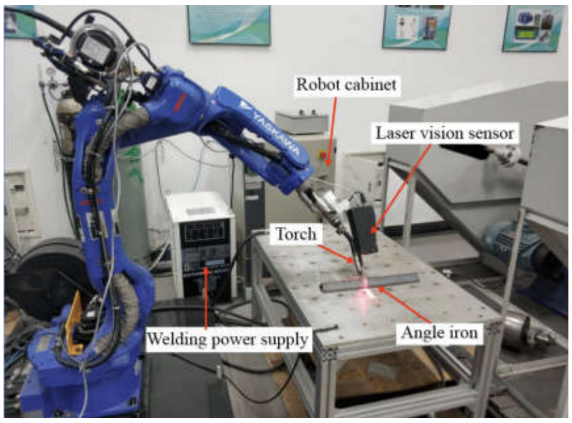


Figure 4.5. *Robot welding arm. [33]*

The trained system is tested using YASKAWA welding robot. Figure 4.5 shows the used test set-up. The robot is able to weld curved lines purely based on visual input. The system shows a control error below 0.8 mm, which is considered as the limit for safe usage. [33]

4.2 Method comparison

All methods compared are introduced in detail in Chapter 3.

Four different exploration strategies in simulated environment are compared in paper by Amigoni, [1]. The test setup consists of a holonomic mobile robot in two-dimensional environment. The robot is equipped with sensors providing a 360° view on range r. In this experiment the robot is considered as dimensionless. The robot aims to create a map of the surroundings in three different environments with different radius: office-like environment with corridors, large open space, and large space with multiple obstacles. The environments described are presented in Figure 4.6. [1]

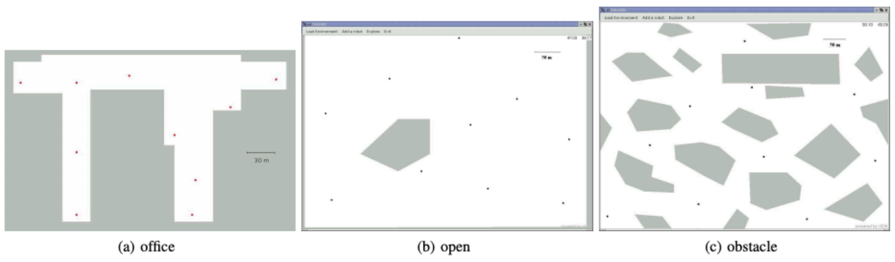
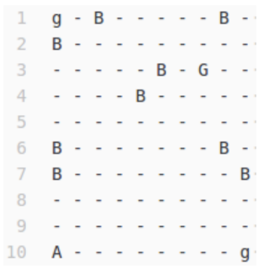


Figure 4.6. *Used test environments. [1]*

The strategies compared are random, greedy, GB-L, and A-C-G. In all environments random exploration performs the worst and A-C-G succeeds to explore the most. Large-environments and environments with obstacles are most successfully explored by GB-L and A-C-G. These methods note the cost of computing. A-C-G is the most efficient method in open environments. Greedy method works efficiently in smaller spaces. [1]

The results show that small environments are not always explored faster than large ones. Results also indicate that environments with obstacles are explored faster than open environments. This is due the used end condition, that ends the exploration when a certain percentage of the area is successfully explored. If the larger environment contains obstacles, the percentage of free area is smaller and the aimed goal percentage will be reached faster. The results also suggest that methods balancing utility and cost are commonly more efficient than methods that only use utility. This explains the performance difference between GB-L and greedy method. [1]

Similar results were obtained using maze solving in a simulated test environment [28]. The comparison of four different exploration strategies, UCB-1, softmax, pursuit and ϵ-greedy, with a task of using Q-learning to solve random stochastic mazes. The mazes are 10 $\times$ 10 mazes with obstacles, two sub-optimal goals g and goal G, which is optimal. The maze is pictured in Figure 4.7. Average steps required to reach the goal and average cumulative reward were calculated for each type of strategy. The strategies are first tested with non-optimized parameters and later with optimized parameters. When tested with non-optimal parameters, ϵ-greedy shows the slowest increase in performance and requires the most time before converging. Pursuit method converges the fastest, but provides overall the worst. After finding the optimal parameters for each method, all methods are able to reach the optimal goal state. ϵ-greedy commonly shows the worst performance, while softmax consistently reaches the optimal goal with best performance. [28]



```
g - B - - - - - B -
B - - - - - - - - -
- - - - - B - G - -
- - - - B - - - - -
- - - - - - - - - -
B - - - - - - - B -
B - - - - - - - - B
- - - - - - - - - -
- - - - - - - - - -
A - - - - - - - - g
```

Figure 4.7. *Stochastic random maze. [28]*

Overall Boltzmann method outperforms the other methods, but requires a well optimized parameters. Boltzmann and pursuit are more dependent on set parameters than ϵ-greedy and UCB-1, which makes the more difficult to tune to different settings. In this test-set, similarly to the tests in [1], ϵ- greedy performs the worst, but does not require well defined parameters similarly with UCB-1. [28]

5 CONCLUSION

The exploration problem introduced and discussed in this thesis is one of the most persistent issues of deep learning in robotics. We have assessed different exploration options for various deep reinforcement learning problems, specifically application in robotics problems covering issues in navigation and manipulation.

Based on results shown in Section 4.2 Method comparison we can conclude that the most used exploration strategies are simple strategies such as random exploration, ϵ-greedy and optimal policy with added noise. These strategies are rarely the most efficient or reach the best possible results. However, they commonly reach an adequate level of knowledge of the surroundings and are able to perform wanted tasks. Simpler strategies do not require as careful parameter tuning, which makes them easy to use in different situations and supports the popularity. More advanced methods, such as GB-L and A-C-G, outperformed these methods. More exact parameter tuning allows the algorithms explore more precisely and easily exceed the results provided by methods based on randomness. Methods utilizing more advanced algorithms showed promising results especially in simulated environments.

Many exploration strategies perform well in simulated environments, but struggle in real-world environments. Simulated environments are commonly easier to explore due to the possibility to repeat the exploration process multiple times in similar environment without safety limitations. In a simulated environment experiments can be done without a possibility of dangerous situations and constant rewards can be ensured to support exploration. These are conditions that are not commonly full-filled in real-world exploration. This requires different safety and support measures to be accounted in real-world settings, such as safety functions and reward compensation.

This topic leaves plenty of room for future research. Many of the introduces problems are not solved fully. Methods able to generalize the results and methods able to close the gap between simulated and real-world solutions are especially interesting since they will enable easier production of large-scale real-worls applications.

REFERENCES

[1] F. Amigoni. Experimental evaluation of some exploration strategies for mobile robots. *IEEE. International Conference on Robotics and Automation* (2008). DOI: 10.1109/ROBOT.2008.4543637.

[2] K. Arulkumaran, M. Deisenroth, M. Brundage and A. Bharath. Deep Reinforcement Learning: A Brief Survey. *IEEE. Signal Processing Magazine (Volume: 34, Issue: 6)* (2017). DOI: 10.1109/MSP.2017.2743240.

[3] J. Asmuth, L. Li, M. Littman, A. Nouri and D. Wingate. *A Bayesian Sampling Approach to Exploration in Reinforcement Learning*. 2009. arXiv: 1205.2664.

[4] C. Atkeson and S. Schaal. Robot Learning From Demonstration. *ICML. (Volume: 97, pp. 12-20)* (1997).

[5] H. Bing-Qiang, C. Guang-Yi and G. Min. Reinforcement Learning Neural Network to the Problem of Autonomous Mobile Robot Obstacle Avoidance. *IEEE. 2005 International Conference on Machine Learning and Cybernetics* (2005). DOI: 10.1109/ICMLC.2005.1526924.

[6] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova and R. Siegwart. Receding Horizon “Next–Best–View” Planner for 3D Exploration. *IEEE. 2016 International Conference on Robotics and Automation (ICRA)* (2016). DOI: 10.1109/ICRA.2016.7487281.

[7] S. Borowiec. AlphaGo seals 4,1 victory over Go grandmaster Lee Sedol. (2016). [Online reference, cited 25.04.2020]. URL: https://www.theguardian.com/technology/2016/mar/15/googles-alphago-seals-4-1-victory-over-grandmaster-lee-sedol.

[8] N. Casas. *Deep Deterministic Policy Gradient for Urban Traffic Light Control*. 2017. arXiv: 1703.09035.

[9] R. Farlane. A Survey of Exploration Strategies in Reinforcement Learning. (2003).

[10] S. Fujimoto, D. Meger and D. Precup. Off-Policy Deep Reinforcement learning without Exploration. (2019). arXiv: 1812.02900.

[11] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel and S. Levine. Composable Deep Reinforcement Learning for Robotic Manipulation. *IEEE. International Conference on Robotics and Automation (ICRA)* (2018). DOI: 10.1109/ICRA.2018.8460756.

[12] T. Haarnoja, H. Tang, P. Abbeel and S. Levine. Reinforcement learning with deep energy-based policies. *JMLR.org. ICML'17: Proceedings of the 34th International Conference on Machine Learning. (Volume: 70, pp. 1352-1361)* (2017).

[13] S. Ishii, W. Yoshida and J. Yoshimoto. Control of exploitation-exploration meta-parameter in reinforcement learning. *Neural Networks (Volume: 15, Issue: 4-6, pp.665-687)* (2002). DOI: https://doi.org/10.1016/S0893-6080(02)00056-4.

[14] L. Kaelbling, M. Littman and A. Moore. Reinforcement Learning: A Survey. *Springer, Berlin, Heidelberg. Journal of Articial Intelligence Research (Volume: 4, pp. 237-285)* (1996). DOI: https://doi.org/10.1007/978-3-642-24455-1_33.

[15] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J. Allen, V. Lam, A. Bewley and A. Shah. Learning to Drive in a Day. *IEEE. 2019 International Conference on Robotics and Automation (ICRA)* (2019). DOI: 10.1109/ICRA.2019.8793742.

[16] S. Kim, A. Coninx and A. Doncieux. From exploration to control: learning object manipulation skills through novelty search and local adaptation. (2019). arXiv: 1901.00811.

[17] N. Lianqiang and L. Ling. Application of Reinforcement Learning in Autonomous Navigation for Virtual Vehicles. *IEEE. Ninth International Conference on Hybrid Intelligent Systems* (2009). DOI: 10.1109/HIS.2009.118.

[18] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra. Continuous control with deep reinforcement learning. (2016). arXiv: 1509.02971.

[19] M. Lopes, T. Lang, M. Toussaint and P. Oudeyer. Exploration in Model-based Reinforcement Learning by Empirically Estimating Learning Progress. *Advances in Neural Information Processing Systems (NIPS) (Volume: 25)* (2012).

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602.

[21] A. Nair, B. McGrew, M. Andrychowicz and W. Zaremba. Overcoming Exploration in Reinforcement Learning with Demostrations. *IEEE. International Conference on Robotics and Automation (ICRA)* (2012). DOI: 10.1109/ICRA.2018.4463162.

[22] F. Niroui, K. Zhang, Z. Kashino and G. Nejat. Deep Reinforcement Learning Robot for Search and Rescue Applications: Exploration in Unknown Cluttered Environments. *IEEE Robotics and Automation Letters. (Volume: 4, Issue: 2, pp. 610-617)* (2019). DOI: 10.1109/LRA.2019.2891991.

[23] M. Selin, M. Tiger, D. Duberg, F. Heintz and P. Jensfelt. Efficient Autonomous Exploration Planning ofLarge Scale 3D-Environments. *IEEE Robotics and Automation Letters (Volume: 4, Issue: 2, pp. 1699-1706).* (2019). DOI: 10.1109/LRA.2019.2897343.

[24] X. Shuqiang, C. Shukun, G. Kuizeng and C. Wenlong. Welding scheme design of wheat harvester chassis frame based on welding robot. *IOP Publishing Ltd. IPSECE 2019, Journal of Physics: Conference Series (Volume: 1449)* (2020). DOI: 10.1088/1742-6596/1449/1/012122.

[25] O. Sigaud and O. Buffet. *Markov Decision Processes in Artificial Intelligence*. John Wiley Sons, Inc., 2013. DOI: 10.1002/9781118557426.

[26] W. Smart and L. Kaelbling. Effective Reinforcement Learning for Mobile Robots. *IEEE. Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)* (2002). DOI: 10.1109/ROBOT.2002.1014237.

[27] V. Talpaert, I. Sobh, B. Kiran2, P. Mannion, S. Yogamani, A. El-Sallab and P. Perez. Exploring applications of deep reinforcement learning for real-world autonomous driving systems. (2019). eprint: `1901.01536`.

[28] A. Tijsma, M. Drugan and M. Wiering. Comparing exploration strategies for Q-learning in random stochastic mazes. *IEEE Symposium Series on Computational Intelligence* (2016). DOI: `10.1109/SSCI.2016.7849366`.

[29] M. Tokic. Adaptive -Greedy Exploration in Reinforcement Learning Based on Value Differences. *Springer, Berlin, Heidelber. In: Dillmann R., Beyerer J., Hanebeck U.D., Schultz T. (eds) KI 2010: Advances in Artificial Intelligence. KI 2010. Lecture Notes in Computer Science (Volume: 6359)* (2010). DOI: `https://doi.org/10.1007/978-3-642-16111-7_23`.

[30] M. Tokic and G. Palm. Value-Difference based Exploration: Adaptive Control between epsilon-Greedy and Softmax. *Springer, Berlin, Heidelberg. Advances in Artificial Intelligence. Lecture Notes in Computer Science (Volume: 7006)* (2011). DOI: `https://doi.org/10.1007/978-3-642-24455-1_33`.

[31] B. Yamauchi. A frontier-based approach for autonomous exploration. *IEEE. Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'* (1997). DOI: `10.1109/CIRA.1997.613851`.

[32] J. Zagal, J. Ruiz-del-Solar and P. Vallejos. Back to reality: Crossing the reality gap in evolutionary robotics. (2004). DOI: `https://doi.org/10.1016/S1474-6670(17)32084-0`.

[33] Y. Zou and R. Lan. An End-to-End Calibration Method for Welding Robot Laser Vision Systems with Deep Reinforcement Learning. (2019). DOI: `10.1109/TIM.2019.2942533`.