Model based path planning using Q-Learning

Avinash Sharma, Kanika Gupta, Anirudha Kumar, Aishwarya Sharma and Rajesh Kumar, Senior Member, IEEE
Department of Electrical Engineering
Malaviya National Institute of Technology
Jaipur, 302017 India
Email: avinashmnit30@gmail.com, kanika100388@gmail.com, anirudha136@gmail.com,
aishwaryas7611@gmail.com, rkumar@ieee.org

***Abstract*—Though the classical robotics is highly proficient in accomplishing a lot of complex tasks, still it is far from exhibiting the human-like natural intelligence in terms of flexibility and reliability to work in dynamic scenarios. In order to render these qualities in the robots, reinforcement learning could prove to be quite effective. By employing learning based training provided by reinforcement learning methods, a robot can be made to learn to work in previously unforeseen situations. Still this learning task can be quite cumbersome due to its requirement of the huge amount of training data which makes the training quite inefficient in the real world scenarios. The paper proposes a model based path planning method using the ϵ greedy based Q-learning. The scenario was modeled using a grid-world based simulator which is being used in the initial training of the agent. The trained policy is then improved to learn the real world dynamics by using the real world samples. This study proves the efficiency and reliability of the simulator-based training methodology.**

***Index Terms*—Model Based Control, Q-learning, Reinforcement Learning, Neural Network, Grid-World**

I. Introduction

Robots based on classical robotics can accomplish difficult tasks in limited domains. They can only accomplish the tasks for which they are programmed as they don't have a generalized analytical ability. The real challenge today is to impart natural intelligence or generalized analytical ability to the robots. In other words, efforts are underway to ascertain approaches to endow human-like capability to interact with the physical environment. Application of reinforcement learning in robotics has helped in making robots realize if the desired result has been achieved by a series of certain actions. It stores this information, and when it encounters a similar situation, it tries to make a correct move. Its aim is to obtain the optimal policy by interacting with the environment. These methods[1] can be classified into model-free[2] and model-based[3], [4], [5] methods. Model-free methods obtain the optimal policy without learning the model. On the other hand, model-based methods employ the information of model which is known to obtain the optimal policy. Another aspect of the complex mechanical systems or robots is that they cannot be commanded to take arbitrary actions, one of simplest reason being, system's degree of freedom exceeds the number of actuators present in the system.

To address these issues, we have designed a task of autonomous robot navigation. But training in real world environment using conventional reinforcement learning techniques could be quite cumbersome and inefficient, the reason being that the training data is required in abundance. The aforementioned issue can be addressed by formulating a model of real world scenario in the form of grid-world like simulator. This model is used to learn the fundamental dynamics of the real world scenario using Q-learning based reinforcement learning and the weights of the neural network trained in the process are utilized to train a similar structured neural network, the function approximator employed for reinforcement learning in the real world scenario.

Several model-free and model-based approaches have been employed in the past for optimal control of robots. Abbee et al.[6] used Differential Dynamic Programming (DDP), an extension of the Linear Quadratic Regulator (LQR)[7] to obtain a controller which is optimized for the resulting model and reward function. Masuta et al.[8] put forward a perceptual system which is a combination of the retinal model and the spiking-neural network to control the robot arm equipped with a 3D-range camera. Gu et al.[9], [12] proposed a genetic algorithm (GA) approach to evolving robot behaviors and used Fuzzy Logic Controllers (FLCs) to design robot behaviors. Hachour et al.[10] employed a navigation technique based on the combination of Fuzzy Logic (FL) and Expert System (ES) for navigation of intelligent robots in unknown environments. In [11], Sato et al. constructed interpersonal communication-based system that uses pointing gestures as information. [13], [14] gives a brief review of the extensive field of computational swarm intelligence and its applications in swarm robotics, which is an interesting alternative to classical methods of robotics. Reinforcement learning and policy search methods[17], [19], [20] can cope with challenges of robot learning like high-dimensional state and action spaces. In [15], Wang et al. built a car simulation system to train and test the car on the tracks with various RL algorithms, including Actor-Critic method, SARSA(0) and SARSA(λ). [18] shows that hierarchical relative entropy policy search can learn versatile solutions and can achieve improved learning speed and quality of the found policy as compared to the non-hierarchical techniques. These methods have been implemented in robotics for playing games such as table tennis, pig-pong[21], [22], object manipulation. Survey of policy search in robotics has been provided by recent papers[17], [1]. In practice, for perception, state estimation, and low-level control, policy search applications often demand hand-engineered

978-1-5090-5320-9/17/$31.00 ©2017 IEEE

components. To improve performance, Levine et al. [16] has trained CNN(Convolutional Neural Network) using partially observed guided policy search(GPS) method, which transmutes policy search into supervised learning.

In section II, a summary of ϵ greedy Q-learning, the reinforcement learning method used for this study has been provided. Section III explains the methodology and the experimental setup used for this study. Results have been discussed in section IV. The findings of the study have been concluded in section V.

II. ϵ GREEDY Q-LEARNING

Q-learning [2] is a type of temporal difference based reinforcement learning [23] algorithm. Reinforcement learning can be seen as a machine learning algorithm that learns by interacting with its environment. It is used to train agents to take actions such that the rewards accumulated by it are maximized. The agent is the entity which interacts with the environment and receives rewards for each of its action. It updates its policy based on its interaction with the environment such that the cumulative rewards over various episodes are maximized. Table I indicates the various symbols used in the literature. As shown in eq. 1 the policy can be defined as the

TABLE I
IMPORTANT SYMBOLS USED IN THE PAPER

Symbol	
π	Policy
S_t	State at time t
A_t	Action at time t
R_t	Reward due to action A_t time t
v_t	Rewards accumulated till time t
Q_t	Q-value accumulated till time t
α	Learning Rate
γ	Discount Factor
$q(s,a)$	Q-value function

mapping of the current state to the optimal action.

$$\pi\left(a/s\right) = P\left[A_t = a | S_t = s\right] \tag{1}$$

The state is the physical information that determines a particular condition of the system at a specific time. The Markov property of memorylessness (as in eq. 2) is used to represent the state and finding the optimal policy by the iterative learning.

$$P\left[S_{t+1}|S_t\right] = P\left[S_{t+1}|S_1, ..., S_t\right] \tag{2}$$

Q leaning updates the action-value function by bootstrapping the return from an incomplete episode. The basis of Q-learning is derived from the Bellman equation of successive decomposition of the future rewards.

$$v(s) = E\left[R_{t+1} + \gamma v\left(S_{t+1}\right)|S_t = S\right] \tag{3}$$

Q-learning can be used to find an optimal action-selection policy for any given (finite) Markov decision process (MDP) [24].The learning starts with randomly generating Q-value of the policy or a reference value set by the designer. Each time the agent selects an action, it observes a reward and a new state that may depend on both the previous state and the chosen action and updates the corresponding Q-value. Q-value is updated using the eq. 4

$$Q_{t+1}(s_t, a_t) = Q(s_t, a_t) + \alpha_t(s_t, a_t)(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)) \tag{4}$$

where $Q_t(s_t, a_t)$is the old value, $\alpha_t(s_t, a_t)$ is the learning rate, R_{t+1} is the Reward, $\max_a Q_t(s_{t+1}, a)$ is the estimate of optimal future value, γ is the discount factor, $R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a)$ is the learned value.
R_{t+1} is the reward observed after performing a_t in s_t, and $\alpha_t(s, a)(0 < \alpha \leq 1)$ is the learning rate (may be the same for all pairs). An episode of the algorithm ends when system reaches a final state or "absorbing state".
The optimal q-value function $q^*(s, a)$ (as given by eq. 5) is the maximum action-value function over all policies. It tells us about the maximum amount of reward that we can extract, considering the agent is in state s and taking action a.

$$q^*(s, a) = \max_\pi q_\pi(s, a) \tag{5}$$

By using the actions corresponding only to the maximum Q-value, policy may get stuck in a local optima and will end up generating a non-generalised policy. To avoid this discrepancy, ϵ greedy approach has been used. This approach compares a randomly generated number with ϵ, based on which it opts between random and greedy action. The greedy action is the action corresponding to the maximum estimated reward. The value of ϵ is decreased linearly with each iteration to maintain a balance between exploration and exploitation.
A recent novel approach[25] is to use neural networks as a function approximator to approximate the state-action policy. This reduces the requirement of a large storage space to store the policy lookup table. The use of deep neural networks has become quite popular as they can update complex policies with the help of GPU and CPU based parallel computing.

III. METHODOLOGY

Implementation of the model based path planning involves the formulation of a proper scenario. For this, a scenario involving a self-driving bot has been implemented. The state of the system has been derived by use of an overhead camera. The image obtained from the camera is divided into multiple grids, whose properties are binary coded to define the state of the system. Such a formulation is much easier to approximate in the form of a grid-world like scenario. The simulator is then used for learning the fundamental dynamics of the system using the Q-learning based reinforcement learning.

A. Scenario Environment

The environment consists of 3 main components:

- **Agent:** The bot Firebird V 2560 has been used as the agent for learning the dynamics of the scenario. To ease the detection of the bot location along with its direction using the overhead camera, a location detector containing

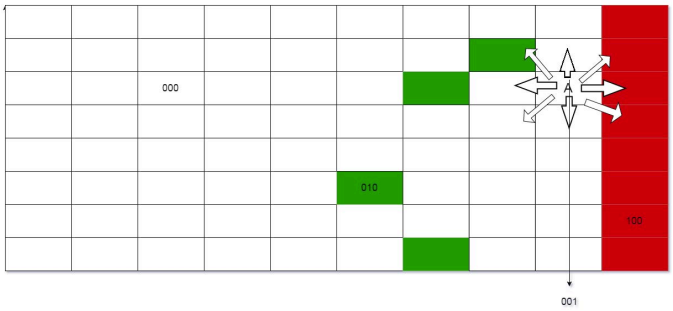

Fig. 1. Layout of a Grid-World

two colors (blue and red) was placed over the bot. The bot makes use of Atmel ATMEGA2560 and Atmel ATMEGA8 as master and slave micro-controller respectively. The bot has two DC geared motors in differential drive configuration with an RPM of 75. Motion control of bot includes both velocity and direction control achieved using PWM (Pulse Width Modulation) and L293D dual motor driver respectively.

- **Goal:** The goal of the scenario is defined by an end line of the driving arena.
- **Obstacles:** While driving towards the goal the agent has to avoid obstacles that comes in its way. These obstacles were green colored to be able to get detected in the image obtained using the overhead camera.

The objective of the agent is to learn to reach the end line (goal) of the arena while avoiding the obstacles in its path such that the reward received by its movements gets maximized. For this, the agent has to properly learn the dynamics of the system. For implementing the objective, a proper reward was defined for each step taken up by the agent. For each action, a small negative reward (penalty) has been defined. Further, on reaching the goal a highly positive reward is given to the agent. On colliding the obstacle or crossing the boundary of the arena a highly negative reward is given to the agent. All these together form the reward function of the environment. For defining the state of the system, all the grids were binary coded using 3 bits representing the condition of the grid. The individual bits were used for representing the presence of agent, goal or obstacle in the grid. Also this eased the approximation of the model to form a grid-world like scenario.

B. Simulator Design

The scenario environment was modeled in the form of a grid-world like simulator. For this, the image from the overhead camera was divided into 80 grids as shown in fig 1 with each grid being defined in the form of 3 binary bits indicating the presence of a particular object (agent, goal, obstacle) in it. Bit 1 is reserved for agent, bit 2 for goal and bit 3 for obstacle. For instance, 000 indicates that none of the objects is present in the grid. The simulator thus formed was used for training using the Q-leaning based reinforcement learning. As shown in fig. 2 for approximating the Q-value function, a

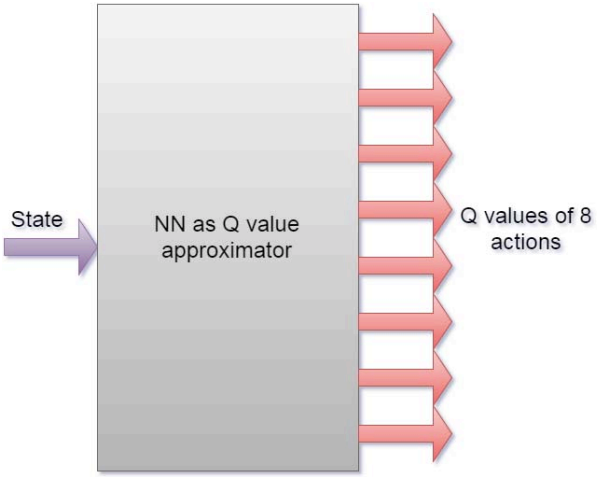

Fig. 2. Neural Network as a Function Approximator

3-layer neural network has been used. The architecture of the neural network is defined below.

C. Neural Network Architecture

The Network used consisted of 2 hidden layers of size 164 and 150 neurons. The output layer has eight neurons representing Q-values of individual actions. The number of inputs of the network was same as the number of features representing the state of the system i.e. 240. After each hidden layer, a 'Rectified Linear unit or relu' activation function was used to normalize outputs of the individual neuron. After the output layer the activation used was 'Linear function'. Such a strategy for defining the output of the system was first suggested by Mnih et al. [25]. To reduce the amount of over-fitting a newly proposed technique called dropout [26] has been used with a dropout probability of 0.2. The neural network has been trained using the 'RMSProp' [27] algorithm by minimizing the mean square error of the data.

D. Neural Network Training

Within the arena, the bot is allowed to take eight different actions defined in the form of the angle of movement. These movements include both straightforward as well as diagonal movements. For the four diagonal movements the negative reward given to it 1.41 times that given while it moves in the straight-forward direction. The reward given to the agent is -1 and -1.41 for straight-forward and diagonal movements respectively. The reward for reaching the goal is set to be +25, and that for colliding an obstacle is -25. Further on crossing the boundary a negative reward of -5 is given to the agent. These values are decided based on the geometry of the training environment and agent. The straight-movement reward (-1) is used as the reference reward based on which rewards for other cases have been formulated. As the training area is divided into square grids, the distance covered during diagonal movement is 1.41 ($\sqrt{2}$) times that covered during straight-movement. Thus, the reward corresponding to the diagonal movement is taken to be -1.41. The reward corresponding to the goal is

decided on the basis of size of the training arena and the movement rewards. The large negative pit rewards favour the learning process of the agent due to which it learns to avoid the obstacle/pit.

$$reward = \begin{cases} -1 & in\ case\ of\ forward\ movement \\ -1.41 & in\ case\ of\ diagonal\ movement \\ -25 & in\ case\ of\ obstacle \\ +25 & in\ case\ of\ goal \end{cases} \quad (6)$$

The formulated model simulator is first used to train the neural network. For this different environmental scenario were used with varying obstacle and agent position. For this ϵ greedy based Q-learning approach has been used. Further to make the agent learn the real world dynamics of the system the training was transferred to the real world by using the bot as the very agent to interact with the given environment. The algorithm for implementing Q Learning with neural network has been explained in algorithm 1.

Algorithm 1 Q Learning with neural network

```
procedure OPTIMIZE Q FUNCTION
    for number of epochs do
        while Game in Progress do
            Generate the Q value from the neural network.
            Find action a using ϵ greedy approach.
            Take the action a and move to new state s'
            Find reward r_{t+1}
            Find Q value for (s,a,s')
            Update buffer and generate batch using buffer.
            Apply batch based Q-value iteration to train the
neural network.
```

IV. Results

For training using the grid-world based simulator different scenario's with different kinds of constraints were first tested. The Table II shows the effect on testing accuracy if the location of the obstacles and agent were fixed along with the number of obstacles in the arena. The table clearly shows that the accuracy got easily improved in such a scenario. Even when the number of obstacles used were 8 the accuracy easily got to 100% by the time the number of epochs of training reached 7500. In further tests, as shown in Table III the agent location was made random in the arena. This made the task of training a bit difficult but still the accuracy reached the 100% mark easily within the 20000 epochs for all the scenarios.

TABLE II
Simulator Training for Fixed Obstacle and Fixed Agent Scenario

Obstacles	2	4	6	8
Epochs				
1000	88.5	84.6	84.5	82.1
2000	98.7	97.6	92.7	89.6
5000	100.0	100.0	98.6	94.6
7500	100.0	100.0	100.0	100.0
10000	100.0	100.0	100.0	100.0

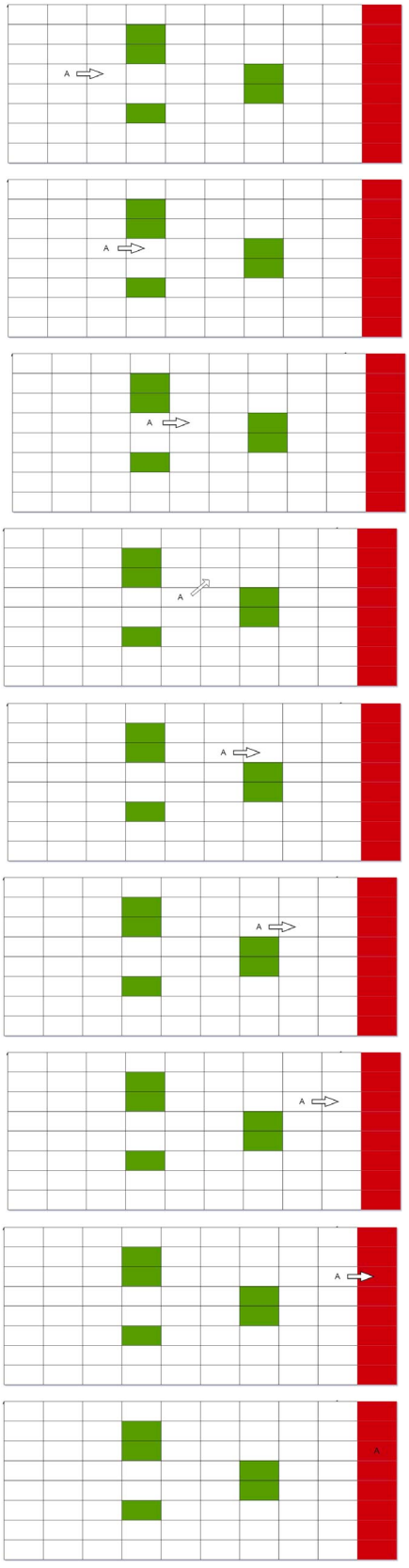

Fig. 3. Robotic motion in the arena

TABLE III
SIMULATOR TRAINING FOR FIXED OBSTACLE AND DYNAMIC AGENT SCENARIO

Obstacles	2	4	6	8
Epochs				
1000	78.5	75.5	68.8	68
2000	93.5	94.4	88.9	80.6
5000	98.8	88.6	92.5	89.6
7500	100	98.4	96.6	95.7
10000	100	100	100	98.5
20000	100	100	100	100

For further increasing the difficulty of the scenario the obstacle location and player location were both made random (dynamic) as in Table IV. As evident from the Table IV this made the training a lot more difficult as compared to the previously tested cases as now for an accuracy of 90% around 70000-80000 epochs are required. The simulator training required for the real world application needs to have the arena to have random number of obstacles within the arena along with them to have dynamic location. This scenario has been tested in the last column of the Table IV. The results clearly show it to be the hardest scenario to be trained. The Fig. 4 shows the non linear increase in the test accuracy which gets saturated after initial increase in the training epochs. The saturation overhead bot control accuracy is near 95%. But this accuracy is achieved after 100000 epochs of simulator training. This type of training is not possible in the real world environmental conditions as each epoch in this scenario is quite expensive. Fig. 3 shows the robotic motion in the arena. It clearly shows that post-training, the robot is able to take optimal action and can traverse the arena to reach the goal.

TABLE IV
SIMULATOR TRAINING FOR DYNAMIC OBSTACLE AND DYNAMIC AGENT SCENARIO

Obstacles	2	4	6	8	Random
Epochs					
5000	58.5	45.8	43.6	34.5	28.8
10000	72.2	61.1	56.7	52.6	51.7
20000	82.1	74.5	66.9	62.5	62.9
30000	88.3	80.6	75.7	70.3	68.2
40000	92.2	85.5	81.2	75.6	76.7
50000	95.5	89.4	84.7	80.1	81.2
60000	93.8	93.5	88.3	84.2	85.5
70000	96.6	92.2	91.6	87.2	87.2
80000	97.8	93.5	93.6	90.1	90.5
90000	95.2	94.1	93.1	92.5	93.1
100000	96.8	95.1	94.8	94.1	94.8

The neural network weights derived from the simulator based training were used for further training in the real world scenario to enable the agent to learn the complex real world dynamics.

V. CONCLUSION

For path planning in a real world scenario, model based methods provide more efficient training methods without the requirement of inefficiently large number of agent-environment interactions. The ϵ greedy Q-learning approach can prove to be quite handy for training the agent once the model for the scenario has been developed. The model of the agent-environment interaction was simulated in the form of a grid-world based simulator. The samples from this simulator were then used to find an optimal policy by using the ϵ greedy Q-learning approach. The method provides the agent with capability to learn from its experience once the initial simulator based training has been done. The real-world training done in the aftermath of the simulator based training helped the agent to learn the non-linear dynamics of its interaction with the environment. The results manifests that the ϵ greedy Q-learning approach is quite effective in case of model based scenarios as even in the totally random scenario the agent was able to generate a success rate of almost 95%. The proposal proved out to be quite efficient in generating an optimal policy for the above mentioned scenario but is limited by the fact that the state of the system is generated by an overhead camera. This constraints the system working to a pre-built arena. In future a scenario with on-board camera can be modelled to make the system flexible to work in any environment.

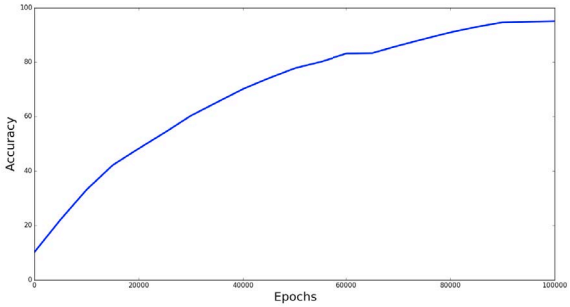

Fig. 4. Accuracy vs Epochs for Simulator based Training

REFERENCES

[1] Kober, J., Bagnell, J.A. and Peters, J., 2013. Reinforcement learning in robotics: A survey. The International Journal of Robotics Research, p.0278364913495721.

[2] Watkins, C.J. and Dayan, P., 1992. Q-learning. Machine learning, 8(3-4), pp.279-292.

[3] Brafman, R.I. and Tennenholtz, M., 2003. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. The Journal of Machine Learning Research, 3, pp.213-231.

[4] Kearns, M. and Singh, S., 2002. Near-optimal reinforcement learning in polynomial time. Machine Learning, 49(2-3), pp.209-232.

[5] Strehl, A.L. and Littman, M.L., 2005, August. A theoretical analysis of model-based interval estimation. In Proceedings of the 22nd international conference on Machine learning (pp. 856-863). ACM.

[6] Abbeel, P., Coates, A., Quigley, M. and Ng, A.Y., 2007. An application of reinforcement learning to aerobatic helicopter flight. Advances in neural information processing systems, 19, p.1.

[7] Kwakernaak, H. and Sivan, R., 1972. Linear optimal control systems (Vol. 1, p. 608). New York: Wiley-interscience.

[8] Masuta, H. and Kubota, N., 2010, October. Perceptual system using spiking neural network for an intelligent robot. In Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on (pp. 3405-3412). IEEE.

[9] Gu, D., Hu, H., Reynolds, J. and Tsang, E., 2003, July. GA-based learning in behaviour based robotics. In Computational Intelligence in Robotics and Automation, 2003. Proceedings. 2003 IEEE International Symposium on (Vol. 3, pp. 1521-1526). IEEE.
[10] Hachour, O., 2009. The proposed Fuzzy Logic Navigation approach of Autonomous Mobile robots in unknown environments. International journal of mathematical models and methods in applied sciences, 3(3), pp.204-218.
[11] Sato, E., Yamaguchi, T. and Harashima, F., 2007. Natural interface using pointing behavior for humanrobot gestural interaction. Industrial Electronics, IEEE Transactions on, 54(2), pp.1105-1112.
[12] Dusko Katic, Miomir Vukobratovic,Genetic Algorithms in Robotics, International Series on Microprocessor-Based and Intelligent Systems Engineering Volume 25, 2003.
[13] Jevti, A. and Andina de la Fuente, D., 2007. Swarm intelligence and its applications in swarm robotics.
[14] Beni, G., 2004. From swarm intelligence to swarm robotics. In Swarm robotics (pp. 1-9). Springer Berlin Heidelberg. Vancouver
[15] Wang, Z., 2003. Car Simulation Using Reinforcement Learning. University of British Colombia, Vancouver, Canad.
[16] Levine, S., Finn, C., Darrell, T. and Abbeel, P., 2015. End-to-end training of deep visuomotor policies. arXiv preprint arXiv:1504.00702.
[17] Deisenroth, M.P., Neumann, G. and Peters, J., 2013. A Survey on Policy Search for Robotics. Foundations and Trends in Robotics, 2(1-2), pp.1-142.
[18] Daniel, C., Neumann, G. and Peters, J.R., 2012. Hierarchical relative entropy policy search. In International Conference on Artificial Intelligence and Statistics (pp. 273-281).
[19] Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8(3-4), pp.229-256.
[20] Gullapalli, V., 1990. A stochastic reinforcement learning algorithm for learning real-valued functions. Neural networks, 3(6), pp.671-692.
[21] Kober, J., Mlling, K., Krmer, O., Lampert, C.H., Schlkopf, B. and Peters, J., 2010, May. Movement templates for learning of hitting and batting. In Robotics and Automation (ICRA), 2010 IEEE International Conference on (pp. 853-858). IEEE.
[22] Kober, J., Oztop, E. and Peters, J., 2011, July. Reinforcement learning to adjust robot movements to new situations. In IJCAI Proceedings-International Joint Conference on Artificial Intelligence (Vol. 22, No. 3, p. 2650).
[23] Sutton, R.S. and Barto, A.G., 1998. Reinforcement learning: An introduction. MIT press.
[24] Bellman R., A Markovian decision process Journal of Mathematics and Mechanics, 6 (1957), pp. 679684.
[25] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.
[26] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), pp.1929-1958.
[27] Hinton, G., Srivastava, N., and Swersky, K., 2012. Overview of mini-batch gradient descent. In: Neural Networks for Machine Learning.