A Robot Exploration Strategy Based on Q-learning Network

Tai Lei
Department of Mechanical and Biomedical Engineering
City University of Hong Kong
Email: lei.tai@my.cityu.edu.hk

Liu Ming
Department of Mechanical and Biomedical Engineering
City University of Hong Kong
Email: mingliu@cityu.edu.hk

***Abstract*—This paper introduces a reinforcement learning method for exploring a corridor environment with the depth information from an RGB-D sensor only. The robot controller achieves obstacle avoidance ability by pre-training of feature maps using the depth information. The system is based on the recent Deep Q-Network (DQN) framework where a convolution neural network structure was adopted in the Q-value estimation of the Q-learning method. We separate the DQN into a supervised deep learning structure and a Q-learning network. The experiments of a Turtlebot in the Gazebo simulation environment show the robustness to different kinds of corridor environments. All of the experiments use the same pre-training deep learning structure. Note that the robot is traveling in environments which are different from the pre-training environment. It is the first time that raw sensor information is used to build such an exploring strategy for robotics by reinforcement learning.**

I. Introduction

Mobile robot exploration of an unknown environment is a quite common problem for robot applications, like rescue, mining etc. Normally, with information from vision or depth sensors, robot requires complicated logic about the obstacles and topological mapping of environments [1] [2] designed by human-beings. However, there is no high-level human-brain-like intelligence in these traditional approaches. Recently, machine learning has attracted more and more attentions. In this paper, we want to develop a machine learning method for robots to explore an unknown environment using raw sensor inputs.

Regarding the requirements mentioned above, Deep Reinforcement Learning, merging reinforcement learning and deep learning, is a proper method to apply in this scenario. For example, Google DeepMind implemented a Deep Q-Network (DQN) [3] on 49 Atari-2600 games. This method outperformed almost all of other state-of-the-art reinforcement learning methods and 75% human players, without any prior knowledge about the Atari 2600 games. It showed great potential to apply this algorithm in other related fields including robotic exploration.

Not like the DQN mentioned above, we apply this learning approach in two steps in the robotics exploration for an unknown environment. Firstly, we build a supervised learning model taking the depth information as input and the command as output. The datum is manually labeled with control commands to tune the moving directions of the robot. This supervised learning model is implemented with a Convolution Neural Network [4]. Secondly, a neural network structure with three fully-connected hidden layers was used to mimic the reinforcement learning procedure taking the feature maps as input. The feature maps are the output of the last second layer of the supervised learning model trained before. This reinforcement learning framework is defined as a Q-Network. In this paper, we will mainly introduce the second step. Particularly, we stress the following contributions:

- We design a revised version of DQN network for a moving robot to explore an unknown environment. The project is implemented in Gazebo and ROS-based interfaces. Feature learning is based on Caffe [5], a popular tool-kit for deep learning.
- The model is validating in several simulated environments. We also discuss the future work, such as adding noise to verify the robustness of the system and apply it in real environments.

II. Related Work

A. Reinforcement Learning in Robotics

Reinforcement Learning (RL) [6] is an efficient way for robotics to acquire data and to learn skills. With an appropriate and abstract reward, the robot can learn a complex strategy without ground truth as references. It was just applied on mastering the strategy of GO (an ancient Chinese board game which was regarded as the most challenging task for artificial intelligence) [7]. It indicates the great feasibility of reinforcement learning in other fields. RL was also applied on an autonomous helicopter flight [8] and autonomous inverted helicopter flight [9], by collecting the flight data and learning a non-linear model of the aerodynamics.

Reinforcement learning was also proved to improve the motion behaviour of a humanoid robot to react for visually identified objects substantially [10], by building an autonomous strategy with little prior knowledge. In this application, The robot showed a continuously evolved performance with time.

Most of reinforcement learning methods for robotics are based on state information. To our knowledge, raw image sensor information has never been considered directly.

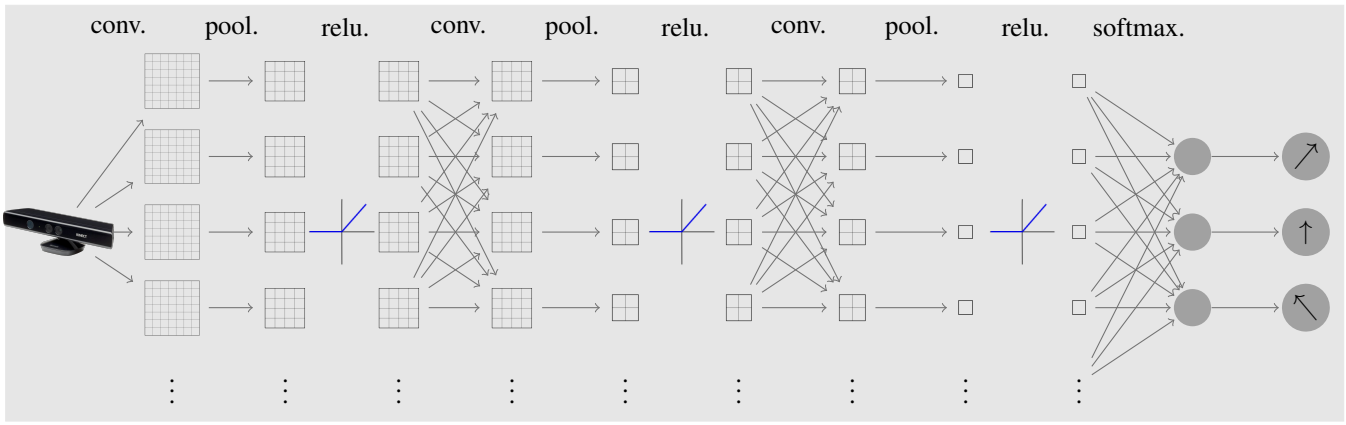


Fig. 1. Structure of the CNN layers. Depth images after down sampling will be fed into to the model. Three Convolution layers with pooling and rectifier layers after are connected together. After that, feature maps of every input will be fully connected and fed to the softmax layer of the classier.

B. CNN in Perception

Convolution Neural Network (CNN) is a classic visual learning method. With the development of large-scale computing and GPU accelerating, huge CNN frameworks can be set with tens of convolution layers.

Normally, CNN was used to solve a classification problem with a softmax layer, such as imagenet classification [11] [12] and face recognition [4]. With a regression layer to optimal the euclidean loss, the feature maps extracted by CNN can also be applied to key points searching problem [13] [14]. Computer vision based recognition methods are mainly feature detection and extraction [15] [16] [17], while CNN extract this feature model by self-learning.

In terms of robotics, CNN was also used to perceive environment information for visual navigation [18]. However, a supervised-learning-based method requires a complicated and time-consuming training period and the trained model cannot be applied in a different environment directly.

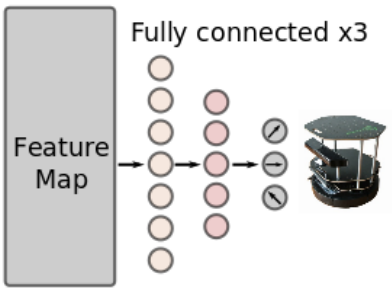


Fig. 2. Feature map extracted from the supervised learning model is the input and is reshaped to a one dimension vector. After three fully-connected hidden layers of a neural network, it will be transformed to the three commands for moving direction as the outputs

Algorithm 1 Q-network algorithm

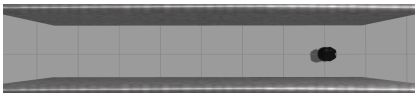
Initialize action-value Q-network with random weights θ
Initialize the memory D to store experience replay
Set the distance threshold $l_s = 0.6$m
for episode $= 1, M$ **do**
  Set the Turtlebot to the start position.
  Get the minimum intensity of depth image as l_{min}
  while $l_{min} > l_s$ **do**
    Capture the real time feature map x_t
    With probability ε select a random action a_t
    Otherwise select $a_t = \text{argmax}_a Q(x_t, a; \theta)$
    Move along the selected direction a_t
    Update l_{min} with new depth information
    if $l_{min} < l_s$ **then**
      $r_t = -50$
      $x_{t+1} = Null$
    else
      $r_t = 1$
      Capture the new feature map x_{t+1}
    end if
    Store the transition (x_t, a_t, r_t, x_{t+1}) in D
    Select a batch of transitions (x_k, a_k, r_k, x_{k+1}) randomly from D
    if $r_k = -50$ **then**
      $y_k = r_k$
    else
      $y_k = r_k + \gamma \max_{a'} Q(x_{k+1}, a'; \theta)$
    end if
    Update θ through a gradient descent procedure on the batch of $(y_k - Q(\phi_k, a_k; \theta))^2$
  end while
end for

III. Implementation Details

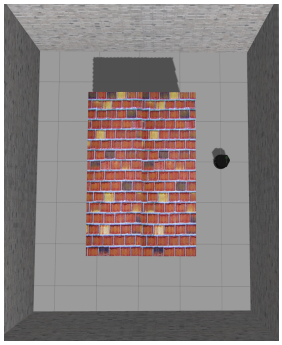
Travelling in an unknown environment with obstacle avoidance ability is the main target of this paper. This task is defined as controlling a ground-moving robot in an environment without any collisions with the obstacles. The input information here is the feature maps extracted by the CNN supervised learning model which was trained in our prior

work. Supervised learning model can be used to perceive an environment [18], so the feature maps can be regarded as abstracted information for the environment to some extends. Robot will perform obstacle avoidance by Q-network learning of itself in other environments which are a little different from the training environment. The implementation of the experiment includes three parts:

- a simulated 3D environment in Gazebo for a robot to explore.
- a Q-network reinforcement learning framework.
- a simulated Turtlebot with a Kinect sensor in Gazebo controlled by the Q-network outputs.



(a) Straight Corridor



(b) circular Corridor

Fig. 3. Simulation Environment

TABLE I
SETTING OF REWARD

State	Reward Value
collision or stop	-50
keep-moving	1

A. DQN

DQN defined the tasks between the agents and the environments [19] in Atari 2600 games. The environment was set as ε. At each step, the agent selected an action a_t from the action sets of the game and observed a displayed image x_t from the current screen. The change of the game score r_t was regarded as the reward for the action. For a standard reinforcement learning method, we can complete all of these game sequences s_t as Markov decision process directly, where $s_t = x_1, a_1, x_2, ..., a_{t-1}, x_t$. Defining the discounted reward for the future by a factor γ, the sum of the future reward until the end would be $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$. T means the termination time-step of the game. The target was to maximize the action-value function $Q^*(s,a) = \max_\pi \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$, where π is the strategy for choosing of best action. From the Bellman equation, it is equal to maximize the expected value of $r + \gamma Q^*(s', a')$, if the optimal value $Q^*(s', a')$ of the sequence at the next time step is known.

$$Q^*(s', a') = \mathbb{E}_{s' \sim \varepsilon}[r + \gamma \max_{a'} Q^*(s', a') | s, a]$$

Not using iterative updating method to optimal the equation, it is common to estimate the equation by using a function approximator. Q-network in DQN was such a neural network function approximator with weights θ and $Q(s, a, \theta) \approx Q^*(s, a)$. The loss function to train the Q-network is:

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)}[(y_i - Q(s, a; \theta_i))^2]$$

y_i is the target, which is calculated by the previous iteration result θ_{i-1}. $\rho(s, a)$ is the probability distribution of sequences s and a. The gradient of the loss function is shown below:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \varepsilon}[(y_i - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i)]$$

TABLE II
TRAINING PARAMETERS AND THEIR VALUE

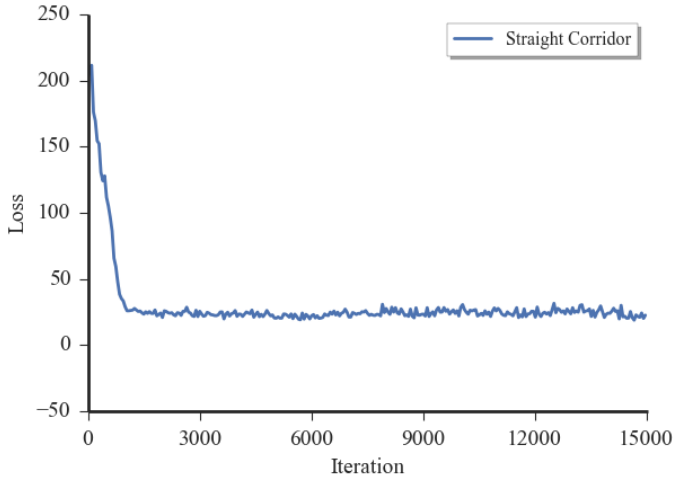
Parameter	Value
batch size	32
replay memory size	5000
discount factor	0.85
learning rate	0.000001
gradient momentum	0.9
max iteration	15000
step size	10000
gamma	0.1

B. Q-network-based Learning System

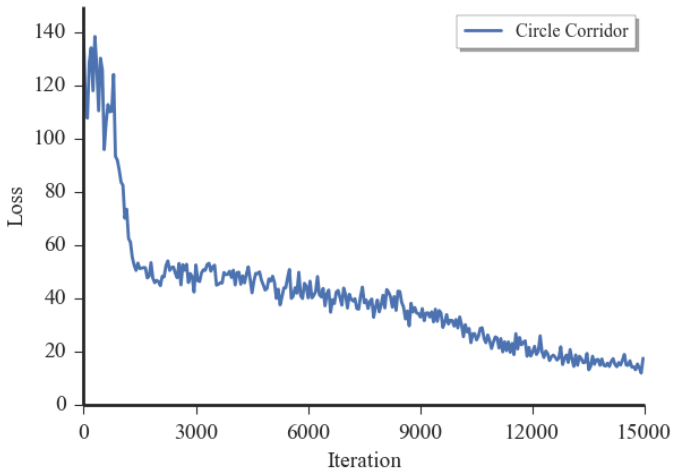
To accomplish the task of exploration, we simplify the DQN [3] into two steps. At first, a 3-layer Convolution Neural Network framework is built to do the pre-processing procedure. Fig. 1 shows the CNN structure in detail. By three times convolution, pooling, and rectifier activation, the feature maps of the inputs are extracted. A softmax layer is implemented to get the output distribution of the moving commands. When training the network, we control the robot to explore in an environment and label the depth image from Kinect sensor. The related control command from a human being was labelled as the ground-truth. The parameters, training and analysis of this CNN model will not be introduced here.

In the second step, the reinforcement learning part, we use the trained model mentioned above to forward every input image of real time and get the feature map of the depth image. Feature map is the output of the last ReLU layer in Fig. 1. Note that we are not using the output command from the first step directly. The feature map consists of $64 \times 20 \times 15$ matrices. Fig. 2 shows the structure of the neural network to estimate the Q-value of Q-network. In the exploration period, the pre-processed depth images from Kinect will be memorized with the related action, reward and the pre-processed depth image captured after executing the action as a transition. At the same time, we randomly choose a batch of transitions to update the weights of Q-network by gradient descent.

The reward function for the Q-network system has two different feedback values, one for normal moving and one for



(a) Straight Corridor Training Loss



(b) circular Corridor Training Loss

Fig. 4. The converging curves of batch loss in iteration procedure

the collision with the obstacles. The system will stare at the Turtlebot state by checking the minimum depth between the Turtlebot and the obstacles from the Kinect. Table I shows the declaration of the reward setting. Here we set the threshold to be 0.6 meter, keeping enough space between the robot and the obstacle. When the depth is lower than the threshold or the robot position does not change for a period of time, we set the robot state to a termination and robot is reset to the start position. The reward for a termination is -50. At that time, we think there is a collision between the robot and the obstacle. Otherwise, if the robot keeps moving, the reward is 1. In the exploration procedure, the only target is to achieve obstacle avoidance, so the feedback of collision must be much larger than the normal moving. When to update the weights, if the reward is -50, that means the robot collide with an obstacle, and the target value for the state and action in this transition will also be -50. On the other hand, the target will be calculated by Bellman Equation if the robot state in this transition is keep-moving. Algorithm 1 presents the whole framework of the Q-network training procedure. In every episode of exploration, randomly choosing the command will increase the variety of the training batch. The randomness will less and less with the decreasing of ϵ. Every time, after the execution of the chosen moving command, the new feature map will be captured with rewarding 1, or the robot collide with the obstacle with rewarding -50. After storing the transition, the weights of Q-network will be update by the batch of transitions which are chosen randomly as well.

C. Environment Design in Gazebo

Gazebo is used to build a simulation environment in this project. The robot is a Turtlebot with two differential wheels and a Kinect sensor. The whole project is implemented on Robot Operation System(ROS). Q-network system tracks the depth information from Turtlebot in Gazebo simulated environment. The command related to the highest Q-value will be transformed to the angel velocity for the Turtlebot by ROS topic.

IV. Experiments and Result

To evaluate the Q-network learning system, two different kinds of environments are designed as shown in Fig. 3. The first one consists of a direct straight corridor. The other one consists of a circular connected corridor with more complicated depth information. The table II shows the training parameters and their values in gradient descent procedure of the Q-network learning implemented by Caffe. The step size means that the learning rate will multiply gamma after first 10000 iterations, which means the learning rate of last 5000 iterations is 0.0000001.

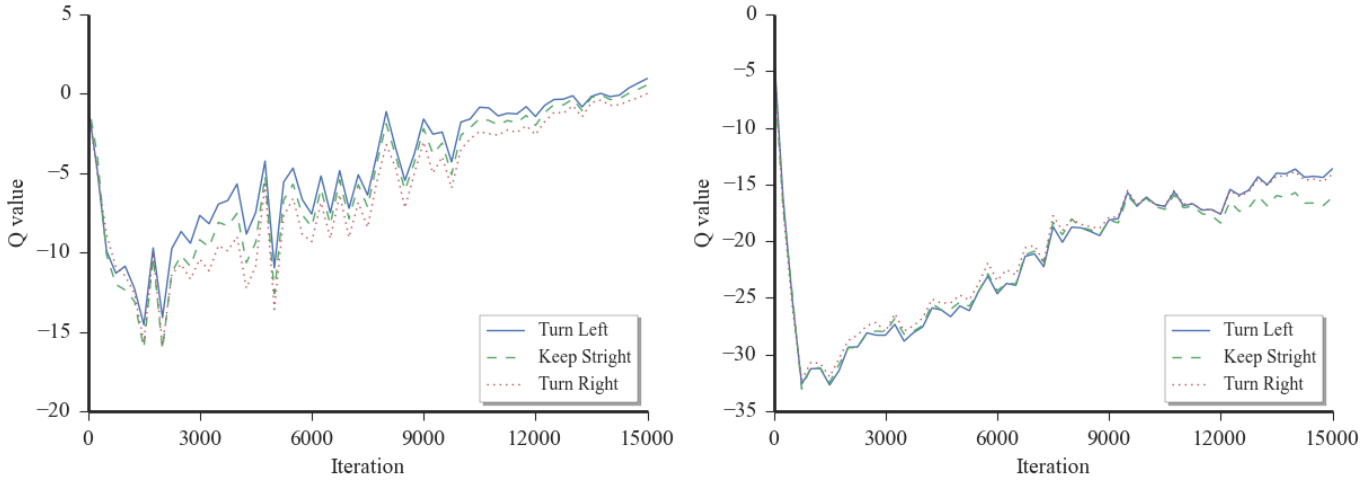
A. Training Result

The Q-network-based learning system experienced 15000 iterations in both of the two simulated environments.

Fig. 4 shows the loss converging curves in the whole learning period. In every iteration step, the euclidean loss between target Q-value and the predicted Q-value is calculated. The loss in Fig. 4 is the average value of the whole training batch in related iteration step. Because we choose the batch randomly in every step, the different training batch sets between the successive gradient descent steps lead the apparent fluctuation in Fig. 4 in both of the environments. It also shows that the loss of both of the environments decreased rapidly in first 1500 iteration steps. And after that, the loss in the straight corridor is stable. But the convergence of the loss in the circular corridor environment is still decreasing apparently. That should be caused by the complexity of the depth information which need more time to train the model.

B. Test Result

In the training process, the weights of the Q-network were saved regularly in every 300 steps of iteration. For both environments, we randomly choose several pre-processed depth



(a) Q-value test for Straight Corridor

(b) Q-value test for Circular Corridor

Fig. 5. Q-value of the evaluation sets calculated by the model at different iteration stages

information in different states as the test set, which correspond to different positions of the Turtlebot in the simulated world. Fig. 5 shows the average Q-value of the test set for 3 different targets commands by using the trained weights of every 300 steps of iteration.

We can observe that all of the 3 target values converge towards a stable state with a certain value. The convergences of Q-value proved the stability of the Q-network system. Along with the shrinking of the learning rate after 10000 steps of iteration, the fluctuation of the value is also reduced, especially for the straight corridor environment. The increment of the Q-value in the training procedure indicates the Q-network system is much more reliable with longer training time.

With the reward mentioned above, a direct feedback is the final scores the robot can achieve, with the trained strategy to choose the command related to the highest Q-value. The higher scores also mean that the Turtlebot can keep moving in the environment for a longer time and avoid more obstacles. But there is no apparent relation between the scores and moving distance, because the Turtlebot may move along a tortuous path. The width of the road in the simulated world is narrow enough that the robot cannot keep turning along the same direction rewarding positive infinite scores.

The trained model in every 1000 steps of iteration is tested 5 times in the related environment and the result is shown in Fig. 6. It shows that the test scores of both environments increase at the first 2000 iterations. Scores tested in straight corridor world keep consistent near -20, which means the robot moves 30 steps, with terminated reward -50. It is enough to arrive at the end of the straight corridor. The other one in the circular corridor environment is increased discontinuously. The highest test value in circular corridor environment is 80, which means that the robot moves 130 steps, with terminated reward -50. The Turtlebot should have finished a lap in that situation. We can imagine that the robot should keep moving forever along the circular corridor environment with infinite scores as a perfect model. But in the test, the robot will finally collide with obstacles every time. It is possible that the model is still not robust enough.

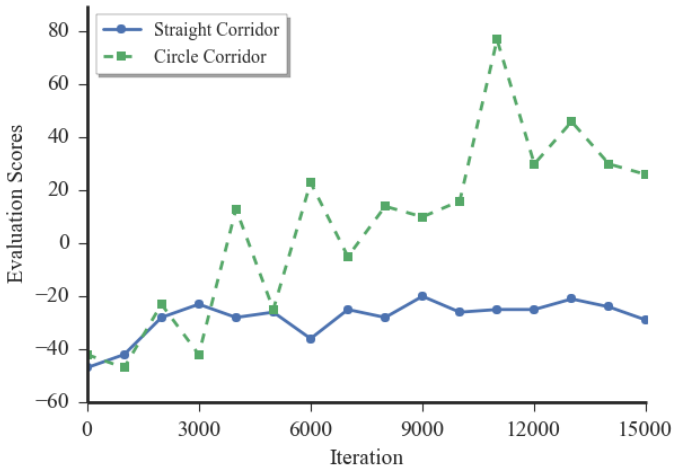


Fig. 6. Average score of 5 times evaluation by using the trained model of every 1000 iteration steps

V. Conclusion

The Q-network is reliable to train a moving robot achieving obstacle avoidance ability in the simulation environment. In this project, we only use a pre-processed CNN model to extract the feature map in a certain environment, and another 3-layer fully connected neural network model is used to train the real-time learning process in other more complicated environments. The test results show that the Turtlebot achieved obstacle avoidance ability and it can travel freely in the simulated environment with the strategy learnt by itself.

In the future, we will envision the implementation of the whole DQN framework in real-time exploration directly. That means there would be totally no pre-processing for the feature map. In real-time learning, the raw image of depth will be fed into the model to predict the Q-values. With GPU accelerating, this should be also feasible.

The reward function should be redesigned as well, because now every non-collision state is rewarded with the same score. To make the model more stable, we should add more man-made and random noise in the input information. However, there is only random Gaussian filter in the Caffe framework now. Finally, we will also test this model in real world scenario in the future.

ACKNOWLEDGMENT

This work is supported by the Research Grant Council of Hong Kong SAR Government, China, under project No. 16206014 and No. 16212815; National Natural Science Foundation of China No. 6140021318, awarded to Prof. Ming Liu.

REFERENCES

[1] M. Liu, F. Colas, L. Oth, and R. Siegwart, "Incremental topological segmentation for semi-structured environments using discretized gvg," *International Journal of Autonomous Robots*, 2014.

[2] M. Liu, F. Colas, F. Pomerleau, and R. Siegwart, "A markov semi-supervised clustering approach and its application in topological map extraction," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 4743–4748.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[4] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *Neural Networks, IEEE Transactions on*, vol. 8, no. 1, pp. 98–113, 1997.

[5] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[6] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[8] H. Kim, M. I. Jordan, S. Sastry, and A. Y. Ng, "Autonomous helicopter flight via reinforcement learning," in *Advances in neural information processing systems*, 2003.

[9] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *Experimental Robotics IX*. Springer, 2006, pp. 363–372.

[10] L. Jamone, L. Natale, F. Nori, G. Metta, and G. Sandini, "Autonomous online learning of reaching behavior in a humanoid robot," *International Journal of Humanoid Robotics*, vol. 9, no. 03, p. 1250017, 2012.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[12] Y. Z. M. L. Shaohua Li, Huimin Huang, "A fast multi-scale convolutional neural network for object recognition," in *Real-time Computing and Robotics (RCAR), 2015 IEEE International Conference on*. IEEE, 2015.

[13] Y. Sun, X. Wang, and X. Tang, "Deep convolutional network cascade for facial point detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 3476–3483.

[14] H. Chen, P. Wang, and M. Liu, "From co-saliency detection to object co-segmentation: a unified multi-stage low-rank matrix recovery approach," in *Robotics and Biomimetics (ROBIO), 2015 IEEE International Conference on*. IEEE, 2015.

[15] M. Liu, D. Scaramuzza, C. Pradalier, R. Siegwart, and Q. Chen, "Scene recognition with omnidirectional vision for topological map using lightweight adaptive descriptors," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 2009, pp. 116–121.

[16] M. Liu and R. Siegwart, "Topological mapping and scene recognition with lightweight color descriptors for an omnidirectional camera," *Robotics, IEEE Transactions on*, vol. 30, no. 2, pp. 310–324, 2014.

[17] M. Liu, B. T. Alper, and R. Siegwart, "An adaptive descriptor for uncalibrated omnidirectional images - towards scene reconstruction by trifocal tensor," in *IEEE International Conference on Robotics and Automation, 2013*, 2013.

[18] A. Giusti, J. Guzzi, D. Ciresan, F.-L. He, J. P. Rodriguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots," *Robotics and Automation Letters, IEEE*, vol. PP, p. 1, 2015.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.