COMP4801 Final Year Project
Final Report

SimNav: Simulation Training Framework for Collision Avoidance in Robot Navigation

Anson Wong
Department of Computer Science
The University Of Hong Kong

UID: 3035098264

Supervisors

Prof. Wenping Wang
Dr. Y.K. Choi

15 April 2018

Abstract

The development of robotics has been rapid recently in different areas, for instance, industrial, social and medical. Robot navigation is a fundamental part among robotic applications in all these domains. The task of navigating from one point to another with collision-free trajectories is referred as collision avoidance. Conventional methods exploited geometric rules or required excessive amount of real-world data. Without the need of expensive real-world data, this project aims to show that a robot can be trained with synthetic data only in simulation environment and navigate safely. Deliverable includes a simulation environment learning framework, two high-quality simulation environments and two trained navigation policies, which serve as a decision maker to navigate the robot and avoid collision.

Acknowledgments

I would like to express my special thanks of gratitude to Prof. Wenping Wang and Dr. Loretta Choi, who gave me the opportunity to work on this project, with your patience and professional advice.

Contents

List of Figures

Chapter 1

Introduction

Applications of robotics have been increasing in different fields. Among there, robot navigation is one of the most important capability. This project focuses on collision avoidance in robot navigation.

1.1 Outline of the Report

This report is structured as following. Chapter 1 will provide an introduction on the topic. Chapter 2 will discuss the methodology. Chapter 3 will cover the experiments and result, where two different trained network is compared in two different simulation environments. Finally Chapter 4 will investigate potential future work, followed by a conclusion.

1.2 Background

Safe navigation in environments with obstacles is fundamental for mobile robots to perform various tasks. Conventional approaches generally search for optimal control to avoid collision based on the geometry or topological mapping of the environment. Environments were perceived as a geometrical world and decisions were only made with preliminary features detected. Robots often follow specific rules and thus it would be hard to adapt to a new environment that would require strenuous effort for different settings.

With the advance of machine learning, people begin to adapt machine learning techniques on robotic problems. Additionally, simulation techniques have been improved along with computer hardware upgrades, enabling computers to simulate and render authentic graphics.

One of the biggest constraints in robotics is hardware. It can be dangerous if a robot performs a task poorly in real world, causing collision and even more serious consequences. Collision avoidance, in particular, requires the robot to explore and navigate in an environment full of obstacles with collision-free trajectories. As a result, safety is one of the biggest concerns in collision avoidance.

1.3 Previous Work

Works in the past in collision avoidance focused mainly on the safety issue. Recently people shifted the focus to social-friendliness, emphasizing the need that the robot should not only avoid collision, but also imitate the way human avoid colliding each other.

Potential-field based

[2] mapped sensor reading from robot into a histogram grid. It then selected the sectors with obstacle density low enough for safe passage and with direction best matching the objective's. [7] made use of the concept of potential field from physics to represent the admissible velocities.

Dynamic based

[6] selected an optimal solution in the search space that is restricted to safe circular trajectories that can be reached within a short time interval and are free from collisions. [5, 21, 15, 22] focused on computing the set of collision-free velocities between all entities, and choose the one closest to the original preferred velocity. This is particularly suitable in multi-agent simulation when all obstacles information are fully observable.

Learning-based

Convolutional neural network (CNN) has been performing well in tasks related to robotics (will be discussed in Section 2.1.2). For collision avoidance, [9] trained a CNN network with collision avoidance data collected by a multi-agent simulator with different parameter settings. [18] trained a CNN network with real-world manually labelled depth images.

Deep Reinforcement Learning (DRL) works robustly in numbers of robotic problems (will be discussed in Section 2.1.5). Using DRL in collision avoidance, [20], [19] and [10] used laser range findings, depth images and predicted depth from RGB images to train a deep network from simulation, respectively. [3] designed a reward function that respected common social norms in human walking and trained a deep network to exhibit socially compliant behaviors.

1.4 Motivation

Reinforcement learning works well in numbers of robotic problems. However, safety issue is still the stumbling block of its usage. The introduction of simulation is beneficial as illustrated by [14] in training a drone to fly. Similar idea is applied on a robot in work such as [20, 3, 10]. One of the potential improvement among the works is the simulation environment. Currently all other simulations only contained simple geometric shapes with a low variety, or fixed map.

1.5 Scope

This project aims to develop a simulation framework where policies can be obtained and provide realistic simulation environments. The policy will serve as a decision maker and enables a robot to navigate safely, in the form of a neural network.

Chapter 2

Methodology

This section presents the algorithm used, named Double Dueling Deep Q-Learning (DQN), combining the work from [12, 24, 23]. An overview of each component will be discussed.

2.1 Terminology

2.1.1 Deep Neural Network

Artificial Neural networks (ANN) is inspired by biological theories and serves as a programming paradigm which enables a computer to learn from existing data. It usually consists of multiple layers between input and output.

Deep neural network (DNN) is an ANN with multiple hidden layers between input and output layers. DNNs can model complex non-linear relationships and the architectures generate compositional models where the object is expressed as a layered composition of primitives. The extra layers enable composition of features from lower layers, potentially modeling complex data with fewer units than a similarly performing shallow network.

2.1.2 Convolutional neural network

Convolutional Neural Network (CNN) is a type of hierarchical neural networks for feature extraction. It works well on extracting the underlying information from high-dimensional data such as images. In general, three operations are involved: convolution, non-linear activation and pooling.

Convolution

The convolution operation takes weighted sum on data, for example, pixel values on an image, and returns a feature map. Considering a two-dimensional context, the mathematical expression is denoted by

$$y_{ij} = (W * x_{ij}) + b$$

where y_{ij} represents the value at coordinate (i, j) of the resulting feature map, W represents the convolution kernel, x_{ij} is the (i, j) patch of the input and b is the bias vector of the convolution kernel .

Non-linear activation

Inspired by the biological nerve system inside our brain, an element-wise non-linear activation function is applied to the output feature maps. Common activation functions includes the sigmoid function $s(x) = \frac{1}{1+e^{-x}}$, the hyperbolic tangent function $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and the rectifier $f(x) = max(0, x)$.

Pooling

The function of a pooling layer is to progressively reduce the spatial size of the representation, thus reduce the amount of parameters and computation in the network, and hence to also control overfitting. Usually a pooling layer will take the maximum over patches of customized size while the depth dimension will remain unchanged. Pooling layer can also perform other functions, such as averaging.

2.1.3 Reinforcement learning

Reinforcement learning (RL) [17] is one of the machine learning methods that are used to solve sequential decision making problems. In general, a sequential decision making problem can be formulated as a Markov decision process (MDP), which is defined by the following: $< S, A, P, R, \lambda >$, where S is the state space, A is the action space, P is the state-transition model, R is the reward function, and λ is the discount factor which represents how important the previous action is, relative to the current state.

2.1.4 Q-Learning

Q-learning is one of the reinforcement learning techniques. It can be used to find an optimal action-selection policy for any given (finite) Markov decision process (MDP). A policy refers to a function that makes decision given the perception of the current state. For example, in the context of collision avoidance in robot navigation, the policy decides movement direction and velocity given the current captured images perceived by the robot.

The action-value function $Q(s, a)$ represents the maximum discounted future reward when we perform an action at in state s. The function is denoted as

$$Q(s, a) = r + \gamma max_{a'} Q(s', a')$$

where r is the immediate reward by performing action a in state s, s' is the next state and γ is the discount factor. By learning the action-value function, it eventually results in an optimal policy by selecting the action with the highest action-value in each state.

2.1.5 Deep Reinforcement Learning

Deep reinforcement learning (DRL) methods generally use deep neural networks as function approximator on components of reinforcement learning, for example, the action-value function. It stabilizes the training of action-value function approximation with deep neural networks.

2.1.6 Deep Q-Learning

Deep Q-learning is an good example of DRL. In the context of Q-Learning, a DNN can be used to replace the action-value function. This enables processing of high-dimensional data and thus Q-Learning can be applied on more complex problems. It stabilized the training of action value function approximation with the help of experience replay [8] and target network[11], which will be discussed below.

Experience replay

Experience replay refers to the playback of the experiences stored in a replay memory. After each action, an experience in the form of $< s, a, r, s' >$ will be saved, which are current state, action performed, reward and the next state, respectively. The experiences are then used to train the network. One way is to select the replays subsequently. However, it may cause overfitting or lead to local minimum. Instead, drawing minibatches from the replay memory randomly would break the similarity of subsequent training samples and avoid the problems above.

Target network

The target network refers to the usage of an extra network to store the action-values. The idea is to separate one network into two, where one used to choose actions and the another one is responsible to store the action-values. In contrast, frequent shift of network values will cause destabilization when using a single network. Therefore, by separating the network and updating the target network slowly, [20, 23] found that it stabilized the training process. The update of action-value then becomes

$$Q(s,a) = r + \gamma Q'(s', argmax(Q(s',a)))$$

where Q and Q' represent the two separate networks.

It presented an end-to-end reinforcement learning approach, only required minimal domain knowledge, for instance, images or game scores. In addition, the trained network with the same structure and hyperparameters was illustrated to be capable of being applied to many different tasks, which is 49 Atari games in [1], and achieved good results, even comparably to a human professional player.

2.2 Simulation Environment

This project uses Unreal Engine 4 (UE4) to simulate the virtual training environments, a game engine that allows game developers to design and build games, simulations, and visualizations. UnrealCV[13] is a open-source plugin that enables access and modification of the internal data structures of the games. This project uses UnrealCV for communication between UE4 and the reinforcement learning module implemented with Keras[4], a high-level neural networks API written in Python and capable of running on top of TensorFlow, CNTK, or Theano. Figure 2.1 shows some examples of the simulation environment.

2.3 Network Structure

Figure 2.2 illustrates the network structure. It takes four consecutive depth images as input, processed by a CNN followed with a dueling DQN. The output of the network are the q-values (or likelihood) of each linear and angular action. The best action is simply the one with highest q-value. The following two extensions were not present in the original DQN. They were

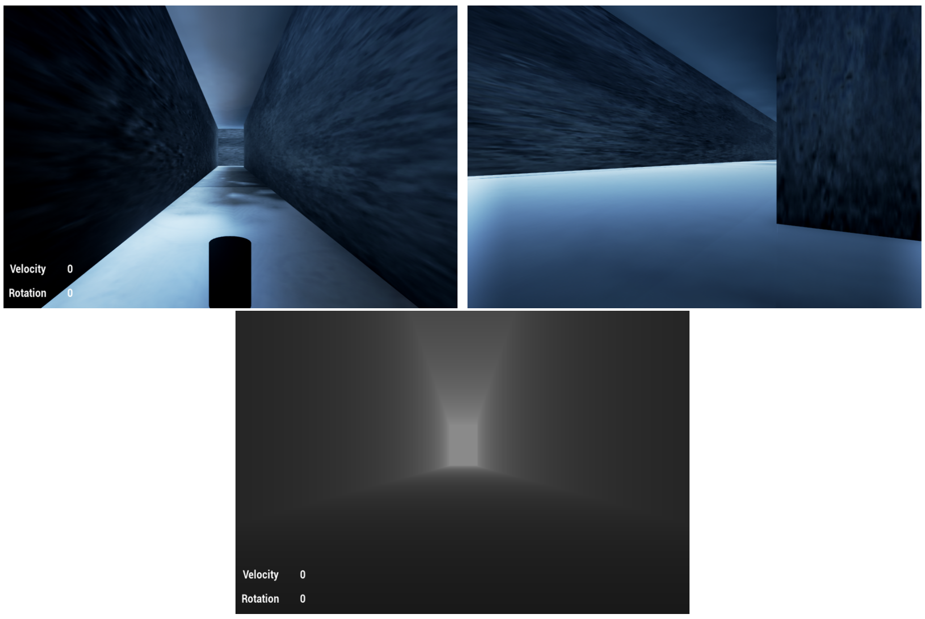


Figure 2.1: (left) Third-person view. The black cylinder is the robot (right) First-person view (bottom) depth image

adopted after experiments which proved the extensions to be beneficial to the performance of the network.

Dueling network

[24] proposed the idea of state-value function V(s) and advantage function A(s,a), namely the dueling network architecture, in contrast to the conventional action-value function Q(s, a). The state-value function V(s) represented how good it is to be in the state s and advantage function A(s,a) represented how much better taking a certain action would be compared to the other possible actions. The two functions were combined to estimate Q(s, a), for faster convergence. The idea can be better illustrated in figure 2.3. The corresponding action-value function then becomes

$$Q(s, a) = V(s) + A(s, a)$$

Dropout

[16] proposed this idea to avoid overfitting in training phrase. The key idea was to randomly drop units (along with their connections) from the neural network during training.

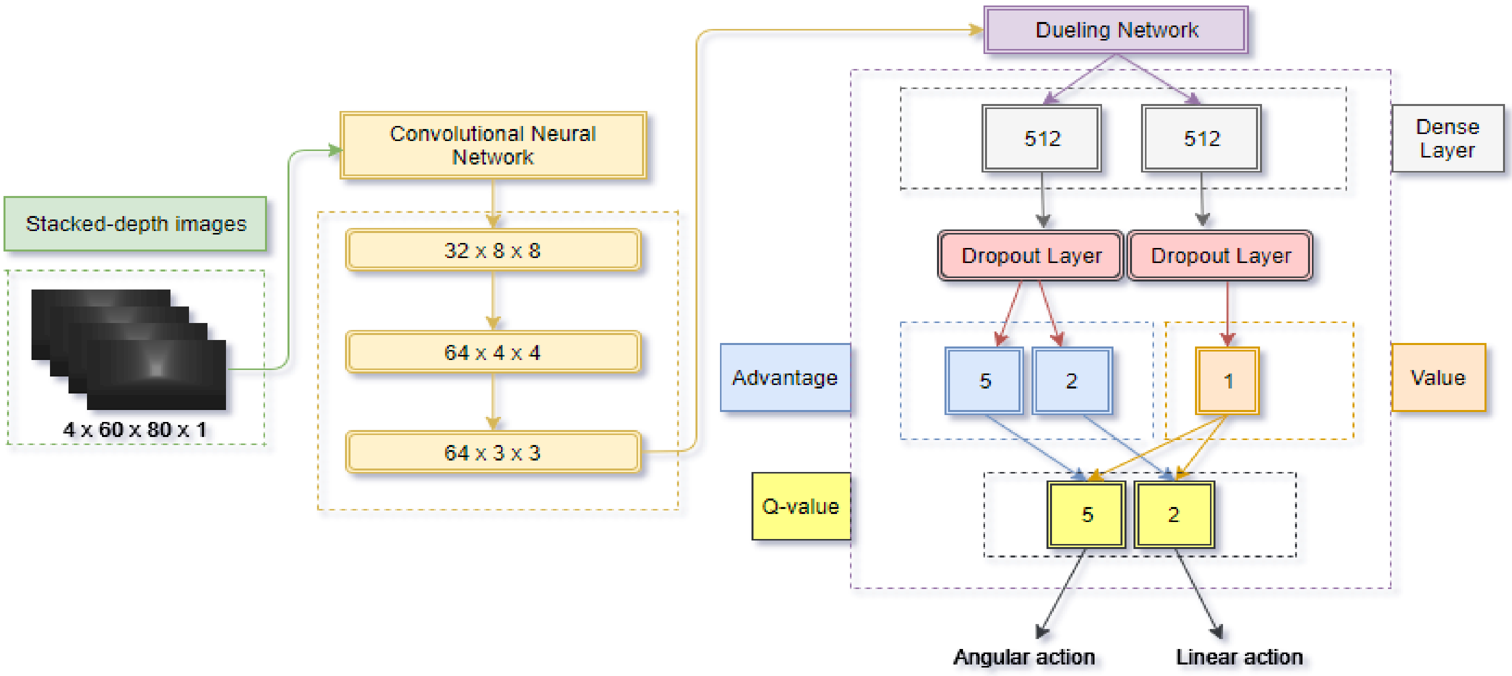


Figure 2.2: The network structure.

2.4 Overview of learning

The agent interacted with the environment and chose random actions based on a probability index which decreased over time. After the agent chose an action, reward was given to the agent and once collision was detected, the episode will restart and the agent will be spawned at next random available location. During training, the agent will store its experience into a buffer and learn from the buffer at the same time. Intuitively the agent will keep learning by distinguishing actions with high rewards under different circumstances.

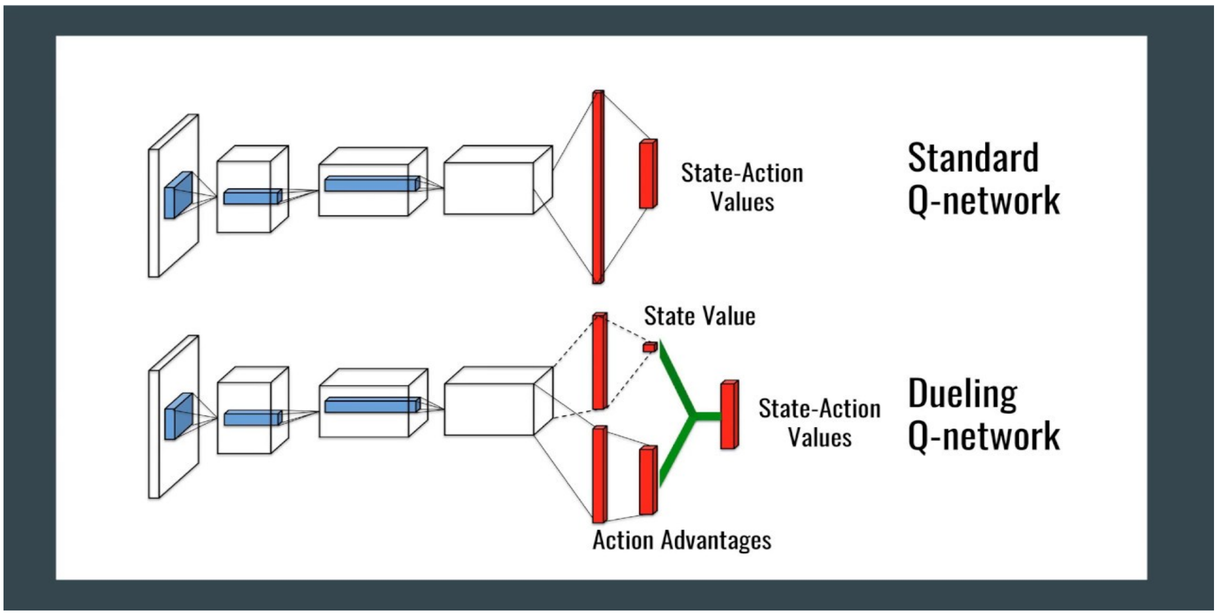


Figure 2.3: Dueling network structure. (Top) The standard network structure with Q(s,a) as action-value function. (Bottom) The dueling network structure with state-value function and advantage function.

Chapter 3

Experiments and Results

3.1 Experiment

Experiments were carried out to evaluate the performance of two network for collision avoidance in two different simulation environments, where each of them is trained separately from the two environments.

The details of experiments, including the environment, action space, task rules and reward function will be discussed here.

3.1.1 Environment-1: Corridor

Setup

The simulation environment is a corridor setting. The agent was spawned at a random available location and no specific tasks or orders were assigned to them. The agent can choose among five different angular actions ($0°$, $\pm 10°$, $\pm 20°$) and two different linear actions (move forward or stay). For simplicity, the distance travelled for moving forward was fixed to be 20 units. When collision was detected, the episode will restart and the agent will be spawned at next random available location. Agent was given images from the previous 3 frames appended with the current frame. Figure 3.1 shows the top-view and perspective-view of the map.

Reward

Reward refers to the score the agent obtained according to an action in order to evaluate how well an action is with respect to the current state the agent is in. The reward is defined as $R = k \cdot v \cdot cos 4\theta$ where v is the velocity, θ is

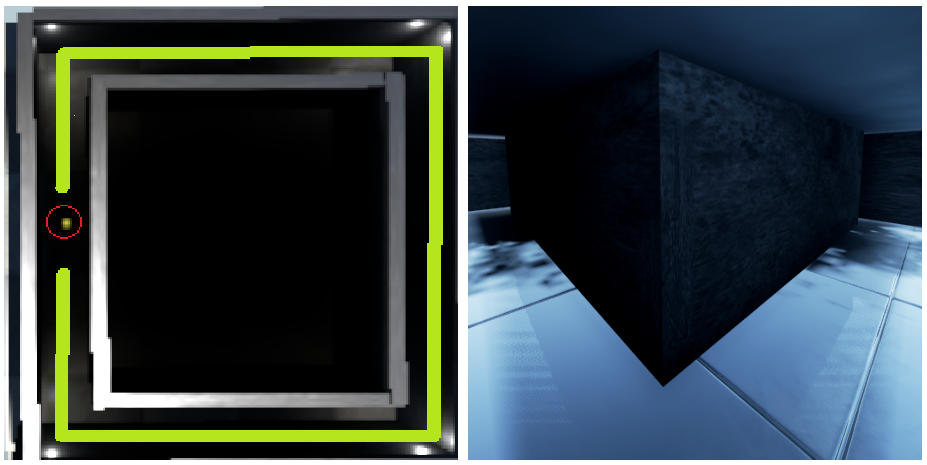


Figure 3.1: The corridor top-view and perspective-view. The red circle refers to the agent and the green line indicates the available space.

the angular velocity and k is a constant for reward normalization. Reward for collision is -10.

3.1.2 Environment-2: A generic map

Setup

The second environment is a generic map, generated at run time by combining square blocks into different shape. The implementation detail will be explained in the next section.
For this map, the agent was spawned at a fixed location and no specific tasks or orders were assigned to them. To reduce problem complexity, the number of actions that agent can choose is reduced to three different angular actions ($0°$, $\pm 10°$) and two different linear actions (move forward or stay). Similarly, the distance travelled for moving forward was fixed to be 20 units. When collision was detected, the episode will restart and the agent will be spawned at the fixed location. Agent was given 1 image only, which is the current frame.

Generic map generation

A map is generated from 4 kinds of blocks: begin/end, left, right and straight (refer to Figure 3.2) with random size obstacles generated at random location. There exists only one obstacle in one block. Figure 3.4 shows an overview of a map and Figure 3.3 shows some more examples.

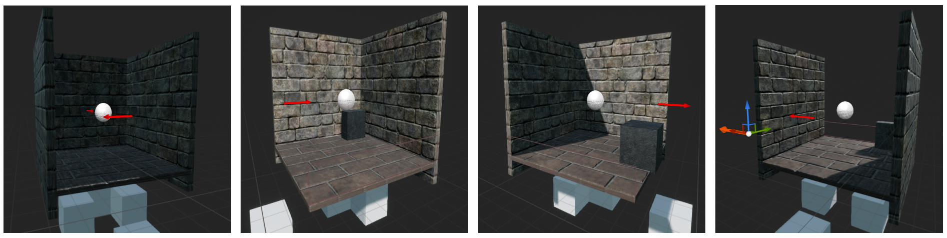


Figure 3.2: (from left to right)begin/end block, left block, right block and straight block. A map will be built by randomly combining these blocks.

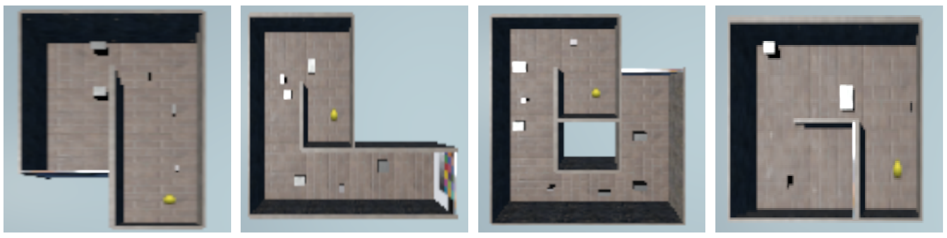


Figure 3.3: Some examples of the dynamic environment.

Reward

Similarly, the reward is defined as $R = k \cdot v \cdot cos4\theta$ where v is the velocity, θ is the angular velocity and k is a constant for reward normalization. Reward for collision is -10.

3.2 Results

3.2.1 Tuned hyperparameter

Discount factor is also a crucial factor to the network performance. Discount factor represents the importance of an action relative to its following actions, known as λ in the action-value function

$$Q(s, a) = r + \gamma max_{a'} Q(s', a')$$

Intuitively, a larger discount factor means the previous action is more important and accountable for its future actions. In the context of collision avoidance, collision may be caused by a sequence of actions, instead of a single action. Therefore, consideration of previous actions is necessary for a robust policy. After some trial, discount factor of 0.95 was found to have the best performance in general, illustrated in Figure 3.5.

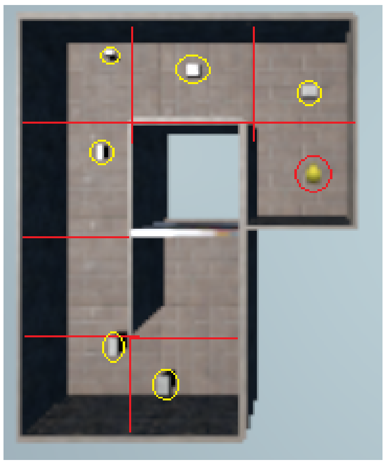


Figure 3.4: The generic map environment top-view. The red line indicates the block boundaries, yellow circle indicates the obstacles and the red circle represents the agent.

3.2.2 Comparison

Two algorithms were compared on two different simulation environment. Although the action space is different, the reward function for both environments are the same. For convenience, P_{Cor} refers to the policy trained in Corridor while P_{Gen} refers to the policy trained in Generic map.

Policy	**Tested in**	$Reward_{average}$*	**SD***
P_{Cor}	Corridor	**9.84**	**0.035**
P_{Gen4}	Corridor	9.78	0.083
P_{Gen1}	Corridor	6.73	3.40
P_{Cor}	Generic map	9.25	1.47
P_{Gen4}	Generic map	**9.72**	**0.81**
P_{Gen1}	Generic map	6.90	2.65

Table 3.1: Performance comparison for two different network. Average reward and standard deviation were obtained from 100 episodes, each episode with 500 steps. Maximum average reward is 10.00.

The result showed an unexpected correlation between training and testing environment. Each of the policy was expected to perform better at the environment which it was trained in. However, P_{Cor} outperform P_{Gen} in both environments.

3.2.3 Intuition

Intuitively Corridor is of less complexity, hence the features are less vague and the agent may learn better. In addition, 4 consecutive images are fed to P_{Cor} while only 1 was fed to P_{Gen}. Motion information was captured in the 4 consecutive images which may cause P_{Cor} to be more stable (smaller SD).

Along with training input and environment, reward function here may also be a crucial factor in determining the performance. The reward function was designed to encourage walking straight without collision. Corridor environment somehow provided a clear and suitable environment to the agent to learn that, since there was mainly straight road and no obstacles throughout the path. Meanwhile, Generic map consisted of numbers of corners and random obstacles. This, however, caused the agent to get use to turning, yielding a smaller reward.

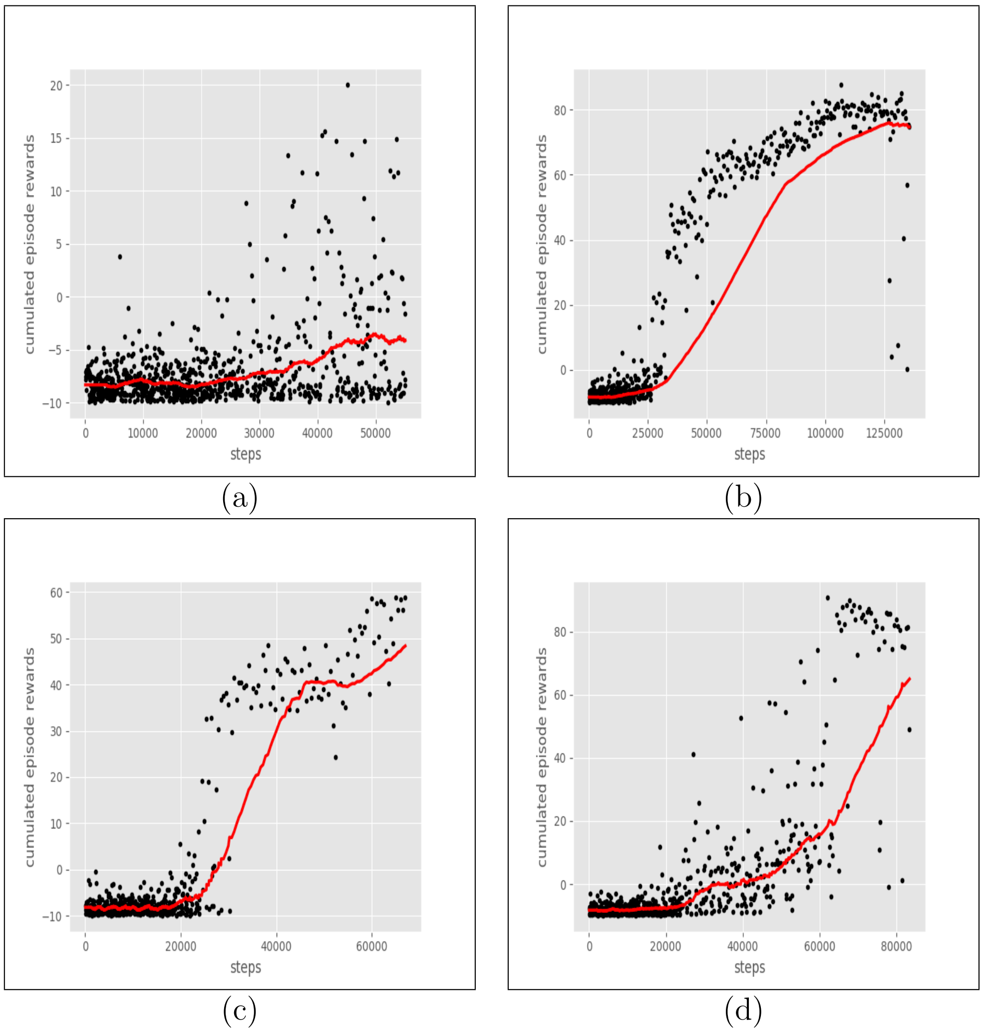
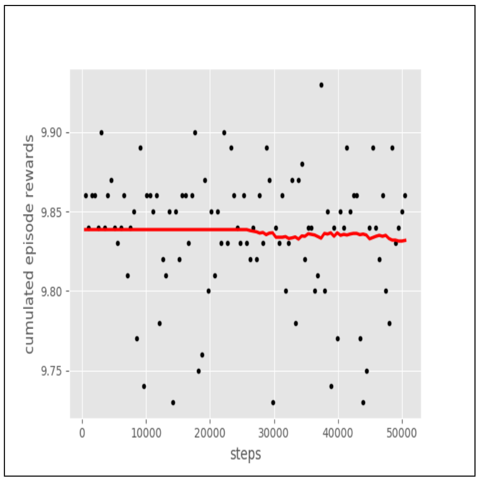
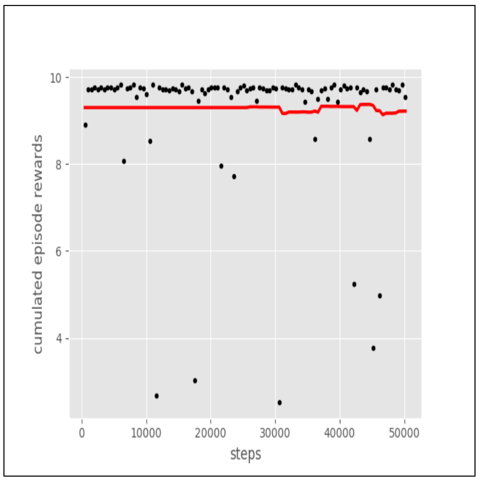


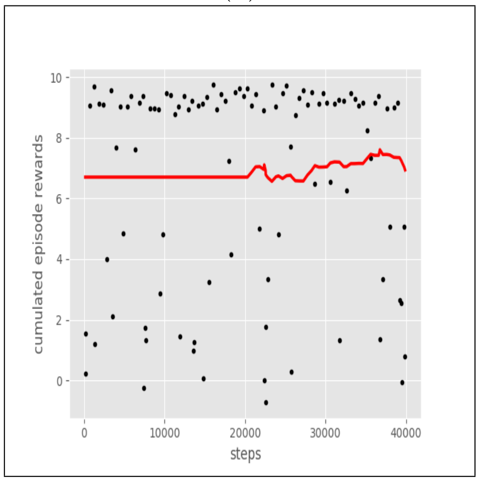
Figure 3.5: The figure shows the average cumulative episode reward across training time. (a) λ=0.9 with original network structure (b) λ=0.9 with dueling network (c) λ=0.99 with dueling network (d) λ=0.95 with dueling network. Notice that (d) has a maximum score of around 90, which is the highest score among all. Although the curve is less smooth, higher maximum score represents the policy produces better trajectories.



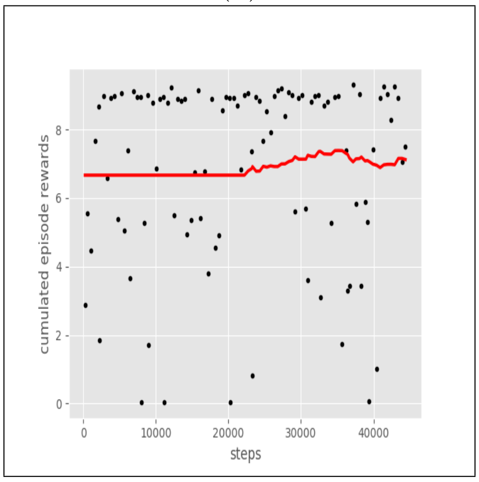
(a)



(b)



(c)



(d)

Figure 3.6: Performance of network(a) P_{Cor} tested in Corridor (b) P_{Cor} tested in Generic map (c) P_{Gen} tested in Corridor (d) P_{Gen} tested in Generic map

Chapter 4

Future Work

The SimNav framework is flexible, providing a training environment along with communication tool between software modules. Eventually, the trained network can be applied onto a robot.
The robot will be using Kinect, a motion sensing input device developed by Microsoft, for obtaining real time depth images or RGB images, according to the setting.
However, one would need to overcome the sensor noise issue and reduce the difference between real and simulation sensor feedback. In simulation, the environment is deterministic. The depth images can be perfectly obtained. However, the sensor data in real world usually contain noisy data. Effort would be needed to mitigate the difference. Figure 4.1 shows an example of a noisy depth image obtained by a Kinect device.

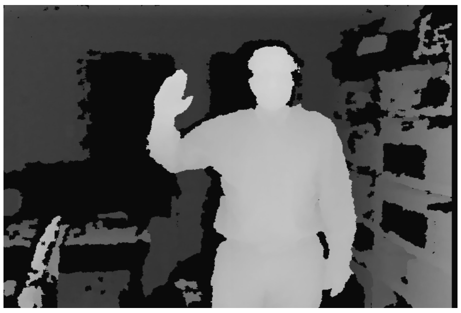


Figure 4.1: (Left) Kinect device. (Right) An example of noisy depth image obtained by Kinect.

Chapter 5

Conclusion

Collision avoidance in robot navigation is an essential area in robotic applications. People in the past adopted constraint-based methods for this problem, while recently majority tended to use learning-based methods. This report proposes a simulation training framework, namely SimNav, and investigates the possibility to train a robot to navigate safely by performing training in SimNav, without any real world data. State-of-the-art reinforcement learning algorithms with different variation will be compared.

References

[1] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012.

[2] Johann Borenstein and Yoram Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, 7(3):278–288, 1991.

[3] Y. F. Chen, M. Everett, M. Liu, and J. P. How. Socially Aware Motion Planning with Deep Reinforcement Learning. *ArXiv e-prints*, March 2017.

[4] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[5] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using the relative velocity paradigm. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 560–565 vol.1, May 1993.

[6] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33, Mar 1997.

[7] S. S. Ge and Y. J. Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, 13:207–222, 2002.

[8] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, May 1992.

[9] P. Long, W. Liu, and J. Pan. Deep-Learned Collision Avoidance Policy for Distributed Multi-Agent Navigation. *ArXiv e-prints*, September 2016.

[10] Jeff Michels, Ashutosh Saxena, and Andrew Y. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 593–600, New York, NY, USA, 2005. ACM.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. *ArXiv e-prints*, December 2013.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

[13] Weichao Qiu and Alan Yuille. Unrealcv: Connecting computer vision to unreal engine. *arXiv preprint arXiv:1609.01326*, 2016.

[14] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *CoRR*, abs/1705.05065, 2017.

[15] J. Snape, J. v. d. Berg, S. J. Guy, and D. Manocha. The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, 27(4):696–706, Aug 2011.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[17] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[18] L. Tai, S. Li, and M. Liu. A deep-network solution towards model-less obstacle avoidance. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2759–2764, Oct 2016.

[19] L. Tai and M. Liu. Towards Cognitive Exploration through Deep Reinforcement Learning for Mobile Robots. *ArXiv e-prints*, October 2016.

[20] Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. *CoRR*, abs/1703.00420, 2017.

[21] J. van den Berg, Ming Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935, May 2008.

[22] Jur van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. *Reciprocal n-Body Collision Avoidance*, pages 3–19. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[23] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[24] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.