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

20 January 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 real robot can be trained with synthetic data only in simulation environment and have comparable performance. Deliverable includes a learning framework to facilitate the training and testing between simulation and real world experiments, a high-quality simulation environment to enhance domain transfer between virtual and real, and a policy applicable on a real robot, which serves 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

Abstract

Acknowledgements

List of Figures

1 Introduction

1.1 Background

1.2 Previous Work

1.2.1 Potential-field based

1.2.2 Dynamic based

1.2.3 Learning-based

1.3 Motivation

1.4 Scope

2 Methodology

2.1 Terminology

2.1.1 Deep Neural Network

2.1.2 Convolutional neural network

2.1.3 Reinforcement learning

2.1.4 Q-Learning

2.1.5 Deep Reinforcement Learning

2.1.6 Deep Q-Learning

2.2 Simulated Environment

2.3 Network Structure

3 Current Status

3.1 Experiment

3.1.1 Setup

3.1.2 Reward

3.2 Discovery
3.2.1  Impact of network structure ........................................ 11
3.2.2  Impact of hyperparameters .......................................... 11
3.3  Problem encountered ...................................................... 12
3.3.1  High Expectation ....................................................... 12
3.3.2  Lack of reference ...................................................... 13
3.3.3  Sensor noise ........................................................... 14

4  Future Work .............................................................. 15
   4.1  Training framework ................................................... 15
   4.2  Domain transfer ....................................................... 15
   4.3  Environment Complexity ............................................. 15
   4.4  Real world testing .................................................... 16

5  Conclusion ............................................................... 17

References ................................................................. 18
List of Figures

2.1 (left) Third-person view. The black cylinder is the robot
(right) First-person view (bottom) depth image .............. 8
2.2 The network structure. ............................................ 9
2.3 Dueling network structure. (Top) The standard network struc-
ture with Q(s,a) as action-value function. (Bottom) The duel-
ing network structure with state-value function and advantage
function. ................................................................. 9
3.1 The corridor environment setting. ............................ 11
3.2 The first row shows a turn-right scenario and the second row
shows a slight-left scenario. ................................. 12
3.3 The figure shows the average cumulative episode reward across
training time. (top-left)(a) $\lambda=0.9$ with original network struc-
ture (top-right)(b) $\lambda=0.9$ with dueling network (bottom-left)(c)
$\lambda=0.99$ with dueling network (bottom-right)(d) $\lambda=0.95$ 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. ............................... 13
3.4 (Left) Kinect device. (Right) An example of noisy depth image
obtained by Kinect. .................................................. 14
4.1 Turtlebot ................................................................. 16
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.

This report is structured as following. Chapter 1 will provide an introduction on the topic. Chapter 2 will discuss the methodology. Chapter 3 talks about current status, findings and problems encountered. Finally chapter 4 will investigate the future work needed.

1.1 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.

1.2 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.

1.2.1 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. [8] made use of the concept of potential field from physics to represent the admissible velocities.

1.2.2 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.

1.2.3 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, [10] 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 [11] 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.

1.3 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, 11]. One of the potential improvement among the works is the simulation environment. Currently all other simulations only contained simple geometric shapes, but not things frequently seen in real life, for example, human. One of the objectives is to enhance the simulation environment for training the robot and deploy on a real robot. This would reduce the discrepancy between robot’s perception in simulation and the real world.

1.4 Scope

This project aims to develop a policy which servers as a decision maker and enables a real robot to navigate safely. The policy will be in the form of a neural network and trained in simulation environments. It focuses on indoor collision avoidance with both static and dynamic obstacles, for example, walls and pedestrians.
Chapter 2
Methodology

This section presents the algorithm used, named Deep Q-network (DQN) [12], improved with several extensions. An overview of each component will be discussed.

2.1 Terminology

Various technical terms involved will be introduced and explained in detail.

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 \( \text{tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \) 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 \( \lambda \) 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 \( a \) 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 \( \gamma \) 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 in 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 [9] and target network, 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] found that it stabilized the training process. The update of action-value then becomes

\[ Q(s, a) = r + \gamma Q'(s', \text{argmax}(Q(s', a))) \]

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

**Dueling network**

[23] 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.

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

Current Status

Experiments were carried out to evaluate the network performance for collision avoidance in simulation environment.

3.1 Experiment

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

3.1.1 Setup

The simulation environment was a corridor setting, as illustrated in Figure 3.1. The agent was spawned at a random location and no specific tasks or orders were assigned to them. It interacted with the environment and chose random actions based on a probability index which decreased over time. The agent can choose among five different angular actions ($0^\circ$, $\pm 5^\circ$, $\pm 10^\circ$) and two different linear actions (move forward or stay). One example is to move forward in the direction of $5^\circ$. For simplicity, the distance travelled for moving forward was fixed to be 20 units. 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 a random location. Agent was given images from the previous 3 frames appended with the current frame. Samples from navigating in simulation are shown in Figure 3.2.

3.1.2 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 $R = k \cdot v \cdot \cos \theta$ where $v$ is the velocity, $\theta$ is the angular velocity and $k$ is a constant for reward normalization. Reward for collision is -10.

### 3.2 Discovery

There were several discoveries obtained from experiments.

#### 3.2.1 Impact of network structure

Performance with different network structure is shown in Figure 3.3. The learning rate, which is a parameter that controls the size of weight and bias changes in learning of the training algorithm, was fixed to be 0.0000001. In the context of collision avoidance, it would be hard for fast convergence in the learning due to high dimension data. Results showed that the agent obtained the best performance with dueling network and dropout.

#### 3.2.2 Impact of hyperparameters

In addition to network structure, 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 $\lambda$ 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. Results in Figure 3.3 showed that a discount factor of 0.95 performs better than 0.9 and 0.99.

3.3 Problem encountered

Although network performance in simulation was satisfactory, there were a few problems encountered.

3.3.1 High Expectation

The initial expectation in the project plan was too high. Before the implementation phrase, everything seemed achievable and manageable. However, hardware and engineering difficulties occurred during implementation of the algorithm. For example, one loop of network training took approximately 8 to 12 hours, which was a very long time. Therefore, intermediate goals were introduced to modularize the problem. For instance, simulation envi-
ronment does not contain dynamic obstacles currently. When agents perform well in easier environments, difficulty will be increased gradually and it will be trained in more complex environments.

Figure 3.3: The figure shows the average cumulative episode reward across training time. (top-left)(a) $\lambda=0.9$ with original network structure (top-right)(b) $\lambda=0.9$ with dueling network (bottom-left)(c) $\lambda=0.99$ with dueling network (bottom-right)(d) $\lambda=0.95$ 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.

3.3.2 Lack of reference

There were not much similar works available online. Therefore, a large portion of work were built from scratch. This increased the difficulty of the
project and time needed to spend on engineering effort.

3.3.3 Sensor noise

The real robot will be using Kinect, a motion sensing input device developed by Microsoft, for obtaining real-time depth images. 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 3.4 shows the Kinect and an example of a noisy depth image obtained by it.

Figure 3.4: (Left) Kinect device. (Right) An example of noisy depth image obtained by Kinect.
Chapter 4

Future Work

Basically the project is on schedule. There are several tasks to achieve in the next few months.

4.1 Training framework

Eventually, this project aims to develop a policy that can be applied on a real robot to navigate and avoid collision. To facilitate the project, a framework from simulation training to real robot testing is needed. It includes the development of the network, the testing of the agent in simulation, the engineering effort on the real robot and the testing of the real robot navigation. This can serve as a standard procedure to develop, deploy and test a particular network structure.

4.2 Domain transfer

Discrepancies exist between virtual environment and the real world. For the robot to work in both conditions, the environment needs to be transferable. The perception of the robot in simulation needs to be similar to that in real world. In this case, the depth images obtained by the robot must be similar. Further verification and synchronization are required and if necessary, processing of depth images in virtual world may be needed.

4.3 Environment Complexity

The current environment complexity is relatively trivial. In order to be able to navigate in real world, the robot would need to learn in a more complex
environment.

4.4 Real world testing

This project will be using Turtlebot [7] for real world testing, as shown in Figure 4.1. Currently testing has been carried out in simulation only. Real world testing is needed.

Figure 4.1: Turtlebot
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 some tended to use learning-based methods. This report investigates the possibility to train a robot to navigate safely by performing training in simulation, without any real world data. In particular, state-of-the-art reinforcement learning algorithms will be used. Currently this project has successfully trained a policy for navigating in a simulated corridor environment. In the future, more complicated simulation environments will be introduced and extensive experiments on a real robot will be carried out.
References


