Imitation Learning based on Generative Adversarial Networks for Robot Path Planning

Thesis by

Xianyong Yi

In Partial Fulfillment of the Requirements

For the Degree of

Masters of Science

King Abdullah University of Science and Technology

Thuwal, Kingdom of Saudi Arabia

November, 2020
The thesis of Xianyong Yi is approved by the examination committee

Committee Chairperson: Dominik L. Michels
Committee Members: Peter Wonka, Mikhail Moshkov
ABSTRACT

Imitation Learning based on Generative Adversarial Networks for Robot Path Planning
Xianyong Yi

Robot path planning and dynamic obstacle avoidance are defined as a problem that robots plan a feasible path from a given starting point to a destination point in a nonlinear dynamic environment, and safely bypass dynamic obstacles to the destination with minimal deviation from the trajectory. Path planning is a typical sequential decision-making problem. Dynamic local observable environment requires real-time and adaptive decision-making systems. It is an innovation for the robot to learn the policy directly from demonstration trajectories to adapt to similar state spaces that may appear in the future. We aim to develop a method for directly learning navigation behavior from demonstration trajectories without defining the environment and attention models, by using the concepts of Generative Adversarial Imitation Learning (GAIL) and Sequence Generative Adversarial Network (SeqGAN). The proposed SeqGAIL model in this thesis allows the robot to reproduce the desired behavior in different situations. In which, an adversarial net is established, and the Feature Counts Errors reduction is utilized as the forcing objective for the Generator. The refinement measure is taken to solve the instability problem. In addition, we proposed to use the Rapidly-exploring Random Tree* (RRT*) with pre-trained weights to generate adequate demonstration trajectories in dynamic environment as the training data, and this idea can effectively overcome the difficulty of acquiring huge training data.
ACKNOWLEDGEMENTS

The Master student life will end soon, and looking back on the period, I have benefited a lot and made much progress from the professors’ teaching and the enthusiastic help of the classmates and friends. I would like to express my gratitude to my supervisor Dominik L. Michels for helping me to gain study experience, which will very benefit my future study. I also would like to thank my friends for helping me during my study life. Thanks very much.
## TABLE OF CONTENTS

Examination Committee Page 2

Copyright 3

Abstract 4

Acknowledgements 5

Table of Contents 6

List of Abbreviations 8

List of Figures 9

1 Introduction 10
   1.1 Research Gap Statement and Motivation 10
   1.2 Goals of the Thesis 13
   1.3 Contributions of the Thesis 14
   1.4 Outline of the Thesis 14

2 Background and Literature Review 15
   2.1 Traditional Robot Navigation 15
   2.2 Human-aware Robot Navigation 16
   2.3 Machine Learning 18
   2.4 Learn from the experience of image generation based on traditional GAN 19
   2.5 Imitation Learning 20
   2.6 Sequence Generative Adversarial Nets 23
   2.6.1 Generator of SeqGAN 25
   2.6.2 Discriminator of SeqGAN 26
   2.7 Use the RRT* with Pre-trained Weights to Aquire Training Data 26
      2.7.1 RRT* with Pre-trained Weights 27
      2.7.2 Features in Cost Function 28
3 Proposed Planning Network

3.1 Overview of the Idea ............................................ 30
3.2 Problem Setting .................................................. 31
3.3 Neural Network Architecture .................................. 31
3.4 Feature Count Objective ....................................... 33
3.5 Policy Gradient ................................................... 34
3.6 Use Monte Carlo Simulation to Estimate the Action-Value Function . 38
3.7 Generator ......................................................... 39
3.8 Discriminator ..................................................... 39

4 Experimental Work ............................................... 42

4.1 The ROS Environment and Information Pre-process ............. 42
4.2 Comparison Metrics ............................................. 43
4.3 Results and Analysis .......................................... 44

5 Conclusion ......................................................... 50

5.1 Summary ......................................................... 50
5.2 The Future Work ................................................ 51

References ............................................................ 53

Appendices ........................................................... 58
## LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>DQN</td>
<td>Deep Q-Network</td>
</tr>
<tr>
<td>FCN</td>
<td>Fully Convolutional Network</td>
</tr>
<tr>
<td>GAIL</td>
<td>Generative Adversarial Imitation Learning</td>
</tr>
<tr>
<td>GAN</td>
<td>Generative Adversarial Network</td>
</tr>
<tr>
<td>GRU</td>
<td>Gate Recurrent Unit</td>
</tr>
<tr>
<td>IRL</td>
<td>Inverse Reinforcement Learning</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short-Term Memory</td>
</tr>
<tr>
<td>MCTS</td>
<td>Monte Carlo Tree Search</td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>ROS</td>
<td>Robot Operating System</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>RRT</td>
<td>Rapidly-exploring Random Tree</td>
</tr>
<tr>
<td>RRT*</td>
<td>Rapidly-exploring Random Tree*</td>
</tr>
<tr>
<td>SeqGAIL</td>
<td>Sequence Generative Imitation Learning</td>
</tr>
<tr>
<td>SeqGAN</td>
<td>Sequence Generative Adversarial Network</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

2.1 Features used in the optimized RRT* path planning method . . . . . . . 28
3.1 The Process of Training the Discriminative Model . . . . . . . . . . . 32
3.2 Overview of the SeqGAIL Architecture . . . . . . . . . . . . . . . . . 32
3.3 The Process of Reinforcement Learning . . . . . . . . . . . . . . . . . 35
3.4 Details of Generator . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
4.1 Result Visual Comparisons between FCN and SeqGAIL . . . . . . . . 45
4.2 Selected Cases Generated by SeqGAIL . . . . . . . . . . . . . . . . . 47
4.3 Averaged Distance Metric Comparison . . . . . . . . . . . . . . . . . 48
4.4 Averaged Feature Counts Errors Comparison . . . . . . . . . . . . . . 48
4.5 SeqGAIL Performance in Different Path Lengths . . . . . . . . . . . . . 49
Chapter 1

Introduction

This thesis develops and implements a Generative Adversarial Network (GAN) based approach with Imitation Learning and Sequence Generative Adversarial Network (SeqGAN) concepts for robot path planning in a dynamic environment, in which the robot needs to plan a feasible path to goal destination and avoid moving obstacle or humans. We model the path planning as a sequence generation problem, more specifically, as a sequential decision making via sequence generation [1]. The robot path planning in dynamic environment using conventional methods also suffers from the instability problem [2] and we perform the refinement method to increase the stability of the path planning [3]. In addition, an objective forcing method is applied on the generation process. This chapter briefly states the main content of the thesis, including a general introduction to the problem, motivation and goals of the research. The contributions of this thesis are also highlighted.

1.1 Research Gap Statement and Motivation

With the rapid development of science and technology, especially information science and technology, mobile robots have played an increasingly important role in our daily life. From simple sweeping robots and robotic wheelchairs to driverless cars, the application areas of mobile robots are rapidly expanding. In order to successfully complete various tasks, it is necessary to avoid collisions with obstacles in the environment, and to complete navigation from one point to another. The application
of path planning can make the mobile robot obtain the best navigation [4], reduce unnecessary curse paths, and improve the work efficiency of the mobile robot.

With the expansion of the application scenarios of mobile robots, the working environment of mobile robots is becoming more and more complex. This requires robots to have strong autonomous learning capabilities, so that they can have a certain degree of adaptability to the environment and plan a real-time optimal path. At present, the path planning methods that have achieved good results in practical applications are: random tree method [5], artificial potential field method [6], genetic algorithm [7], cell decomposition [8], Deep Q-Network (DQN) [9] and other methods. The above methods are all obtained on the basis of handcrafted models and do not have the adaptability to the environment. For this reason, designing a method to obtain the optimal path through autonomous learning of robots is the key to improving the adaptability of mobile robots to the environment.

Among the above mentioned approaches, the DQN method performs well in dynamic path planning in unknown environment. Actually, in conventional Reinforcement Learning (RL) tasks, the optimal strategy is learned by simply and directly calculating the cumulative reward, without training samples, and the training data comes from the agent’s interactive behavior in the environment. In this case, it is difficult for the agent to reach the target position by planning a long term path, and it cannot get rewards frequently. It is also because of this that it is impossible to obtain model parameter updates in a short time, and a lot of redundant work will be performed, resulting in the model parameter learning is difficult. In the path planning problem, there is often a huge search space, which further increases the difficulty of the application of the reinforcement learning algorithm. Imitation learning changes the learning method of the agent in traditional reinforcement learning, and the agent learns directly from the expert sample. Therefore, this thesis uses the concept of imitation learning to realize path planning.
Imitation learning refers to learning from expert samples given by humans or other agents. For a path planning problem, its solution is a sequence containing a series of location information and the actions to be taken. Extracting state-decision pairs and forming a new set as fragments can use states as features and decisions as markers for supervised learning, so that the behavior or action sequence of the machine agent can be close to the required standard behavior or action sequence. Different from the classic DQN, this thesis chooses to use expert samples to train the neural network instead of interacting with the environment in traditional reinforcement learning, so as to accelerate the learning process of neural network weights and make the weights easy to control. Learning directly from trajectory examples is the strength of Generative Adversarial Imitation Learning (GAIL) [10]. However, GAIL is suitable for motion planning, but not for the sequence generation, nor the path planning. The reason is that the standard GAIL model is based on the Inverse Reinforcement Learning (IRL) concept that only considers the current state of the agent for deciding next action [11], while the path planning depends on not only the current position but also the previous positions [12]. In this case, the idea of SeqGAN should be an appropriate approach as a basis to perform the sequential decision making task, path planning in our setting, by learning directly from trajectory examples, given that sequential decision making and sequence generation can be combined with the imitation learning concept [13][14]. In this thesis, we combine the concepts of Imitation Learning and SeqGAN to establish a Sequence Generative Imitation Learning (SeqGAIL) model.

The motivation behind the idea of SeqGAIL is that although we cannot completely manually define the data distribution of demonstration trajectories, we can use the discriminative network to automatically learn the characteristics of demonstration trajectories. In SeqGAIL, we jointly train two sub-adversarial models: the Generator generates the path points sequence based on the environment information; and the Discriminator predicts the probability that the generated path sequence is a real
sequence, which is a feasible planned path in the dynamic environment. In the training process, the Generator aims to deceive the Discriminator into believing that its output is an real path sequence, while the Discriminator strives to avoid being fooled by improving its ability to distinguish between the generated sequence and the real sequence. When the Generator and Discriminator reach the Nash equilibrium, this adversarial training achieves a win-win situation [15] [16][17].

Additionally, in recent years, Fully Convolutional Network (FCN) has gained researchers’ attention on path planning, and shown good results in terms of off-line path planning both in static and dynamic environments [18][19][20][21]. [19] proposed an new approach based on FCN to learn path planning in dynamic environment from demonstration trajectory examples generated by pre-trained Rapidly-exploring Random Tree* (RRT*). In this research, we would like to compare our method with the FCN-based method.

1.2 Goals of the Thesis

In this thesis, we focus on the robot path planning in dynamic environments. The goals of this research are as following:

• Based on Imitation Learning and SeqGAN concepts, to develop and implement a path planning method for robots in dynamic environment, without explicitly defining the environment and attention models. The method in this thesis is expected to be able learn the demonstration trajectory examples from RRT* with pre-trained wights, to imitate the expert’s behaviors while crossing the crowd.

• The proposed approach could generate a feasible path, given that the environment information including moving persons or obstacles, and the start and target points are provided.

• The proposed off-line (global) path planning method in this research is expected to outperform the off-line method based on FCN as mentioned above.
1.3 Contributions of the Thesis

With the completion of the project, the goals of the thesis have been accomplished, and a path planning framework based on GAN has been successfully developed. So the contributions of this thesis is as following:

- We successfully adopted the Generative Adversarial Network concept on the robot dynamic path planning problem, to imitate the demonstration behaviors.
- We take the refinement measure to solve the instability problem.
- We proposed to use a objective forcing method to guide the Generator besides of the reward from Discriminator.
- In addition, we proposed to use the RRT* with pre-trained weights to generate adequate demonstration trajectories in dynamic environment as the training data, and this idea can effectively overcome the difficulty of acquiring huge training data.

1.4 Outline of the Thesis

The following are the descriptions of the structure of the thesis. Chapter 1 states the main content of the thesis, including a general introduction to the problem and goals and contributions of the research. Chapter 2 provides detailed literature review and background information about robot path planning. Moreover, the basic machine learning concepts and Sequence GAN model are also introduced. As one of the main contributions of this thesis, Chapter 3 discusses the implementation details of the SeqGAIL network. It illustrates the methodology of policy gradient and Monte Carlo Simulation in the path sequence generation. In Chapter 4, the simulation environment Robot Operating System (ROS) and results of the approach are introduced, and we compare with the FCN-RRT* approach to test the performance of the SeqGAIL. In Chapter 5, the conclusion and future work of the thesis are introduced.
Chapter 2

Background and Literature Review

2.1 Traditional Robot Navigation

In a broad sense, robot navigation includes navigation system components such as map construction, robot positioning, path planning, and motion control. The robot navigation concept in this research refers to the path planning of the robot from the current position to the target position. Path planning algorithms usually require global environmental information to plan a collision-free path connecting the starting position and the target position, and then precisely control the robot to execute according to the planning result to reach the target point. In the research of path planning, the most typical method is the A* search algorithm, which divides the map space into equal-sized cell, and then generates a cost map based on the cost value of each cell [22]. The A* algorithm uses a heuristic function to quickly search in the state space and find a path that connects the starting point and the target with the least cost. The A* algorithm assumes that the map environment information does not change during the search process. Once the environment information changes during the search process, the A* algorithm needs to be re-run for re-planning [22]. In a real environment, environmental information will always change. In order to improve the planning efficiency of the A* algorithm in a dynamic environment, [23] proposed D* algorithm that can be applied in a dynamic environment. In the re-planning stage, the intermediate state nodes that have been expanded in the previous planning process can be used, thereby improving the planning efficiency in a dynamic environment.
Both A* and D* planning algorithms are deterministic path planning algorithms, and the solutions obtained can ensure completeness and optima, but they are only suitable for state search in low-dimensional space.

In the path planning problem, its state space is huge, and usually they cannot be solved efficiently in real time. In order to solve the high-dimensional state space planning problem, some planning methods based on random sampling have been proposed. A method based on random sampling is Rapidly-exploring Random Tree (RRT) method. The algorithm starts from the starting point and expands a tree-like branch outward, and the direction of the branch is obtained by random sampling in the planning space. The tree branches continue to expand until reaching the target point, so the feasible path can be obtained by backtracking the tree structure.

Because the sampling process of the RRT algorithm has the randomness characteristic, the generated path by RRT is usually only a feasible path rather than an optimal path [24]. To optimize the generated trajectories, RRT* algorithm is proposed by [25], which is an improved version of RRT. The main difference between RRT* and the basic RRT algorithm is that the RRT* algorithm has a search process for the neighboring nodes of the newly generated node [25], which is used to select a low-cost parent node. Besides, a rewiring process is adopted to further reduce the path cost. It is a breakthrough method to solve the problem of high-dimensional optimal path planning [24]. Given enough running time, the RRT* will always find the optimal solution [25]. But in a dynamic environment, the path generated by the pure RRT* maybe is not the ideal one, because it could not take the intention of moving objects or persons into account.

2.2 Human-aware Robot Navigation

As more and more robot technology is applied on the daily life environment from the factory environment, robot navigation technology is also facing new challenges.
Traditional robot navigation usually only considers the safety of the robot itself and the surrounding environment while all other people or objects in the environment are regarded as obstacles. The robot under this navigation design framework cannot adapt to the current pedestrian-centered environment [26]. In order to solve this problem, there have been more and more researches on human-aware-based robot navigation in recent years. Human-aware robot navigation is a cross-research direction of human-computer interaction and path planning. It also considers the problem of robot navigation and the problem of interaction with pedestrians in the environment during navigation [26]. In robot navigation applications, such as robot guidance services, robot escort services, or robot transportation services, robots need to have the ability to interact with pedestrians while they are moving [27]. Human-aware navigation research is aimed at improving this ability.

The purpose of human-aware-based robot navigation is that the navigation behavior of the robot is more acceptable to pedestrians in the environment. Specifically, the feasible navigation behavior of the robot in crowd has the following characteristics [26]: (1) Comfort: The interactive behavior of robot navigation will not be felt by pedestrians disturbance or nervousness; (2) Naturalness: The navigation interaction behavior of the robot can be similar to the interaction behavior between people: (3) Sociality: The navigation behavior of the robot can conform to interaction habits when facing the human. In order to meet these Human-aware robot navigation requirements, the human-aware-based costs and constraints are included into path planners. But the cost functions need to be pre-defined, which is not appropriate for imitating navigation behaviors in crowd [26]. Then the proximity model is proposed, but proximity focuses on person-to-person interaction, and does not work well in navigating among people. In order to solve the path planning and dynamic obstacle avoidance problems of robots, we adopted the concepts of machine learning, specifically Imitation Learning and Sequence Generative Adversarial Networks. In
the following sections, I would like to first introduce the Machine Learning, Imitation Learning, and Generative Adversarial Nets concepts for better understanding.

2.3 Machine Learning

Machine learning mainly is used to study how to analyze and use given data to improve the performance of existing systems [28]. The content of the data is different, and the form is also different, so the association between the data and the inner rule of the data often cannot be discovered simply by observation. The problem of learning is how to use and process these data. The most human-like system is a system that can learn independently without manual intervention. Traditional systems require humans to provide certain assistance to the system. Machines learning algorithm allows the system to use data to autonomously discover rules and learning experience without manual intervention. Therefore, many experts take machine learning as an important way to artificial intelligence [29]. Machine learning is now widely used in various fields of human life and production.

Based on the type of label existing in the machine learning method, machine learning is divided into four categories, including supervised learning, semi-supervised learning, unsupervised learning and reinforcement learning [30]. Different types of machine learning algorithms have different applicable scenarios. The supervised learning and semi-supervised learning are based on training samples with certain conditions, and corresponding training samples with a certain amount of labels are needed [31]. If the corresponding labeled samples is provided, the supervised and semi-supervised methods can achieve good results. For some other problems, it is relatively difficult to obtain the labeled training samples required by the algorithm, and it is often impossible to provide a sufficient number of labeled training samples. The supervised and semi-supervised methods cannot solve related problems well. Reinforcement learning does not need labeled samples when facing these problems, but it needs to learn
through the interaction between the system and the environment. The reinforcement learning alleviates the predicament faced by the supervised learning algorithm [32].

Another algorithm that emphasizes autonomous learning is the imitation learning method [33]. Different from traditional reinforcement learning methods, the imitation learning algorithm uses the expert’s policy, and avoids interaction with the environment by analyzing the expert’s policy [33]. Sometimes we often need an agent to be able to complete a certain task in a certain way. Reinforcement learning cannot control the agent’s tendency to choose the corresponding method, which requires the use of imitation learning algorithms. The agent uses the existing expert policy, and builds a policy model. Then the built policy model can control the agent to reproduce the expert trajectories.

The imitation learning algorithm has broadened the scope of the use of reinforcement learning algorithms in the past. Now it has been applied in many fields. For example, it can implement robot movement, human-computer interaction, and policy learning for specific tasks based on imitation learning methods [34]. Before introducing Imitation Learning, I would like to make a brief introduction of Generative Adversarial Nets whose concepts will be adopted in this thesis.

2.4 Learn from the experience of image generation based on traditional GAN

GAN [16] has been widely used in image generation since it was proposed. In order to use GAN to implement policy imitation learning in path planning, we can learn from the experience of GAN in image generation. When using GAN to generate an image, the generator uses randomly distributed data as input to generate an image, then the image output by the generator and the real image are used as the discriminator input, so that the discriminator outputs a probability value, which indicates the probability that the generated image belongs to the real image [16]. The process from
random data to image is a process from image encoding to decoding. The generator can be seen as mainly composed of an encoder and a decoder. The encoder is often constructed by a convolutional neural network, and the decoder mainly includes de-convolution neural network. The expanded noise data is reshaped into image features when passing through the convolutional layer [35]. The image features are extracted through the multi-layer network. After Batch Normalization and deconvolution, the extracted features are restored to high-resolution images. The discriminator actually is a classifier, with a convolutional neural network as the main component. After the convolutional neural network extracts image features, the discriminator finally outputs a probability that the input image belongs to the real image.

The process of pictures generation by GAN is mapping from the noise back to the original image space, and projecting from low-dimensional to high-dimensional, which disassembles the image distribution factors in the training pictures [36], and narrows the JS divergence of the image distribution of real pictures and generated pictures. Each pixel in an image is a variable, and the number of pixels contained in the entire image increases as the resolution of the image increases, and is related to the color space of the image, forming a multivariate distribution [35].

2.5 Imitation Learning

Imitation learning uses the existing expert trajectories to learn policy. By converting the given expert trajectories into example data and encoding the sample, the sample data can be transmitted between the agent and the expert. In the theory based on reinforcement learning algorithm, imitation learning algorithm can effectively improve the performance of the policy function learned by reinforcement learning algorithm [33]. In the setting of imitation learning, the agent needs the expert samples to learn strategies. According to different ways of processing state-action pairs in expert strategies, different methods of imitation learning can be implemented, and the most
important problem of imitation learning is how to use expert sample data. There are three main methods for us to study imitation learning: Behavioral Cloning method, Learning based on Inverse Reinforcement Learning and Generative Adversarial Imitation Learning (GAIL) [10].

**Behavioral Cloning**

Behavioral Cloning is one of the main ways to implement the imitation learning algorithm [33]. The expert strategy provided in advance is used as the cloning object. The agent analyzes the state in the expert data and moves the classifier adopted by the expert in the corresponding state. Through the classifier, the agent can correctly classify the state when facing a similar environmental state, and the category is the action that should be performed in the corresponding state [37]. This method can implement imitation learning well and does not require interaction with the environment. The imitation learning algorithm implemented in this way has been used in training quadcopters, and the trained aircraft can fly through the woods. The same algorithm has also been tried to be used in automatic driving, using human driving data [37]. For example, the driving trajectory is used as the expert data, while the image taken by the front camera is used as the state, and the expert’s control action is the corresponding label, so as to train the corresponding strategy network to implement the automatic driving. Although the behavioral cloning method is dealing with many difficult problems, it still has certain defects. The behavioral cloning method requires expert data to provide corresponding labels in all states, that is, the expert is required to guide the agent in all possible thoughtful situations. The expert data must include all states and the corresponding label. In many environments, such complete expert data is often not available, so the behavioral cloning method cannot be applied to this type of scenario [38].
Inverse Reinforcement Learning

The method based on Inverse Reinforcement Learning is also an important way to implement the imitation learning [10] [39]. The basic idea of the inverse reinforcement learning method is to try to find a unique reward function. The characteristic of this reward function is that it can always make the reward value of the expert strategy higher than the reward value of other strategies [39]. If the reward function value is used as the evaluation criterion, expert strategy is always the best strategy. After learning this reward function, the agent needs to use this reward function to re-execute the reinforcement learning algorithm and find the optimal strategy through the reinforcement learning algorithm. Like the behavioral cloning method, the inverse reinforcement learning method still assumes that all the state-action pairs in the corresponding scene can be obtained. [40] adopted the inverse reinforcement learning method, assuming that both the agent and the expert perform tasks at exactly the same speed. At this time, the agent can fully synchronize with the expert to track the trajectory, and the reward function sought by the inverse reinforcement learning is defined as the similarity between the action performed by the agent and that by the expert in each step, and then a reasonable reward value is obtained. Although this method can complete the task, the inverse reinforcement learning method still encounters the difficulty of obtaining the reward function. In the reinforcement learning process after obtaining the reward function, the system needs to solve the Bellman equation. The solution process of the equation will become impossible to solve or the solution is too complex due to the increase in the dimension of the state action, and the agent is also very complicated [10]. It may encounter the problem that the state action space is too large.
Generative Adversarial Imitation Learning

IRL is usually not efficient, because even if the cost function is learned well, the RL method is still needed to train the strategy. GAIL can learn strategies directly from expert data [10]. GAIL describes a policy through the reinforcement learning process of cost function. Such a cost function is learned by the maximum causal entropy IRL learning framework. This description process gives a learning framework for directly learning policy from data without going through other intermediate steps. The core of GAIL is that although it uses the idea of adversarial, there is no explicit generator in it, and the agent’s policy is acting as the generator [10]. GAIL is roughly divided into two steps, in which the first step is to train the Discriminator through adversarial training between the data sampled by the current policy and the expert data; then, the Discriminator is used as a surrogate reward function to train the policy.

However, GAIL is suitable for motion planning, but not for the sequence generation, nor the path planning, as mentioned in the last chapter. Sequence Generative Adversarial Nets (SeqGAN) should be an appropriate approach to perform the sequential decision making task, more specifically, path planning in this thesis, given that sequential decision making and sequence generation can be combined with the imitation learning concept [13][14].

2.6 Sequence Generative Adversarial Nets

As a new training method of generative model, GAN adopts the adversarial concept to guide the training of Generator through Discriminator, and has achieved good results in generating data. Nevertheless, when the target is a non-continuous sequence to be generated, this method will show its limitations [41]. For non-continuous sequence generation, such as text generation, there are two main difficulties to use traditional GAN:

1. In traditional GAN, Generator starts with random sampling, and then per-
forms deterministic conversion according to the parameters of the model. Through the output of the generator, the loss value calculated by the discriminator, according to the loss gradient obtained to guide the generator to make slight changes, so that generator can produce more realistic data. In text generation tasks, generator usually uses Long Short-Term Memory (LSTM), so what generator passes to discriminator is a sequence of discrete values. After the output of each LSTM unit passes through softmax, a specific word is obtained based on probability sampling, which makes it difficult to deal with gradient descent [41].

(2) The traditional GAN can only evaluate the score/loss of the entire generated sequence, and cannot be refined to evaluate the quality of the intermediate generated token and the impact on subsequent generation [41].

Reinforcement learning can solve the above two points well. Policy Gradient uses reward as feedback to increase the probability of the action with a large reward and reduce the probability of the action with a small reward. If we have a reward, we can perform training and update the weights. The Policy Gradient algorithm can be adopted when generator generates a word. During training, given a feedback reward, we can update the weights of generator through this reward, and no longer need to rely on the backpropagation to update the weights. So in this case, the first difficulty is solved. For the second difficulty, when a word is generated, the Monte Carlo tree search can be used (Alpho Go also uses this method) to immediately evaluate the quality of the current word, instead of waiting until the end of the entire sequence to evaluate the word good or bad [41].

Therefore, the combination of reinforcement learning and adversarial ideas can theoretically solve the problem of non-continuous sequence generation, and the SeqGAN model is a model that combines the above ideas and can be used for sequence generation. In the following, I take the word generation as an example to introduce the concept of SeqGAN.
2.6.1 Generator of SeqGAN

Generator generally chooses a Recurrent Neural Network (RNN) structure, LSTM or Gate Recurrent Unit (GRU) can be used. For the input sequence, we first get the embedding of the words in the sequence, and then input it into each cell, and combine with a full connected hidden layers to get the probability of outputting each word [41]. With this probability, the Generator can sample a batch according to it.

The sequence generated by the Generator is obtained by probability sampling, rather than a fixed value obtained by argmax for each output. This is consistent with the idea of policy gradient.

In each cell, we can get a probability distribution. We select an action or a word based on it. For determining whether the word obtained based on this probability distribution is bad or good, a reward is needed to determine the probability of this word being selected. Getting this reward requires Discriminator and Monte Carlo tree search methods. As mentioned earlier, the calculation basis of the reward is that, as much as possible to let Discriminator think that the data generated by the Generator is real-world data. Here we set the label of the real data to 1, and the label of the data generated by the Generator to 0.

If the current cell is the last cell, that is, we have obtained a complete sequence, then pass this sequence to Discriminator, and calculate the probability that the output is 1 to get the reward value. If the current cell is not the last cell, that is, the current word is not the last word, and we have not yet got a complete sequence. In this case, to estimate the reward of the current word, we can use the Monte Carlo Tree Search method, which uses the previously generated sequence, and starts sampling from the next position of the current position to get a bunch of complete sequences. The sampling strategy is called roll-out policy [41], and this strategy is also implemented through a neural network, which we can think of as the Generator. After getting the sampled sequence, the bunch of sequences is input to Discriminator to get a batch
of probabilities that the output is 1, and the average of this bunch of probabilities is the reward.

With the reward, the Generator can continue to be trained, through the Policy Gradient method, by the idea that increase the selection probability of actions with large rewards and reduce the selection probability of actions with small rewards.

### 2.6.2 Discriminator of SeqGAN

The Discriminator model is a classifier. There are many classifiers for classification. The original paper uses a Convolutional Neural Network (CNN). At the same time, in order to make the classification effect of the model better, a highway network is added on the basis of CNN [41]. For Discriminator, since it is a classifier and the output is the probability value of the two categories, the log loss function similar to logistic can be used to train Discriminator.

We noticed that [42] [43] [44] are exploring the potential of GAN in robot path planning too. But there are some differences in training strategies and experimental settings between them and this thesis research. And the most significant difference is that we adopt the sequence generation in RL and Monte Carlo Tree Search techniques.

### 2.7 Use the RRT* with Pre-trained Weights to Acquire Training Data

Since the constructed neural network requires a large amount of trajectory examples, we follow the idea as shown in [45], in which RRT is used to generate training data, and then train LSTM network. We have the similar idea in this research, that is, we use RRT* with pre-trained weights to get demonstration path trajectories, and then use the demonstration data to train our GAN-based neural network.

As we mentioned in the previous section, the basic RRT is an algorithm to quickly search non-convex high-dimensional spaces by randomly constructing Space Filling
Rapidly-exploring Random Trees* (RRT*) is a technique for optimal path planning, which is a variant of RRT algorithm. Each point $p$ has an associated cost and the RRT* algorithm aims to get the trajectory $P^*$ that has the lowest total cost of the path $C(P)$. For this purpose, it generates random samples in the configuration space; in the meanwhile, a tree towards the goal point is established. Then there will be a path from the start point to the goal, represented by a discrete points sequence $P = \{p_1, p_2, \cdots, p_T\}$. As we need to use the RRT* with pre-trained weights, we follow the optimized RRT* in [46] for generating demonstration trajectories.

In the pre-trained RRT*, each point $p$ is associated with a cost function, formed by a set of features $f_j(p)$ and the corresponding weights $\omega_j$, so the cost function of each point $p$ is as shown in the following:

$$c(p) = \sum_{j=1}^{J} \omega_j f_j(p) = \omega^T f(p)$$  \hspace{1cm} (2.1)

where $f(p) = [f_1(p), f_2(p), \cdots, f_J(p)]^T$ is based on $J$ feature functions that describe the task to be performed. The total cost of a generated path is the cumulative value of all of points cost in the path sequence.
Figure 2.1: Features used in the optimized RRT* path planning method. \( d_1 \) indicates the distance between the robot and the goal destination; \( d_2 \) indicates the distance between the robot and the moving obstacles or human, while \( \alpha \) is the angle between the moving obstacles front and the current point; \( d_3 \) represents the distance between the robot and the closest static obstacle.

\[
C(P) = \sum_{i=1}^{T-1} \frac{c(p_i) + c(p_{i+1})}{2} \|p_{i+1} - p_i\| \\
= \omega^T \sum_{i=1}^{N-1} \frac{f(p_i) + f(p_{i+1})}{2} \|p_{i+1} - p_i\| \\
= \omega^T F(P)
\]

where \( F(P) \) represents the feature count of path \( P \). Therefore, given the trained weights \( \omega \), the RRT* will get a path that has the lowest total cost.

### 2.7.2 Features in Cost Function

As mentioned above, a concept of feature is introduced. It is because for the path planning in dynamic environment, there are different elements except the goal point need to be considered, including the moving obstacles or persons stand in different positions and the robot has to avoid them to reach the goal.

In this case, several features have been considered, referring to [46]. As shown in Figure 2.1, \( d_1 \) represents the distance between current point and the goal point; \( d_2 \) represents the distance between the current point and the moving obstacles or
people, while $\alpha$ is the angle between the moving obstacles front and the current point; $d_3$ represents the distance between the current point and the closest static obstacle, such as the wall. And then these features are considered to get the cost function in the RRT* algorithm. We discuss more details of the feature functions in the following.

The first feature function is the Euclidean distance calculation from the robot position to the goal.

$$f_1 (p) = \|p, p_{\text{goal}}\| \quad (2.2)$$

The second feature function calculates the cost related to moving obstacles. It is defined by a mixture of Gaussian functions referring to [46].

$$f_2 (p) = \prod_{k=1}^{H} (g(d_j, \alpha_j) + 1) - 1 \quad (2.3)$$

where $H$ represents the number of moving obstacles or humans. The $g(\cdot)$ is used to calculate the cost between the current position and the moving obstacles, and it depends on the distance ($d_j$) and relative angle ($\alpha_j$).

The third feature function calculates the distance between the nodes $p$ and closest obstacle.

$$f_3 (p) = \frac{a_1}{\gamma * (\|p, p_{\text{closest,obs}}\| + a_2)} \quad (2.4)$$

where $a_1$, $a_2$, and $\gamma$ are a hyper-parameters and we take $a_1 = 2$, $a_2 = 0.2$ and $\gamma = 10$ separately, referring to [46].

Finally, the cost function for each point is composed by these feature functions with corresponding weights $c (p) = \sum_{i=1}^{3} \omega_i f_i (p)$, where the value of weight $\omega_i$ ranges from 0 to 1, and the sum of total value of weights $\omega_i$ is 1.
Chapter 3

Proposed Planning Network

In this chapter, we discuss the details of the proposed model for path planning, and show how the path points sequence is generated by this model.

3.1 Overview of the Idea

In GAN, the Generator starts with random sampling, and then performs deterministic transformations according to the parameters of the model. Through the output of the generative model $G$, the discriminative model $D$ calculates the corresponding loss value, and according to the obtained loss gradient, $G$ is optimized and improved, so that $G$ produces samples closer to the real data. In sequence generation tasks, we use LSTM as the generative model. As we discussed in the last Chapter, $G$ passes to $D$ a sequence of discrete values, that is, the specific path points sequence generated by each LSTM unit after normalization, and after probability sampling, which makes gradient descent difficult to handle. In addition, GAN can only evaluate the quality or loss of the entire generated sequence, and cannot be refined to evaluate the quality of the current generated sample and the impact on the subsequent generation. To deal with these problems, we adopt the idea of policy gradient, which is to use the reward value as a feedback, then we can update the parameters through gradient training. In addition, when generating a path point, the Monte Carlo Tree Search (MCTS) can be used to immediately assess the quality of the current action, instead of waiting until the end of the entire sequence to evaluate it. Therefore, for our path
sequence generation problem, similar with the scenario of SeqGAN, the combination of Reinforcement Learning and GAN ideas can theoretically effectively solve the above limitations and solve the problem of non-continuous sequence generation. Besides, we add the Feature Count Errors reduction as a forcing objective, and the refinement measure is applied. More details and analysis of the proposed model are shown in the following sections.

### 3.2 Problem Setting

Our goal is to develop a method that can generate the path points sequence based on the environment information $E$, start point $p_s$, and goal point $p_g$. The path points sequence is represented by $P_{1:T}$. The current tokens sequence is $P_{1:t−1}$ at time $t$, while we need to determine the next point $p_t$ during generation process. The direct aim is to learn the parameters of a model in order to generate a feasible path sequence from $p_s$ to $p_g$. We train the weights of neural network using policy gradient as in the following discussion.

### 3.3 Neural Network Architecture

Based on the environment information and the current path points sequence, Generator models the position sequence, and the entire intermediate position sequences are constructed by MCTS, which will be input to the Discriminator to get the probability of being from real data, then we get a reward to choose the next path point. In this case, the neural network is expected to retain the information implicit in the path and fully explore the hidden characteristics of the trajectory. The overview of the SeqGAIL architectures are shown in Figure 3.1 and Figure 3.2, while the Figure 3.1 is a pre-training process for Discriminator.

We choose LSTM as the Generator as RNN has shown the effectiveness and high ability for robot path generation [47], and LSTM is a variant of RNN with the ability
Figure 3.1: The process of training the discriminative model. The discriminator is fed with two kinds of trajectory data, where one is the “real” demonstration data from the pre-trained RRT* algorithm, while another is the “fake” path sequences generated by the initial generator. The blue circles indicates the condition information, e.g. environment information and start-goal points pair. The green and yellow circles indicates the generated path sequences by RRT* and generator accordingly.

Figure 3.2: Overview of the SeqGAIL architecture. When generator generates the path sequence, the MCTS method is applied, and the refine method is used to filter the invalid path sequences which are not met the constraints. The discriminator with feature count objective calculates the probability value that the generated sequence belongs to the “real” path.
to filter information and remember conditional information. Since the many current algorithms for path planning rely on the external features without considering the use of the characteristics of trajectory itself for in-depth mining, we use the natural memory characteristics of LSTM to retain the most important information in the path point sequence. For Discriminator, we choose another LSTM network for performing classification.

Noted that the standard SeqGAN is not suitable for directly being applied on the path planning problem, so in this research, we incorporate some useful ideas of SeqGAN in sequence generation and designed a variant based on it. There are three major significant changes compared with the standard SeqGAN. One of which is that the state includes not only the current produced tokens, but also the environment information. The second change is that we add a constraint on the candidate tokens while generating intermediate sequences by MCTS, that is the points candidates are restricted in a range that the current point can reach in an appropriate distance. The third one is that after generating the entire intermediate sequence by MCTS, we use a refine method to filter the invalid sequences that did not meet the requirements of a path, such as the steering and distance between the last generated path point and the goal point should be in an appropriate range. For an invalid sequence, we give it a penalty value as the reward.

In addition, the idea of using conditional GAN to do the path planning in [48] is very similar to ours. We carried out our design work on their basis, but our work is different from them in that: we use Monte Carlo to expand the node; we add a feature count objective; and we design and implement experiments in the ROS environment.

3.4 Feature Count Objective

We apply the Feature Counts Errors reduction as the specific objective for the Generator, which is a key evaluation metric for the generated path as shown in [49] [46]
[19], and the calculation method is shown in Chapter 2. Given the generated path points sequence $P_g$ and the ground true sequence $P_d$, a reward $Q(P_g, P_d)$ based on the Feature Count Objective reduction will be got, whose value will range from 0 to 1.

### 3.5 Policy Gradient

Before introducing the policy gradient method used here, I would like to start with the related concepts of Reinforcement Learning. Reinforcement Learning is a branch of deep learning that emphasizes how individuals act based on the environment to maximize the expected reward.

The four elements of reinforcement learning: state, action, policy and reward. Given states $S$ and possible behaviors $A$, the states are mapped to the best behavior through reinforcement learning to maximize the reward. This is a closed-loop system, where behavior affects the result, and the result will provide a reward for making the next choice. There is no need for external supervision in this closed-loop system, that is, the choice of behavior can be completely derived from the reward from the result. As shown in Figure 3.3, the agent interacts with the environment, and the agent constantly gains experience in trial and error like a human, to make the best choice. In such a learning process, there is no need to directly tell the agent what to do in a specific environment.

#### Policy

The policy is the agent’s behavior, which is the mapping from state to action $\pi: s \rightarrow a$. There are two kinds of policy, deterministic policy $a = \pi(s)$ and stochastic policy $\pi(a | s) = P[A_t = a | S_t = s]$ [50]. In our path planning task, policy is consistent with LSTM cell calculation. State is the environment info and generated sequence tokens $(E, p_1, \ldots, p_{t-1})$, action is the process of generating $p_t$. So the policy in our
Figure 3.3: The process of Reinforcement Learning. The agent performs an action, and the environment changes the state and provides a reward to the agent, then the agent acts further according to this feedback.

problem is:

Stochastic policy: \( \pi(a \mid s) = P(A_t = p_t \mid S_t = (E, p_1, p_2, \ldots, p_{t-1})) \)

That is, given the state \((E, P_{1:t-1})\), the probability distribution of the path point \(p_t\) will be generated. This is actually the basic function of a cell of LSTM.

**Action-Value Function**

The value function in RL is a prediction of future reward, which is used to evaluate states, and then choose the appropriate action, as shown in the following [50]:

\[
v_\pi(s) = \mathbb{E}_\pi \left[ R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s \right]
\]

That is, in a certain state \(s_t = s\), \(v_\pi(s)\) represents the expectation of the sum of all future rewards under the policy \(\pi\), where \(\gamma\) is the discount coefficient.

Besides, we need to define a return value \(G_t\), which is the total discounted reward from time-step \(t\).
\[ G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \]

The discount \( \gamma \in [0, 1] \) is the present value of future rewards. The value of receiving reward \( R \) after \( k + 1 \) time-steps is \( \gamma^k R \) [51].

So similarly, in our path planning problem, we use the action-value function \( q_\pi(s, a) \), which is the expected reward for taking actions \( a_t \) (in our setting, they are points \( p_t \)) in states \( s_t \), when following policy \( \pi \)

\[ q_\pi(s, a) = \mathbb{E}_\pi [G_t \mid S_t = s, A_t = a] \]

Without the feature count objective, the optimization goal at initial state is:

\[ J(\theta) = [R_T \mid s_0, \theta] = \sum_{p_1 \in P} G_\theta(p_1 \mid s_0) \cdot R^{G_\theta}_{D_\phi}(s_0, p_1) \quad (3.1) \]

which is the value function when the initial state \( s_0 \) and the parameter \( \theta \) are given. The initial state \( s_0 \) can be seen as the state when the above LSTM RNN only has environment information including goal point and start point. \( G_\theta(p_1 \mid s_0) \) is the probability of generating output \( p_1 \), at the initial state \( s_0 \). The term \( R^{G_\theta}_{D_\phi}(s_0, p_1) \) in formula 3.1 is the sum of the subsequent rewards of the point \( p_1 \) in the state of \( s_0 \), which is the action-value function \( q_\pi(a \mid s) \). The optimization goal is to enlarge probability expectations that the generated path points sequence through the generator is considered by the discriminator to be a real path sequence. In which the parameter \( \theta \) determines the policy of Generator.

Introduce the Feature Counts Errors reduction objective, and the final objective function is:

\[ J(\theta) = \sum_{P_{1:T}} G_\theta(P_{1:T} \mid E) \cdot R^{G_\theta}_{D_\phi}(P_{1:T-1}, E, p_T, P_d) \quad (3.2) \]
where $R_{D,Q}^G$ is the reward from Discriminator with $Q$ objective forcing, $E$ represents the environment information, $P$ is the generated path points sequence by $G$ with parameter $\theta$, and $P_d$ is the ground true path sequence used to calculate the $Q$ reward.

In our optimization objective 3.2, the optimization goal $J(\theta)$ is actually the same as the policy objective functions in reinforcement learning, and we can use the policy gradient method.

**Policy Gradient**

Policy gradient algorithms [51] search for a local maximum in $J(\theta)$ by ascending the gradient of the policy with parameters $\theta$

$$\Delta \theta = \alpha \nabla_\theta J(\theta)$$

where $\nabla_\theta J(\theta)$ is the policy gradient, and $\alpha$ is a step-size parameter.

The Generator in our model is a Markov Decision Process (MDP), and in one-step MDP, the likelihood ratios is used to compute the policy gradient.

$$J(\theta) = E_{\pi_\theta}[r]$$

$$= \sum_{s \in S} d(s) \sum_{p \in P} \pi_\theta(s, p) R_{s,p}$$

$$\nabla_\theta J(\theta) = \sum_{s \in S} d(s) \sum_{p \in P} \pi_\theta(s, p) \nabla_\theta \log \pi_\theta(s, p) R_{s,p}$$

$$= E_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, p) r]$$

(3.3)

To generalize the likelihood ratio approach to multi-step MDPs, we replace the instantaneous reward $r$ with long-term value $R^G(s, p)$, and the policy gradient is:

$$\nabla_\theta J(\theta) = E_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, p) R^G_\theta(s, p)]$$

(3.4)
It can be seen here that for our optimization goal \( J(\theta) \), the gradient can be calculated, since the strategy is differentiable, and there is no need to calculate the gradient of the action-value function \( R(\cdot) \), only the value is required.

### 3.6 Use Monte Carlo Simulation to Estimate the Action-Value Function

For a complete path points sequence that has been generated, its action-value function can be obtained by:

\[
R_{D,Q}^G(P_{1:T-1}, E, p_T, P_g) = \lambda(D(E, P_{1:T}) - b(E, P_{1:T})) + (1 - \lambda)Q(P_{1:T}, P_g)
\]

(3.5)

where \( b(\cdot) \) is the baseline value to reduce the variance of the reward, and the value of \( b(\cdot) \) is a constant 0.5, which is referred to the idea in [15].

After the state \( s = (E, P_{1:T-1}) \), the last point \( p_T \) is generated, and the complete path \( P_{1:T} \) is obtained. The complete path can be judged by the Discriminator to be a real path \( D\phi(E, P_{1:T}) \).

But if it is incomplete path, the Discriminator cannot process it. In this case, we can use Monte Carlo simulation to estimate \( R \), by simulating the results of \( N \) times and take the average of the final reward for \( N \) times (here \( N \) is a hyperparameter).

\[
\{P_{1:T}^1, \ldots, P_{1:T}^N\} = MC^G_\theta(P_{1:t}; N)
\]

(3.6)

\[
R_{D,Q}^G(P_{1:t-1}, E, p_T, P_d) = \\
\begin{cases} \\
\frac{1}{N}\sum_{n=1}^{N} \lambda(D(E, P_{1:T}^n) - b(E, P_{1:T}^n)) + (1 - \lambda)Q(P_{1:T}, P_d) & t < T \\
\lambda(D(E, P_{1:t}) - b(E, P_{1:t})) + (1 - \lambda)Q(P_{1:t}, P_d) & t = T
\end{cases}
\]

(3.7)
Use formula 3.8, we can get the gradient, and use formula 3.9 to update the Generator’s parameters by gradient ascending.

\[
\nabla J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{p_t} R_{D,Q}^{G_\theta} (P_{1:t-1}, E, p_T, P_d) \cdot \nabla \theta (G_\theta (p_t \mid P_{1:t-1}, E)) \\
= \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}_{p_t \in G_\theta} \left[ R_{D,Q}^{G_\theta} (P_{1:t-1}, E, p_T, P_d) \cdot \nabla \theta \log G_\theta (p_t \mid P_{1:t-1}, E) \right] \tag{3.8}
\]

\[
\theta \leftarrow \theta + \alpha_h \nabla J(\theta) \tag{3.9}
\]

The Generator is incrementally updated using the reward \( R_{D,Q}^{G_\theta} \) to generate more feasible path points sequences, and after g-steps, we can re-train discriminator further.

### 3.7 Generator

The Generator is a LSTM RNN network. It defines the policy that generates the path sequence \( P \) given the input state which is a flattened vector including the environment information and the current path points. In details, the LSTM network maps the input embedding representations into a hidden states sequence by using the update function recursively [41].

The overall view of the Generator is as shown in Figure 3.4, where \( h_t \) represents the hidden state. The output is represented by a softmax transformation from the hidden states, in order to get the path points distribution.

### 3.8 Discriminator

The Discriminator is another LSTM model used to distinguishes a path sequence that is from either the real trajectories or the planned fake sequences. The non-linear sigmoid is applied on the final output layer, then we get the probability that the input path belongs to the real trajectories. The whole discriminative model has two hidden
Figure 3.4: The generator is fed with environment information and $p_{t-1}$-$p_g$ pair, which is the current point and goal point pair, to generate the next point $p_t$. The $h_t$ indicates the hidden states in the hidden layers. There are two hidden layers and one softmax layer to calculate the path points distribution.

Following [52], we adopt the weight clamping to have parameters $\omega$ of the Discriminator lies in a compact space for the optimization reason, by clamping the weights to a fixed box $([-\epsilon, \epsilon])$ after each gradient update (here we set $\epsilon$ as 1.0).

The complete process of the SeqGAIL algorithm is shown in Algorithm 1. In order to better update the policy network, the discriminative model should be trained first during the update process, and the Generator can be updated under the condition that the discriminative model is optimized.

Noted that the problem we are solving is mainly the GAN module, and our novelty is on this part. So in this thesis, we did not explain too much about the feature extractor module, as we focus on the GAN module, and we directly leveraged other’s current feature extractor module [53] to implement the method.
Algorithm 1 Sequence Generative Adversarial Imitation Learning

**Require:** Environment information examples in flattened vector presentation extracted by feature extractor module \( E = \{E_1, \ldots, E_D\} \) in \( D \) scenarios, start point \( p_s \), goal point \( p_g \), generator policy \( G_\theta \), discriminator \( D_\omega \)

Initialize the Generator \( G_\theta \), and the Discriminator \( D_\omega \). Initialize the corresponding network parameters \( \theta_0 \) and \( \omega_0 \)

Through the RRT* with pre-trained weights, get the demonstration trajectory examples \( P_d = \{\hat{p}_1, \hat{p}_2, \cdots, \hat{p}_T\} \)

Use the maximum likelihood estimation to pre-train \( G_\theta \) on the demonstration trajectory examples until the best generation is achieved.

Generate negative samples using \( G_\theta \) to pre-train \( D_\omega \) via minimizing the cross entropy

```
for episode = 1 to M do
    for g-steps do
        The Generator executes the policy based on the parameter \( \theta_0 (\theta_i) \), and obtains a path points sequence using MCTS, \( \{p_1, p_2, \cdots, p_T\} \). In this step, the refine method is used to check if it meets the requirements between the last generated point and the goal point \( p_g \).
        for \( t = 1 \) to \( T \) do
            Compute \( R_{D,Q}^{G_\theta} \) by Equation 3.5
        end for
        Update \( G_\theta \) via Policy Gradient by Equations 3.8 3.9
    end for
    for d-steps do
        Use current \( G_\theta \) to generate negative examples and combine with given positive examples \( P_d \) to re-train Discriminator \( D_\omega \) for \( k \) epochs
    end for
    Check whether the number of training rounds reaches the set value, if it exceeds, it will end
    Check whether the SeqGAIL network converges, and if it converges, end the process
end for
```
Chapter 4

Experimental Work

4.1 The ROS Environment and Information Pre-process

Robot Operating System (ROS) [54] is an open source robot framework, which belongs to the secondary operating system. It provides operating system-like services, including hardware abstraction, underlying device control management, inter-process message transfer, and package management. It also provides some tools and library functions required for compiling, for obtaining, writing information and multi-machine integration execution. Code reuse and modularization are the design goals of ROS, so ROS uses a distributed process framework, and each process is called a node. A robot control system usually contains many nodes, and these nodes can run on different machines at the same time, and all nodes are encapsulated in a package or stack. The function package is the main form of organization software in ROS. It contains ROS dependent libraries, data sets, configuration files, and other files. There is a loose block and point-to-point communication structure between nodes. ROS mainly has several different communication methods, including synchronous Remote Procedure Call (RPC) communication mode based on service mechanism, topic publish-subscribe communication mode based on asynchronous streaming media data, and parameter server communication mode. The topic publish-subscribe communication mode is a many-to-many communication mode; the synchronous RPC communication mode is used to provide query-reply structure interaction, which is different from the topic publish-subscribe communication mode. All nodes are managed by the master
node. The master node is mainly used to save registration information and lookup tables for node topics and services. Registered nodes can find the node that matches their registration information through information query, and then communicate with their matched node through the master node [54].

We follow the method and framework [45] [53] to convert the environmental information into a flattened vector. In this step, we use the existing feature extraction module framework [55], and ROS point cloud feature extraction module [56] to preprocess the environment information. The pre-trained RRT* is from [57], while the comparison target FCN-RRT* is from [58].

To simulate the robot’s behavior, we also use Rviz to do the experiments. The environment data is from Service Robotics Lab [59]. The hardware employed was an Intel Broadwell processor with 2 vCPU and 8 GB memory.

4.2 Comparison Metrics

To get the analysis of comparison between the planned paths by SeqGAIL and the demonstration paths generated by pre-trained RRT*, we use two kinds of metrics to measure them, referring [49] [46] [19].

- The first one is Distance Metric as shown in Equation 4.1:

\[ \mu(P_1, P_2) = \frac{D(P_1, P_2) + D(P_2, P_1)}{2} \]  
\[ D(P_1, P_2) = \frac{\sum_{i=1}^{T} d(P_1(i), P_2)}{N} \]

in which \(d(\cdot)\) represents the Euclidean distance between the point \(p_i\) of \(P_1\) and its closest point on \(P_2\), and \(T\) is the number of points of \(P_1\).

- The second one is the Feature Counts Errors, which represents the feature count difference between the generated path and the demonstration path. The feature
count $F(P)$ of a path $P$ is defined as formula 4.2, where $f(P(i))$ indicates the value of the feature function for point $p_i$ of path $P$, and $\|P(i+1) - P(i)\|$ is the Euclidean distance between $p_i$ and $p_{i+1}$ of the path $P$.

$$F(P) = \sum_{i=1}^{N-1} \frac{f(P(i)) + f(P(i+1))}{2} \|P(i+1) - P(i)\|$$  \hspace{1cm} (4.2)

Note that although in the pre-trained RRT*, all of the three feature functions mentioned in Chapter 2 are taken into account to generate the training trajectories, but here, in our setting, we only use the first feature function (euclidean distance from the current generated position to the goal point) to calculate the feature counts errors, as in the feature extraction module, the static obstacle and moving obstacles are not specified, and there are no other physical constraints, as the process in [53] [45].

### 4.3 Results and Analysis

We first train the FCN and SeqGAIL approaches with the training set generated by the RRT* with pre-trained weights, and then compare the results using the testing set (totally around 150 scenarios). For a clearly comparison, we divided the testing trajectories into four sets, and perform comparison separately on each of them. For a straightforward view on the comparison, we selected some of the cases in which our method outperforms the FCN method, as shown in Figure 4.1, which is the visual comparison in 8 different independent scenarios. In most cases, our method wins, although in some cases, our method cannot perform well, and the extensive cases of SeqGAIL’s performance are in Appendix A.

I would like to briefly explain some cases about SeqGAIL’s performance, as shown in Figure 4.2. In most cases, the SeqGAIL can perform demonstration behavior, in other words, it can imitate the trajectories in some extent, although it cannot
Figure 4.1: Visual comparison in 8 different independent scenarios (two images for each scenario, one for FCN and another for SeqGAIL). In each scenario, the red path indicates the demonstration trajectory, while the green path indicates the generated path by FCN or SeqGAIL accordingly. The green cylinder with an arrow represents the moving object or human with a moving direction, and the yellow arrow indicates the goal point, while the start point is the center of the scenario image. The pink blocks represent the static obstacles as the scenario image is based on a cost-map.
completely reproduce the demonstrations and maybe fails to generate a feasible paths in some complicated scenarios. But in this thesis, we do not take the visual results as a main metric, and we use the metrics in existing works, the average Distance Metric and average Feature Counts Errors.

By using the average Distance Metric and the average Feature Counts Errors, the results are shown in the Figure 4.3 and Figure 4.4, respectively. The SeqGAIL in most sets outperforms the FCN-RRT* method. But in Set-2, the SeqGAIL does not perform well, and as for the reason, we also investigated it in the following.

Although SeqGAN performs well in the word sequence generation area, but the path planning still is a little bit different to the sentence generation, as the planned path may be very long, while the generated sentence has an appropriate length that approximately is below 20 or 30. In other words, the proposed network works well given that the path sequence is not long. If the length is very long, the results seems not ideal for navigation, as shown in Figure 4.5. Actually this problem also arrives in the standard SeqGAN, so is this case, it brings one of the drawbacks of SeqGAIL in path planning problems, that is the SeqGAIL is not suitable for path planning in a long path with many points. However, this issue can be theoretically resolved by reducing the resolution of the environment images or directly restricting the point cover range of the environment image.

In addition, there are other reasons related to the training data that could explain the SeqGAIL does not perform well in the Set-2, or in the scenarios where the path is too long. That is, the SeqGAIL is not suitable for the complicated environment, where the path is too tortuous or there are too many obstacles. Actually, the environment data we use is monotonous, which is from only one lab, and their environment data contains complicated scenarios (not all, but some), which will influence the performance of SeqGAIL that has not been designed to be very robust yet. Besides, our training data is not expert data, but expert-like data generated by RRT* with
Figure 4.2: Selected 16 different independent cases generated by SeqGAIL. In most cases, the SeqGAIL can perform well, while in some cases the SeqGAIL cannot generate feasible path like the demonstration, e.g. in the 14th scenario.
Figure 4.3: Averaged distance metric comparison results. In Set-1, Set-3, Set-4, SeqGAIL achieves the lower distance difference from the demonstration trajectory, than that of FCN-RRT* method. In Set-2, the SeqGAIL does not outperform the FCN-RRT* method.

Figure 4.4: Averaged feature counts errors comparison results. Similar with the comparison result of averaged distance metric, in Set-1, Set-3, Set-4, SeqGAIL outperforms the FCN-RRT*, while in Set-2 it has higher averaged feature counts errors than that of FCN-RRT*.
Figure 4.5: SeqGAIL performance in different path lengths. When the planned path has less points, the navigation behavior is more like the demonstration, which is measured by the Distance Metric and Feature Counts Errors. Note that there is no strict positive correlation between the Distance Metric and Feature Counts Errors.

pre-trained weights. Although this method alleviates the difficulty in obtaining training data to a certain extent, the quality of training data cannot be guaranteed to be completely feasible. In addition, the feature extraction method in the [53] [45] will also affect our results, because the feature extraction of environmental information is one of the keys. If the feature is not properly extracted, then the results of our path planning approach will not be good.
Chapter 5

Conclusion

This chapter discusses the summary of the thesis, limitations of the proposed model and future following work.

5.1 Summary

In this thesis, we have developed and implemented a GAN-based approach that can imitate the trajectories to do the global path planning in dynamic environments, which is one of the main contributions of this thesis. We adopt the concept of imitation learning to directly learn from demonstration trajectories, without defining the environment and attention models. The approach expands the next path point using Monte Carlo Search Tree, and an objective forcing method is taken into account besides the reward from the Discriminator, which is a novel idea on this problem. We also proposed to use the pre-trained RRT* algorithm to generate the demonstration training data, which alleviate the difficulty of acquiring huge amount of data. Compared with the recently proposed FCN-based path planning model, the SeqGAIL proposed in this thesis can get better results, given the path length is not too long. And the goals mentioned in this thesis are all achieved.

However, the limitation of Monte Carlo tree search in our method is that every time a path point is generated, the sample generation is required to perform $N$ times, which is very time-consuming. And when calculating the reward estimate of the subsequent points sequence, the points generated in the earlier period will be repeatedly
calculated, which leads to the overfitting problem for the earlier subsequence. In addition, the traditional model is generally an optimization function, and there is an optimal solution for convex optimization. For GAN, it is to find the Nash equilibrium point in two players, but it is not certain that the Nash equilibrium point will be found, so it is unstable during training.

Taking these limitations into account, we discussed the possible improvement directions for the reference of future work.

5.2 The Future Work

As we can see in the results as shown in Figure 4.2 and Appendix A, the SeqGAIL needs to be optimized further to be applied on the real application. Here, we have the following considerations for future work reference:

- The paths generated by SeqGAIL has a visually apparent characteristic is that there are some sharp steering on the path, which maybe is because the hyperparameters related to steering control is not optimized enough, although we controlled the last generated point connected to the goal point.

- And if the path lengths are very long, the results are also disappointing, which maybe is because of the nature of SeqGAN. This would need to be optimized further.

- In addition, the metrics of this research needs to be expanded. The subjective point of view maybe can also be a metric in the comparison, as the feasible navigation behavior in crowd is judged by the human in daily life, not a specific measuring metric based on calculation.

- The training data maybe is not sufficiently social compliant, and the environment information is not complete, which will influence the performance of our
proposed model. So in the future, if there are some better training data, the model can be optimized further.
REFERENCES


[42] M. Mohammadi, A. Al-Fuqaha, and J.-S. Oh, “Path planning in support of smart mobility applications using generative adversarial networks,” in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 878–885.


APPENDICES

A  Extensive Cases of SeqGAIL’s Performance

This appendix shows more extensive path planning cases of SeqGAIL. These cases are different and independent with each other. As described in Chapter 4, in each scenario based on the cost-map, the red path is the demonstration path; the green path is the generated path by FCN or SeqGAIL accordingly. The green cylinder indicates the moving object or human, and the arrow on the cylinder indicates the moving direction. The yellow arrow represents the goal point, while the start point is at the center of the scenario image. The pink blocks represent the static obstacles.