Spring 5-22-2021

Neuromorphic Systems For Pattern Recognition And Uav Trajectory Planning

Yilan Li
Syracuse University, yli41@syr.edu

Abstract

Detection and control are two essential components in an intelligent system. This thesis investigates novel techniques in both areas with a focus on the applications of handwritten text recognition and UAV flight control. Recognizing handwritten texts is a challenging task due to many different writing styles and lack of clear boundary between adjacent characters. The difficulty is greatly increased if the detection algorithms is solely based on pattern matching without information of dynamics of handwriting trajectories. Motivated by the aforementioned challenges, this thesis first investigates the pattern recognition problem. We use offline handwritten texts recognition as a case study to explore the performance of a recurrent belief propagation model. We first develop a probabilistic inference network to post process the recognition results of deep Convolutional Neural Network (CNN) (e.g. LeNet) and collect individual characters to form words. The output of the inference network is a set of words and their probability. A series of post processing and improvement techniques are then introduced to further increase the recognition accuracy. We study the performance of proposed model through various comparisons. The results show that it significantly improves the accuracy by correcting deletion, insertion and replacement errors, which are the main sources of invalid candidate words.

Deep Reinforcement Learning (DRL) has widely been applied to control the autonomous systems because it provides solutions for various complex decision-making tasks that previously could not be solved solely with deep learning. To enable autonomous Unmanned Aerial Vehicles (UAV), this thesis presents a two-level trajectory planning framework for UAVs in an indoor environment. A sequence of waypoints is

selected at the higher-level, which leads the UAV from its current position to the destination. At the lower-level, an optimal trajectory is generated analytically between each pair of adjacent waypoints. The goal of trajectory generation is to maintain the stability of the UAV, and the goal of the waypoints planning is to select waypoints with the lowest control thrust throughout the entire trip while avoiding collisions with obstacles. The entire framework is implemented using DRL, which learns the highly complicated and nonlinear interaction between those two levels, and the impact from the environment. Given the pre-planned trajectory, this thesis further presents an actor-critic reinforcement learning framework that realizes continuous trajectory control of the UAV through a set of desired waypoints. We construct a deep neural network and develop reinforcement learning for better trajectory tracking. In addition, Field Programmable Gate Arrays (FPGA) based hardware acceleration is designed for energy efficient real-time control.

If we are to integrate the trajectory planning model onto a UAV system for real-time on-board planning, a key challenge is how to deliver required performance under strict memory and computational constraints. Techniques that compress Deep Neural Network (DNN) models attract our attention because they allow optimized neural network models to be efficiently deployed on platforms with limited energy and storage capacity. However, conventional model compression techniques prune the DNN after it is fully trained, which is very time-consuming especially when the model is trained using DRL. To overcome the limitation, we present an early phase integrated neural network weight compression system for DRL based waypoints planning. By applying pruning at an early phase, the compression of the DRL model can be realized without significant overhead in training. By tightly integrating pruning and retraining at the early phase, we achieve a higher model compression rate, reduce more memory and computing complexity, and improve the success rate compared to the original work.

NEUROMORPHIC SYSTEMS FOR PATTERN RECOGNITION AND UAV TRAJECTORY PLANNING

By

Yilan Li

B.S., Xi'an Jiaotong University, 2012

M.S., Syracuse University, 2015

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Syracuse University

May 2021

Acknowledgements

First of all, I would like to sincerely thank my supervisor, Dr. Qinru Qiu, for providing me with guidance throughout my Ph.D. research. Dr. Qinru Qiu offered me the chance of doctorate study on January 2015 when I decided to move step forward in the academic. She kept me constantly engaged with my research ideas and provided me with refreshing insight, critical questions and all-round support. I have been very fortunate with her all the way through the completion of my Ph.D. My sincere gratitude also goes to Dr. Jay K Lee, who gave me a lot of learning guidance when I first came to the United States, and encouraged me to continue academic research.

I would like also to show my appreciation to the members of my exam committee, Dr. C.Y. Roger Chen, Dr. Mustafa Cenk Gursory, Dr. Sucheta Soundarajan, Dr. Fanxin Kong and Dr. Amit K Sanyal for their valuable comments. My appreciation also extends to my collaborators, Dr. Hossein Eslamiat and Hongjia Li. In addition, a thank you to all my labmates: Dr. Qiuwen Chen, Dr. Khadeer Ahmed, Dr. Zhe Li, Dr. Ziyi Zhao, Dr. Krittaphat Pugdeethosapol, Dr. Amar Shrestha, Haowen Fang, Zhao Jin, Mingyang Li, Chen Luo, Daniel Patrick Rider, Zaidao Mei and Yue Ma. There is no way to express how much it meant to me to have been a member of Advanced Microprocessor and Power-aware Systems (AMPS) lab.

Finally, I would like to sincerely express my everlasting appreciation to my family for their enduring support, unconditional trust and timely encouragement during my doctoral work. This journey would not have been possible without them, and this thesis is dedicated to my husband Yantao Lu, my baby boy Eiden Wenxin Lu, and my parents, Junli Li and Quanli Li.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ADAM Adaptive Moment Estimation

ADMM Alternating Direction Method of Multipliers

AMPS Advanced Microprocessor and Power-aware Systems

BVLOS Beyond Visual Line-of-Sight

CNN Convolutional Neural Network

DNN Deep Neural Network

DQN Deep Q Network

DRL Deep Reinforcement Learning

FPGA Field Programmable Gate Arrays

GA Genetic Algorithm

GNSS Global Navigation Satellite Systems

HMM Hidden Markov Models

HoGs Histograms of Oriented Gradients

IoT Internet of Things

LGVI Lie Group Variational Integrator

RC Reactive Control

ReLU Rectified Linear Units

RL Reinforcement Learning

RNN Recurrent Neural Network

SDNN Space Displacement Neural Network

SoC System-on-Chip

TP Trajectory Planning

UAV Unmanned Aerial Vehicles

Chapter 1

Introduction

Neuromorphic systems refer to emerging neural network-based models and computing systems. This evolving field offers exciting possibilities, including many emerging cognitive applications. Two major applications of neuromorphic systems are detection and control. The former refers to the recognition of familiar patterns from noisy input and the later refers to the selection of system actions based on the environment status to maximize the total reward. Both tasks are corner stones of an intelligent system. How to improve their accuracy and reduce the cost have been the focus of many research works. In this thesis, we will study two neuromorphic systems, handwriting text recognition and UAV flight control. Techniques for accuracy enhancement and complexity reduction will be discussed and evaluated.

In this chapter, we first discuss the motivation of the study. Then we introduce background of some learning models and complexity reduction techniques that will be used in this work. Finally, the contributions of the thesis are summarized.

1.1 Motivation

There has been a significant amount of work on neuromorphic systems for different tasks. Recognizing handwritten text is one of them. It is surprisingly challenging

because of many reasons. Firstly, handwritten words are largely variable because everyone has their own unique writing style. In the past, many techniques, such as Hidden Markov Models (HMM) and deep learning networks, have been applied to this problem. Unfortunately, the performance of these previous approaches was restricted due to the lack of the ability to automatically extract high-level features and also because of the expensive training cost. Furthermore, adjacent characters sometimes connect or overlap. If there is no clear boundary, two characters close to each other may be recognized as one character, or one character may be split into two characters. This greatly increases the difficulty for algorithms that detect only based on pattern matching. In the first part of this thesis, we focus on investigating the learning model that improves the accuracy of handwritten text recognition and overcomes the shortcomings of previous works. This is achieved by using a recurrent belief propagation network that is introduced in Chapter 2.

The autonomous systems have attracted much attention recently. Among them, the UAV technology is one of the rapidly growing fields with tremendous opportunities for research and applications. The biggest challenge to safely integrating UAV into the national airspace – with the ability to fly beyond the line of sight of the operator – is developing a system that enables UAV to "sense and avoid" other stationary or moving objects. The system requires the UAV to achieve real autonomy in real time without remote controls, external navigation aids (such as Global Navigation Satellite Systems (GNSS)) and radar systems. In the second part of this thesis, we focus on the autonomous flight control of UAVs. Small UAVs are considered in our work not only because of their wide availability and the most civilian usage, but also because they impose more stringent constraints on the size and energy dissipation of the onboard computing platform. We present two neuromorphic systems for energy efficient UAV trajectory planning and control respectively. The construction, training and evaluation of those systems are described in Chapter 3 and Chapter 4.

Conventional high-performance computing systems are too bulky for small sized UAVs and their power consumption is too high for an on-board computing. There is a gap between the computing capabilities of on-board embedded systems and the computation demand for real-time planning and control of the autonomous UAV. There have been efforts in closing this gap from both sides. On one hand, faster embedded processors with small footage and low energy consumption, such as NVIDIA Jetson TX2 and NX, have been used for cyber physical applications on UAVs [3][4]. On the other hand, more efficient computing models have been investigated by eliminating unnecessary weights to compress the model size and to reduce inference energy consumption of the over-parametrized networks without or with negligible accuracy loss [5][6]. Small networks with good performance reduce energy costs, computation complexity, storage requirements, and inference latency, all of which facilitates the deployment on small UAVs. Furthermore, network compression improves model generalization by regularizing over-parametrized functions. Therefore, compressing over-parameterized neural networks is an important step towards successful training and real time on-board processing [7]. However, almost all of previous works on DNN weight pruning solely focused on supervised image classification, leaving it unclear if the compression techniques are applicable to DRL. Furthermore, most of them apply pruning on fully trained model and require iterative retrain of the pruned model. Due to the error and trial nature of Reinforcement Learning (RL), the training of a DRL model converges slower than conventional DNN. And the three-step process of model compression, i.e. training, pruning, and re-training, exacerbates such disadvantage. In Chapter 5 of this thesis, we address this problem by presenting an early phase integrated neural network compression system for DRL models.

1.2 Cogent Confabulation

Our handwriting text recognition system is built on top of the cogent confabulation model. The cogent confabulation theory [8] mimics the Hebbian learning, the information storage and inter-relation of symbolic concepts, and the recall operations of the brain. Based on the theory, the cognitive information process consists of two steps: learning and recall. During the learning, the knowledge links are established and strengthened as symbols are co-activated. During recall, a neuron receives excitations from other activated neurons. A "winner-takes-all" strategy takes place within each lexicon. Only the neurons (in a lexicon) that represent the winning symbol will be activated and the winner neurons will activate other neurons through knowledge links. At the same time, those neurons that did not win in this procedure will be suppressed.

A computational model for cogent confabulation is proposed in [8]. Based on this model, a lexicon is a collection of symbols. A *knowledge link* (KL) from lexicon A to B is a matrix with the row representing a source symbol in A and the column representing a target symbol in B. The ij_{th} entry of the matrix represents the strength of the synapse between the source symbol s_i and the target symbol t_j. It is quantified as the conditional probability $p(s_i|t_j)$. The collection of all knowledge links is called a *knowledge base* (KB). The knowledge bases are obtained during the learning procedure. During recall, the excitation level of all symbols in each lexicon is evaluated. Let l denote a lexicon, F_l denote the set of lexicons that have knowledge links going into lexicon l, and S_l denote the set of symbols that belong to lexicon l. The excitation level of a symbol t in lexicon l can be calculated as:

$$I(t) = \sum_{k \in F_l} \sum_{s \in S_k} I(s)[ln(\frac{p(s|t)}{p_0}) + B], t \in S_l. \tag{1.1}$$

The function I(s) is the excitation level of the source symbol s. Due to the "winner-

takes-all" policy, the value of I(s) is either "1" or "0". The parameter p_0 is the smallest meaningful value of $P(s_i|t_j)$. The parameter B is a positive global constant called the *bandgap*. The purpose of introducing B in the function is to ensure that a symbol receiving N active knowledge links will always have a higher excitation level than a symbol receiving $(N-1)$ active knowledge links, regardless of the strength of the knowledge links.

1.3 Deep Reinforcement Learning

Reinforcement learning is a goal-oriented algorithm, which is well-suited for decision-making where supervised learning or unsupervised learning alone can't do the job. It solves the difficult problem of correlating immediate actions with the delayed returns they produce. In RL, an agent interacts with its environment and receives reward based on its behavior. This enables the agent to adapt its policy so as to increase the potential of reward in the future. Reinforcement learning can be empowered by DNN directly, and have been applied to applications with high dimensional states like Atari video games, Dota-2, as well as the Go game [9]. In DRL, DNNs are used to approximate the value function $V(s;\boldsymbol{\theta})$ or $Q(s,a;\boldsymbol{\theta})$ [10]. $\boldsymbol{\theta}$ represents parameters of the DNN model. A state value $V(s;\boldsymbol{\theta})$ is the expected return by following policy π with initial state s, where the policy π is a mapping from states to actions. The action value $Q(s,a;\boldsymbol{\theta})$ is the expected return for selecting an action a in a state s and then following policy π. The value function is used to measure the selection quality of each state or state-action pair. Policy optimization is to find the best mapping that maximizes the accumulated discounted rewards.

Deep Q-learning is the first DRL method proposed by DeepMind [9] and it adopts a DNN to derive the relationship between each state-action pair (s,a) of the system under control and its value function $Q(s,a)$. The Q-value is the expected cumulative

(with discounts) reward when system starts at state s and follows action a (and certain policy thereafter). To be more specific, the DRL agent performs inference using the DNN to predict the Q value of all possible actions under the current state and select the action a_k with the highest Q value at each decision epoch. To train a DRL model, at the next decision epoch t_{k+1}, the DRL agent updates the Q value of state action pair $Q(s_k, a_k)$ based on the received reward $r_k(s_k, a_k)$ using the Bellman equation in (1.2) and stores it in the replay buffer.

$$Q(s_k, a_k) = r_k + \max_{a_{k+1}} \gamma Q(s_{k+1}, a_{k+1}) \tag{1.2}$$

where r_k is the reward achieved in time slot k, and $\gamma < 1$ is the future reward discount factor. At the end of the execution sequence, the DRL agent performs mini-batch updating [11][12] that updates the DNN samples from the replay buffer. To deal with continuous action space, actor-critic models have been proposed as effective means of combining the policy search with learned value estimation. The actor-critic based reinforcement learning framework [10] learns policy and state-value function by training two interacting models, i.e. actor and critic, and is usually used to solve problems with continuous action space.

1.4 Neural Network Pruning in Deep Learning

Weight pruning have been explored extensively by previous works for DNN compression. Most of past approaches use magnitude pruning, in which the weights with the smallest magnitude are pruned first [13][14][15][16]. [17] greedily prunes weights by approximating change in the loss function and [6] uses variational methods to prune models. [18] proposes an ADAM-ADMM framework for structured weight compression for DNNs. The framework is incorporated with the stochastic gradient decent training process and can be understood as a dynamic regularization method in which the regularization target is analytically updated in each iteration. It can prune the

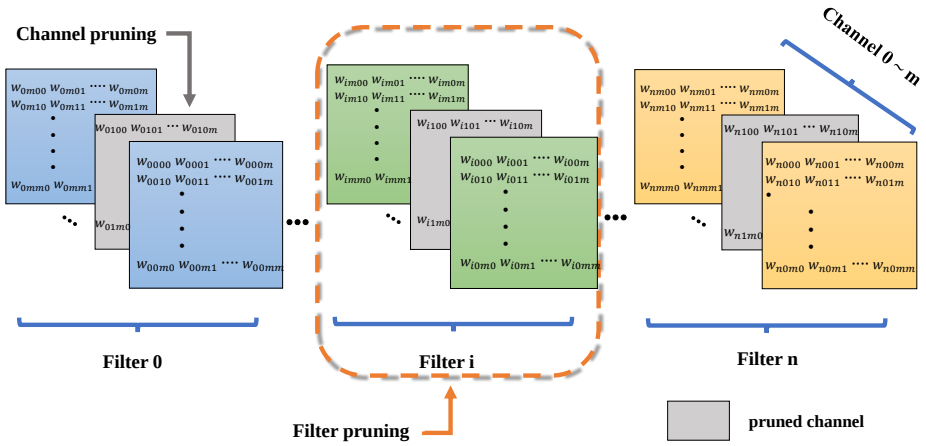


Figure 1.1: Structured weight pruning strategies defined on filter and channel levels respectively on a convolution layer

weights to create different types of structured sparsity such as filter-wise, channel-wise, column-wise sparsity as well as non-structured sparsity. Figure 1.1 illustrates structured sparsity defined on filter and channel levels respectively on a convolution (*conv*) layer. The specific structure sparsity and prune ratio are enforced during the training as constraints, and solved using the Alternating Direction Method of Multipliers (ADMM) method. ADMM is a powerful mathematical optimization technique, which decomposes an original constrained optimization problem into two subproblems that can be solved separately and efficiently [19]. It can effectively deal with a subset of combinatorial constraints and yield optimal (or at least high quality) solutions. When applied to DNN weight pruning, the first subproblem is to minimize the loss of DNN training with dynamic regularization that minimizes the distance between the weight coefficients and a set of auxiliary variables. The first subproblem can be solved using gradient descent [20]. The second subproblem can be formulated as a Euclidean projection onto a set with special structure and it can be solved analytically.

Structured weight pruning, such as filter pruning, channel pruning, and column pruning, are believed to be more effective than unstructured weight pruning. This is because structured weight compression maintains certain regularity. Furthermore,

the main advantage of structured weight compression is that a full matrix will be maintained in general matrix multiplication with dimensionality reduction, without the need of indices, thereby facilitating hardware implementations. It is also worth mentioning that filter compression and channel compression are correlated [18] as pruning a filter in layer i (after batch norm) results in the removal of corresponding channel in $i + 1$. In general, *conv* operations are commonly transformed to matrix multiplications by converting weight tensors and feature map tensors to matrices, named general matrix multiplication (GEMM). Therefore, the complete matrix can be retained in GEMM without indexing when the dimension is reduced by the reduction function, which promotes the hardware implementation.

1.5 Contributions

In this thesis, we investigate different computing paradigms to explore efficient deep learning models for different applications. To be specific, recurrent belief propagation is applied as the model for the intelligent handwritten text recognition. Deep Q-network is applied for energy efficient trajectory generation. Actor-critic based deep reinforcement learning is used to learn fast and accurate trajectory control policy. An early phase integrated weight compression scheme is developed to achieve real time on-board trajectory planning of UAVs. The organization and contributions are concluded as the following.

In Chapter 2, we present the belief propagation network which is incorporated with dictionary information to recognize handwritten text images. A layered approach is adopted. The bottom layer is a CNN for pattern recognition and the upper layer is a recurrent belief propagation network that searches for the possible words, which can be formed using the detected characters. Our recurrent belief propagation model does not rely on accurate separation of the characters. With the proposed post-pruning

techniques, the size of the output set of word candidates is reduced and the ranking of the correct word within the output set increases.

In Chapter 3, we deploy a two-level optimization framework to ensure the safe and effective operation of the drone, by generating obstacle free trajectories that allow the drone to maintain stability and energy efficiency. At the higher level, a series of waypoints are selected, which lead the UAV from its current location to its destination. The energy efficient trajectory is analytically generated between each pair of adjacent waypoints at the lower level. The entire framework is implemented using deep reinforcement learning that learns the highly complex and non-linear interaction between these two levels. In addition, a progressive learning strategy is investigated, which does not only reduce the convergence time, but also improves the quality of the results. We also provide results to show the effectiveness of using genetic algorithms to adjust the gain in the optimal trajectory scheme.

In Chapter 4, we present an actor-critic reinforcement learning framework that controls UAV trajectory through a sequence of desired waypoints. A deep neural network is constructed to learn the best tracking strategy, and reinforcement learning is developed to optimize the resulting tracking scheme. The proposed trajectory tracking framework improves UAV's response time and robustness. Compared to tradition PID controller, it achieves lower position error and less system power consumption with faster attainment. With the consideration of only linear operations in actor network, FPGA based hardware acceleration is also designed for energy efficient real-time control.

In Chapter 5, an early phase integrated structured weight compression scheme for DRL based waypoints planning is presented. By applying pruning at the early phase, the system improves computational performance of training and inference without decreasing success rate. The integrated training and pruning framework achieves a significant performance improvement by having high compression ratio, better success

rate and less memory and computation requirements. The convergence speed of the new framework is 34.14% faster with two refactoring techniques. With closely integrated pruning and retraining at the early phase, the framework does not only achieve higher compression ratio, but also gives better success rate. This converts into more reduction of the floating point operations for the inference as well as a measured run time reduction.

In Chapter 6, we summarize the work in this thesis and provide some future research directions.

Chapter 2

Assisting Fuzzy Offline Handwriting Recognition Using Recurrent Belief Propagation

2.1 Introduction

In recent years, many studies focus on recognizing handwritten words. Handwritten words demonstrate high variabilities because each person possesses his/her own unique writing style. Furthermore, clear boundaries cannot always be found between characters in handwritten text. Adjacent characters sometimes are connected or overlapped. This significantly increases difficulty for detection algorithms solely based on pattern matching. Without clear boundary, two characters that are close to each other may be recognized as one character or one character may be split into two characters. These errors are usually referred as insertion or deletion errors. Recognition of handwritten characters in offline situation are more challenging, because it does not have dynamic representations of hand writing trajectories, which is a useful feature for classification.

Some of recent approaches apply Histograms of Oriented Gradients (HoGs) [21], discrete HMM [22][23] or deep neural networks [24][25] to recognize handwritten words. In general, these works investigate two directions to improve the recognition accuracy: 1) Searching for the set of more robust features that are orientation, distortion insensitive; 2) Incorporate language and dictionary information with the pattern recognition. S. Yao [26] used a method based on sequences of HoG feature vectors. This method normalizes and divides the input image into equal-sized cells, and then organizes HoG descriptors into horizontal and vertical directional features vectors. Discrete HMM has been successfully used for handwritten Arabic word recognition by M. Dehghan et al. [27]. They use the histogram of 4-directional chain code as feature vectors, by using a moving window scanning the input image from left to right. However, directional features of handwritten words are variable and they are hard to recognize with rotation and distortion. A. Gupta [28] improved this by using Fourier descriptors. However, exactly segmenting words into individual characters is essential, which is less likely to achieve when the input is noisy. Y. Lecun [29] [30] proposed an efficient multilayer CNN for recognizing both handwritten digits and characters.

To integrate dictionary information with character recognition, [29] applies standard grammar graph to select the output from the CNN based character recognizer and form the selected characters into words. Although effective, only if both recognition graph and standard grammar graph reach the end nodes will an acceptable sequence of input symbols be selected and the standard grammar graph is not recurrent, therefore, it is not capable of correcting the deletion errors, which is very likely to occur when no well-defined character boundaries are found. This is improved by the multidimensional Recurrent Neural Network (RNN) proposed by Alex Graves [31]. Trained with the image of whole words, the RNN is able to recall the spatial pattern of adjacent characters, which improves recognition accuracy. Esam Qaralleh

et al. [32] tuned the recurrent neural network to a deep neural network with three hidden layers and two subsampling layers. Their approach segments the input word into sub-words first and then recognized sub-words using RNN. In this way, the complexity is greatly reduced. The RNN provides a comprehensive solution for spatial temporal pattern recall, however, its training is quite expensive. In [33] [34][35], a layered framework is developed that utilizes cogent confabulation model in the upper layer to form correct words and sentences based on the characters detected by the bottom layer using pattern matching. However, similar to [29], the confabulation model assumes that the images and characters in the real word has one to one correspondence, it cannot correct the insertion and deletion error.

In this work, we aim at incorporating dictionary information to assist recognition of handwritten text images. We generalize the definition of handwritten text to any text image with irregular fonts and possible overlapped characters. To avoid expensive training of an RNN, we adopted similar layered approach as [29] and [33]. The bottom layer is a CNN for pattern recognition and the upper layer is a recurrent belief propagation network that searches for the possible words that can be formed using the detected characters. The belief propagation network generates fuzzy outputs. The belief propagation network generates fuzzy outputs. The output is a set of possible words recognized from the given image and their scores. The fuzzy output can further be refined if sentence level information is provided. The belief propagation network is a neural network constructed based on a given dictionary and the construction complexity is linear to the size of the dictionary. Unlike the models in [28] and [33], our network does not rely on accurate separation of the characters. Each neuron maintains a memory. The state of a neuron is not only determined by its input, but also its historical value. These improvements not only enable the model to correct insertion and deletion errors, but also replacement errors. If a wrong spelled word that is not in the dictionary, our system will give several most likely word candidates

with the correct spelling. Some examples are given in Section 2.3.4.

The goal of our research is to reduce the size of the output set of word candidates and to increase the ranking of the correct word within the output set. This is achieved by a carefully designed inference network and certain pruning techniques such as post Gaussian Mixture Estimation. Compared to the standard grammar graph based word detection, we have more than 5% improvement in word accuracy. Compared to the result without post Gaussian estimation, 46.57% unrelated word candidates are pruned with additional 7.2% ranking increase.

The rest of this chapter is organized as the follows. Section 2.2 provides the details about system architecture and algorithm. Experimental results are given in Section 2.3 and Section 2.4 summarizes the current work.

2.2 System Architecture and Algorithm

2.2.1 Network Overview

The overall framework has three layers: (1) Segmentation layer using Local Peak Finder Algorithm [36]; (2) Recognition layer using LeNet-Structured CNN; and (3) Inference layer using recurrent belief propagation network. Figure 2.1 shows the flowchart of the overall framework. Its input is handwriting word images. The output is a fuzzy recognition. It consists of a set of possible word candidates and their ranking and scores. Using Local-peak finder algorithm, we divide a word image into a sequence of segments. Then the LeNet-based recognizer gives possible class labels with probabilities. The segmentation step is a best effort approach to separate characters. Since there is not always clear boundary between characters in handwritten texts, the separation is not perfect. It is possible that a segment consists of multiple characters. Those segments are detected and processed using Space Displacement Neural Network (SDNN) [37]. The details will be introduced in Section 2.2.2. Lastly,

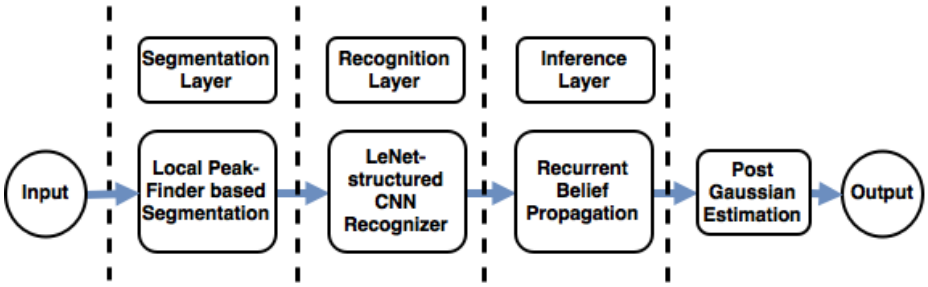


Figure 2.1: Overall framework

the belief propagation network recalls words with the highest likelihood based on the predictions from the CNN and report their ranking and scores.

2.2.2 Segmentation Based on Local Peak Finder Algorithm

SDNN has been proposed in [37] to apply CNN on text images with connected digits. The neural network is used to process each sub-image selected by a series of sliding widows, and the output is connected using a Viterbi alignment module. Naively applying SDNN to the original word image will unnecessarily increase the computation complexity. In the first step, we segment the image into separate characters or character groups and then apply SDNN on each segment. In this work, we improved the local peak finder algorithm in [36] to search for segmentation point. The algorithm is shown in Algorithm 1.

In this work we consider black-white images with white background and black foreground. The white pixel value is 255 and black pixel value is 0. Firstly, we calculate the summation of the pixel values for each column and record the positions and values of peak (column with local maximum pixel value) and valley (column with local minimum pixel value). The maximum difference between the peak and valley are calculated and denoted as Δ. If a column's pixel value is less than 0.6Δ, it will be removed from consideration. Therefore, only those columns with large number of white pixels will be considered as potential point for segmentation. Starting from the

Algorithm 1 Local peak finder algorithm: pseudo code

```
True pixel peak set = ∅;
Obtain column pixel value summation vector C;
Record positions and values of every valley V_i and peak P_t in C;
Calculate maximum different Δ between peak and valley;
if column pixel value < 0.6Δ then
    Remove the column from consideration;
end
while position V_i in valley set not the end do
    Find peak position P_t prior previous to V_i;
    Mark position V_imin of min valley between P_t and V_i;
    if V_imin ≠ V_i then
        Set frame between V_imin between V_i as decision region D;
        Mark position P of max peak in decision region D;
        if pixel value of P ≥ threshold T then
            Add P to true peak set T_p;
        end
    end
end
Separate input image at position value in set T_p;
```

first valley position V_i, we look for the first peak P_t to the right of V_i and the region between these two is set as decision region. When the pixel values in this region are all greater than the value of V_i, V_i is move to the next valley position. If there exists a point V_{imin} with smaller pixel value than V_i and the peak between $[V_{imin}, V_i]$ is greater than a particular threshold T, we will segment the image at the location of the peak, and continue processing the remaining image using the same algorithm.

The segmentation algorithm is only a best effort approach. It is possible that multiple connected characters will be included in the same segment. Those segments whose width is less than the average character width T_w will be processed directly by the CNN for pattern matching. T_w is obtained from the training set. For segments whose width is greater than T_w, a moving window that is T_w wide is used to scan through the segment from left to right with one pixel step size. Each image selected by the window will be processed by the CNN.

Figure 2.2 shows an example of the segmentation step. The input word image is

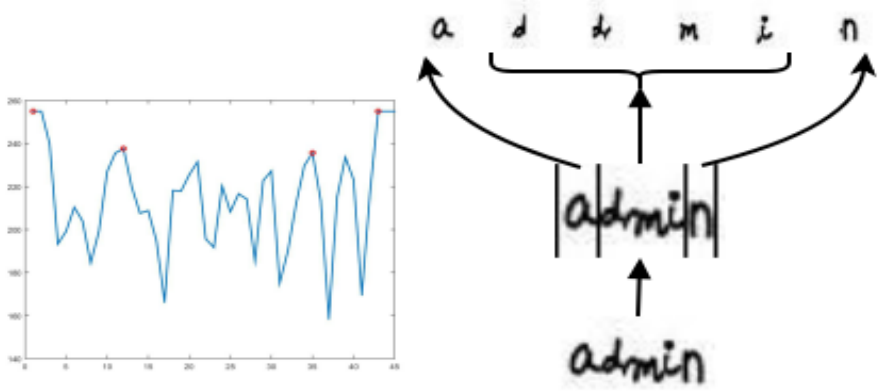


Figure 2.2: Example for segmentation layer

shown at the bottom. The column pixel value is plotted and three segmentation point are identified and highlighted. The middle segment is wider than average. Applying a moving window to this segment, we got four images for this area. Overall six images will be sent to CNN for pattern recognition.

2.2.3 LeNet-Structured CNN Recognizer

We use the CNN structure defined by Berkeley Vision and Learning Center [38] for pattern recognition. It is trained to recognize 26 English alphabets. The structure is the same as LeNet. The input is a 28×28 image and output is a set of possible characters and their probabilities. Please note that the image generated from the segmentation layer has with T_w, which is less than 28. They are padded with white space to make the size 28×28. The base learning rate of the training is set to 0.001 and number of iterations is set to 5000.

Each recognizer predicts a vector of N most likely labels for every segment. Therefore the input of belief network in next layer is a sequence of N dimensional vectors. Each vector represents a set of possible candidates perceived at specific location in the input image. Using the segmentation results from Figure 2.2 as an example, we show how the recognition layer works in Figure 2.3. Here N is set to 2.

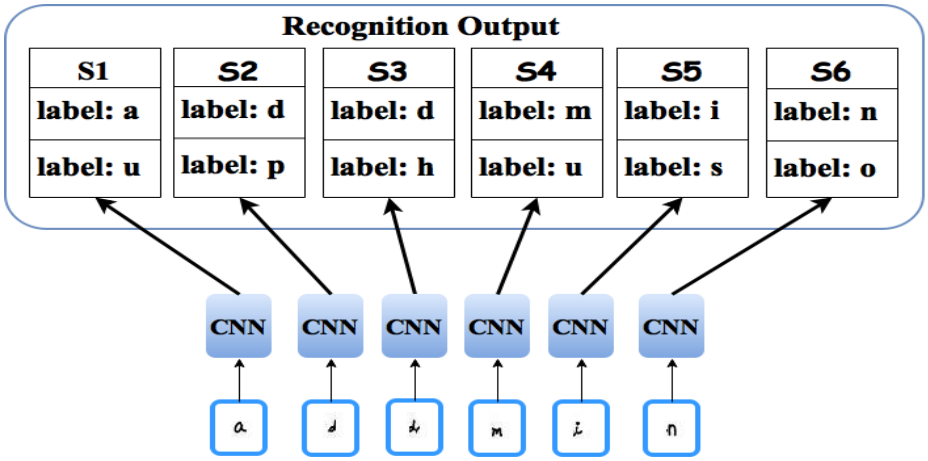


Figure 2.3: Example for recognition layer

2.2.4 Recurrent Belief Propagation Network

The inference layer of belief propagation is a neural network that consists of three types of neurons: input neurons ($\mathcal{I}$), interpretation neurons ($\mathcal{P}$), and dictionary neurons ($\mathcal{D}$).

Every substring of characters in a dictionary word starting from the first character corresponds to a dictionary neuron. We denote a dictionary neuron as $\mathcal{D}_\alpha$, where α is the substring associated to the neuron. For example, the word "admin" is associated to five dictionary neurons, $\mathcal{D}_a$, $\mathcal{D}_{ad}$, $\mathcal{D}_{adm}$, $\mathcal{D}_{admi}$, and $\mathcal{D}_{admin}$. We can further divide the dictionary neurons into two categories: neurons that represent real dictionary words or neurons that represent substring of real words.

All dictionary neurons are connected in a Trie [39] structure, and all connections are bi-directional. That means, neurons i and j are connected, if i is the prefix of j or vice versa. Furthermore, if i is the prefix of j, then we call the link from i to j as prediction link, and the link from j to i as feedback link. For example, there is a prediction link from neuron $\mathcal{D}_a$ to neuron $\mathcal{D}_{ad}$, and a feedback link from $\mathcal{D}_{ad}$ to $\mathcal{D}_a$. However, there is no connection between $\mathcal{D}_a$ and $\mathcal{D}_{adm}$ or any other neurons in previous example. The bidirectional connections form a recurrent network.

In this work, the dictionary neurons are generated using Mieliestronk's list [40].

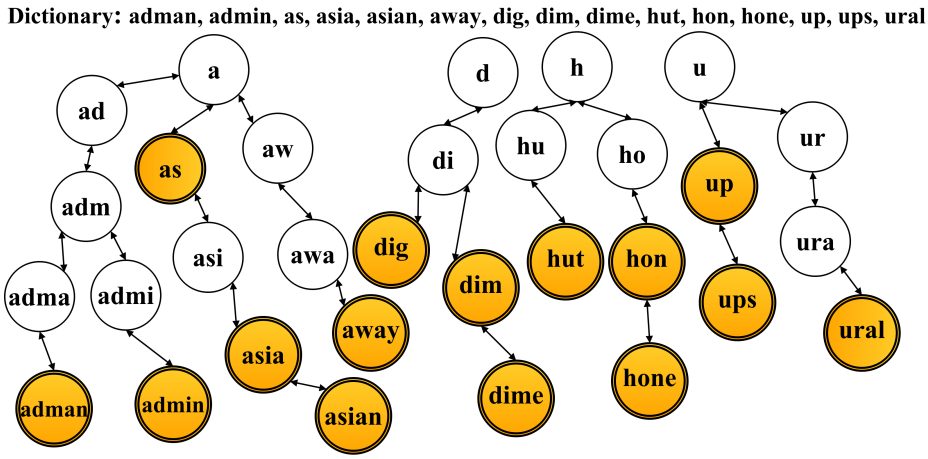


Figure 2.4: Example for neuron pool

This dictionary has 58027 English words with average length of 8 characters. It generates approximate 470000 dictionary neurons. We refer to the overall dictionary neurons as the neuron pool. Figure 2.4 shows all dictionary neurons and their connections generated from a small dictionary with only 15 words. All neurons that correspond to real word are highlighted in orange. As we can see, the network has a tree structure.

As shown in Figure 2.5, each directional link between two neurons is associated with a weight, which is set to be the log conditional probability $\log[p(s|t)/p_0]$ between the source and target neurons of the link. The $p(s|t)$ for prediction links is collected statistically from the dictionary. For feedback links, this value is always 1. There are 26 input neurons, each represents a possible character candidate detected by the CNN recognizer. An input neuron is denoted as $\mathcal{I}_\beta$, where β is one of the 26 English alphabet. Considering the possible errors in recognition, we add a set of 26

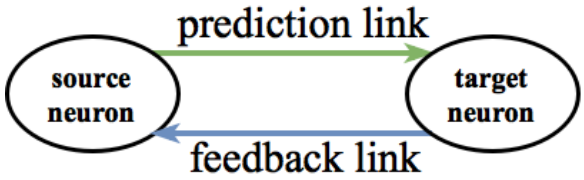


Figure 2.5: Knowledge link

interpretation neurons, denoted as $\mathcal{P}_\gamma$, where each γ is an English character. Links are established from $\mathcal{I}_\beta$ to $\mathcal{P}_\gamma$, the weight of the link is the probability that γ is recognized as β. This information is collected from the training process. There is also knowledge links from the interpretation neuron to the dictionary neuron. If γ is the last character in the substring α, then there is a link from interpretation neuron $\mathcal{P}_\gamma$ to dictionary neuron $\mathcal{D}_\alpha$.

An example of all 3 types of neurons and their connections is shown in Figure 2.6 for the 15-words dictionary. The interpretation and dictionary neurons are represented by rhombus and circles respectively. To make the figure readable, we do not show the connections between interpretation neurons and dictionary neurons, but they are reflected by neuron colors, i.e. there is a link between interpretation neuron and dictionary neuron with the same color. In this example, there is a 0.076 probability that a letter "d" will be recognized as "a", therefore, the link from $\mathcal{I}_a$ to $\mathcal{P}_d$ has weight 0.076. The interpretation neuron $\mathcal{P}_d$ will excite all dictionary neurons that end with letter "d", therefore, it has a connection to two dictionary neurons, $\mathcal{D}_d$ and $\mathcal{D}_{ad}$.

Each neuron maintains its excitation level. The excitation level of each input neuron is directly set to the corresponding class probability reported by the CNN.

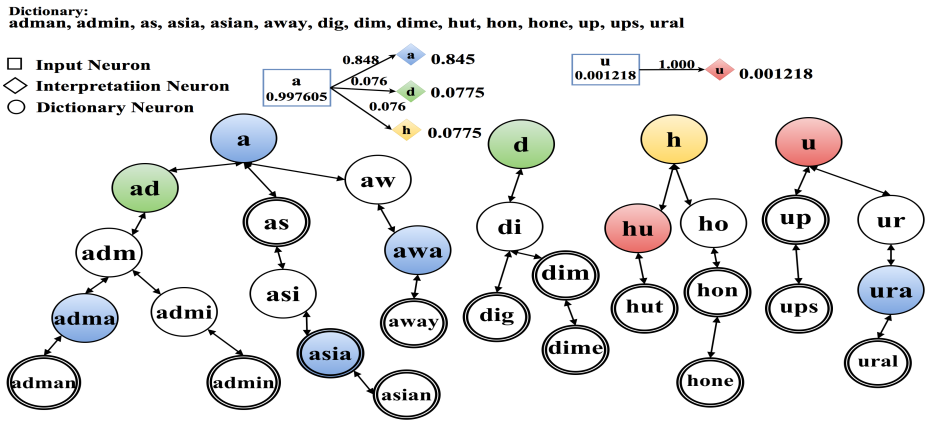


Figure 2.6: Example for inference layer

The excitation levels of all the other neurons are calculated as:

$$I(t) = \sum_{k \in F_l} [\sum_{s \in S_k} I(s) ln\big(\frac{p(t|s)}{p_0}\big) - ln\big(\frac{p(s|t)}{p_0}\big) + B] + (1 - \alpha) I(t), t \in S_l, \qquad (2.1)$$

where Variable $I(s)$ is the normalized excitation level of neuron s, and is referred as activation level. The normalization is carried out across all neurons of the same type. As we can see from the equation, the excitation level of a neuron t is determined by both the input excitation and the neuron's current activation. In other words, a neuron has memory. Even if it is not being excited externally, it will still remain active. However, its activation level will diminish a specific percentage α if it does not receive input excitation. During the normalization procedure, the activation of this neuron will be inhibited as other neurons that have been excited externally keeps on increasing their excitation level. For a dictionary neuron $(\mathcal{D})_\alpha$, its input comes the prediction link, the feedback link and the link from the interpretation neuron. The predictive signal predicts the next character that may be perceived, the feedback signal confirms the previous perception based on current inputs and the interpretation signal simply represents the sensory input from the recognizers. The proposed model is to certain degree similar to the HTM model [41]. However, the HTM model uses a one-bit flag to indicate prediction status, while the prediction in our model is lumped in the neuron excitation level.

During recall, the excitation level of all neurons will be updated each time a new input is received from the pattern matching layer. Please note that the normalization is performed after each update, therefore, the neurons with higher excitation will suppress those with lower excitation in a soft winner-takes-all manner. At the end of recall, a sequence of neurons will be highly activated, which form a path (or multiple paths) that lead to the predicted word(s). Please note that if we unroll the recurrent network over time, it is actually similar to a confabulation network [42][43][44]. Using

the network for word "admi" as an example. The belief network is shown in Figure 2.7b. It consists of seven dictionary neurons. Assume six images are separated (either by segmentation or sliding window) for pattern recognition as shown in Figure 2.2, then the belief network will be evaluated six times. If we unroll the recurrent network over time and create a copy of all dictionary neurons for each evaluation interval, then we obtain a confabulation network with six lexicons as shown in Figure 2.7a, each lexicon has seven symbols corresponding to all dictionary neurons. Each symbol only connects to symbols in its adjacent lexicons and there is no connection between symbols in the same lexicon. The connections corresponding to predictive and feedback links are illustrated in Green and Blue. There is also a Red connection that links the same neuron in adjacent lexicons. This models the memory of the neurons, i.e. the neuron's previous excitation level affects its current excitation level. To make the figure simple, we only show links between Lexicon 1 and Lexicon 2, however, these links should repeat between all other adjacent lexicons. The path shown by solid arrows leads to the correct word.

Assume that the recognizer recalls only one matching pattern for each image, i.e. the output from the recognition layer is a sequence of six 1-D vectors, and the output of the recognition layer is the sequence "a, d, d, m, i, n". Also assume that the same set of characters are triggered in the interpretation layer. These input signal will dynamically changes the excitation level of neurons. Figure 2.7c plots how the neuron activation level changes over the time. For example, neurons "a" and "adma" first get excited when the first input "a" is received. Neurons "ad" and "adman" are being predicted then, and they further predict downstream neurons. In the next, the input "d" is received, and neuron "ad" is further excited. It sends feedback signal to neuron "a" to confirm the previous observation and continue predicting neuron "adm". The excitation level of "adma" diminishes gradually even though it got excited at the beginning, because there is no feedback or input to confirm the observation (or

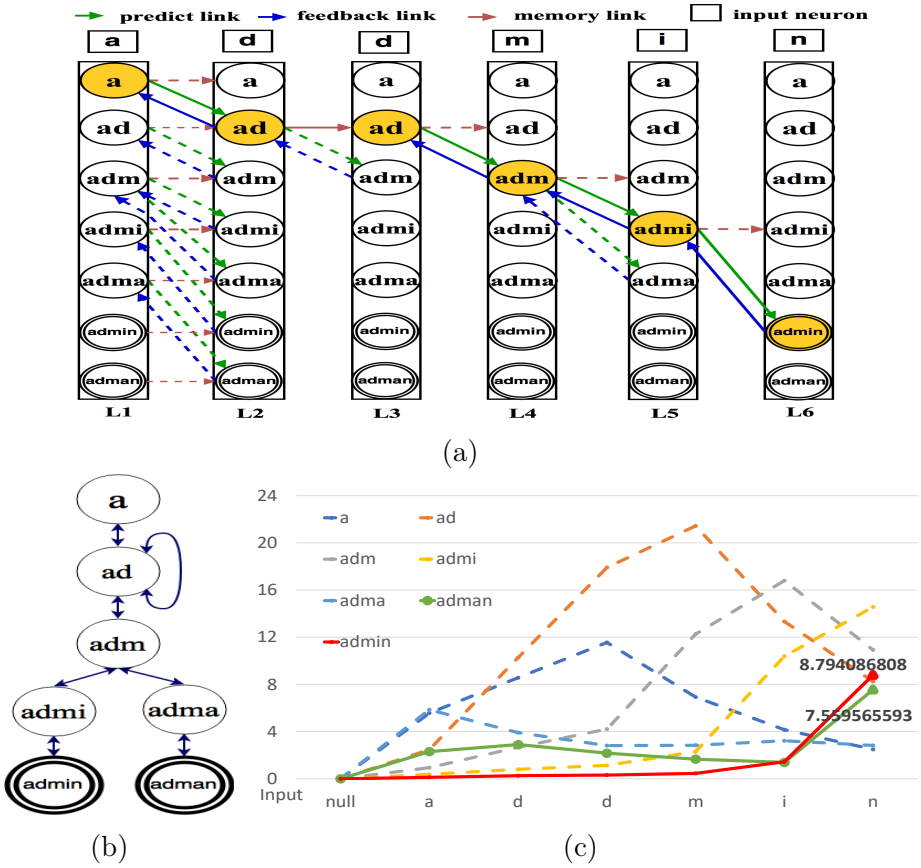


Figure 2.7: (a) Confabulation network (b) Recurrent belief propagation network (c) Example for neuron excitation level evaluation

prediction). At the end, the word "admin" accumulates the highest excitation. Also the set of neurons "a", "ad", "adm", "admi", and "admin" has the highest activation and they identify a path that leads to the correct word.

2.2.5 Further Improvements

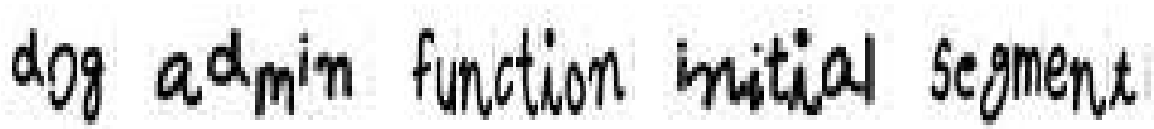
Sometime, certain common combination of characters or high-frequency words will reach high excitation simply from prediction, even though some of their characters are not reported by the recognizer. The last step of our work is to lower the ranking of these words or eliminate them from the candidate list. This is achieved by adding pre-processing and post-processing.

We denote estimation of the word length and start-end position constraint as pre-

processing constraints. We firstly estimate the probable number N_c of characters in the word, which equals the quotient of input image pixel width divided by average character width T_w. If the word candidate has more than N_c characters, it will be eliminated. Furthermore, start-end position constraint is used to strengthened the excitation process of neurons. In the first evaluation interval, excited neurons containing only one character will get more excitation than others. Similarly, excited neurons representing a real word will get more excitation if the input from the pattern matching layer is the last. For example, adding start-end position constraint will help to differentiate the first excitation levels of "a" and "adma" in Figure 2.7c, as "a" will get more excitation increase than "adma".

The post-processing is achieved by considering the location distribution of each character. For any English character x, we consider its location in a word is a random variable. The probability that α will be the l_{th} letter in the word is denoted as $pr_x(l)$. We assume that this distribution follows a Gaussian Mixture Model (GMM), and the information reported by the recognizer is a sample of the distribution, based on which the whole distribution is constructed [45].

Using the GMM, a variable $Prob(w) = \prod_{i=1}^{N} wpr_{x_i}(i)$ is calculated for each word w, where x_i is the i_{th} character in w. This variable indicates the possibility that w is the correct word by considering where each character in w is located. The excitation level of word w is then adjusted based on the probability by calculating: $el'(w) = el(w) + ln\frac{prob(w)}{p_0}$. Please note that the excitation level $el(w)$ is actually the log probability of w estimated using the inference network, therefore, the adjustment is actually calculating the product of the two probabilities to combine the prediction results from different approaches. Our experimental results show that combining with GMM will improve the ranking of the correct word by 7.2%.



(a) dog (b) admin (c) function (d) initial (e) segment

Figure 2.8: Word examples

2.3 Evaluations

2.3.1 Environment Setup

The dataset that we use for training and testing are generated based on images from Chars74k [46]. Our dictionary is the Mieliestronk's list [40]. In this list, the British spelling was preferred and American versions are deleted. Only lower words are tested in our experiments. The word images are generated using the method as mentioned in [29]. We first randomly select a word from the Mieliestronk's list. For every character in the word, we then randomly select a character image from different writing styles in the Chars74k dataset and put them close together. We allow adjacent characters to connect to each other. We keep the height of the word image to be 28 pixels by scaling without changing its aspect ratio. Some sample input word images are shown in Figure 2.8.

2.3.2 Tranining of CNN-based Recognizer Component

We train the CNN-based recognizer with a subset of Chars74K [46]. Chars74K contains 26 classes and 55 samples for each class. All the 3410 hand drawn characters in the dataset are lower-case English characters, from a to z. We use 32 samples per class for training and the rest 13 samples per class for testing. We choose 28×28 as the size of input images, which is the same as defined in [38] LeNet caffemodel.

During testing, the CNN report a set of class labels and their matching probabilities. The test accuracy of various training learning rates and iterations is reported in

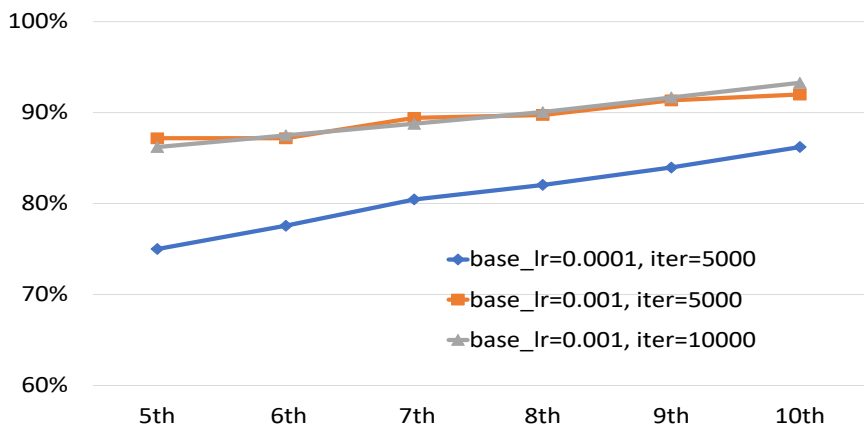


Figure 2.9: Test accuracy of CNN-based recognizer

Figure 2.9. "i_{th}" means that the correct labels score is within the i_{th} highest predicted score. According to these results, we choose 0.001 as base learning rate and 5000 iterations to train the CNN model. To limit the complexity of probabilistic inference, we only send six highest possible class labels to the next layer.

2.3.3 Word Recognition Accuracy

We test the whole system under an environment using GeForce GTX/750/PCIe/SSE2. In the experiments, 187200 input images containing 6240 different words which are randomly selected in [40] are generated. Word candidates are selected from the highest excitation level to the lowest. We report the results from three aspects: (1) the chances that the correct word is within top 5, 10 and 20 predictions; (2) the number of word candidates; (3) expected ranking of the correct word.

Firstly, Figure 2.10 shows the chances that the correct word is within top 5, top 10 and top 20 predictions respectively after adding improvement techniques. As we can see from the figure, we got 46.07% average accuracy increase if the correct word is predicted within Top 5. 36.84% accuracy improvement is got when the word is correctly predicted within Top 10. The accuracy will increase 18.75% if the correct word is predicted within Top 20. As the end of belief propagation, the dictionary neurons

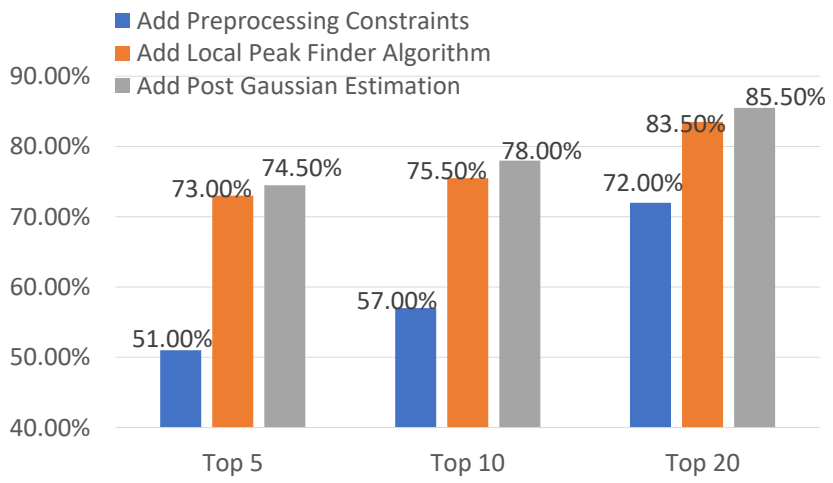


Figure 2.10: Accuracy improvement

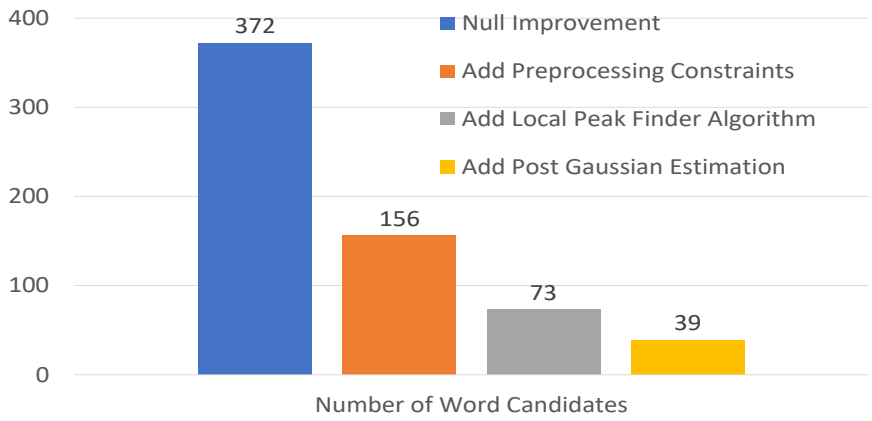


Figure 2.11: Average number of word candidates improvement

corresponding to real words and whose excitation level is non-zero are reported as candidates. Large amount of word candidates means high complexity and ambiguity. Therefore, we show how the pre- and post-processing techniques can help improve the validity of our network.

Figure 2.11 shows the decrease of average number of word candidates after applying different improvement techniques. Adding preprocessing constraints can help prune 58.06% irrelevant word candidates. Applying Local Peak Finder based segmentation will further reduce 53.21% word candidates. Combining these with post

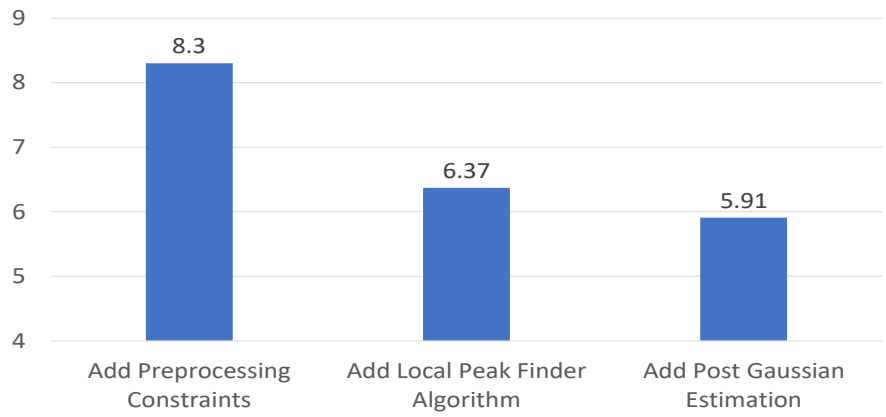
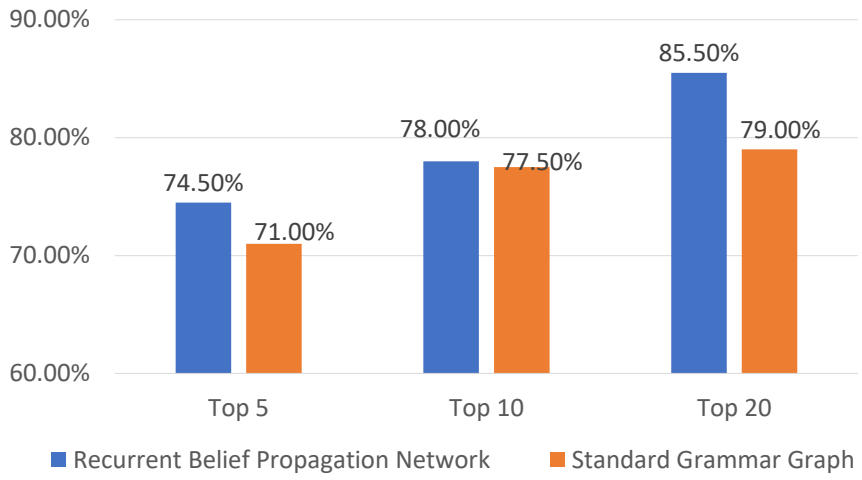


Figure 2.12: Expectation ranking improvement of correct word

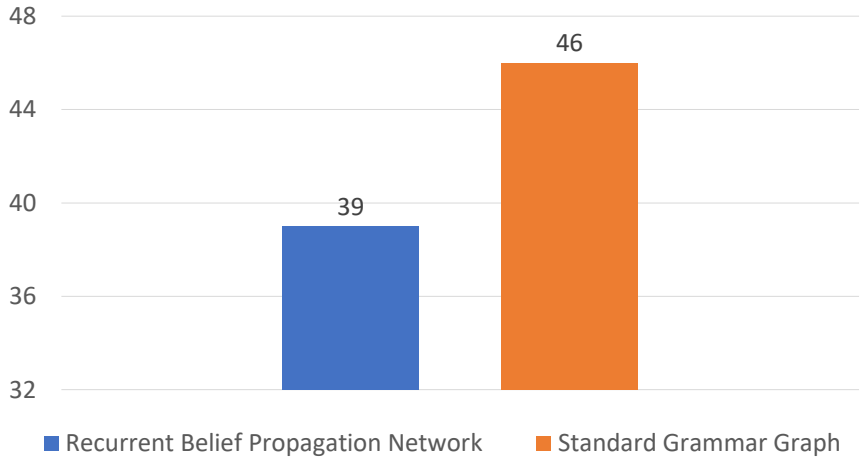
Gaussian estimation gives another 46.57% reduction. Overall, the number of word candidates can be reduced to 39 on average.

Figure 2.12 shows the expectation ranking improvement of the correct word if it is reported within Top 20 predictions. The lower ranking means better recognition quality. The figure indicates that the expected ranking of the correct word is 8.3 when only preprocessing constraints are applied. With segmentation, this value drops to 6.37. Finally, after using post Gaussian estimation, the expected ranking of correct word reduces to 5.91.

As a base line reference, we also implement the standard Grammar Graph, which is a type of finite-state transducers [47], mentioned in [29], and use it to replace the belief network. Figure 2.13a compares the accuracy of the two approaches after applying preprocessing constraints, local peak finder algorithm and post-processing. Again, the accuracy is measured by the chances that the correct word is within the Top 5, 10 and 20 predictions. The results show that our approach is 5 8% better than the standard Grammar Graph. Figure 2.13b reports that our approach has 15.2% less word candidates than that of standard grammar graph.



(a) Accuracy comparison



(b) Average number of word candidates comparison

Figure 2.13: Results comparisons with standard grammar graph

2.3.4 Recognition Results of Recurrent Belief Propagation Network

In this section, we list recognition examples generated by our recurrent belief propagation network. Table 2.1 lists the top predicted word candidates for some input images. The columns labeled as "Output Word Candidates" and "Excitation Level" give word candidates in the output set with their corresponding excitation level (i.e. score) in descending order. The correct word is highlighted in bold.

Table 2.1: Word candidates of sample words

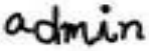
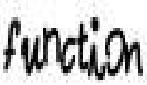
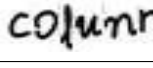
Input Images	Output Word Candidates	Excitation Level
	**best**	**10.3500**
	test	10.1219
	beet	9.9260
	stet	8.5189
	**admin**	**7.8904**
	adman	5.1902
	**initial**	**5.4023**
	fantail	-1.0919
	lenient	-3.4354
	strict	9..4421
	script	**8.5002**
	spirit	8.4568
	junction	9.9152
	function	**9.8459**
	fraction	-30.9105
	friction	-71.5689
	**segment**	**9.5512**
	tensest	8.7067
	tempest	-23.0852

Table 2.2 lists the recognition for some word images with wrong spellings. These wrong spelled words are randomly selected from [48]. Again, the highest predicted words are reported with their excitation levels listed in descending order. The words in bold font are the actual correct ones.

Table 2.2: Word candidates of sample wrong spell words

Input Images	Output Word Candidates	Excitation Level
	**describe**	**12.3928**
	disperse	11.9450
	perspire	11.8388
	effete	12.6600
	effect	**11.9770**
	defect	10.5627
	colour	11.8852
	column	**9.2361**

2.4 Conclusion

In this chapter, we introduce a recurrent belief propagation system for handwriting recognition. The system construction, processing algorithm and recall process are presented. In our self-structured system, allowing multiple neurons got excited helps improve quality of knowledge link information and maintain a relatively high accuracy at the same time. Because neurons can retain the knowledge information and reduce the chance of wrong recognition caused by CNN. The proposed preprocessing and post processing techniques effectively reduces the size of predicted word candidates and improves the expected ranking of correct words. The presented belief propagation network is general enough to be applied in other applications for sequence prediction and recall. One of our future work is to extend it for speech recognition. The similar segmentation and deep neural network can be used for pattern matching. We will also consider other promising methods, such as incorporating higher level context, to improve the accuracy and generality of the framework.

Chapter 3

Autonomous Waypoints Planning and Trajectory Generation for Multi-rotor UAVs

3.1 Introduction

Onboard real-time trajectory planning is particularly important in Beyond Visual Line-of-Sight (BVLOS) operations, and applications that require unmanned vehicles to move in cluttered and dynamic environments [49]. Such applications include indoor operations [50], package delivery in urban and suburban areas, monitoring of civilian infrastructures like bridges and highways, autonomous landing on moving platforms [51], and tracking wildlife in forested areas. Autonomous trajectory generation has received increased attention in the past decade [52][53][54], especially for autonomous systems such as UAV. Safe and effective operations of these UAVs de-

This work is partially supported by the National Science Foundation under Grant CNS-1739748 and SRC task 2893.001.

mand that we consider trajectory generation as a constrained optimization problem. While there are many different ways to formulate the objectives and constraints, the basic requirement is to consume minimum flight energy while avoiding obstacles and maintaining the UAV stability during the journey. To solve such problem analytically is extremely difficult if impossible when the UAV needs to fly large distance in a complex environment. Some of the classical approaches apply rapidly-exploring random trees [55] and voronoi graph [56]. In general, these approaches generate the feasible path by traversing the graph made by all possible paths and searching for possible links between path nodes. The convergence to the energy efficient path takes much time, and hence it cannot be applied during the real flight time. Furthermore, they either uses a large margin for obstacle avoidance, or adopt a trial-and-error based iterative approach, which will further increase the complexity. [57] implements a gradient decent approach which converges more quickly without loss of robustness. However, non-smooth trajectory are produced that are difficult for UAVs to precisely follow. Neither does it consider other objective functions such as minimum control thrust.

Recently deep learning has drawn extensive attention in areas of robotics applications for its outstanding abilities to learn representations of complex environment. Such representation is essential for environment awareness for applications such as UAV trajectory generation. Among others, deep reinforcement learning is extremely suitable to solve goal-oriented robotics tasks that has close interaction with environment dynamics [58]. Such interaction provides feedback, which is useful to improve the performance of the task being learned. Reinforcement learning has been used for robot path planning in some previous works. Real-time model-based reinforcement learning framework [59] as well as Q-learning [60] are adopted to find path in 2D surface. These models consider only the robot (or UAV) status as the system state. The environment information (e.g. the location of the obstacles) is not part of the

system state. In other words, these models learn the environment instead of learning the relationship between optimal control and the surrounding environment. When the environment changes, the model needs to be trained again. Such blindness to the surrounding environment is not realistic in today's UAVs. With the availability of 3D map and sensors such as the image and depth camera, the UAVs will have partial information about the environment. Environment information is used as an input in [61] and [62], where Deep Q Network (DQN) and deep deterministic policy gradient network, are deployed. However, without a lower level optimization, they are not capable to maintain the stability of UAVs or minimize the flight energy. At lower-level, [63][64] have decoupled time and geometry, constructed a geometric trajectory and then parameterized it in time. Utilizing differential flatness of dynamics to generate a trajectory is also adopted in [65]. Furthermore, [66] develops a high level trajectory generator in conjunction with a motion primitive generator to choose an optimized trajectory among different motion primitives. However, planning scheme with consideration only of the actuation model of UAVs is not enough, especially when maneuvers that go beyond hovering or level flight are required. In situations where large maneuvers are required, fast online trajectory planning schemes like the one proposed in this work become necessary.

In this work, we tackle the problem of UAV trajectory generation in a known 3D environment by solving it as a two-level optimization. Figure 3.1 shows the overall framework. At the upper-level, a sequence of waypoints is selected that lead the UAV from its current position to the destination, and at the lower-level, the efficient trajectory between each pair of adjacent waypoints is generated analytically. While the goal of trajectory generation is to maintain the UAV stability with minimum flight energy, the goal of the waypoints planning is obstacle avoidance with minimum control thrust in a (partially) known environment over the entire trip.

The two level approach is optimized together and it effectively reduces complexity

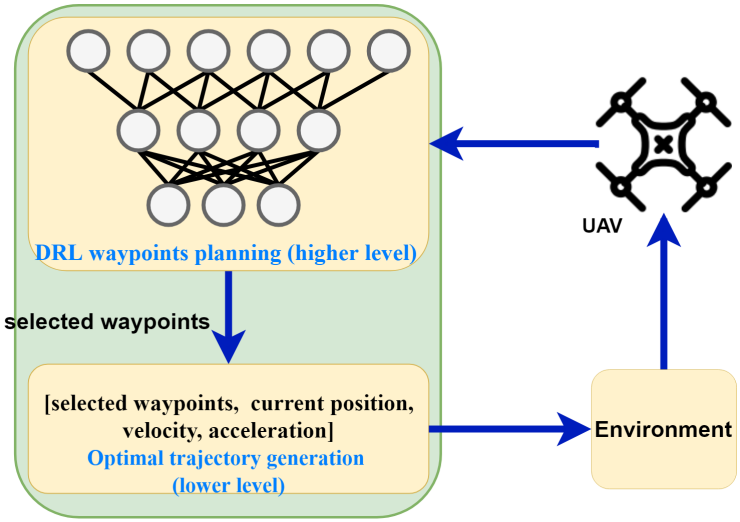


Figure 3.1: Overall framework of proposed autonomous waypoints planning and trajectory generation scheme

of lower level optimization, where detailed aerodynamic model of UAVs is applied to generate a short trajectory in a localized free space without worrying about obstacles. The upper-level optimizer (i.e. the waypoints planner) ensures that a global efficient solution can be obtained by connecting the sequence of locally optimized short trajectories. This is a combinatorial optimization problem with exponential search space, and its results are heavily affected by the lower level optimization. In this work, we use DRL for waypoints planning. The DRL framework not only learns the highly complicated and nonlinear interaction between the upper and lower level optimizers, but also learns and generalizes the impact from the environment (e.g. the location of obstacles) to the waypoints selection. While the DRL model also relies on trial-and-error approach and detailed aerodynamic model for training, this is done offline. During the mission, the waypoints are selected only based on the surrounding environment, the target location, and the flight status of the UAV.

We formulate the DRL as a DQN, which approximates the efficient selection of actions in time based on the instantaneous configuration of the environment. With awareness of this configuration and the feedback at every time step, the UAV modifies its behavior, i.e. the selection of the next waypoint. The learning of the DRL is

carried out in a controlled environment in a progressive manner so that the UAV can first discover its own dynamics and then learn how to cope with the external environment. After the generation of waypoints, an optimal trajectory is calculated, where the control inputs actuate the three degrees of rotational motion and one degree of translational motion in a body-fixed coordinate frame. The translational motion is controlled by a single thrust along a body-fixed direction vector, which can be controlled by the attitude of the vehicle. This actuation model covers a wide range of unmanned vehicles like fixed-wing and quadcopter UAVs, unmanned underwater vehicles, and spacecraft.

3.2 DRL-based Waypoints Generation

3.2.1 Problem Formulation

In this work, we focus on trajectory generation for multi-rotor UAVs. These UAVs have fixed plane of rotors that actuate the vehicle in three-dimensional transnational and rotational motion, hence they have the property of under-actuation. Given a closed environment, the UAV takes off from an arbitrary position and reaches a target position which is preassigned, without colliding with obstacles. As stated before, the first step is to select waypoints based on the environment. The entire 3D environment is divided into $N \times N \times N$ grids. The environment is described by a function $\mathbf{M}()$ maps a grid (x, y, z) to a real value $\mathbf{M} : (x, y, z) \rightarrow \mathbf{R}$. A grid, g, that contains obstacle will be mapped to -10, $\mathbf{M}(g) = -10$. The destination grid that the agent needs to reach is mapped to 10, and the grid where the UAV is currently located is mapped to 1. All other grids are mapped to zeros in the discretized environment block. Let $W_0, W_1, \cdots, W_{N-1}$ be the sequence of generated waypoints, where each one is a $3D$ vector corresponding to a grid in the environment. Let $f(W_i, W_j)$ denote the control thrust for the UAV to follow the trajectory between waypoints W_i and W_j

generated by the lower level optimizer. Also, let $\mathbf{G}(W_i, W_j)$ denote the set of grids that the generated trajectory between W_i and W_j will pass through. The total thrust cost along the trajectory is denoted as $\mathbf{F}$. The problem of waypoints generation can be formulated as the following:

Problem 1 (Efficient obstacle avoidance waypoints planning) *Minimize*

$$\mathbf{F} = \Sigma_{i=0}^{N-2} f(W_i, W_{i+1}), \tag{3.1}$$

subject to

1. *reaching the target position from current position,*

$$\mathbf{M}(W_0) = 1, \quad \mathbf{M}(W_{N-1}) = 10, \tag{3.2}$$

2. *reaching the target position without colliding with obstacle,*

$$\mathbf{M}(g) \neq -10, \quad g \in \mathbf{G}(W_i, W_{i+1}), \quad 0 \leq i \leq N-2, \tag{3.3}$$

To find the set of W_i, $0 \leq i \leq N-1$ is a combinatorial problem. The goal is to achieve minimum control thrust without obstacle collision if the UAV flies along the waypoints and trajectory. A large reward will be received at the end of the flight if the UAV reaches the destination. While this is the problem formulation of the upper level optimizer, the functions $f(W_i, W_j)$ and $\mathbf{G}(W_i, W_j)$ are determined by the lower level optimizer. Reinforcement learning provides a way to solve such constrained optimization problem with delayed reward. Incorporating with deep neural network, an efficient policy is learned to guide the UAV to the next selected actions (i.e. waypoints) that can lead to maximum future rewards.

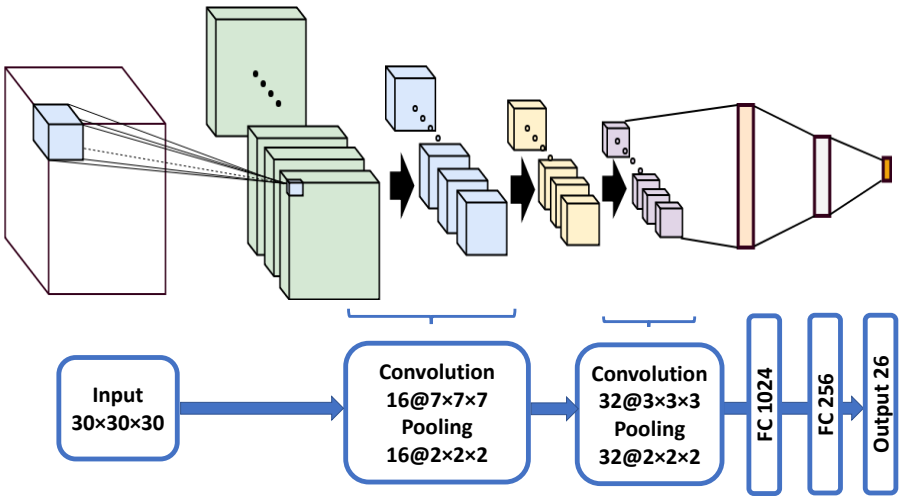


Figure 3.2: Network structure of proposed deep Q network

3.2.2 Network Structure

The detailed structure of the DQN is shown in Figure 3.2. The input of DQN is the state, which represents current known knowledge of the surroundings and the status of the UAV. The state is a $3D$ matrix with size $N \times N \times N$. Each entry (x, y, z) of the matrix is the mapped value $\mathbf{M}(x, y, z)$ of the corresponding grid in the $3D$ environment previously discussed. The state has the information about the relative position between agent and obstacles. A UAV can choose any of the $3 \times 3 \times 3$ grids around its current location as the waypoint. Therefore, there are 26 possible actions. This input state is fed into two $3D$ convolutional layers and each is followed by a pooling layer. The intermediate output of the second pooling layer is fed into two fully-connected layers with the size 1024 and 256 respectively. The output is a fully-connected layer with the size 26. Each output neuron estimates the $Q(state, action)$ values for one of the 26 actions at the given state.

Our goal is to generate trajectory for the UAV with minimum control thrust under the premise of reaching target position without hitting any obstacle. Therefore, our reward function is defined as the combination of position reward and control reward

as following:

$$\mathbb{R}(state, action) = \alpha\mathbb{R}_p(state, action) + \beta\mathbb{R}_c(state, action), \tag{3.4}$$

where α and β are the coefficients of position reward and control reward respectively, $\alpha = \beta = 0.5$ in our experiment. $\mathbb{R}_p(state, action)$ is the position reward of taking action in current state. It is defined as following:

$$\mathbb{R}_p(state, action) = \begin{cases} 10, & \text{reach target position} \\ -10, & \text{collide with obstacles} \\ 0. & \text{others} \end{cases}$$

And $\mathbb{R}_c(state, action)$ indicates the control reward. It is calculated as the negative $L1$ norm of thrust cost, which is calculated by (3.21) in efficient trajectory generation scheme proposed in Section 3.4.

3.2.3 Learning of DQN

Since the problem complexity, i.e. the total number of state action pairs, is $O(26\times N^3)$ which is relatively larger than many other existing problems [67][68][69], it is crucial to maximize exploration at the beginning of learning. Therefore $\epsilon-$greedy [11] is applied during the learning. Based on $\epsilon-$greedy, more random actions (i.e. exploration) are taken at the beginning of learning and more actions with maximum $Q(state, action)$ values (i.e. exploitation) are chosen as learning progresses.

To improve the learning, we also decrease the learning rate lr gradually because it becomes harder to improve performance with large learning rate as the gradient reaches plateau. In our approach, there are $30,000$ learning episodes in total. Instead of using a fixed learning rate, the learning rate starts from $1e-4$ and decreases every $5,000$ episodes based on $lr_{new} = (lr - \frac{lr}{epoch})_{epoch=i}$, $i \in \{5e+3, 1e+4, 1.5e+4, 2e+$

$4, 2.5e + 4\}$ and i is the i_{th} learning episode. This helps to prevent the learning from over correct after 5,000 iterations, which allows to maximize the exploitation.

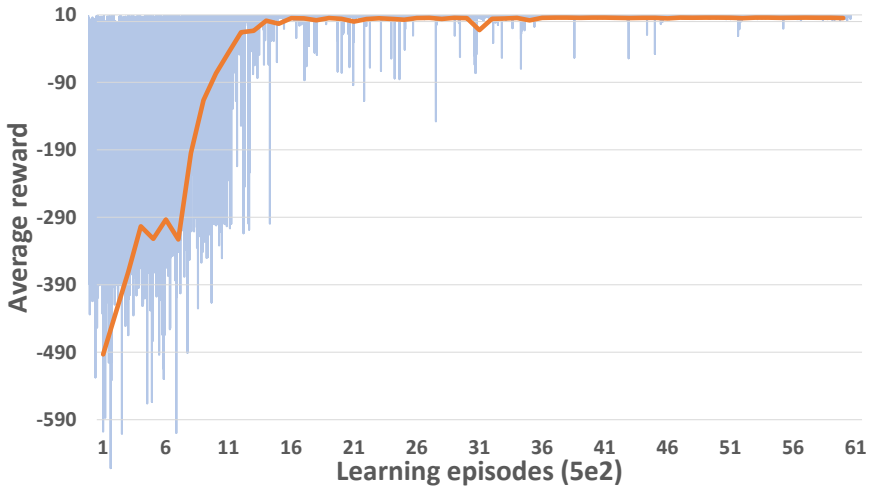
Instead of randomly initializing model weights based on a uniform distribution, we initialize the weights of model based on a normal distribution. It initializes weights with relatively small values, and prevents outputs of the model from being either too large or too small. Batch normalization is used before the second convolutional layer and the first fully-connected layer to reshape the input of those hidden layers. In order to bound the training time, for every single training episode, the maximum steps W_{max} that the UAV can take is fixed. If the UAV has taken W_{max} steps but has not reached the target position, it will be forced to start a new learning episode. The start and target positions are randomly selected. So do the locations of obstacles. In this way the environment configuration of every episode is different, therefore each learning episode is independent. During learning, an experience replay is used to save last thousand times of performance and a randomly sampled mini-batch of size 32 is used to train the network. At each time step within an episode, the UAV takes an action and receives a +10 position reward if it reaches the target position and -10 if colliding with obstacles. Otherwise the position reward is 0. The control reward is determined based on the current and next position, velocity and acceleration of the UAV. The weighted sum of these two rewards and corresponding UAV state and action is saved in experience replay buffer.

3.2.4 Progressive Learning in a Controlled Environment

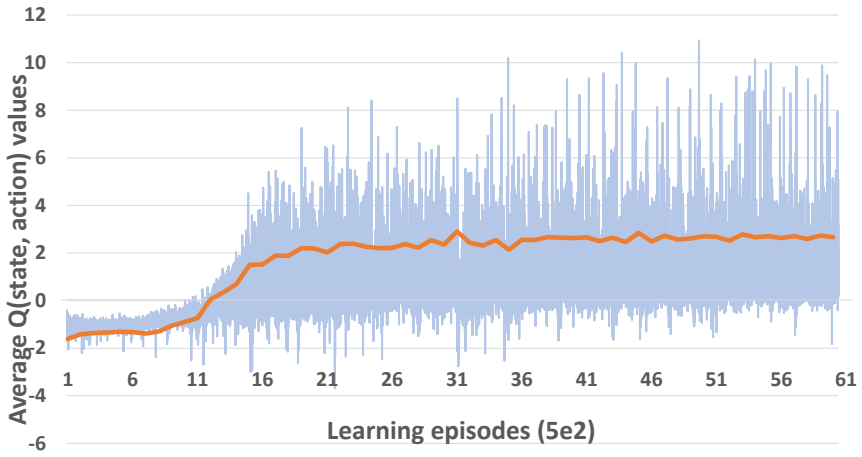
The waypoints planner is trained from scratch together with lower-level trajectory generation scheme. At the beginning of the training, the optimizer does not only have no idea about the efficient route to the target, it also does not know how the trajectory generation layer (i.e. the lower layer) will react to different waypoints selection, and if the UAV can keep its stability. It even does not know that the goal is to reach the

target while avoiding obstacles. All of these must be taught globally using reward and trial. Before a baby learns walking, we put her in a small area free of obstacles, where she can roll and crawl and discover how to coordinate muscle movements. Then she can learn how to reach target and avoid obstacles. We believe that the same should be applied to teach a UAV fly and we refer this as ***progressive learning in a controlled environment***. It consists of two measures, "*progressive learning*" and "*controlled environment*". The progressive learning requires us to start learning with very low UAV mobility and gradually increase it as the learning progresses. By only allow the UAV to travel short distance, it can get quick feedback. Although it won't be able to travel long enough to reach the target and get the large position reward, based on the received control reward it learns how to coordinate with the lower level controller. Based on this learned knowledge, it will then learn how to reach target when the mobility increases. The controlled environment means to start the learning with a free space and gradually increase the number of obstacles. In this way, it can reach the target sooner with less failure. The received position reward helps the UAV to gain the knowledge of its goal.

In our original learning approach, the maximum number of steps W_{max} that the UAV can take is set to 1000. Using the progressive learning technique, the W_{max} is initialized to be 100 at the beginning of learning, and every 5,000 iterations, its value increases 50%. The new value is calculated as $(W_{max})_{new} = round(1.5W_{max})$. With such short travel distance, if the UAV is able to reach the target, it will gain the knowledge and increase the value of locations close to the target. If the UAV is not able to reach the target, it will still gain the knowledge about the control cost. Using the controlled environment technique, the number of obstacles is set to 0 at the beginning of the learning. The number increases by 5 every 5000 iterations. With the help of controlled environment, the UAV can reach target much quicker at the beginning of learning. Again, this helps it to learn the inherent relation and



(a) Average reward agent received in each learning episode



(b) Average predicted $Q(state, action)$ values of selected actions

Figure 3.3: Learning results using progressive learning in a controlled environment

interaction between the upper and lower layer optimizer faster and better. Later in the learning process, when the UAV has more knowledge of its capability, it will learn how to avoid obstacles more quickly. Figure 3.3a gives the average reward the agent received in each learning episode during the entire learning process. And Figure 3.3b gives the average predicted $Q(state, action)$ values of those selected actions during the learning process. These figures indicate that as the learning goes on, the UAV receives more and more rewards and selects better and better actions. The learning converges at around 7,500 iterations. Experimental results also show that using progressive

learning in controlled environment not only reduces learning time but also improves learning quality. Details can be found in Section 3.5.

3.3 Dynamics Model of Multi-rotor UAV

3.3.1 Continuous Time Dynamics

The rigid body model considered in this work has four control inputs for the six degrees of freedom. These control inputs include a torque for the three degrees of freedom of rotational motion and one thrust along a body-fixed thrust vector. It can be applied to several unmanned vehicles, and the particular case of a quadrotor UAV is considered in Section 3.5 for numerical results.

In this work, $b \in \mathbb{R}^3$ denotes the rigid body's position vector expressed in an inertial coordinate frame and $\mathcal{R} \in \mathrm{SO}(3)$ is the rigid body's attitude expressed as the rotation matrix from inertial frame to body-fixed frame. Without loss of generality, it is assumed that the thrust vector is along the third body-fixed coordinate frame axis. The translational dynamics motion equation is:

$$m\dot{v} = mge_3 - fr_3, \tag{3.5}$$

where g is gravitational acceleration, $v \in \mathbb{R}^3$ is the translational velocity in inertial frame, $e_3 = [0,\ 0,\ 1]^T$, $u = fr_3 \in \mathbb{R}^3$ is the control thrust vector of magnitude f acting on the body, and r_3 is the unit vector along the third axis of the body-fixed coordinate frame, expressed in the inertial frame. Note that r_3 is also the third column of the rotation matrix $\mathcal{R}$. ((3.5)) can be rewritten as:

$$\dot{v} = ge_3 - \frac{1}{m} fr_3. \tag{3.6}$$

The velocity kinematics for the translational motion expressed in inertial coordinate

frame is simply $\dot{b} = v$. Consider a "triple integrator" dynamics model for position trajectory generation, given by

$$\dot{b}(t) = v(t), \tag{3.7}$$

$$\dot{v}(t) = a(t), \tag{3.8}$$

$$\dot{a}(t) = u(t), \tag{3.9}$$

where the vectors b, v, a, $u \in \mathbb{R}^3$ represent position, velocity, acceleration, and jerk respectively. Let $x \in \mathbb{R}^9$ denote the state vector, i.e., $x = \begin{bmatrix} b^{\mathrm{T}} & v^{\mathrm{T}} & a^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}$. The resulting system can be compactly expressed as follows:

$$\frac{dx}{dt} = Ax + Bu, \tag{3.10}$$

$$y = Cx. \tag{3.11}$$

where

$$A = \begin{bmatrix} \mathbf{0}_{3\times3} & I_{3\times3} & \mathbf{0_{3\times3}} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & I_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}, B = \begin{bmatrix} \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} \\ I_{3\times3} \end{bmatrix},$$
$$C = \begin{bmatrix} I_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix},$$

where $I_{3\times3}$ is the 3×3 identity matrix. A trajectory is to be generated for this system to pass through a given set of k waypoints in position, where $k \geq 1$. The set of waypoints consisting of positions in $\mathbb{R}^3$ with respect to an inertial frame, are generated by the method described in previous section. To facilitate numerical computation of the system, the dynamics expressed in (3.10)-(3.11) is discretized in the next subsection.

3.3.2 Discretization of Dynamics

Consider a fixed step size in time, h, and a fixed time interval $[0, T]$ over which the trajectory is to be generated in discrete time. Without loss of generality for the system, the initial time is assumed to be 0. Thus time is discretized as $t_n = nh$ with $T = m_k h$, so that m_k is a positive integer that corresponds to the final time at which the generated trajectory passes through the final waypoint. Let the discrete-time state variable be given by $x_n = x(nh)$, where $n \in \mathcal{N}$ and $\mathcal{N} = \{0, 1, \ldots, m_k\}$. Denote the discrete time instants at which the trajectory passes through the given position waypoints by m_i, $i = \{1, \ldots, k\}$, with $\{m_1, \ldots, m_k\} \subset \mathcal{N}$. The discrete system representation of (3.10)-(3.11) can be obtained as:

$$x_{n+1} = A_d x_n + B_d u_n, \tag{3.12}$$

$$y_n = C_d x_n, \tag{3.13}$$

where

$$A_d = e^{Ah}, \quad B_d = \int_0^h e^{A\sigma} B d\sigma, \quad C_d = C,$$

Due to the nilpotent nature of A, only the first three terms of the exponential series are needed to calculate e^{Ah} exactly. Therefore, the above discretization leads to an *exact discretization* of the continuous time system (3.10)-(3.11). The optimal control problem is formulated and its solution is presented in the next section.

3.4 Optimal Trajectory Generation and Gain Selection

3.4.1 Position Trajectory Through Waypoints

The problem of trajectory generation amounts to constructing a feasible discrete-time desired trajectory through the given set of k waypoints generated by DRL-based algorithm. Let the set of k waypoints be given by tuples $(y^w_{m_1}, m_1), (y^w_{m_2}, m_2), \cdots (y^w_{m_k}, m_k)$, where the time instants corresponding to these waypoints are denoted by the subscript $m_i \in \mathcal{N}$, with $i = 1, \cdots k$. We construct a discrete optimal control problem such that the output y_n passes through the given waypoints in specified time instants, i.e. $y_{m_i} = y^w_{m_i}$, for $i = 1, \cdots k$. Let the initial state be given by $x(0) = x_{init}$. The boundary condition at the end point is determined by the last waypoint, $y^w_{m_k}$. The optimal control problem can be formulated as follows:

Problem 2 (Discrete-time Optimal Trajectory Generation) *Minimize*

$$\begin{aligned}\mathcal{J}^d =& h\sum_{i=0}^{m_k}\frac{1}{2}(x_i^T Q x_i + u_i^T R u_i) \\ &+ \frac{1}{2}\sum_{j=1}^{k}\left(C_d x_{m_j} - y^w_{m_j}\right)^T S\left(C_d x_{m_j} - y^w_{m_j}\right),\end{aligned} \tag{3.14}$$

subject to

1. *satisfying the dynamical model,*

$$x_{i+1} = A_d x_i + B_d u_i, \tag{3.15}$$

2. and the boundary conditions given by,

$$x_0 = x_{init}, \tag{3.16}$$

$$C_d x_{m_k} = y^w_{m_k}. \tag{3.17}$$

Here $Q \in \mathbb{R}^{9\times 9} \geqslant 0$, $R \in \mathbb{R}^{3\times 3} > 0$ *and* $S \in \mathbb{R}^{3\times 3} \geqslant 0$ *are square, symmetric matrices.*

In problem 2, high values of the position, velocity, acceleration and the derivative of acceleration (also known as "jerk"), are penalized. Additionally, at the time instances corresponding to the waypoints, the error between actual position and the desired position waypoint is penalized. The problem 2 can be approached from the first principles of optimal control. Let the augmented performance index be written as,

$$\mathcal{J}^d_a = \mathcal{J}^d + \sum_{i=0}^{m_k-1} \lambda^{\mathrm{T}}_{i+1}(A_d x_i + B_d u_i - x_{i+1}), \tag{3.18}$$

here $\lambda_i \in \mathbb{R}^9$ is a vector of co-states. The optimal control input is found to be (details are removed for brevity):

$$u_i = -[R + (B_d)^{\mathrm{T}} P_{i+1} B_d]^{-1} (B_d)^{\mathrm{T}} (P_{i+1} A_d x_i + \eta_{i+1}). \tag{3.19}$$

This control input generates an optimal, smooth trajectory between waypoints.

Remark 1 *Let* $K_i = [R + (B_d)^T P_{i+1} B_d]^{-1} (B_d)^T$, *then the optimal control can be written as*

$$u_i = -K_i\big((P_{i+1} A_d x_i + \eta_{i+1})\big)$$

After applying the optimal control, the dynamics of the discrete system given in (3.12)

becomes,

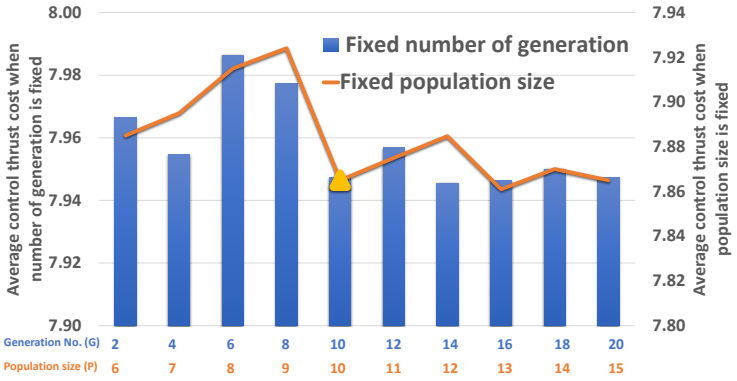
$$\begin{aligned} x_{i+1} &= A_d x_i - B_d K_i (P_{i+1} A_d x_i + \eta_{i+1}), \\ &= \big(A_d - B_d K_i P_{i+1} A_d\big) x_i - B_d K_i \eta_{i+1}. \end{aligned} \tag{3.20}$$

Remark 2 *Throughout all steps, thrust force can be calculated by:*

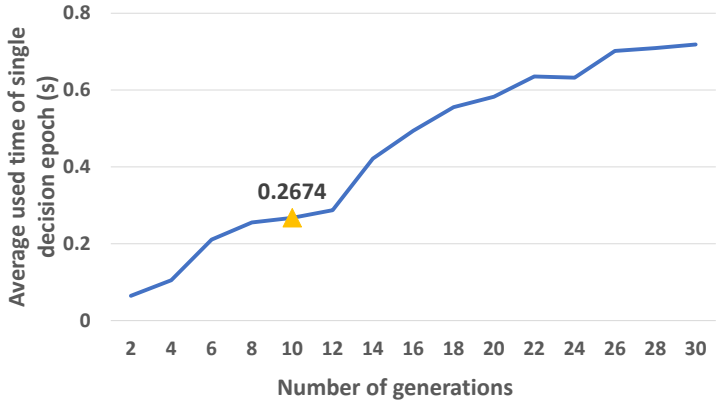
$$f_i = m\|a_i - g e_3\|. \tag{3.21}$$

3.4.2 Gain Selection of Optimal Trajectory Generation

During the trajectory generation, the system gives the acceleration of the UAV while solving Problem 2. Based on (3.9) and (3.6), we derived (3.21) to calculate the control thrust, which is the feedback reward for higher level waypoints planner during learning. It is necessary to mention that in the lower level of our module, optimal trajectory generation scheme, Q, R and S are three positive definite gain matrices. Each of these gain matrices penalize different aspects while generating the trajectory as indicated in (3.14). Q is a matrix penalizing high values of position, velocity and acceleration. The higher the values in Q, the harder these parameters are penalized. The input jerk is penalized when R matrix has large eigenvalues and how much the waypoints will affect the trajectory is weighted by S matrix. From (3.14) we can see that these three gain matrices have significant impact on the performance index $\mathcal{J}^d$ of the trajectory. Their values need to be tuned in order to minimize the control thrust f. In our experiment, f is calculated from the lower level scheme, while it will be measured by sensors in real field learning. We apply Genetic Algorithm (GA) [70] to optimize the gain matrices. The best set of (R, Q, S) satisfies $\arg\min_{R,Q,S} \sigma f(W_i, W_{end}, v_i, a_i)$ where W_i, v_i, a_i are initial position, velocity and acceleration of the UAV and W_{end} are the destination for UAV.



(a) Trade-off among average control thrust cost, number of generations and population size



(b) Relation between number of generations and time consumption for single decision epoch

Figure 3.4: Genetic algorithm utilization analysis

Without loss of generality, we let the gain matrices be identity matrices scaled by different factors. Therefore, each chromosome (i.e. solution) in the population has 3 genes, one for each gain matrix. For our problem, we randomly select 10 chromosomes based on a uniform distribution at the beginning. The negated L1 norm of thrust cost is used as the fitness of each solution. Based on the fitness value, we select the best set of (R, Q, S) within the current population as parents for mating. Next step is to apply GA variants (i.e. crossover and mutation) to produce the offspring, creating new population by appending parents and offspring. In our approach, one point crossover and uniform mutation are adopted. Repeating these steps for several

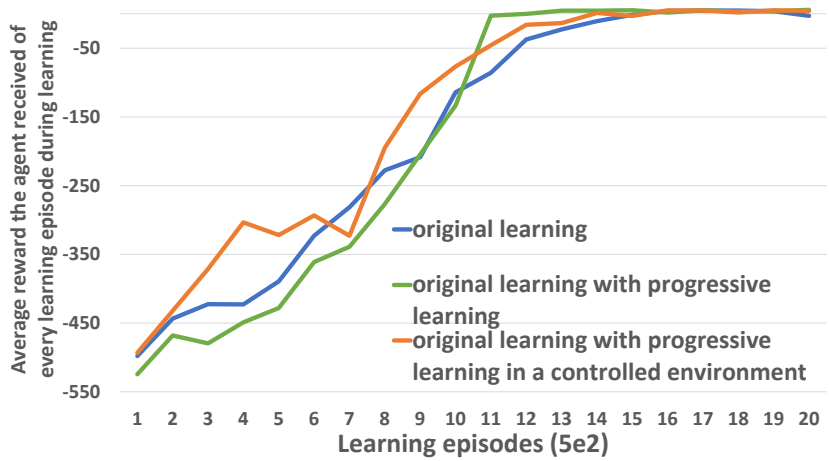
iterations, the returned efficient set of (R, Q, S) results in minimum control thrust when UAV flying from the W_i to W_{end}. As a stochastic optimization algorithm, the more generations and larger population evaluated by the GA, the better solution can be found. Figure 3.4a shows how the cost (i.e. inverse of fitness) reduces as the generation (blue labels) and population size (orange labels) increases. As we can see that the quality of the solution saturated when the size of population and the number of generations are both beyond 10. Figure 3.4b shows that the runtime of the GA is almost a linear function of the number of generations. Based on the above analysis we set the population size to 10 and maximum generations also to 10 indicated as yellow triangle in Figure 3.4.

3.5 Evaluations

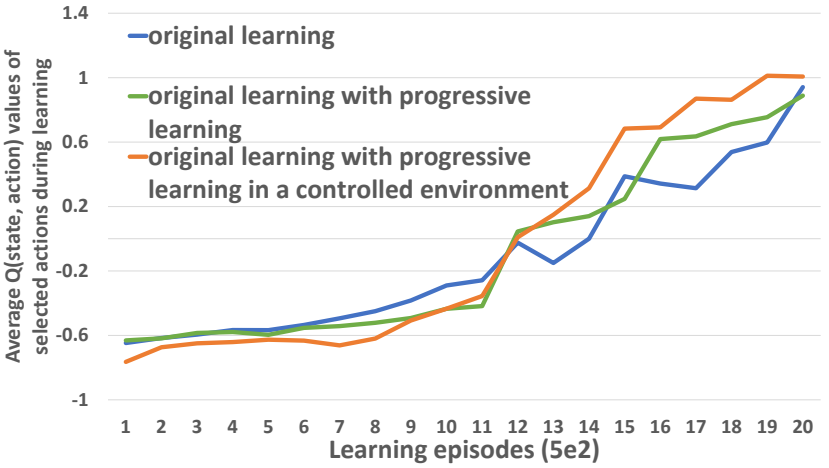
In this section, we demonstrate the performance of our proposed model. The training and testing were done on NVIDIA TitanX (Pascal). In the experiments, the environment is divided into $10 \times 10 \times 10$ and the unit distance δ_d is 10 meters. Within each testing scenario, the number of obstacles is randomly generated and obstacles are placed randomly within the environment boundary. Besides, the start and target positions are also randomly selected. We report the results of two aspects: (1) the improvements achieved by using progressive learning in a controlled environment; (2) the results compared with some existing approaches.

3.5.1 Impacts of Progressive Learning in a Controlled Environment

Figure 3.5a and 3.5b compare the change of reward and Q values of learning with and without gradually increased UAV mobility (i.e. progressive learning) and environment complexity (i.e. controlled environment). The results of traditional learning,



(a) Average reward the agent received



(b) Average $Q(state, action)$ values of selected actions.

Figure 3.5: Learning results of first 10,000 episodes comparisons before and after using improved learning

which adopts none of those two improvements, are represented by blue curves and the results after adopting only the progressive learning are represented by green curves in the figure. The orange curves show the results of applying both progressive learning and controlled environment techniques. To make it clear to see, we show the result of first 10,000 episodes with average of every 500 learning episodes. With the help of progressive learning, the learning converges after 7,500 episodes. It is 16.67% less compared to the traditional learning that converges after 9,000 episodes. Because of the reduced mobility, an episode at the beginning of progressive learning is much

Table 3.1: Average selected waypoints number comparison over different distances

Normalized distance from start to destination	(0,4]	(4,8]	(8,12]	(12,16]
traditional learning	3.43	6.49	12.11	13.52
improved learning	3.40	6.43	11.88	13.42

Table 3.2: Average control thrust cost comparison over different distances.

Normalized distance from start to destination	(0,4]	(4,8]	(8,12]	(12,16]
traditional learning	1.4321	3.5951	6.2375	6.2459
improved learning	1.3479	3.4859	6.0139	6.1138

shorter than an episode in the traditional learning. Therefore, the reduction in computing time is even more. It uses around 10 hours total learning time with both improvement techniques which has 47.4% reduction in learning time. From Figure 3.5b we can see that the progressive learning and controlled environment not only speed up the convergence, the quality of learning is also better because the Q values are more stable than that of traditional learning.

To evaluate the quality of learned model, we generated thousands different testing scenarios and divide the flight of the UAV into four groups: (1) the UAV collides into obstacles without reaching target; (2) the UAV collides into obstacles but eventually reaches the target; (3) the agent does not collide into obstacles neither does it reach the target; (4) the UAV does not collide into obstacles and it successfully reaches the target. If the UAV travels 100 steps without reaching the target, it is regarded as a failure. Only type (4) flights are considered as successes. We refer both type (4) and type (2) as achieved as they both achieved the goal, i.e. reaching the target. Figure 3.6 shows how the success rate and achieve rate improved after using the progressive learning and controlled environment. As indicated in the figure, the success rate increases from 93.8% to 95.8% and the achieve rate increases from 94.0% to 96.0% after optimization.

With respect to the number of steps taken and control thrust consumption during

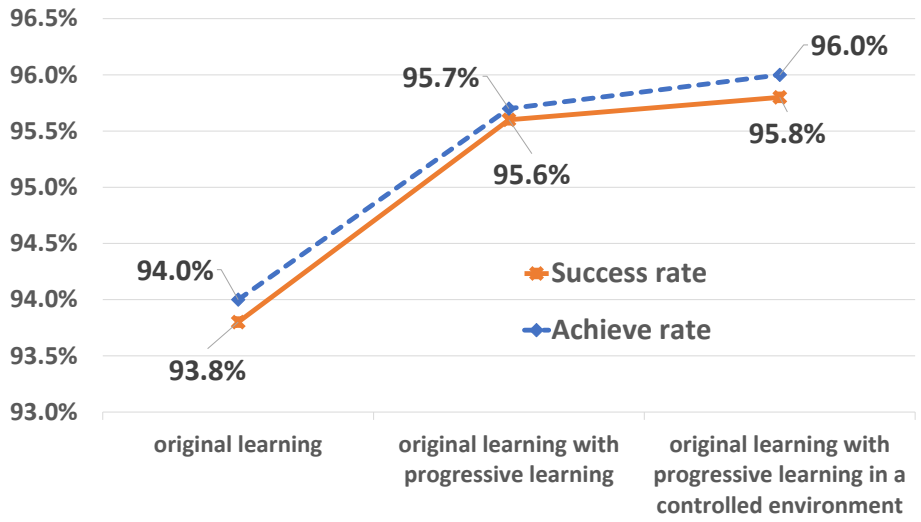
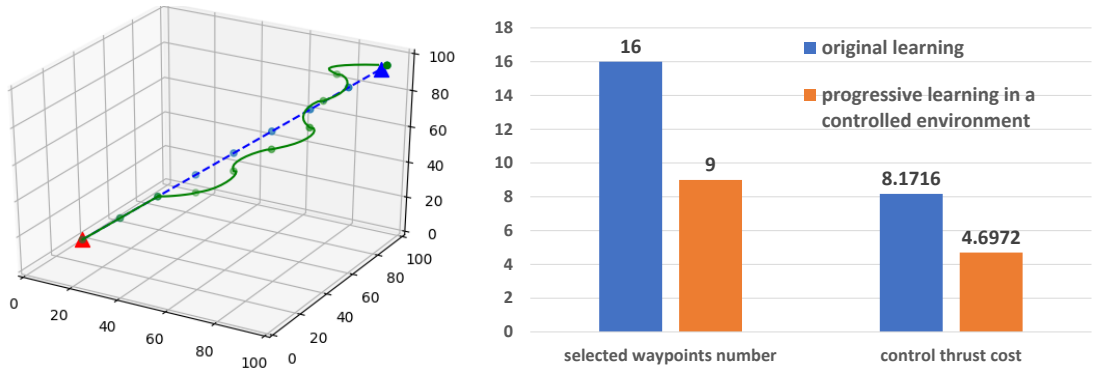


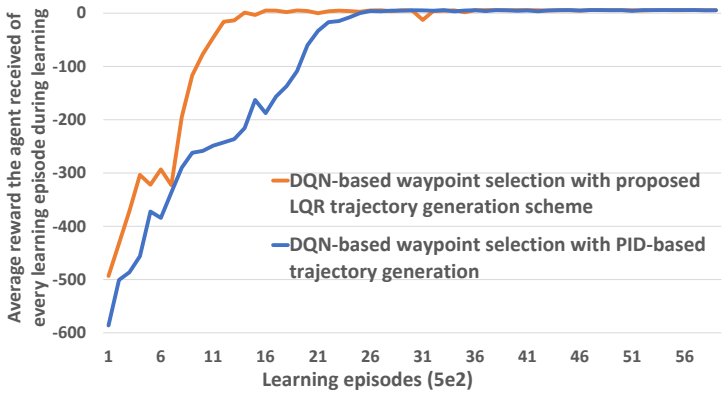
Figure 3.6: Success rate and achieve rate improvements using progressive learning in a controlled environment techniques. Situation (4): the agent reaches the target position and does not collide into obstacles

the flight, progressive learning in a controlled environment can also lead to improvement. Table 3.1 compares the number of waypoints generated by the traditional learning and the improved learning over different distances. The less number of waypoints means the fewer steps for the UAV to fly to the target. In general the UAVs learned using the improved learning need less number of steps. The reduction of waypoints usage becomes larger as the distance between start and target increases. Table 3.2 compares the average thrust cost if the UAV follows the trajectory, To illustrate it clearer, an example of a scenario in an obstacle-free environment is shown in Figure 3.7, which compares results of model trained in traditional way and one trained with controlled environment technique. In the example, the environment has the same configuration, i.e. same start (i.e. red triangle) and target position (i.e. blue triangle). The green curve indicates the trajectory along waypoints generated by traditional learning, and the blue curve shows the trajectory along waypoints generated by improved learning.

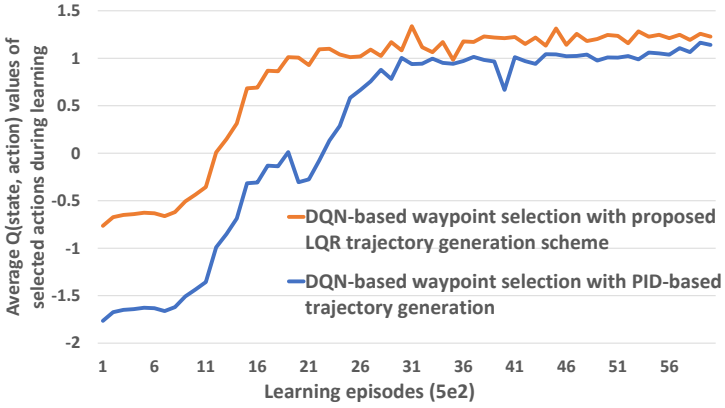


(a) (b)

Figure 3.7: (a) Example of trajectory generation through waypoints selected by traditional learning (*green*) and improved learning (*blue*) respectively (b) Comparisons tracking number of selected waypoints and thrust cost during generation



(a) Average reward the agent received every learning episode



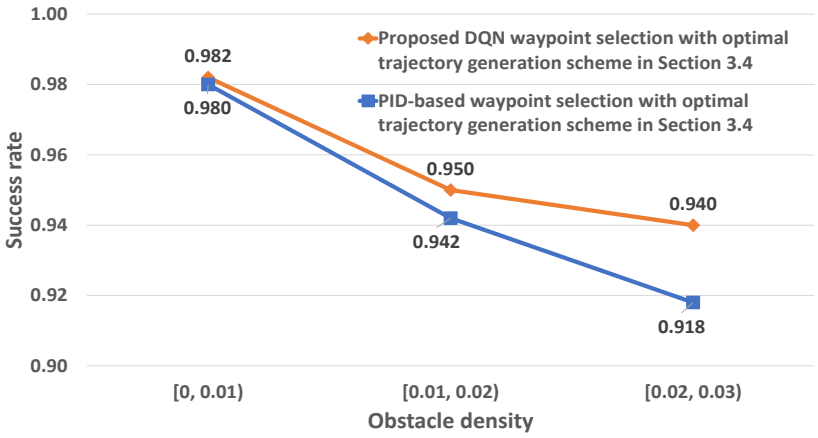
(b) Average $Q(state, action)$ value of selected actions

Figure 3.8: Learning results of proposed DQN waypoints selection together with proposed energy efficient trajectory generation scheme and PID-based trajectory generation baseline respectively

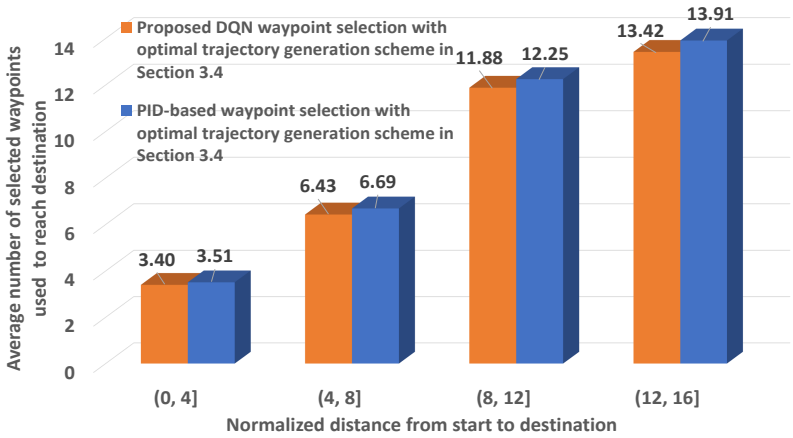
3.5.2 Results Comparison

As a baseline of reference, we also implement a control scheme based on PID theory [71] to estimate the control thrust consumption along waypoints, and use it to replace energy efficient trajectory generation of Section 3.4 scheme as the lower level optimizer. We define position as measurable variables and velocities when reaching each waypoints are regarded as controllable variables. Every time the position of the agent is updated by the feedback of the environment, and the feedback is calculated by the environment simulation based on Kinematic theory [72]. Figure 3.8 shows the learning results of DQN when our proposed trajectory generation scheme or PID are used in the lower level. As indicated in the figure, the learning of our scheme converges 44% faster than using PID, because the proposed energy efficient trajectory generation scheme will not over correct the trajectory and there is no control latency, hence its performance is more stable and predictable. Also the DQN with efficient trajectory control achieves higher reward, i.e. it consumes less control thrust, because of higher fidelity of the proposed energy efficient trajectory generation scheme in Section 3.4. Figure 3.9 (a)$\sim$(c) reports the comparisons of success rate, the average number of selected waypoints to reach the target and the average control thrust cost to go through these waypoints. The comparisons are divided into four groups based on the Euclidean distance between start and target position.

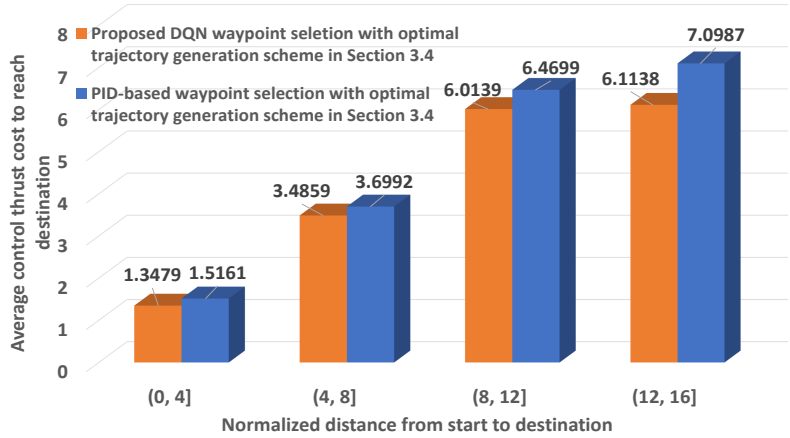
Since the critical roles of gain matrix in proposed energy efficient trajectory generation scheme in Section 3.4, it is crucial to improve the performance by tuning gain matrices. The control thrust of UAVs with and without optimized grain matrices is compared in Figure 3.10. The average thrust cost of using genetic algorithm is indicated as blue labels, while orange indicates the result of manually selecting gain matrices. As shown in the figure, the average control thrust consumption decreases from 11.0531 to 10.9740 in a $30 \times 30 \times 30$ discretized environment block. Figure 3.11a gives an example of trajectories without gain optimization (blue), with medium op-



(a) Comparison tracking success rate of proposed DRL with proposed energy efficient trajectory generation scheme and with PID baseline



(b) Average selected waypoints number to reach destination



(c) Average control thrust cost along selected waypoints

Figure 3.9: Tracking results comparisons of different trajectory generation schemes, i.e. proposed energy efficient trajectory generation scheme (orange) and PID baseline (blue), with the same proposed DQN waypoints selection process

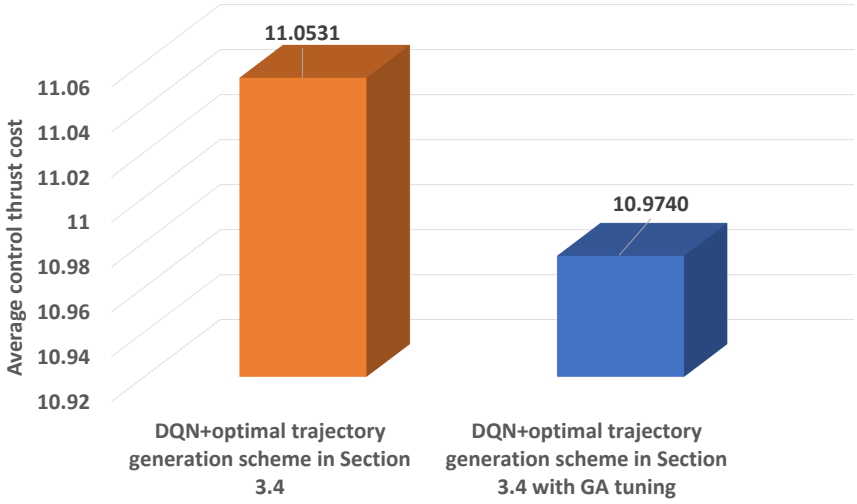
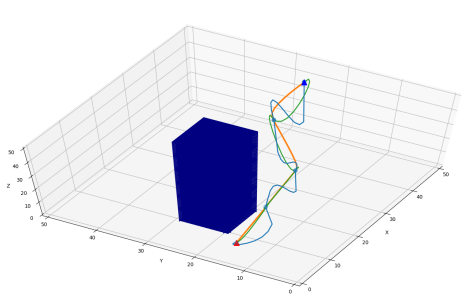
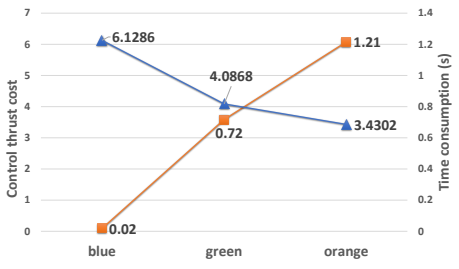


Figure 3.10: Improvements achieved with tuning gain matrices of efficient trajectory generation scheme in Section 3.4 using genetic algorithm

timization (green) and with heavy optimization (orange). The corresponding control thrust and optimization time is given in Figure 3.11b. In the example, four waypoints (blue dots) are selected to reach the target. The start and target points are shown in red and blue triangles respectively and the cylinder represents obstacles. The blue curve shows the trajectory generated with fixed R, Q and S, which has large overshoot and some sharp curves. The control thrust for the blue trajectory is 6.1286 force cost as indicated in Figure 3.11b. If we apply GA between current waypoints W_i and next waypoint W_{i+1} to select the efficient set of R, Q and S just for each segment, the

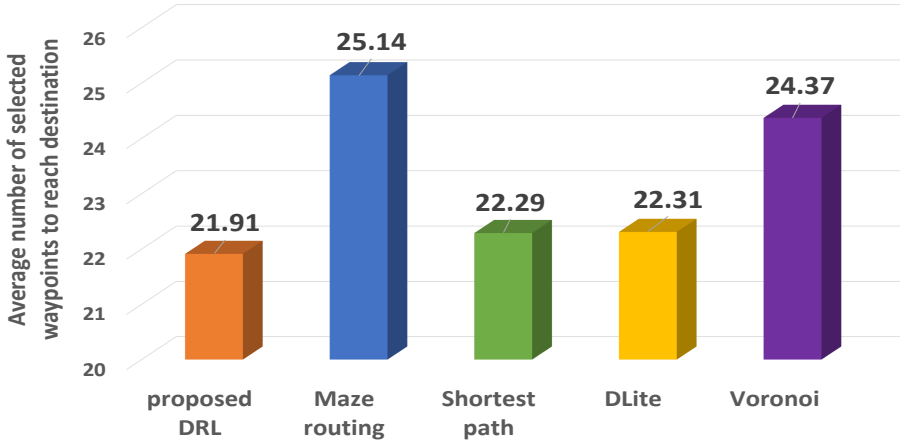


(a) Example trajectories

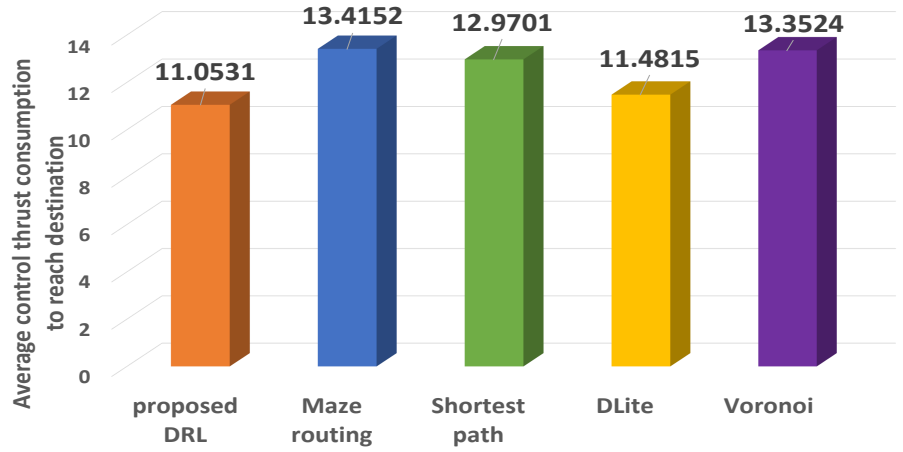


(b) Control thrust cost and time consumption comparison

Figure 3.11: Example of trajectory generation without and with gain matrices tuning by genetic algorithm. Blue: trajectory without gain optimization; Green: trajectory with medium gain optimization; Orange: trajectory with heavy gain optimization



(a) Average number of selected waypoints to reach destination



(b) Average control thrust cost along trajectory generated through waypoints

Figure 3.12: Results comparison between proposed DQN scheme, routing, shortest path, DLite and Voronoi

control thrust cost reduces to 4.0868 (i.e. green curve), after 7 generations of GA search. The control thrust cost further reduces to 3.4302 after 10 generations of GA search (i.e. orange curve). And the orange line in Figure 3.11b gives time cost for the GA optimization.

Finally, we compare the DRL based waypoints selection with four traditional waypoints selection approaches in aspects of average number of steps needed to reach target, and the average control thrust cost along the trajectory. These four approaches include maze routing [73], shortest path [74], DLite algorithm [75] and voronoi path

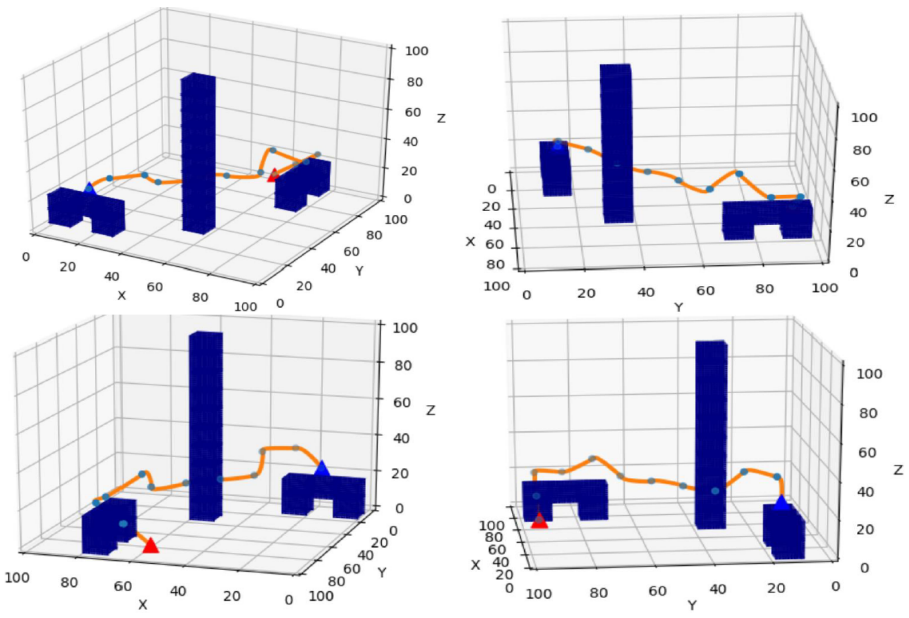


Figure 3.13: An example of autonomous waypoints planning and trajectory generation using proposed framework

[76]. To make it more convincing, the size of discretized environment is set as $30 \times 30 \times 30$. We generated 1000 different test scenarios by randomly select different start positions, destination positions, types and locations of obstacles. Figure 3.12 compares the average number of selected waypoints to reach destination and the average control thrust cost for the UAV to go through these waypoints. For all waypoints selection approaches,efficient trajectory generation scheme in Section 3.4 is used for trajectory generation. In Figure 3.12a, the results show that our approach only needs an average 21.91 waypoints which is 6.6% less than other approaches. In Figure 3.12b, the comparison of average control thrust consumption is reported. As indicated in the figure, our approach use least control thrust which has 13.33% reduction than other approaches. According to these results, our proposed scheme with fewer selected waypoints is less possible to be over constrained and less times of lower level scheme invocation is needed.

A more realistic scenario is given in Figure 3.13. In this scenario, a door is set as the start position which is indicated by a red triangle in the figure. The UAV takes off

from the door, and then land on the center of the table near the door first. After that, it takes off again to reach the final destination which is indicated by blue triangle in the figure. As shown in the figure, blue dots are selected waypoints provided by our proposed DQN scheme. And the smooth orange curve shows the trajectory generated by efficient trajectory generation scheme proposed in Section 3.4. The UAV does not collide into obstacles during the flight. In order to display the trajectory clearly, we show four different views of the 3D trajectory plot.

3.6 Conclusion

A two-level framework to generate navigation trajectory for UAVs to follow in a complex environment is introduced. The framework's construction, processing and analysis are presented. The proposed waypoints planning and trajectory generation framework effectively avoids obstacles in complex indoor environment and reduces the control thrust consumption during flight. Also, it is general enough to be applied in other robotics tasks such as parcel delivery and conflicting routing of high-density UAVs [77].

Chapter 4

Fast and Accurate Trajectory Tracking for Unmanned Aerial Vehicles based on Deep Reinforcement Learning

4.1 Introduction

Recently the applications of UAVs have been widely used in numerous real world applications where human operations are limited. With the increasing of data volume and accuracy requirements for practical applications, the reliable operations of UAV, i.e. the stable autonomous guidance and control, have been considered as one of the most critical. Efficient tracking algorithms enable a smooth trajectory and hence a lower system power/energy dissipation during the flight. Traditionally, the PID control mechanism is the state-of-the-art choice for industrial UAV trajectory tracking system. PID controllers are easy to be implemented on FPGA and sufficient for many

This work is partially supported by the National Science Foundation under Grant CNS-1739748.

control problems. They work well when process dynamics are benign and the performance requirements are modest [78][79]. However, the PID controller cannot treat processes with large time delay efficiently and it shows poor performance for tracking problems requiring aggressive dynamic configurations, including uncertain internal disturbance compensation and imbalances retrieval [80][81]. For some applications, modified PID models implemented have been explored to improve the performance [82][83].

Meanwhile, it is a big challenge to control UAVs stability in general using low power cost platforms, especially under uncertain disturbance from environments. The main reason is that it is hard to obtain a high fidelity mathematical model of a UAV which has an under-actuated system with nonlinear dynamics [84]. To improve the stability and real-time control, DNN embedded on different hardware platforms are introduced [85][86]. Through large data training, the DNN-based control system achieves adaptability and robustness that guarantee the stability of the flight [87]. Additionally, the controllers are able to follow the desired trajectory with the tolerance of unexpected disturbance. Similar to PID controllers, the DNN based controller estimates the control actions based on past flight experience to reduce the instantaneous tracking imperfections. None of them considers how the chosen action will affect the subsequent rewards. Hence, they are likely to generate suboptimal solutions.

RL provides a mathematical framework for learning or deriving policies that map situations (i.e. states) into actions with the goal of maximizing an accumulative reward [88]. Unlike supervised learning, in RL the *agent* (i.e. learner) learns the *policy* for decision making through interactions with the *environment*. The aim of the agent is to maximize the cumulative long-term *reward* by taking the proper action at each time step according to the current *state* of the environment while considering the trade-off between explorations and exploitations. Q-learning is one of model-free RL strategies storing finite state-action pairs and corresponding Q-values in a

look-up table and it has been applied for thermal and energy management in autonomous computing systems [89][90]. The combination of conventional Q-learning and deep neural network, i.e. Deep Q-network [9], provides a breakthrough in DRL. The neural network in DQN needs to accumulate enough samples of values and the data needed for its training can either come from a model-based simulation or from actual measurement [12]. Originally developed by DeepMind, the DRL provides a promising data-drive, adaptive technique in handling large state space of complicated control problems [11]. The actor-critic deep reinforcement learning [62] has overcome difficulties in learning control policies of systems with continuous state and action space, which provides a potential solution for effective real-time mission control of autonomous UAVs.

In this work, we propose an actor-critic DRL model to track trajectories of UAVs through sets of desired waypoints, and its implementation using FPGA. The detailed framework is discussed in Section 4.2. Based on the model provided by [1], we aim at actuating one degree of translation motion and three degrees of rotation motion for quadrotor body-fixed UAVs. We construct a fully-connected neural network to learn the optimal tracking policy based on DRL. We choose different sets of desired waypoints as test benchmarks. The experimental results in Section 4.3 show that compared to the baseline, our proposed approach can achieve 58.14% less position tracking error and 9.23% faster attainment. The efficient tracking leads to 21.77% power saving during the flight time. In addition, our actor network can easily be mapped to FPGAs for hardware acceleration and the input/output size of network is constrained because of limited dimensions of state/action that an agent/environment can physically process. With a low-cost FPGA, one single decision can be made within 0.0002 second at only 0.03mW power consumption in a decision epoch. The speed and power consumption allows the proposed actor-critic framework to be used for real-time on-board control of autonomous systems.

4.2 System Architecture and Hardware Design

4.2.1 Problem Formulation

In this work, we consider the trajectory tracking for under-actuated aerial vehicles through a set of given desired waypoints. To state the problem without losing generality, we aim at quadrotor fixed-wing UAVs with four control inputs, one degree of translation motion and three degrees of rotation motion (i.e. pitch, roll and yaw.) The directions of four motions are illustrated in Figure 4.1 using different colors (yellow: *translational freedom*; green: *rotation freedom*). It is extremely difficult to integrate detailed mechanistic model of complicated dynamics with classic control theory, a model-free solution for the control problem is preferred. Because actions of the tracking problem are continuous variables (e.g. turning force, thrust etc.), the actor-critic reinforcement learning is adopted, which learns to find the optimal set of actuations that move the UAV towards desired trajectory. The technique presented in [1] is used to generate C^2 trajectory based on a given set of predefined waypoints T_d where each waypoint gives the desired position of the UAV at time t. Meanwhile, desired velocity vd_t, desired acceleration dvd_t and desired attitude Rd_t are extracted from T_d based on kinematics. The positions and attitudes together form the pose of the UAV. The goal of our model is to minimize the differences between desired poses and actual poses during tracking. We define $sd_t = \{pd_t, vd_t, dvd_t, Rd_t\}$ as desired state and $s_t = \{p_t, v_t, dv_t, R_t\}$ as actual state of UAV at time step t. Each of first three variables in the sd_t and s_t are 3-dimensional vectors and the last one in the sd_t and s_t is 3×3-dimension, hence the desired state and the actual state are variables in 18-dimensional space. All variables used at time t are summarized in Table 4.1.

The concatenation of sd_t and s_t forms the agent state $\mathbb{S}_t$. Furthermore, we define action at time t as $\mathbb{A}_t = \{fm_t, \tau_t\}$, where the translational force fm_t and torques τ_t are applied to UAV. The translational force fm_t is a force perpendicular to the

Table 4.1: Variables summary

pd_t : desired position	p_t : achieved position
vd_t : desired velocity	v_t : achieved velocity
dvd_t : desired acceleration	dv_t : achieved acceleration
Rd_t : desired attitude	R_t : achieved attitude
sd_t : desired state	s_t : achieved state
fm_t : applied force	τ_t : applied torques
$\mathbb{A}_t$: applied action	$\mathbb{S}_t = \{sd_t, s_t\}$: agent state
$\mathbb{E}$: environment simulation	

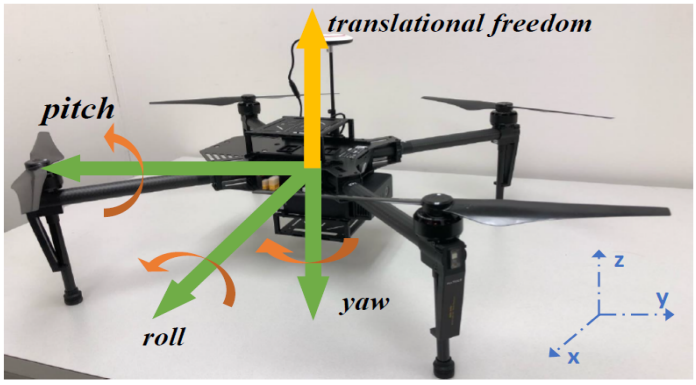


Figure 4.1: Illustration of UAV control inputs

top surface of UAV and the three components in τ_t are applied to roll, pitch and yaw directions respectively. The reward $Reward(\Delta_t)$ at time t is defined as the Manhattan distance between the desired pose and the actual pose, i.e. $Reward(\Delta_t) = f(|p_t - pd_t| + |v_t - vd_t| + |R_t - Rd_t|)$.

4.2.2 Network Structure

Since the control variables (i.e. fm_t, τ_t) of UAV are in a continuous space which are infinitely large, we build an actor-critic reinforcement learning model instead of discretizing the action space. The actor model is a feed-forward deep neural network of three fully-connected hidden layers with Rectified Linear Units (ReLU) as the activation function. It is used to predict the optimal action based on current state $\mathbb{S}_t$. The number of neurons in fully-connected hidden layers are 64, 128 and 128 respectively. The size of output layer is 4 and LeakyReLU activation function is used

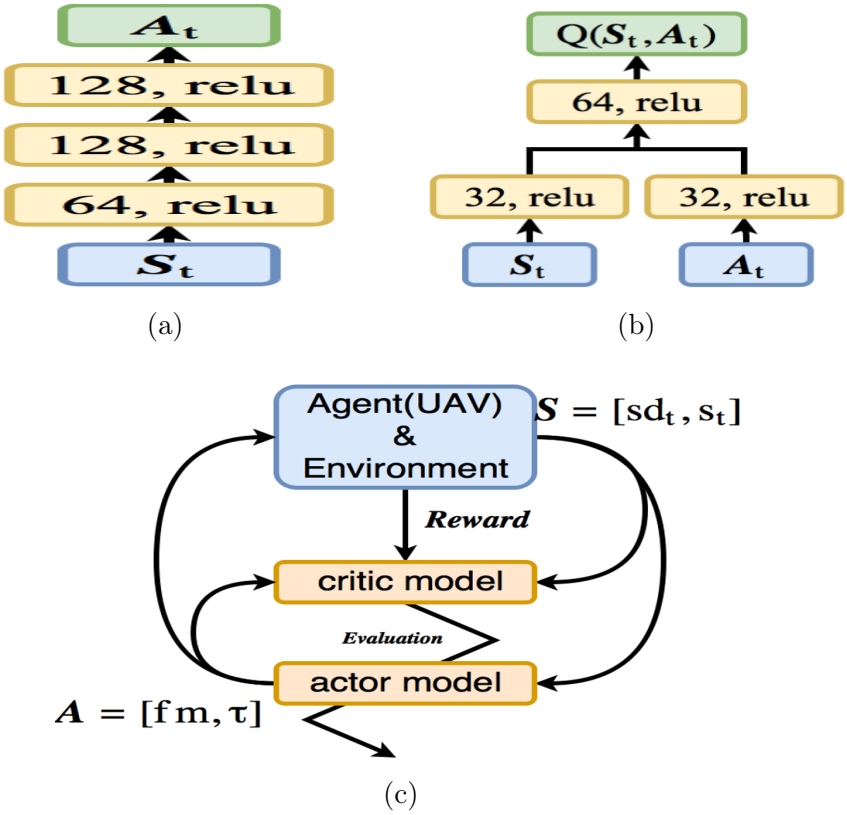


Figure 4.2: Architecture details (a) actor model, (b) critic model and (c) overall framework

in the output layer. The critic model is another feed-forward neural network that computes an evaluation of the action and that evaluation is used by actor model to update its control policy in particular gradient direction. The critic model has two hidden layers, where the first layer contains two separate fully-connected structures and the number of hidden neurons in each is 32. The addition of outputs from the first hidden layer is fed into the second layer which has 64 hidden neurons. The inputs of critic model are $\mathbb{S}_t$ and $\mathbb{A}_t$ and the output is a single value $Q(\mathbb{S}_t, \mathbb{A}_t)$. The size of the critic and actor is optimized as hyper-parameters through cross-validation. The detailed networks of actor model and critic model are shown in Figure 4.2a and Figure 4.2b. The overall framework is shown in Figure 4.2c. During training, the actor model is pre-trained using labeled pair data $(\mathbb{S}_t, \mathbb{A}_t)$ generated from simulation [1] to predict the optimal action $\mathbb{A}_t$ based on current agent state $\mathbb{S}_t$. Next agent state

$\mathbb{S}_{t+1}$ is calculated through environment simulation based on $\mathbb{A}_t$ and is used to predict optimal $\mathbb{A}_{t+1}$ by actor model. The critic model evaluates the resulting $\{\mathbb{S}_{t+1}, \mathbb{A}_{t+1}\}$ pair by predicting a Q-value to fine-tune action prediction. Therefore, the weights in actor model are updated by the gradient between actor and critic model, using chain rule $dQ/dW_{actor} = dQ/dW_{critic} \times dW_{critic}/dW_{actor}$. W_{actor} and W_{critic} indicate the weights of *actor* and *critic* models respectively.

4.2.3 Reward Definition

Our goal is to actuate the UAV to be closer to desired pose T_d along the desired trajectory, i.e. minimizing the value of $\Delta P_t = |p_t - pd_t|$. Besides position error, the stability of the UAV should also be taken into consideration. Therefore the values of velocity errors (i.e. $\Delta V_t = |v_t - vd_t|$) and attitude errors ($\Delta R_t = |R_t - Rd_t|$) must also be minimized. However, our experiments show that simply using a linear combination of ΔP_t, ΔV_t, and ΔR_t as the reward function will make convergence difficult in learning process. According to [91], using geometrically discounted reward will prevent the accumulated reward to become infinite and make the model more tractable. Therefore, we define the reward at each time step following a standard normal distribution, guaranteeing the largest reward is accepted when the total differences between desired trajectory and actual trajectory at time t (i.e. $\Delta_t = \Delta P_t + \Delta V_t + \Delta R_t$) reaches zero and closing to zero reward is obtained when Δ_t increases. The total discounted reward is denoted as $\mathbb{R}$.

$$\begin{aligned} Reward(\Delta_t) &= \frac{1}{\sqrt{2\pi}} exp(-\frac{\Delta_t^2}{2}), \\ \mathbb{R} &= \sum_{t=0}^{\infty} \gamma^t Reward(\Delta_t) \ , \end{aligned} \tag{4.1}$$

where $\Delta_t = \Delta P_t + \Delta V_t + \Delta R_t$.

4.2.4 Proposed Hardware Configuration

Figure 4.3 shows an overview of hardware design of the proposed UAV controller and its connections. Intel (Altera) Cyclone V 5CGX FPGA is selected as our processing platform and all massive parallel computations are implemented on it. We select this FPGA because of its light weight as the UAV payload, relatively high compute capability, and its low energy cost.

Although there are only linear computations (i.e. multiplications and additions) are need in actor model and the mapping to FPGA is straightforward, two issues need to be addressed. Firstly, all computations cannot be done at once due to resource limitation of FPGA, time-multiplexing is essentially required. Secondly, computation latency introduced by time-multiplexing conflicts with the real-time response requirement for UAV control. Our proposed design aims at exploiting the hardware resource to improve parallelism and to minimize latency, taking two issues mentioned above into consideration. In our design, DSP blocks are used as multiplier. Layer-wise computation is done by computation array, which consists of multiple parallel computation units for multiplication and accumulation performance. Results of intermediate layers are buffered in register array and will feed back to the input of computation array for the computation of next layer. DNN controller builds the communication between FPGA and ARM processor. Resource allocator schedules the time-multiplexed computation. The FPGA is connected to the on-board ARM Cortex-A9 processor. ARM Cortex-A9 takes control commands from FPGA as input, calculates actuations in each freedom, and then sends actuations to UAV controller. In addition, UART handles the communication between ARM Cortex-A9 and UAV. It accepts flight state and sensor data from UAV and sends data to FPGA after preprocessing.

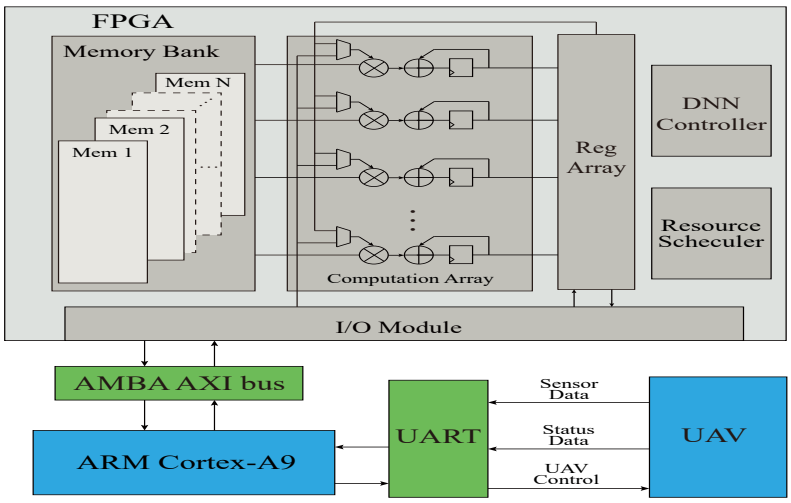


Figure 4.3: Hardware configuration of the UAV controller

4.3 Evaluations

4.3.1 Environment Setup

We trained the actor-critic network and implemented the simulation on Nvidia GeForce GTX1070 using Keras [92]. The data that we use to train and test our model is generated using simulator described in [1]. Our dataset consists one thousand different 3D trajectories of four different shapes, including straight lines, z-shape curves, spiral curves and circles. Each desired trajectory has one thousand desired waypoints, giving enough time for UAV to track it. All desired waypoints are defined by mathematical equations parameterized by time. Eight hundred different trajectories of four different shapes are evaluated in total. The mass of UAV used is $4.34kg$, and its inertial properties $\mathcal{J}$ is a 3×3 diagonal matrix (i.e. $diag[0.820 \quad 0.0845 \quad 0.1377]kgm^2$) which determines the required magnitude of force to accelerate the UAV in each rotation direction respectively. The goal is to minimize the power consumption and time used from trajectory deviated to tracked.

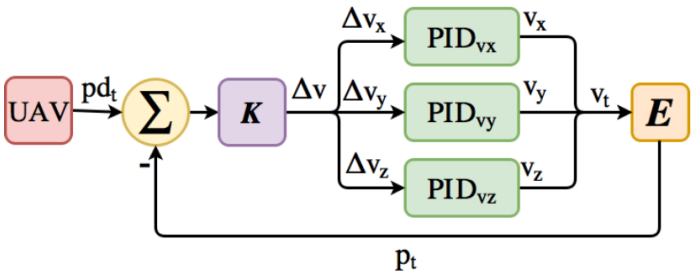
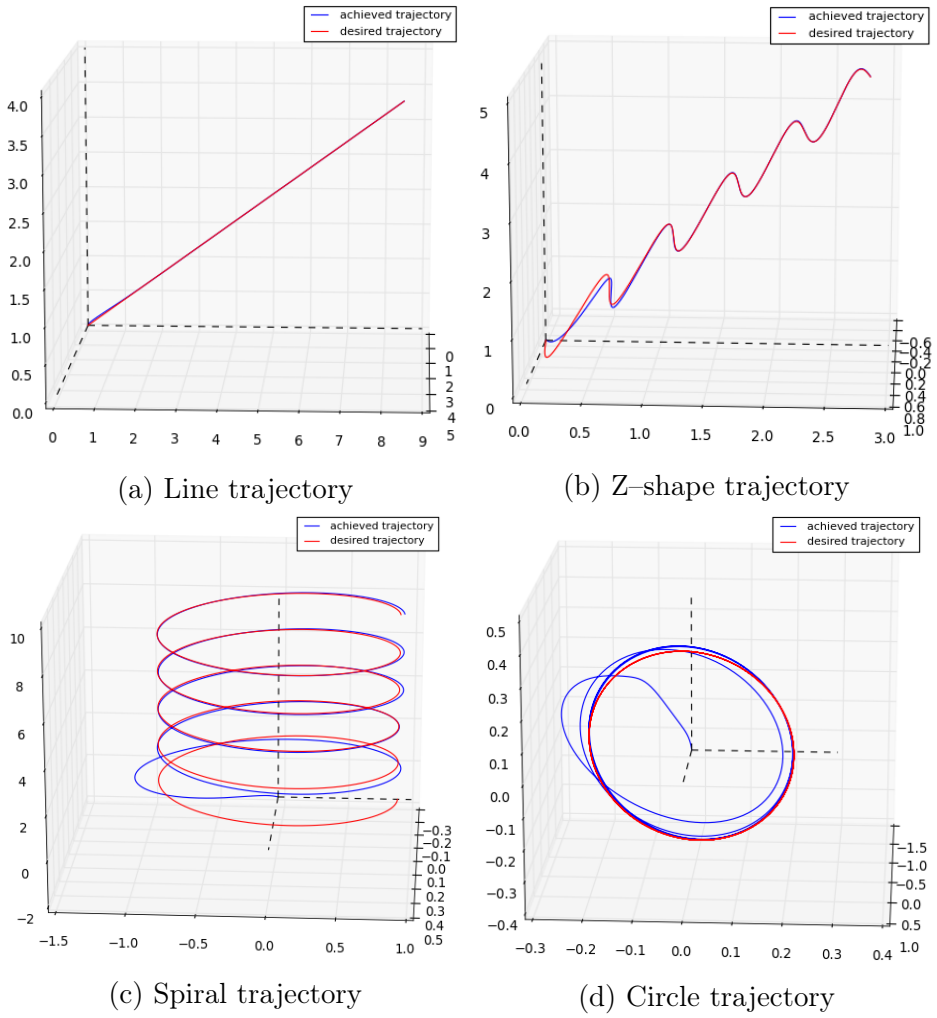


Figure 4.4: Structure of PID-based baseline controller

4.3.2 PID Implementation

As a baseline approach, we also implement control scheme based on PID theory because it has been widely used in industrial applications and real-time control situations. We define desired position pd_t as observable variables (OV) and actual position p_t as measurable variables (MV). Velocities are regarded as controllable variable (CA). The data of pd_t is gathered from simulator in [1] and is used to derivate desired velocity vd_t of UAV at time t. Three PID controllers are used for each component of velocity respectively, which are indicated by green blocks in Figure 4.4. The inner structure of each PID is the same as conventional structure. PID controllers calculate errors of each component between desired velocity and achieved velocity by the UAV, i.e. errors between vd_x and v_x, vd_y and v_y and vd_z and v_z, continuously as references to give velocity corrections. Furthermore, the achieved position p_t is calculated through Environment simulation (i.e. orange block) and used as feedback based on current velocity v_t. As a consequence, the difference between desired position pd_t and actual position p_t is used to update the velocity of UAV through Kinematics block (i.e. purple block). The overall structure of PID-based controller forms a feedback loop and is shown in Figure 4.4



(a) Line trajectory

(b) Z–shape trajectory

(c) Spiral trajectory

(d) Circle trajectory

Figure 4.5: Examples of achieved trajectories and desired trajectories in terms of four different shapes. (*red*: desired trajectories. *blue*: achieved trajectories using proposed DRL-based learning approach.)

4.3.3 Results Comparison

All test samples of desired trajectories are generated the same way as mentioned in Section 4.3.1. Figure 4.5 shows sample testing achieved trajectories (blue curves) using proposed DRL learning approach and corresponding desired trajectories (red curves). We report the results from four aspects: (1) L1-norm of position tracking error; (2) L1-norm of velocity tracking error; (3) Time used to complete tracking; (4) Power consumption.

Figure 4.6 shows the $L1$-norm of position tracking error, where the first four

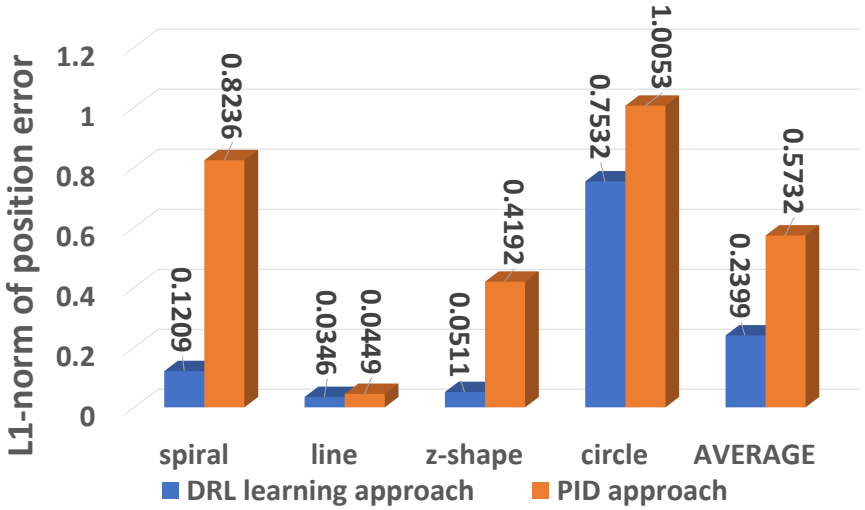


Figure 4.6: Tracking result comparison between proposed DRL-based framework and PID-based baseline in terms of $L1$-norm of position error

columns are comparison results of trajectories with different shapes respectively and the fifth column is average $L1$-norm of position error of all testing trajectories. According to the figure, our approach has lower average position error especially when the trajectory is more complicated. Compared to PID based baseline control, 22.94% less position error is achieved for straight line trajectory tracking and 25.07% less position error is achieved for circular trajectory tracking. The position tracking error reduction increases to 85.32% for spiral shape tracking and 87.81% for z-shape trajectories tracking respectively. On average, our approach outperforms 58.14% better than PID based control in position tracking of different shapes of trajectories.

Figure 4.7 shows the average of $L1$-norm of velocity tracking error. Similar to Figure 4.6, the first four columns are comparison results of each type of trajectory with different shapes, and the last column shows overall $L1$-norm of velocity tracking error averaged over all testing trajectories. It shows 29.79% error reduction is achieved for line trajectory and 67.22% error reduction is achieved for circle trajectory. Up to 91.64% and 93.17% error reductions are achieved for z-shape and spiral trajectories. On average, our approach outperforms 80.60% than PID control with respect to velocity tracking error. Again our learning based approach performs better for the

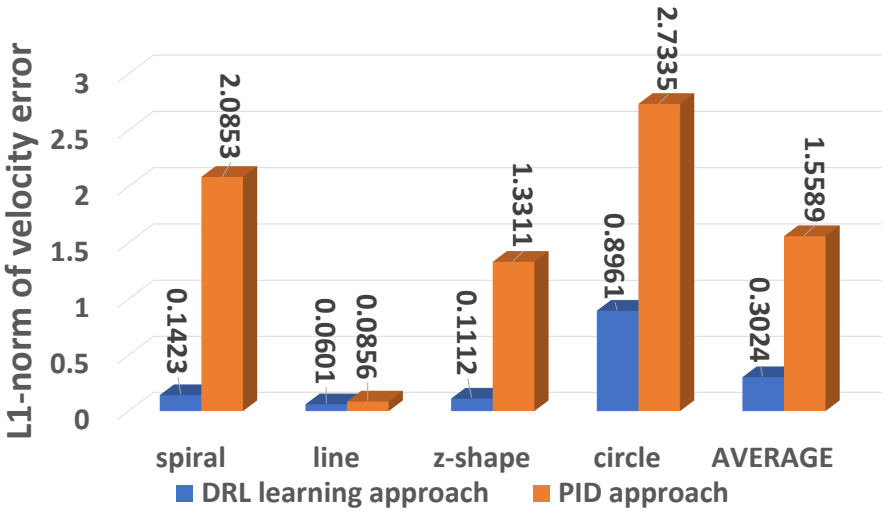


Figure 4.7: Tracking result comparison between proposed DRL-based framework and PID-based baseline in terms of $L1$-norm of velocity error

more complicated trajectories.

Figure 4.8 compares the total time steps used for UAV to follow each shape of desired trajectory and the average time steps used in total to reach stability. We report the number of time steps used for UAV to completely track the desired trajectory. The total time steps for each testing trajectory is one thousand. The time step when trajectory is tracked is regarded as the time t_c after which the $L1$-norm of position error between pd_t and p_t is less than 0.0001. The average tracking time for different types of trajectories, and the average tracking time over all testing trajectories are reported. Our approach is 9.23% faster than PID-based control to achieve stable pose on average. It is especially 13.86% faster for line trajectory and up to 15.58% faster for z-shape trajectory. Moreover, PID has lags in responding current dynamics in all three directions of velocity. Therefore, PID-based controller is not optimally adapted for non-linearity situations, especially not robust in fast dynamic control. It trades off the control performance and time.

Figure 4.9 reports average total power consumption after one trajectory is completely tracked using our approach and baseline PID controller. As indicated in the figure, our approach achieves 11.04% less power consumption when tracking z-shape

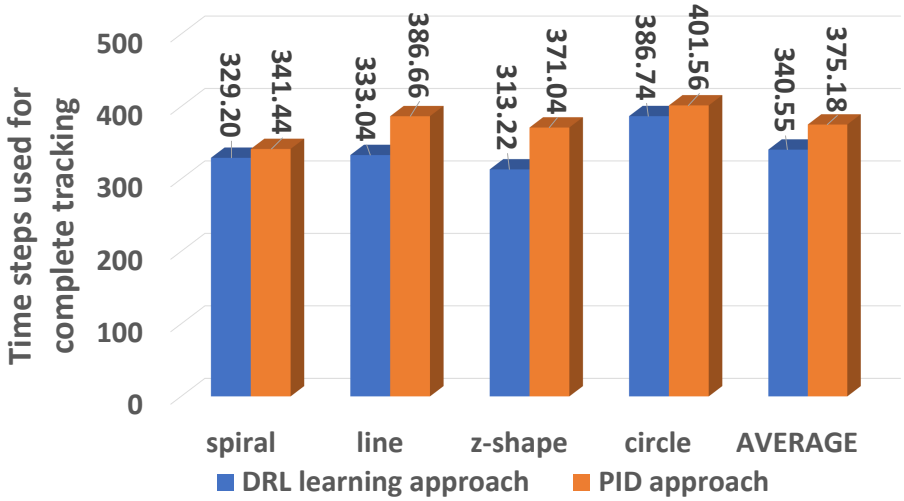


Figure 4.8: Tracking result comparison between proposed DRL-based framework and PID-based baseline in terms of used tracking time steps

trajectory and consumes 18.64% less power for line trajectory tracking. Furthermore, up to 21.63% and 29.11% power consumption improvements have been achieved for circle and spiral trajectories tracking respectively. An average of 21.77% less power is consumed using our approach for tracking all different trajectories. The noticeable result explains that PID control consumes more power because of oscillations of controllable variables during tracking process.

4.3.4 Tracking in a Noisy Environment

To test the robustness of the learning based trajectory tracking, we also add different levels of random Gaussian noise and see if the tracking algorithm can adapt to the changing environment. Our model free DRL approach is compared to the Matlab model based trajectory tracking approach, which calculates the force and torque of the UAV using a 3-dimensional Euclidean space mechanics model in the form of Lie Group Variational Integrator (LGVI) given in [1]. The random noise is a deviation added on the UAV position at same random time step with a duration of 5 time steps. Such noise could be used to model the effect of wind gust, which may deviate the

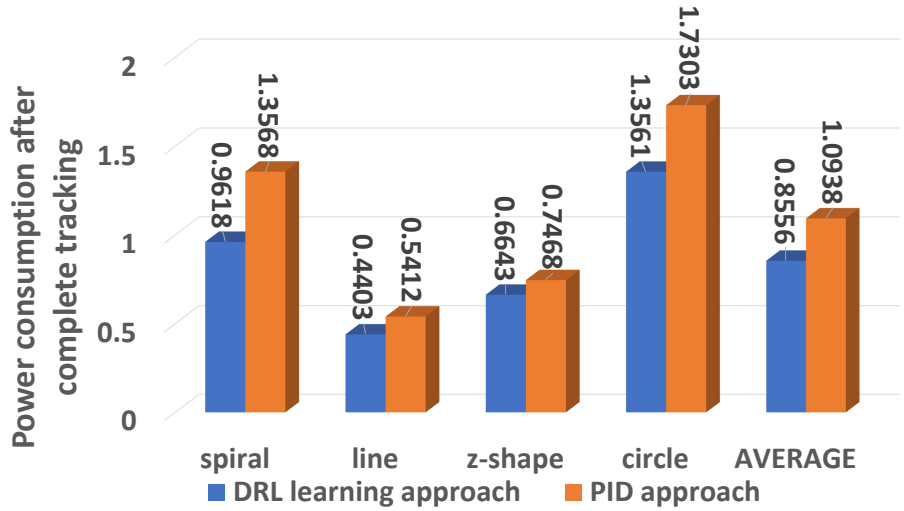


Figure 4.9: Tracking result comparison between proposed DRL-based framework and PID-based baseline in terms of power consumption

UAV away from its current position. Figure 4.10 compares the tracking results in an environment with random noises.

In the first experiment, a relatively small noise is added to the environment. The left figure in Figure 4.10a shows the desired and actual trajectories generated by Matlab simulator for the UAV using model based tracking in [1], while the right one shows the similar information for the UAV using our proposed DRL-based tracking model. As we can see, the UAV using the DRL based tracking follows the desired trajectory more closely. After adding small random Gaussian noise, DRL based system is more stable and achieves smaller position tracking error under considerable error precision. The left figure in Figure 4.10b shows the $L1$-norm of position tracking error for model based tracking (D_{matlab}) and DRL based tracking (D_{DRL}). The right figure in Figure 4.10b shows the difference between these two (i.e. $D_{matlab} - D_{DRL}$). It shows that after deviating from the original trajectory due to random noise, the model based approach will not correct itself right away. Only after the deviation becomes large, it will gradually track back the original trajectory. While the DRL has a relatively more stable position error during all the time. In the second experiment, we add relatively larger noise to the environment. The left and right figures in

Table 4.2: FPGA performance analysis for actor model

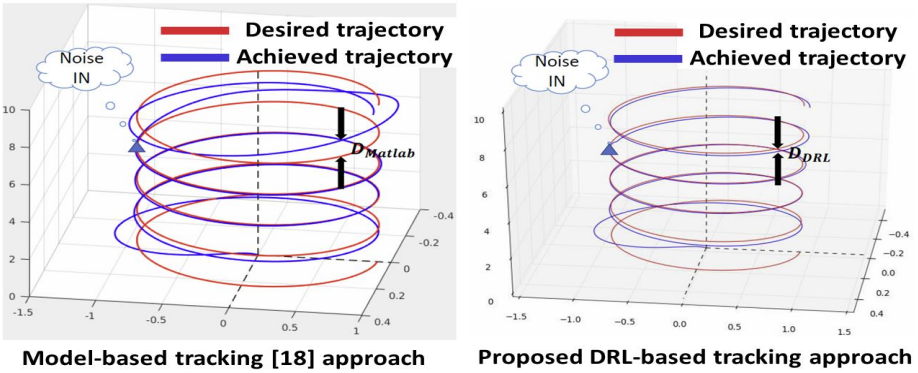
Frequency	373.02MHz
Throughput	204.96 Action/s
Total Power	33.57mW
Logic Utilization in ALM	10.03%
RAM Utilization	20.94%

Figure 4.10c give the original and actual trajectories of the model based and DRL based tracking. As indicated, the model based approach is not able to keep the given trajectory, while the learning based approach can.

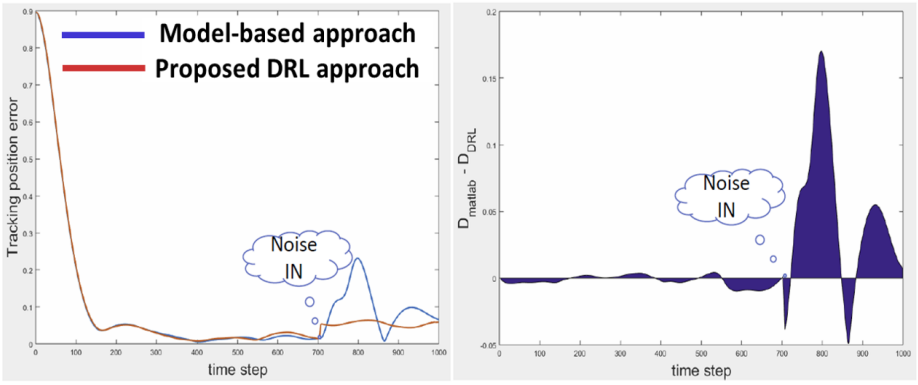
4.3.5 Hardware Performance on FPGA

A truly autonomous UAV has three components, sensing, detection and control. One of the benefits of adopting DRL based trajectory tracking is that it solves the control problem using a deep neural network, which is known to be efficient for detection and sensor signal processing. Using a unified computation model (i.e. DNN) for different tasks allows us to design highly optimized application specific hardware for that computation model, instead of relying on flexible general purpose processor, which is either too bulky or cannot provide enough computation power.

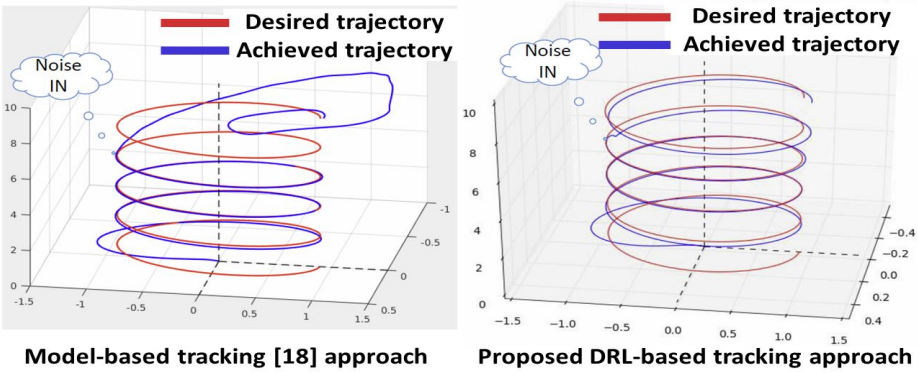
While the training of the DRL framework requires both the actor and critic networks, only the actor network needs to be implemented on the UAV during the runtime and run in real-time. To evaluate the cost, payload and energy impact that the actor network may bring to the UAV, we implement our actor model on the FPGA platform to validate our method on the real-world devices. We choose Intel (Altera) Cyclone V 5CGX FPGA as our evaluation platform. The platform's System-on-Chip (SoC) has the maximum CPU clock frequency of $925MHz$ and embedded DDR3 SDRAM with the memory bandwidth pf around $4,500MB/s$. The actor network has $25,196$ connections, which correspond to $25,196$ multiplication/addition operations. We have to time multiplex the hardware resource in FPGA in order to



(a) Comparison of achieved trajectories with a relatively small noise (*red*: desired trajectory, *blue*: achieved trajectory)



(b) Comparison of tracking position error



(c) Comparison of achieved trajectories with larger noise (*red*: desired trajectory, *blue*: achieved trajectory)

Figure 4.10: Experimental results of trajectories tracking in noisy environment comparisons between using model-based tracking [1] and using proposed DRL-based tracking scheme

evaluate the whole network. The amount of computations that can offload to the FPGA is constrained by the size of the on-chip RAM, which will be used to store the weight parameters and intermediate results. With up to 21% utilization of the RAM resources, we use 10% utilization of the programmable logic resource to achieve 200 actions/s throughput. We present the performance and energy consumption of our FPGA implementation in Table 4.2. In this implementation 16-bit wide fixed-point data precision is used. The results show that, the power consumption of the actor network is very low, hence enables the model to run on the UAV devices, which usually have stringent energy resources. Please note we include the static power in the total power. The real computation power should be much lower.

4.4 Conclusion

In this chapter, we have introduced a framework for UAV trajectory tracking based on deep reinforcement learning using FPGA. The system structure, processing algorithm and software/hardware performance are presented. In our approach, the UAV tracks a desired trajectory through a set of given waypoints, tolerating random Gaussian noise within considerable range. The hardware consumption of the implementation of this scheme is also provided. The proposed scheme is general and applicable to be applied in real UAVs for fast and accurate trajectory tracking system.

Chapter 5

Early Phase Integrated Neural Network Compression for DRL-based UAV Trajectory Planning Framework

5.1 Introduction

Driven by the opportunities and challenges of Internet of Things (IoT), the demand for autonomous UAV increases for research and industrial applications. Although the term "UAV" refers to wide range of systems, from airplane-sized combat drones to insect-sized micro-drones, in this work, we focus on small UAVs with take-off weights 2-20 lbs. This is not only because such small UAVs are commercially available and have the most civilian usage, but also because they impose more stringent constraints on the size and energy dissipation of the onboard system. The safety and efficient control of autonomous UAVs require obstacle-free trajectory planning particularly for BVLOS operations, and applications that require unmanned vehicles to move in cluttered environments [49]. Such applications include indoor operations [50], pack-

This work is partially supported by the National Science Foundation under Grant CNS-1739748 and SRC task 2893.001.

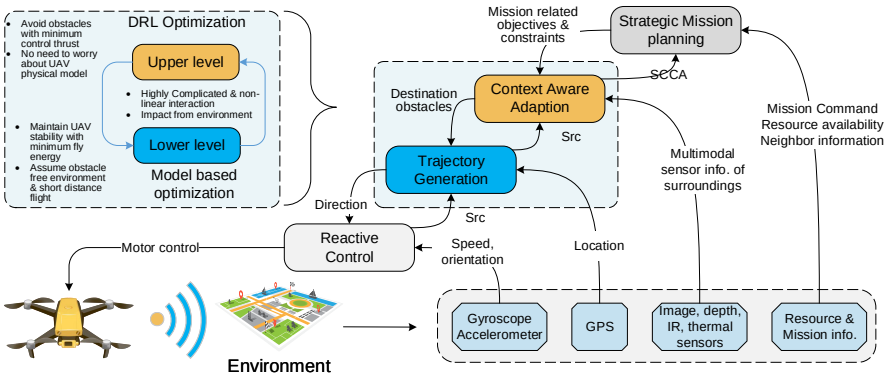


Figure 5.1: Two levels in an autonomy of UAV

age delivery in urban and suburban areas, monitoring of civilian infrastructures like bridges and highways, autonomous landing on moving platforms and tracking wildlife in forested areas.

In this work, we divide the autonomy of UAV into multiple levels shown in Figure 5.1 from autonomous obstacle avoidance to strategic mission planning. The first level autonomy is Trajectory Planning (TP), where the drone dynamically calculates a sequence of waypoints and the trajectory that leads to a (changing) destination with the consideration of its current speed, acceleration, flight condition and potential obstacles. In the lower level, Reactive Control (RC) monitors the gyroscope and accelerometer data, and dynamically adjusts the motor to react to the changing airflow to stabilize the drone in a changing environment.

The trajectory planning is difficult because the UAV needs to maintain it stability, and also ensure safety and energy efficiency in real time. In [2] a two-level framework is proposed for trajectory planning. The upper level framework uses a deep reinforcement learning model to generate a set of optimal waypoints while the low-level framework uses non-optimization to find energy trajectory to connect adjacent waypoints. Based on the sensor information from reactive controller, optimal decisions on flight and sensor control needs to be regenerated by the *TP* layer. These two layers work at the lower abstraction level (i.e. altitude, speed, etc.) of UAV physical

status requiring the fastest response. Therefore, enabling nonlinear guidance running on onboard hardware in real-time is the key to ensure flight safety and assurance.

However, UAVs are not equipped with sufficient computing power due to the limited energy and payload capacity. Conventional high-performance computing systems, mostly GPUs, are either significantly power intensive or too bulky to be placed on small UAVs. The existing gap between the computing capabilities of on-board embedded systems and the real-time computation requirements of the autonomous UAV becomes even wider if we consider the need to process multiple channels of sensor inputs, to search for the optimal decision in DRL framework, not to mention the potential mini-batch updating requirement for DRL. There have been efforts in closing this gap from both sides. On one hand, faster embedded processors with small footage and low energy consumption, such as such as NVIDIA Jetson TX2 and NX, have been used for cyber physical applications on UAVs [3][4]. On the other hand, more efficient computing models have been investigated. A widely used technique is to compress over-parametrized neural networks. Eliminating unnecessary weights from neural networks can decrease the model size and reduce inference energy consumption [5][6] of the trained networks without or with very little accuracy loss. Most commonly it is achieved by setting a particular set of weights to 0 and freezing them for the course of subsequent training. There are multiple motivations for performing compression. Firstly, it supports generalization by regularizing over-parametrized functions. Secondly, by small networks with good performance reduces energy costs, computation complexity, storage requirements, and inference latency, all of which can support the deployment on small UAVs. In summary, compressing over-parameterization is an important step towards successful training and real time on-board processing [7]. However, most of the existing works apply pruning on fully trained model and many of them only focused on later phase of pruning influence. The three-step process, i.e. training, pruning, and re-training, has high computation

and memory complexity. The complexity is even higher when it is applied to a model trained using deep reinforcement learning, as the interaction with the environment and model adaptation take place sequentially.

In this work, we investigate pruning deep neural networks when training DRL network to plan energy efficient UAV trajectories. We present an early phase integrated model compression framework for UAV trajectory planning, incurring less computation overhead and achieving performance improvement. In our system, the drone dynamically calculates a sequence of waypoints and the trajectory that leads to a destination with the consideration of its current speed, acceleration, flight condition and potential obstacles in the trajectory planning scheme. Meanwhile, the planning model is pruned during the training process. Here, we employ Adaptive Moment Estimation (ADAM) – ADMM [18] based compression combined with the trajectory planning scheme. Notably, we verify that ADAM-ADMM based compression generalizes to other research domains, and it can help save DRL training effort moreover and improve inference performance.

This work has the following three main contributions. The first is the improved initialization and loss function. We discovered that a truncated navigation gain and stochastic action reinitialization are two techniques that work effectively with the DRL training of the trajectory planning problem. More than 34.14% improvement of convergence can be achieved using this technique. The second is an optimized layer-wise distribution of compression ratio and general guideline for structured weight pruning for the deep Q-network. Our study shows that weight compression of intermediate layer will perform better compared with weight compression of input or output layer due to the high redundancy in intermediate layers. Results indicates 3.045x prune ratio plus 1.7% success rate improvement by using targeted compression. The third is the integrated structured weight pruning framework on DRL problems at an early phase of training. We factorized the ADAM-ADMM weight pruning algorithm

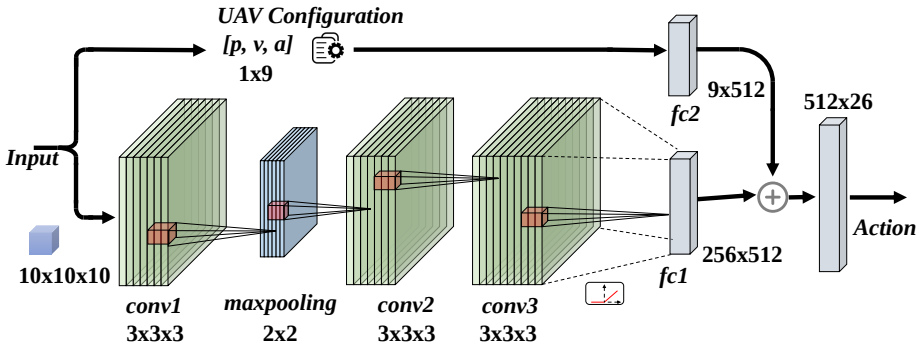


Figure 5.2: Trajectory planning network structure

Table 5.1: Parameters number and FLOPs of each layer in trajectory planning network

Layer#	$conv1$	$conv2$	$conv3$	$fc1$	$fc2$
Param#	3584	884992	524544	131584	5120
FLOPs#	1957888	7081984	524800	132096	5632

to take advantages of the special characteristics after ADMM regularization. Results of different sparsity combinations with DRL based trajectory planning problems are reported in Section 5.3.2. We observed that weight compression together with training from an early phase will uncover necessary weights, and it learns faster than original DRL while reaching better performance.

The rest of this chapter is organized as follows: In Section 5.2, we describe details about our proposed integrated model compression system. Section 5.3 describes our experiments setup and evaluation results. Finally, Section 5.4 gives the conclusions.

5.2 The Improved DRL Framework

In this section, we present the deep reinforcement learning framework with early phase integrated weight compression for the UAV trajectory planning.

5.2.1 Overall System Flow

The goal of our work is to plan the optimal trajectory of UAV with the smallest memory size and the fastest processing speed. Firstly, we formulate the optimal trajectory planning as a two-level optimization problem. The upper level is DRL based optimization to learn how to plan waypoints in a known environment. The waypoints are selected so that if we apply a non-linear optimization to find each trajectory between two adjacent waypoints in the lower level, we do not need to worry about obstacles. During training, the upper level tries to learn the dynamics of the lower level optimization and how lower level dynamics impact the interaction between the UAV and environment. Waypoints will be generated to cope with such dynamics. As a result, the lower level can assume an obstacle free environment and focus only on connecting two adjacent waypoints. Finally all waypoints will be connected to generate a smooth trajectory with optimized energy efficiency. The prior work shows that the model generates obstacle free smooth trajectory that consumes less thrust energy compared to other heuristic waypoint generation methods.

The original network architecture without weight compression in this work is shown in Figure 5.2. The input of the network is the $10 \times 10 \times 10$ array of the environment information and a 1×9 vector of the UAV status. The output is a 1×26 vector of the Q-values for 26 actions corresponding to 26 possible next waypoint locations. All convolutional layers in the network are $conv3D$ layers. Parameters number and FLOPs of each layer are shown in Table 5.1. In this work we integrate weight compression with the DRL training of the trajectory planning framework. The goal is to plan optimal trajectory in real time with less memory, lower computation complexity and computing time, at the same time with manageable training cost. To avoid the cost of training the model twice, it is desirable to prune the model as early as possible. However, training sparse architectures produced by pruning from scratch is known to be very difficult [14]. Instead of pruning from scratch,

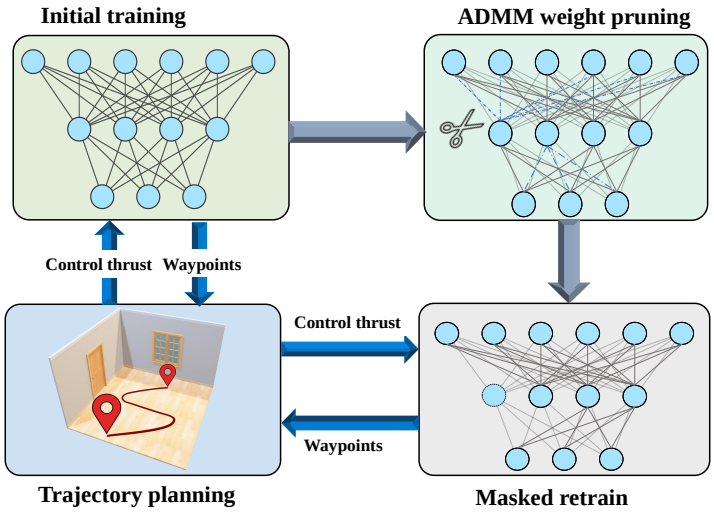


Figure 5.3: Overall flow of our pruning framework

we start to apply structured weight compression at early iterations of DRL training of the original model, allowing the weights to settle at some prior knowledge of the problem. Therefore, the partially trained original model behaves as a teacher to distill the network. The overall flow is shown in Figure 5.3.

5.2.2 Trajectory Planning Refactoring

To reduce the training cost, we further refactorize the prior work [2] using truncated navigation gain and stochastic action reinitialization.

Truncated navigation gain We define the truncated navigation gain as reward in trajectory planning and it contains two parts. One is navigation reward $\mathcal{N}_r$ and the other is navigation effort $\mathcal{N}_e$. First of all, taking obstacle uncertainty into consideration, we assume optimal navigation direction θ follows a truncated Gaussian distribution. Here $\boldsymbol{\theta}$ is a two-dimensional vector indicating polar angle (θ_1) and azimuthal angle (θ_2) respectively. The mean value is denoted as $\hat{\boldsymbol{\theta}}$, and it is the direction of the destination location. We define the variance of the navigation direction as $\boldsymbol{\Sigma}$, it is the deviation of the optimal direction distribution of the UAV. The navigation

reward $\mathcal{N}_r$ is the distribution of $\boldsymbol{\theta}$ as:

$$\begin{aligned}\mathcal{N}_r &= f(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{a}, \boldsymbol{b}) \\ &= \frac{\phi((\boldsymbol{\theta} - \boldsymbol{\mu}), \boldsymbol{\Sigma})}{\Phi((\boldsymbol{b} - \boldsymbol{\mu}), \boldsymbol{\Sigma}) - \Phi((\boldsymbol{a} - \boldsymbol{\mu}), \boldsymbol{\Sigma})}, \\ &\text{subject to} \quad \boldsymbol{a} \leqslant \boldsymbol{\theta} \leqslant \boldsymbol{b}\end{aligned} \tag{5.1}$$

where

$$\phi(\boldsymbol{\theta}) = \frac{1}{2\pi \left|\boldsymbol{\Sigma}\right|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu})}, \tag{5.2}$$

is normal distribution of $\boldsymbol{\theta}$, and is the cumulative distribution of $\phi(\boldsymbol{\theta})$,

$$\Phi(\boldsymbol{\theta}) = \frac{1}{2}(1 - erf(\frac{\sqrt{2}}{2}\boldsymbol{\theta})), \tag{5.3}$$

where erf is the Gaussian error function of $\boldsymbol{\theta}$. Due to symmetry of the UAV scenario, we set

$$\boldsymbol{\mu} = \hat{\boldsymbol{\theta}} = (\hat{\theta_1}, \hat{\theta_2}), \tag{5.4}$$

$$\boldsymbol{\Sigma} = \begin{pmatrix} \tilde{\theta_1} & 0 \\ 0 & \tilde{\theta_2} \end{pmatrix}, \tag{5.5}$$

$$\boldsymbol{a} = \hat{\boldsymbol{\theta}} - \frac{\pi}{2}, \tag{5.6}$$

$$\boldsymbol{b} = \hat{\boldsymbol{\theta}} + \frac{\pi}{2}, \tag{5.7}$$

where $\hat{\theta_1}$ and $\hat{\theta_2}$ are destination directions, $\tilde{\theta_1}$, $\tilde{\theta_2}$ are standard deviation given from UAV intrinsic property. $\hat{\theta_1}$ and $\tilde{\theta_1}$ are in polar angle, $\hat{\theta_2}$ and $\tilde{\theta_2}$ are in azimuthal angle respectively. And $\boldsymbol{a}$ and $\boldsymbol{b}$ are low and above bounding ranges of random variables respectively. Therefore, Equation 5.1 can be written as:

$$\mathcal{N}_r(\boldsymbol{\theta}; \hat{\boldsymbol{\theta}}, \tilde{\boldsymbol{\theta}}) = \frac{\phi(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}, \boldsymbol{\Sigma})}{2\Phi(\frac{\pi \boldsymbol{I}}{2}, \boldsymbol{\Sigma}) - 1}, \tag{5.8}$$

It can be shown that $\mathcal{N}_r(\boldsymbol{\theta})$ reaches the maximum value when $\boldsymbol{\theta}$ equals to $\hat{\boldsymbol{\theta}}$. Secondly, the navigation effort $\mathcal{N}_e$ is calculated using method proposed in [2]. In order to combine these two, we formulate these two parts as composite gain using fusion combination. We denote g as the gain of fusion. Then the truncated navigation gain $\mathfrak{G}$ is denoted as a combined optimization problem below.

$$\begin{aligned} \mathfrak{G}(\boldsymbol{\theta}) &= \begin{bmatrix} 1-g & g \end{bmatrix} \begin{bmatrix} \mathcal{N}_r(\boldsymbol{\theta}) \\ \mathcal{N}_e(\boldsymbol{\theta}) \end{bmatrix}, \\ \text{s.t.} \quad & g \in [0,1] \end{aligned} \tag{5.9}$$

The variance of $\mathfrak{G}(\boldsymbol{\theta})$ can be obtained as

$$Var(\mathfrak{G}(\boldsymbol{\theta})) = (1-g^2)Var(\mathcal{N}_e(\boldsymbol{\theta})) + g^2(\mathcal{N}_r(\boldsymbol{\theta})), \tag{5.10}$$

To minimize the variance of the gain, we take the derivative of Equation 5.10 with respect to g, and set it to zero. Then we can get

$$g = \frac{tr[Cov(\mathcal{N}_e(\boldsymbol{\theta}))]}{tr[Cov(\mathcal{N}_e(\boldsymbol{\theta}))] + tr[Cov(\mathcal{N}_r(\boldsymbol{\theta}))]}, \tag{5.11}$$

where tr represents trace of the matrix and covariance learned for navigation reward and navigation effort are denoted as

$$Cov(\mathcal{N}_r(\boldsymbol{\theta})) = \boldsymbol{\Sigma}[1 + \frac{\boldsymbol{a}\phi(\boldsymbol{a}) - \boldsymbol{b}\phi(\boldsymbol{b})}{\Phi(\boldsymbol{b}) - \Phi(\boldsymbol{a})} - (\frac{\phi(\boldsymbol{a}) - \phi(\boldsymbol{b})}{\Phi(\boldsymbol{b}) - \Phi(\boldsymbol{a})})^2]\boldsymbol{\Sigma}^T, \tag{5.12}$$

$$Cov(\mathcal{N}_e(\boldsymbol{\theta})) = \boldsymbol{\Sigma}\boldsymbol{\Sigma}^T. \tag{5.13}$$

Stochastic action reinitialization Even with techniques such as epsilon-greedy exploration, Conventional deep Q-learning exploits actions with the highest Q-value during the training more often than exploration new actions. The predicted Q-values

of actions are just rough estimations especially during the early phase of training. Relying on the actions with maximum Q values for making decision during the training time will hardly generate satisfactory decisions. The inadequacy of relying only on the highest Q-value actions becomes prominent especially with pruning. It is the source of the slow convergence, as the confidence of Q-value prediction has higher uncertainty under higher compression rates. Therefore, we introduced a stochastic action re-initialization step during training. Every time the UAV fails its mission, it will randomly select an action instead of predicting once again. Also, the agent will always randomly choose the action of the top M actions with the highest values. With this technique, the model converges faster than prior work. Detailed comparison results are included in Section 5.3.1.

5.2.3 Early Phase Integrated Weight Compression System for UAV Trajectory Planning

We start the pruning process in early iterations of the DRL training. This allows the model to gain some prior knowledge of the problem without spending too much time in refining itself. After pruning, retrain or fine-tune the model is an important step, as it allows the model to have a chance to re-adjust to prevent performance degradation. Therefore, in our system the training goes on while pruning and we do not need a fully trained network to start with.

In this work, we adopt structured pruning for two reasons. Firstly, structured pruning has been found to significantly improve pruning performance than other sparsity types [15]. Secondly, structured pruning can speed up computation on standard hardware by removing whole groups of weights such as filters or channels of convolutional layers in neural network. The remaining weight connections maintain the convolution structure and hence the computations can easily be mapped to parallel GEMM operations, while the unstructured pruning will generate sparse weight

matrix that cannot easily be parallelized. In addition, we focus on local pruning instead of global pruning. In other words, we specify compression ratio for each layer separately.

Consider an agent with an action space of $\mathcal{A}$ possible actions and a network with L layers and each layer i as parameters p_i, including weights $\boldsymbol{w_i}$ and bias $\boldsymbol{b_i}$, we minimize the objective function subject to specific structured sparsity constraint on base of parameters in each layer, i.e.

$$\begin{aligned} &\min_{(\mathbf{W},\mathbf{b})} f(\mathbf{W},\mathbf{b}) \\ &\text{s.t.} \quad \mathbf{W} = \left\{\boldsymbol{w}_i\right\}_{i=0}^{L-1}, \mathbf{b} = \left\{\boldsymbol{b}_i\right\}_{i=0}^{L-1}, \\ &\qquad \boldsymbol{w}_i \in \boldsymbol{s}_i, i = 0, ..., L-1 \end{aligned} \tag{5.14}$$

where $\boldsymbol{s}_i$ is the set of $\boldsymbol{w}_i$ with a specific structure and the objective function is defined as

$$\begin{aligned} &f(\mathbf{W},\mathbf{b}) = \\ &\frac{1}{|batch| \cdot \mathcal{A}} \sum_{j \in batch} max(\sum_{a=0}^{\mathcal{A}-1} |q_{aj} - \hat{q}_{aj}|, \sum_{a=0}^{\mathcal{A}-1} (q_{aj} - \hat{q}_{aj})^2) \\ &+ \sum_{i=0}^{L-1} \|\boldsymbol{w}_i\|_F^2, \end{aligned} \tag{5.15}$$

In this case, *batch* is the set of past experiences sampled from the replay buffer, j is one the experiences (i.e. a possible environment state), q is the Q value of action a under state j, and $\hat{q}$ is its corresponding target value. Unlike supervised learning, where labels are provided, the target Q value is updated based on the following equation

$$\hat{q}_{aj} = \mathfrak{G}(\boldsymbol{\theta}) + \gamma max \hat{q}_{aj'} \cdot I(j), \tag{5.16}$$

where $\mathfrak{G}(\boldsymbol{\theta})$ is the reward defined in Equation (5.9), γ is the discounted factor, j' is

the next state of state action pair (a, j), and

$$I(j) = \begin{cases} 0, & \text{if sequence terminates at episode j+1} \\ 1, & \text{others} \end{cases} \tag{5.17}$$

The second term in Equation 5.15 is L_2 weight regularization. The constraint of Equation 5.14 is nonconvex and combinatorial. Using ADMM it can be equivalently rewritten in a form with restriction $\boldsymbol{w}_i = \boldsymbol{z}_i$

$$\min_{\mathbf{W},\mathbf{b}} f(\mathbf{W}, \mathbf{b}) + \sum_{i=0}^{L-1} f_i(\boldsymbol{z}_i), \tag{5.18}$$

where f_i is the indicator function of the specific structure $\boldsymbol{s}_i$. According to [18], the optimization problem can be decomposed into two subproblems.

$$\min_{\mathbf{W},\mathbf{b}} f(\mathbf{W}, \mathbf{b}) + \sum_{i=0}^{L-1} \frac{\rho_i}{2} \left(\|\boldsymbol{w}_i - \boldsymbol{z}_i + \boldsymbol{u}_i\|_F^2 + \lambda \|\boldsymbol{b}_i\|_F^2 \right), \tag{5.19}$$

$$\min_{z_i} \sum_{i=0}^{L-1} f_i(\boldsymbol{z}_i) + \sum_{i=0}^{L-1} \frac{\rho_i}{2} \left(\|\boldsymbol{w}_i - \boldsymbol{z}_i + \boldsymbol{u}_i\|_F^2 + \lambda \|\boldsymbol{b}_i\|_F^2 \right), \tag{5.20}$$

where $0 < \rho_i < 1$ is penalty parameter for layer i. This problem is solved by iterating $\mathcal{K}$ steps and the optimal solution is the Euclidean projection of $\boldsymbol{w}_i^s + \boldsymbol{u}_i^{s-1}$ onto specific sparsity $\boldsymbol{s}_i$. Since the weight magnitude is usually just a noisy representation of the importance of weight, one-shot pruning directly to the desired compression ratio is likely to remove connections that is important. In this work, we apply ADMM multiple times, each time a fraction of the weight coefficients are pruned until we reach the desired prune ratio.

Unlike the supervised learning where training set is provided, the training set of the deep Q-learning (i.e. the state action pairs and the corresponding target Q values)

Algorithm 2 Early phase integrated weight pruning for DRL based trajectory planning

Input : Initially trained DRL based trajecotry planning network with parameters ($\mathbf{W}$, $\mathbf{b}$)

Output: pruned network with specified percentage of zero parameters for each network layer

Load network with initially trained parameters $\boldsymbol{W}$ and $\boldsymbol{b}$;
Initialize penalty parameter to ρ_0;
set admm pruning episode $k = 0$;
foreach *network layer i in pruning configuration* **do**
  set $\boldsymbol{w_i}^k = \boldsymbol{w_i}$, $\boldsymbol{u_i}^k = \mathbf{0}$, $\boldsymbol{z_i}^k = \boldsymbol{w_i}^k$;
for *penalty episode* $t \leftarrow 1$ *to* $\mathcal{T}$ **do**
  set $\rho_t = 10^t \cdot \rho_0$;
  Restore replay buffer $\mathcal{B}$ to capacity $\mathcal{C}$ with converged model from episode $t-1$;
  for *pruning episode* $k \leftarrow 1$ *to* $\mathcal{K}_p$ **do**
    Sample random mini batch of past transitions from $\mathcal{B}$;
    Calculate corresponding target value $\hat{q}$ of each state and action pair;
    foreach *layer i in pruning configuration* **do**
      set $\rho_i^k = \rho_t$;
      if $\boldsymbol{w}_i^{k+1} < prune\_percentile \cdot \left\|\boldsymbol{w}_i^{k+1}\right\|_F^2$ **then**
        set $\boldsymbol{w}_i^{k+1} = 0$;
      set $\boldsymbol{z_i^{k+1}} = \boldsymbol{w_i}^{k+1} + \boldsymbol{u_i}^k$;
      set $\boldsymbol{u_i}^{k+1} = \boldsymbol{w_i}^{k+1} - \boldsymbol{z_i^{k+1}} + \boldsymbol{u_i}^k$;
    Calculate total loss $\mathfrak{L} = f(\boldsymbol{W}, \boldsymbol{b}) + \sum_{i=0}^{L-1} \frac{\rho_i^k}{2} \left(\left\|\boldsymbol{w_i}^{k+1} - \boldsymbol{z_i}^{k+1} + \boldsymbol{u_i}^k\right\|_F^2 + \lambda \left\|\boldsymbol{b_i}\right\|_F^2 \right)$;
    Perform a gradient descent $\partial \mathfrak{L}$ with respect to the network parameters $\boldsymbol{W}$ and $\boldsymbol{b}$;
Restore replay buffer $\mathcal{B}$ to capacity $\mathcal{C}$ with ADMM converged model;
foreach *layer i in pruning configuration* **do**
  if $\boldsymbol{w}_i < prune\_percentile \cdot \|\boldsymbol{w}_i\|_F^2$ **then**
    set $\boldsymbol{w}_i = 0$;
for *retraining episode* $m \leftarrow 0$ *to* $\mathcal{M}_r$ **do**
  Sample random mini batch of past transitions from $\mathcal{B}$;
  Calculate corresponding target value $\hat{q}$ of each (state, action) pair;
  foreach *layer i in pruning configuration* **do**
    if $\boldsymbol{w}_i = 0$ **then**
      set gradient mask $\mathfrak{M}_i = \mathbf{0}$;
    else
      $\mathfrak{M}_i = \mathbf{1}$;
  Perform a gradient descent $\partial f((\boldsymbol{w}_i, \boldsymbol{b}_i) \cdot \mathfrak{M}_i)$ with respect to the masked network $\boldsymbol{W}$ and $\mathbf{b}$;
return sparse network;

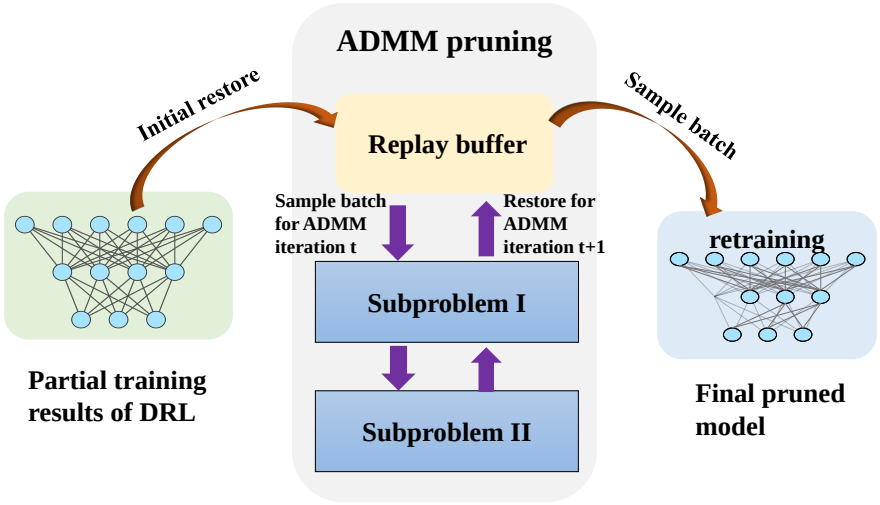


Figure 5.4: Iterative prune-restore flow diagram

is sampled from the replay buffer which is updated through simulation or deploying the learned model. We adopted an iterative prune-restore procedure. Each time after the ADMM prune converges, a pruned model will be simulated and the replay buffer will be updated. The detailed flow diagram is shown in Figure 5.4.

The replay buffer $\mathcal{B}$ is firstly updated to capacity $\mathcal{C}$ using partial training model. The ADMM pruning will be applied $\mathcal{K}$ times. Every time ADMM is applied, the model is pruned by iteratively solving the subproblems 1 and 2 over sampled mini-batch from $\mathcal{B}$. Before each application of ADMM, $\mathcal{B}$ is restored using the converged model obtained from the previous pruning. After ADMM pruning completes, the remaining non-zero weights are retrained [18]. This helps achieve higher compression rate without success rate degradation. It is shown as the last step in the flow diagram. The algorithm 2 shows the overall process of our early-phase integrated weight pruning with DRL based trajectory planning.

5.3 Experimental Results

The proposed system is implemented and evaluated on Linux with two GeForce RTX 2080 Ti. Again, we focus on the pruning of a DRL based optimal trajectory planning model within a closed environment. The evaluation environment is randomly generated, including random placement of obstacles and random selection of the start and target waypoints. The overall environment is discretized to $10 \times 10 \times 10$. Each waypoint is encoded as the center of each environment grid. We use the algorithm proposed in [2] to generate the optimal trajectory with minimal control effort analytically between each pair of adjacent waypoints. In this section, we demonstrate performance of our proposed approach from four aspects.

5.3.1 Performance Improvement of Trajectory Planning Refactoring

To measure the impact of two refactoring techniques for optimal trajectory planning, we first compare with prior work [2] in terms of convergence behavior. We use FLOPs to indicate the computation effort [93][94]. As shown in Table 5.2, the training effort needed of a fully trained model reduces from 2.123e+14 to 1.399e+14, resulting in 34.14% reduction. Besides, Figure 5.5 shows a comparison of convergence in terms of cumulative success rate during training. From the figure, we can see that our model converges faster and become more stable than the prior work with the help of our refactoring techniques. Previously, the model converged at around $30K$ episodes, while our model converges 20% faster than before. Also our model can learn the

Table 5.2: Training convergence behavior comparison in terms of FLOPs between proposed refactoring planning and prior work [2]

	training effort (FLOPs) to convergence
prior work[2]	2.125e+14
refactoring planning	1.399e+14

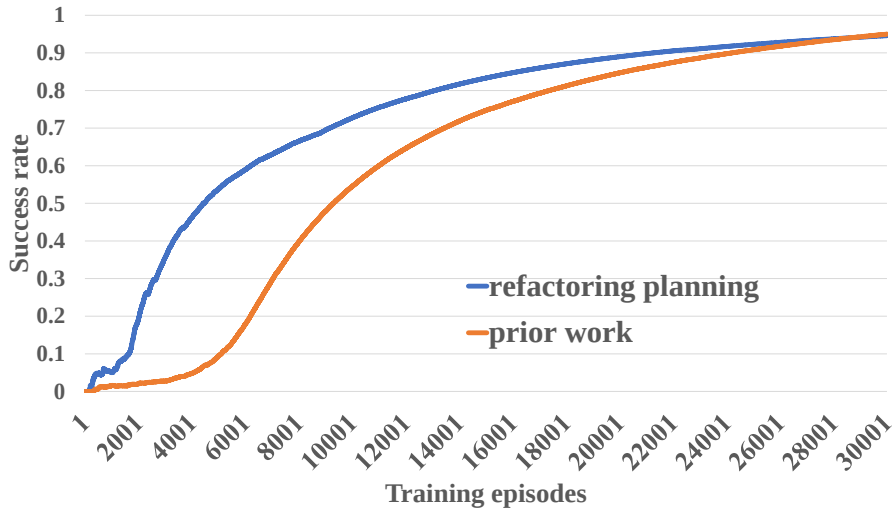


Figure 5.5: Training convergence behavior comparison in terms of cumulative success rate between proposed refactoring planning and prior work [2]

interaction between upper and lower level in trajectory planning 2X faster at the early phase of the training. More specifically, our model can achieve around 50% success rate with only $5k$ episodes while $10k$ episodes are required at least.

5.3.2 Pruning Level Influence for Each Layer of the Neural Network

Next, we investigate how much compression can be tolerated in each layer. We find that the performance of structured pruning varied substantially for each layer. Figure 5.6 shows the relationship between the compression rate and the evaluation success rate for each layer. Dashed lines and solid lines are performance of the network with channel pruning and filter pruning respectively at specific layer. It is worth mentioning that in order to keep all information of input and out layers i.e. $conv1$, $fc2$ and fc_{out}, we only apply filter pruning at the two input layers and no pruning at the output layer. We note that the layer closer to the input or output is extremely sensitive to the pruning. Any pruning at those layers will immediately cause significant performance dropping while the intermediate layers are more robust to pruning.

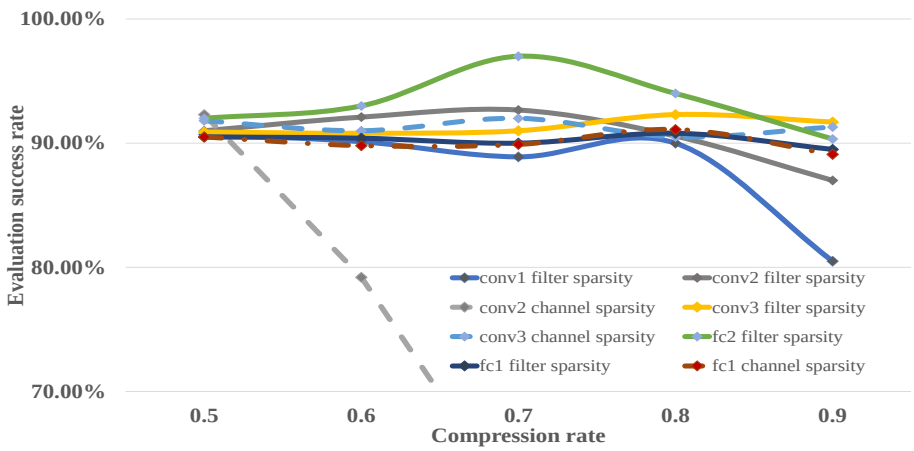


Figure 5.6: Relationship between compression rate and evaluation success rate for each layer of trajectory planning network

For example, pruning of *conv*3 causes less success rate drop than pruning of *conv*2 for both filter and channel pruning. And filter pruning of *conv*1 is less stable than other two *conv* layers. Therefore, the first layer is not the focus of pruning. We also found that filter pruning can achieve higher compression ratio without success rate drop than channel pruning. For example, the success rate of *conv*2 layer with channel pruning drops notably as increasing the compression ratio. It indicates that the level of overparameterization varies dramatically across different layers and "one ratio fits all" compression models may have adverse impacts on performance. In this work, we set different prune ratios for different layers.

Figure 5.7 shows the maximum prune ratio that keeps success rate above 90.70% for each layer per sparsity type. As shown in the figure, *conv*3 can have a maximum 5x compression of filter pruning. Moreover, $fc1$ layer can achieve 80% sparsity both after filter pruning and channel pruning.

Through systematically removing the group of weights which are zero or close to zero in different layers together and retraining the remaining non-zero weights, we can prune more aggressively with a tolerable loss of the success rate [18][5]. Experiments are carried out to validate the effectiveness and efficiency of different combinations of filter and channel structured pruning for the model. We measure the quality of

Table 5.3: Details of structured pruning results with different configuration of layer-wise compression ratio

Structured pruning	Pruned layers					Total sparsity	Prune rate	Success rate
	$conv1$	$conv2$	$conv3$	$fc2$	$fc1$			
filter prune	-	-	-	-	-	0%	-	96.8%
pruning ratio	-	-	-	-	-			
filter prune	-	-	50%	-	-	25.17%	1.336x	94.00%
channel prune	-	-	50%	-	-			
pruning ratio	-	-	4x	-	-			
filter prune	-	-	50%	50%	-	29.40%	1.416x	96.00%
channel prune	-	-	50%	-	-			
pruning ratio	-	-	4x	2x	-			
filter prune	-	-	50%	50%	-	31.49%	1.460x	95.60%
channel prune	-	-	50%	50%	-			
pruning ratio	-	-	4x	4x	-			
filter prune	-	50%	50%	50%	-	59.81%	2.488x	94.50%
channel prune	-	-	50%	50%	-			
pruning ratio	-	2x	4x	4x	-			
filter prune	50%	50%	50%	50%	50%	60.11%	2.507x	95.00%
channel prune	-	-	50%	50%	-			
pruning ratio	2x	2x	4x	4x	2x			
filter prune	-	70%	82%	-	-	67.16%	3.045x	**97.30%**
pruning ratio	-	3.33x	5.56x	-	-			
filter prune	10%	70%	82%	-	-	67.19%	3.047x	97.20%
pruning ratio	1.11x	3.33x	5.56x	-	-			
filter prune	10%	75%	82%	-	-	70.02%	3.335x	94.30%
pruning ratio	1.11x	4x	5.56x	-	-			
filter prune	-	70%	80%	79%	50%	73.32%	3.748x	94.30%
pruning ratio	-	3.33x	5x	4.76x	2x			
filter prune	10%	70%	80%	80%	50%	73.44%	3.764x	95.00%
pruning ratio	1.11x	3.33x	5x	5x	2x			
filter prune	-	70%	85%	85%	50%	75.50%	4.082x	93.60%
pruning ratio	-	3.33x	6.67x	6.67x	2x			

Table 5.4: Training effort(FLOPs) with/without our pruning method for the model with the best performance

Success rate of initial training	Success rate after pruning	pretrain FLOPs		Weight pruning FLOPS	Total training FLOPS
		conv layers	FC layers		
96.8%	-	1.377e+14	2.182e+12	-	1.399e+14
96.8%	97.1%	1.377e+14	2.182e+12	9.401e+11	1.409e+14
87.5%	97.3%	9.182e+13	1.455e+12	9.401e+11	9.422e+13
65.4%	96.7%	4.591e+13	7.275e+11	9.401e+11	4.758e+13
49.7%	95.2%	2.870e+13	4.5487e+11	9.401e+11	**3.009e+13**

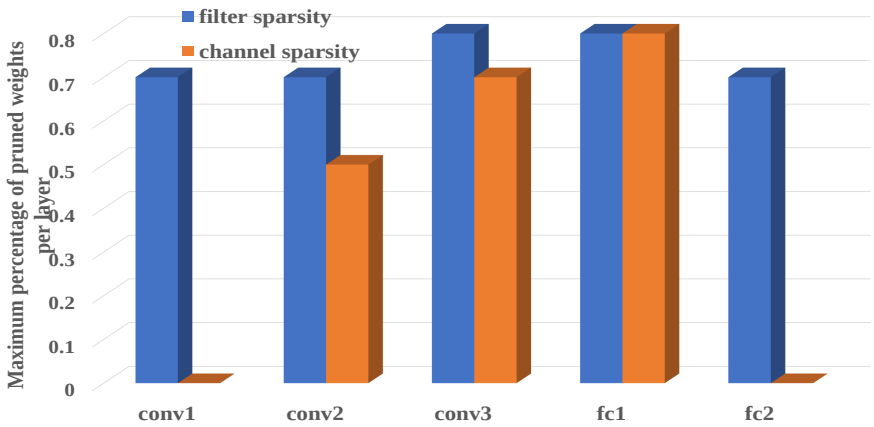


Figure 5.7: Comparison of per-layer maximum portion of pruned weights on trajectory planning network per sparsity types

Table 5.5: Comparison between un-pruned fully trained model and best pruned model with our method

	Prune rate	Model sparsity	Success rate	Achieve rate	Average waypoints	Inference FLOPs	Average measured inference time(s)
prior work[2]	-	0%	95.6%	96%	9.0	1.844e+7	0.002454
un-pruned of our work	-	0%	96.8%	98.0%	6.7	9.716e+6	0.002412
Our work	3.045x	67.16%	97.3%	98.6%	6.92	4.160e+6	0.001427

model using success rate for 1000 randomly generated test cases of UAV navigation. Table 5.3 shows the details of several structured pruning experiments with different configuration of layer-wise compression ratio along with evaluation success rate. All of the experiments are built on top of the same partially trained initial model with success rate of 87.5%. After compression and retrain we not only decreased the model size and computation cost, but also improved the success rate. Success rate can be increased to 97.3% with 3.33x weight pruning of *conv*2 layer and 5.56x weight pruning of *conv*3 layer, resulting in 3.047x weight pruning in total. With a moderate success rate loss within 2.3% compared with the model without best pruned, a total sparsity of 73.44% is achieved, translating into 3.764x weight pruning.

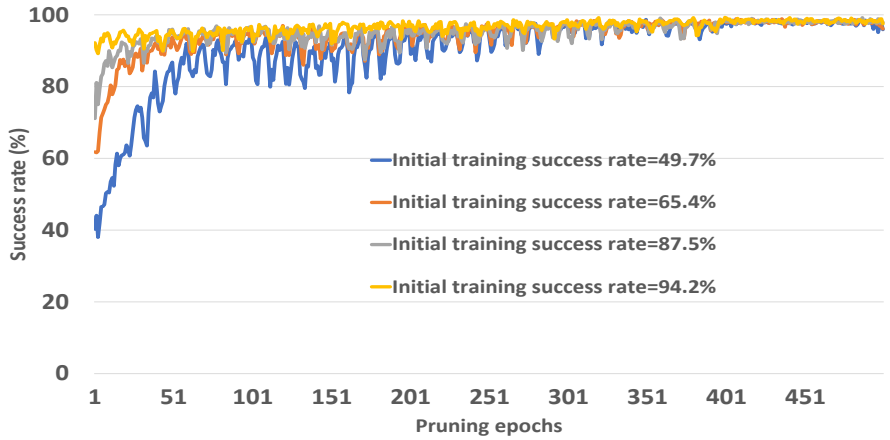
Table 5.6: Average inference time comparison between fully trained model without pruning and model pruned with our method

	Average inference time
Fully trained model without pruning	0.002412s
Model pruned with our method	0.001427s
Inference speedup	**40.84%**

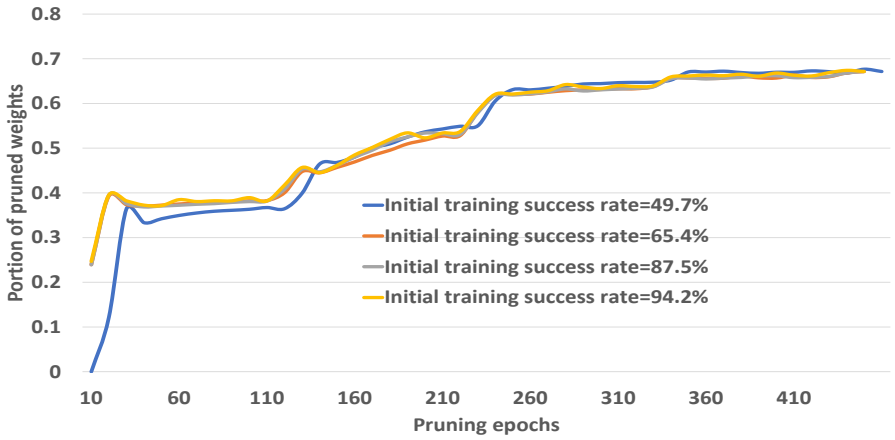
5.3.3 Training Effort Evaluation of Presented System

To evaluate the efficiency of our presented pruning system, we compare the training and pruning cost in Table 5.4. Again, we use FLOPs to indicate computation cost. The total pruning FLOPs of this configuration is 9.401e+11. Table 5.4 compares the training effort with and without presented weight pruning. The success rate of a fully trained model is 96.8% without pruning, and it requires 2.4e+3 training episodes and totally 1.399e+14 FLOPs. The success rate increases to 97.3% after pruning a pretrained model with success rate 87.5% and the FLOPs is reduced by 33.33%. If we allow a 2.1% success rate degradation, the total FLOPs drop to 3.009e+13 which is 79.17% reduction.

We also show how the evaluation success rate change during the pruning with different initial models in Figure 5.8a. These different initial models indicated by different colors in the figure correspond to different initial training efforts. The same prune configuration is adopted by all four initial trained models, i.e, 3.33x pruning of *conv*2 and 5.56x pruning of *conv*3. With an initial success rate of 49.7%, the model will converge in around 250 epochs as indicated in blue line. And with a better trained initial model, the pruning process can improve the performance faster while pruning. Figure 5.8b shows the total sparsity change of these four pruning processes. We can observe that our presented work can speed up the convergence of DRL training significantly, with only 500 epochs in total. This is because pruning can effectively mitigate exploration effort required by the model and reduce the possibility of overfitting.



(a) Success rate change during the pruning with different initial models



(b) Total sparsity change of the model during the pruning with different initial models

Figure 5.8: Success rate and total sparsity comparison during pruning with different initial models

For this pruning configuration, the average inference time with and without pruning is reported in Table 5.6. Both models run on one GeForce RTX 2080 Ti. It shows that with a pruning ratio of 3.045x, the inference time can have 40.8% reduction. This indicates that the pruning result is suitable for GPU acceleration and the model reduction can be transformed into runtime reduction at almost the same scale.

5.3.4 Performance Comparison Between the Best Pruned Model and Prior Work

In the last experiment, we compare a pruned model (pruned), a fully trained unpruned model with truncated navigation gain and stochastic action initialization (unpruned), a model from prior work in [2]. First, we found that the un-pruned model has a success rate of 96.8% and achieve rate of 98%, and both are higher than the prior model. In addition, the average number of selected waypoints is reduced to 6.7 if applying refactoring techniques, which is 25.56% less than prior work plus a 47.3% FLOPs decrease. This improvement is attributed to the refactoring techniques introduced in Section 5.2.2. We also found that the success rate increases from 96.8% to 97.3% with the pruned model, which is trained and compressed by our early phase integrated compression framework. Also, in terms of number of waypoints, our pruned model selects less number of waypoints. With the best pruning configuration, the average waypoints number is 6.92, which is 23.11% less than prior work. Thirdly, compared to the unpruned model, the inference FLOPs is further reduced by 57.18% with efficient pruning. The prune ratio of the overall model with the best pruning configuration achieves 3.045x in total. As we can observe from the results in Table 5.5, our presented work has achieved significant improvement in saving training workload and success rate and give a moderate improvement in energy efficiency of planed trajectory.

5.4 Conclusion

In this work, we present an early phase integrated neural network compression for DRL based trajectory planning system. It motivates a practical success of applying structured weight pruning from the early phase of DRL training process. Experimental results show that our presented system can save at most 78.49% training effort and achieve 40.84% inference speedup. It also has high pruning rates, retaining high

accuracy as well as FLOPs reduction that cannot be achieved before. We also observe that layers far from the input have more redundancy of weights. The results have significant implications for deep reinforcement learning training process. This observation suggests that the pruning starting from the early phase is vital for the success of the training. We think that it is able to represent a broader phenomenon in DNNs.

Chapter 6

Conclusion

6.1 Summary

In this thesis, we investigated different neuromorphic paradigms to solve different real-life deep learning problems, including pattern recognition, autonomous planning and control for UAV. Different framework structures were investigated to achieve higher accuracy, more precise planning and higher computational efficiency.

In Chapter 2, we discussed a recurrent belief propagation based offline handwriting recognition framework. A probabilistic inference network that performs recurrent belief propagation was developed to process the recognition results of deep convolutional neural network and formed individual characters to words. The post processing has the capability of correcting deletion, insertion and replacement errors in a noisy input. The output of the inference network was a set of words with their probability of being the correct one. With the purpose of limiting the number of candidate words, a series of improvements have been made to the probabilistic inference network, including using a post Gaussian Mixture Estimation model to prune insignificant words. With incremental comparison experiments, we proved the proposed framework was efficient to achieve the desired performance.

In Chapter 3, we proposed a two-level framework to generate the navigation trajectory that the UAV follows in a complex environment. The construction of the framework, the processing and analysis of experimental results were introduced. The proposed trajectory planning framework can avoid obstacles in complex indoor environments and use minimal control thrust consumption during flight. It can not only maintain the stability of the UAV, but also ensure its safety and energy efficiency in real time. The experimental results indicated that it was versatile enough to be applied to other robotic tasks, such as package delivery and routing conflicts for high-density UAVs.

In Chapter 4, we applied a deep reinforcement learning framework based on actor critic algorithm to track under-actuated aerial vehicles through a given set of required waypoints. The system structure, processing algorithm and software/hardware performance were introduced. In our approach, the UAV can track the desired trajectory through a set of predefined waypoints and can tolerate random Gaussian noise in a considerable range. In addition, the hardware consumption to implement this scheme was provided. The same as before, our proposed scheme was universal and suitable for applications in real UAVs for fast and accurate trajectory tracking systems.

In order to optimize performance and computational efficiency, an early integrated neural network compression for DRL based trajectory planning system was presented in Chapter 5. We found that from the early phase of the DRL training process, structured weight pruning can be applied to achieve an actual success. The experimental results proved that this system can not only save more training effort and achieve faster inference speed, but also retain high success rate on the basis of greatly reducing the number of FLOPs, which cannot be achieved before. We also observed that layers that far from the input have more weight redundancy. Our results indicated that pruning from the early phase is of great significance for the successful training of DRL. We have reason to believe that it can represent a broader phenomenon in

deep neural networks.

6.2 Future Direction

In this thesis, we studied the autonomous waypoints planning and trajectory generation scheme in known environment. However, more often we cannot obtain complete external environmental information. Driven by extensive robotics technology, there is an increasing demand for real-time on-board autonomy of UAVs without external navigation equipment. Therefore, one unified real-time autonomous system is urgently needed. In future research, firstly, our goal is to expand the generality by combining unknown 3D environments exploration and localization and mapping of the UAV. The action space will be considered continuously without ignoring information of time and space. In such unified system, the UAV not only needs to consider position and control thrust consumption, but also considers the exploration efficiency when taking each action. In addition, the environment placement will be dynamic while planning and exploring. This means that the destination, location of obstacles will not be fixed. The UAV will only have knowledge of areas that have been explored and detected before. As a result, the unpredictability of the environment is taken into consideration during planning and exploration. These reconstruction aspects can be used to further improve the versatility and reality of the system. Secondly, the exploration and localization combined problem will be more time consuming when implementing with DRL. Thus, we will further apply the early phase integrated weight compression system to the unified autonomous system. As it is discussed in this thesis, the early phase integrated weight compression is a vital for success training of DRL problem. Therefore, a real-time and comprehensive unified autonomous system can be realized on the real UAV.

Bibliography

[1] S. P. Viswanathan, A. K. Sanyal, and E. Samiei, "Integrated guidance and feedback control of underactuated robotics system in se (3)," *Journal of Intelligent & Robotic Systems*, 2018.

[2] Y. Li, H. Eslamiat, N. Wang, Z. Zhao, A. K. Sanyal, and Q. Qiu, "Autonomous waypoints planning and trajectory generation for multi-rotor uavs," in *Proceedings of the Workshop on Design Automation for CPS and IoT*, 2019, pp. 31–40.

[3] P. M. Wyder *et al.*, "Autonomous drone hunter operating by deep learning and all-onboard computations in gps-denied environments," *PloS one*, vol. 14, no. 11, p. e0225092, 2019.

[4] H.-H. Wu, Z. Zhou, M. Feng, Y. Yan, H. Xu, and L. Qian, "Real-time single object detection on the uav," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2019, pp. 1013–1022.

[5] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5687–5695.

[6] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," *arXiv preprint arXiv:1701.05369*, 2017.

[7] R. T. Lange, "The lottery ticket hypothesis: A sur-

vey," *https://roberttlange.github.io/year-archive/posts/2020/06/lottery-ticket-hypothesis/*, 2020. [Online]. Available: https://roberttlange.github.io/posts/2020/06/lottery-ticket-hypothesis/

[8] R. Hecht-Nielsen, *Confabulation theory: the mechanism of thought.* Springer, 2007.

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

[10] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

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

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[13] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[14] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[15] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, "Stabilizing the lottery ticket hypothesis," 2020.

[16] H. Yu, S. Edunov, Y. Tian, and A. S. Morcos, "Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp," *arXiv preprint arXiv:1906.02768*, 2019.

[17] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[18] T. Zhang, K. Zhang, S. Ye, J. Li, J. Tang, W. Wen, X. Lin, M. Fardad, and Y. Wang, "Adam-admm: A unified, systematic framework of structured weight pruning for dnns," *ArXiv*, vol. abs/1807.11091, 2018.

[19] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[21] S. Bhowmik, M. G. Roushan, R. Sarkar, M. Nasipuri, S. Polley, and S. Malakar, "Handwritten bangla word recognition using hog descriptor," in *2014 Fourth International Conference of Emerging Applications of Information Technology*. IEEE, 2014, pp. 193–197.

[22] M. Dehghan, K. Faez, M. Ahmadi, and M. Shridhar, "Holistic handwritten word recognition using discrete hmm and self-organizing feature map," in *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*, vol. 4. IEEE, 2000, pp. 2735–2739.

[23] P. Jifroodian-Haghighi, "A discrete hidden markov model for the recognition of handwritten farsi words," Ph.D. dissertation, Concordia University, 2010.

[24] V. Frinken and S. Uchida, "Deep blstm neural networks for unconstrained continuous handwritten text recognition," in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2015, pp. 911–915.

[25] B. Su, X. Zhang, S. Lu, and C. L. Tan, "Segmented handwritten text recognition with recurrent neural network classifiers," in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2015, pp. 386–390.

[26] S. Yao, Y. Wen, and Y. Lu, "Hog based two-directional dynamic time warping for handwritten word spotting," in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2015, pp. 161–165.

[27] M. Dehghan, K. Faez, M. Ahmadi, and M. Shridhar, "Handwritten farsi (arabic) word recognition: a holistic approach using discrete hmm," *Pattern Recognition*, vol. 34, no. 5, pp. 1057–1065, 2001.

[28] A. Gupta, M. Srivastava, and C. Mahanta, "Offline handwritten character recognition using neural network," in *2011 IEEE International Conference on Computer Applications and Industrial Electronics (ICCAIE)*. IEEE, 2011, pp. 102–107.

[29] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[30] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of 2010 IEEE international symposium on circuits and systems*. IEEE, 2010, pp. 253–256.

[31] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," in *Advances in neural information processing systems*, 2009, pp. 545–552.

[32] E. Qaralleh, G. Abandah, and F. T. Jamour, "Tuning recurrent neural networks for recognizing handwritten arabic words," 2013.

[33] Q. Qiu, Q. Wu, M. Bishop, R. E. Pino, and R. W. Linderman, "A parallel neuromorphic text recognition system and its implementation on a heterogeneous high-performance computing cluster," *IEEE Transactions on Computers*, vol. 62, no. 5, pp. 886–899, 2012.

[34] Q. Qiu, Z. Li, K. Ahmed, H. H. Li, and M. Hu, "Neuromorphic acceleration for context aware text image recognition," in *2014 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2014, pp. 1–6.

[35] Q. Qiu, Z. Li, K. Ahmed, W. Liu, S. F. Habib, H. H. Li, and M. Hu, "A neuromorphic architecture for context aware text image recognition," *Journal of Signal Processing Systems*, vol. 84, no. 3, pp. 355–369, 2016.

[36] N. Yoder, "Peakfinder: Quickly finds local maxima (peaks) or minima (valleys) in a noisy signal," 2014.

[37] O. Matan, C. J. Burges, Y. LeCun, and J. S. Denker, "Multi-digit recognition using a space displacement neural network," in *Advances in neural information processing systems*, 1992, pp. 488–495.

[38] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675–678.

[39] M. T. Tree, M. Cut, and P. C. T. Salesman, "From wikipedia, the free encyclopedia."

[40] "Mieliestronk dictionary." [Online]. Available: http://www.mieliestronk.com/wordlist.html

[41] A. M. Zyarah, "Design and analysis of a reconfigurable hierarchical temporal memory architecture," 2015.

[42] Q. Qiu, Q. Wu, D. J. Burns, M. J. Moore, R. E. Pino, M. Bishop, and R. W. Linderman, "Confabulation based sentence completion for machine reading," in *2011 IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB)*. IEEE, 2011, pp. 1–8.

[43] Z. Li and Q. Qiu, "Completion and parsing chinese sentences using cogent confabulation," in *2014 IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB)*. IEEE, 2014, pp. 31–38.

[44] Z. Li, Q. Qiu, and M. Tamhankar, "Towards parallel implementation of associative inference for cogent confabulation," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2016, pp. 1–6.

[45] "Machine learning and pattern recognition density estimation: Gaussians." [Online]. Available: http://www.inf.ed.ac.uk/teaching/courses/mlpr/lectures/mlprgaussian.pdf

[46] T. E. De Campos, B. R. Babu, M. Varma *et al.*, "Character recognition in natural images." *VISAPP (2)*, vol. 7, 2009.

[47] F. Pereira, M. Riley, and R. Sproat, "Weighted rational transductions and their application to human language processing," in *Proceedings of the workshop on*

Human Language Technology. Association for Computational Linguistics, 1994, pp. 262–267.

[48] "240 common spelling mistakes in english." [Online]. Available: Available:http://www.engvid.com/english-resource/common-spellingmistakes-in-english/

[49] M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," *Robotica*, vol. 34, pp. 467–497, 2015.

[50] M. Hehn and R. DAndrea, "Real-time trajectory generation for quadrocopters," *Robotics, IEEE Transactions on*, vol. 31, no. 4, pp. 877–892, 2015.

[51] J. Sanchez-Lopez, S. Saripalli, P. Campoy, J. Pestana, and C. Fu, "Toward visual autonomous ship board landing of a VTOL UAV," in *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*. IEEE, May 2013, pp. 779–788.

[52] L. Singh and J. Fuller, "Trajectory generation for a uav in urban terrain, using nonlinear mpc," in *American Control Conference, 2001. Proceedings of the 2001*, vol. 3. IEEE, 2001, pp. 2301–2308.

[53] M. Hehn and R. D'Andrea, "Quadrocopter trajectory generation and control," in *IFAC world congress*, vol. 18, no. 1, 2011, pp. 1485–1491.

[54] W. Xu, J. Wei, J. M. Dolan, H. Zhao, and H. Zha, "A real-time motion planner with trajectory optimization for autonomous vehicles," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012.

[55] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[56] S. A. Bortoff, "Path planning for uavs," in *American Control Conference. Proceedings of the 2000*, vol. 1, no. 6. IEEE, 2000.

[57] J. Tisdale, Z. Kim, and J. K. Hedrick, "Autonomous uav path planning and estimation," *IEEE Robotics & Automation Magazine*, vol. 16, no. 2, 2009.

[58] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[59] N. Imanberdiyev, C. Fu, E. Kayacan, and I.-M. Chen, "Autonomous navigation of uav by using real-time model-based reinforcement learning," in *Control, Automation, Robotics and Vision (ICARCV), 2016 14th International Conference on.* IEEE, 2016, pp. 1–6.

[60] P. K. Das, S. Mandhata, H. Behera, and S. Patro, "An improved q-learning algorithm for path-planning of a mobile robot," *International Journal of Computer Applications*, vol. 51, no. 9, 2012.

[61] L. Tai and M. Liu, "Towards cognitive exploration through deep reinforcement learning for mobile robots," *arXiv preprint arXiv:1610.01733*, 2016.

[62] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[63] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research.* Springer, 2016, pp. 649–666.

[64] F. Stoican and D. Popescu, *Trajectory Generation with Way-Point Constraints for UAV Systems.* Springer International Publishing, 2016.

[65] M. P. Vitus, W. Zhang, and C. J. Tomlin, "A hierarchical method for stochas-

tic motion planning in uncertain environments," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2263–2268, 2012.

[66] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, Dec 2015.

[67] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3357–3364.

[68] J. Wu, S. Shin, C.-G. Kim, and S.-D. Kim, "Effective lazy training method for deep q-network in obstacle avoidance and path planning," in *Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1799–1804.

[69] R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, and A. Cangelosi, "Toward end-to-end control for uav autonomous landing via deep reinforcement learning," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2018, pp. 115–123.

[70] A. E. Eiben, J. E. Smith *et al.*, *Introduction to evolutionary computing*. Springer, 2003, vol. 53.

[71] L. Pacheco and N. Luo, "Testing pid and mpc performance for mobile robot local path-following," *International Journal of Advanced Robotic Systems*, vol. 12, no. 11, p. 155, 2015.

[72] S. K. Malu and J. Majumdar, "Kinematics, localization and control of differential drive mobile robot," *Global Journal of Research In Engineering*, 2014.

[73] J. A. Nestor, "A new look at hardware maze routing," in *Proceedings of the 12th ACM Great Lakes symposium on VLSI*. ACM, 2002, pp. 142–147.

[74] R. b. Omar, "Path planning for unmanned aerial vehicles using visibility line-based methods," Ph.D. dissertation, University of Leicester, 2012.

[75] S. Koenig and M. Likhachev, "Dˆ* lite," *Aaai/iaai*, vol. 15, 2002.

[76] B. Lau, C. Sprunk, and W. Burgard, "Improved updating of euclidean distance maps and voronoi diagrams," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 281–286.

[77] H. Eslamiat, Y. Li, N. Wang, S. Amit, and Q. Qiu, "Autonomous waypoint planning, optimal trajectory generation and nonlinear tracking control for multi-rotor uavs," 03 2019.

[78] F. Sassi, M. Abbes, and A. Mami, "Fpga implementation of pid controller," *Proceedings-Copyright IPCO*, 2014.

[79] L. Marconi and R. Naldi, "Robust full degree-of-freedom tracking control of a helicopter," *Automatica*, 2007.

[80] ——, "Aggressive control of helicopters in presence of parametric and dynamical uncertainties," *Mechatronics*, 2008.

[81] P. E. Pounds, D. R. Bersak, and A. M. Dollar, "Stability of small-scale uav helicopters and quadrotors with added payload mass under pid control," *Autonomous Robots*, 2012.

[82] Y. Chen and Q. Wu, ""design and implementation of pid controller based on fpga and genetic algorithm"," in *Electronics and Optoelectronics (ICEOE), 2011 International Conference on*. IEEE, 2011.

[83] F. Goodarzi, D. Lee, and T. Lee, "Geometric nonlinear pid control of a quadrotor uav on se (3)," in *Control Conference (ECC), 2013 European.* IEEE, 2013.

[84] N. Mohajerin and S. L. Waslander, "Modelling a quadrotor vehicle using a modular deep recurrent neural network," in *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on.* IEEE, 2015.

[85] M. Collotta, G. Pau, and R. Caponetto, "A real-time system based on a neural network model to control hexacopter trajectories," in *Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM), 2014 International Symposium on.* IEEE, 2014.

[86] A. Din, B. Bona, J. Morrissette, M. Hussain, M. Violante, and M. F. Naseem, "Embedded low power controller for autonomous landing of uav using artificial neural network," in *Frontiers of Information Technology (FIT), 2012 10th International Conference on.* IEEE, 2012.

[87] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, "Deep neural networks for improved, impromptu trajectory tracking of quadrotors," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on.* IEEE, 2017.

[88] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press Cambridge, 1998.

[89] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2013.

[90] S. Yue, D. Zhu, Y. Wang, and M. Pedram, "Reinforcement learning based dynamic power management with a hybrid power supply," in *Computer Design (ICCD), 2012 IEEE 30th International Conference on.* IEEE.

[91] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[92] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[93] T. Yang *et al.*, "Deep neural network energy estimation tool," in *IEEE Conference on Computer Vision and Pattern Recognition CVPR*, 2017, pp. 21–26.

[94] D. H. Dario Amodei, *AI and Compute*, 2018. [Online]. Available: https://openai.com/blog/ai-and-compute/

Vita

Yilan Li received her B.S. degree in Information Engineering in 2012 from Xi'an Jiaotong University, Xi'an, China and M.S. degree in Electrical Engineering in 2015 from Syracuse University, Syracuse, NY, US. She completed her Ph.D. degree in the Department of Electrical Engineering and Computer Science of Syracuse University in 2021. Her research interests are in the neuromorphic systems for cyber physical applications, such as pattern recognition and autonomous systems.