Thesis for Master's Degree

Learning Through Experience:

A Self-Learning Collision Avoidance System

Hyunjun Han

School of Electrical Engineering and Computer Science

Gwangju Institute of Science and Technology

2018

Learning Through Experience:

A Self-Learning Collision Avoidance System

경험을 통한 학습:

자가학습 기반의 충돌 회피 시스템

Learning Through Experience:

A Self-Learning Collision Avoidance System

Advisor : Professor Heung-No Lee

By

Hyunjun Han

Graduate: School of Electrical Engineering and Computer Science

Gwangju Institute of Science and Technology

A thesis submitted to the faculty of the Gwangju Institute of Science and Technology in partial fulfillment of the requirements for the degree of Master of Science in the School of Electrical Engineering and Computer Science

Gwangju, Republic of Korea

2017. 11. 30

Approved by [illegible]

Professor Heung-No Lee

Committee Chair

Learning Through Experience: A Self-Learning Collision Avoidance System

Hyunjun Han

Accepted in partial fulfillment of the requirements for the degree of Master of Science

11. 30. 2017

Committee Chair [illegible]
Prof. Heung-No Lee

Committee Member [illegible]
Prof. Kiseon Kim

Committee Member [illegible]
Prof. Jong Won Shin

MS/EC
20161107

Hyunjun Han (한현준). Learning Through Experience: A Self-Learning Collision Avoidance System (경험을 통한 학습: 자가학습 기반의 충돌 회피 시스템). School of Electrical Engineering and Computer Science. 2018. 36p. Prof. Heung-No Lee

Abstract

Collision avoidance, or obstacle avoidance, is a fundamental problem dealt in autonomous navigation that allows unmanned vehicles, such as UAVs, to perform tasks in unknown environments. In this paper, we introduce simple and effective deep learning based collision avoidance that does not require any pre-acquired training dataset for learning. We take a method based on experiencing adverse events, or events that must not be repeated in the future. This is accompanied by an online, self-learning algorithm combining the concept of both reinforcement learning and supervised learning that allows UAV to gain experiences and learn by itself. Through taking both learning mechanism approach, we make agent learn fast that does not require any human efforts into gathering and labelling of training dataset. In the algorithm, two procedures are performed during its training phase; data gathering procedure and learning procedure. We run simulation tests on its effectiveness in its learning rates and its learning performance. Simulation results show that our system can achieve minimum of 99.25% up to 100% accuracy in classifying collision event from all possible events, allowing UAV to freely fly without collision in simulated unknown environments.

Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation

From military, commercial, to personal uses, Unmanned Aerial Vehicles (UAVs), with its growing popularity, are proven to be extremely beneficial in accomplishing tasks within places unreachable by man. Especially with the aid of rapidly growing deep learning techniques, UAV related technologies are actively being developed that increases its potential to be used as an effective tool. Among those is collision avoidance, or obstacle avoidance, which is a fundamental system that ensures UAVs to autonomously fly without user controls and perform tasks in unknown environments.

In this paper, we propose a new deep learning based collision avoidance system that learns by itself and can adapt to any environments using a single simulated depth sensor. We achieve this through a learning algorithm that combines advantages of both supervised learning and reinforcement learning that ultimately allows agent to interact with its environment, gather data, and train throughout its learning procedure without requiring any pre-acquired training dataset. As a result, we achieved high accuracy in avoiding collision within simulated environments using an LSTM network.

1.2 Thesis Outline

The rest of the paper will be organized as follows:

In Chapter 2, prerequisite knowledges regarding the types of machine learning related to the proposed system will be discussed to help readers better understand the context. Supervised learning, reinforcement learning, and deep learning will be introduced along with some examples. Deep learning will be categorized into three sub-sections, namely Convolutional Neural Network, Deep Q-learning Network, and Recurrent Neural Network.

Chapter 3 will briefly discuss about related works regarding the proposed system with respect to two types of machine learning approach; supervised learning and reinforcement learning.

In Chapter 4 and 5, the proposed collision avoidance system, simulation, and its result will be discussed in detail.

In Chapter 6, we will conclude with summary and future work.

2. Prerequisite Knowledge

2.1 Machine Learning

Machine Learning is "a field of study that gives computers the ability to learn without explicitly programmed." This definition was stated by an American pioneer in the field of computer gaming and artificial intelligence, Arthur Samuel, who coined the term "machine learning" in 1959 [1]. Tom M. Michell, a former Chair of the Machine Learning Department at CMU, quoted that a computer program is said to learn from experience *E* with respect to some task *T* and some performance measure *P*, if its performance on *T*, as measured by *P*, improves with experience *E* [2]. As such, machine learning is a study that evolved from the study of pattern recognition and computational learning theory in artificial intelligence that seeks constructions of algorithms that can learn and make predictions on data [3].

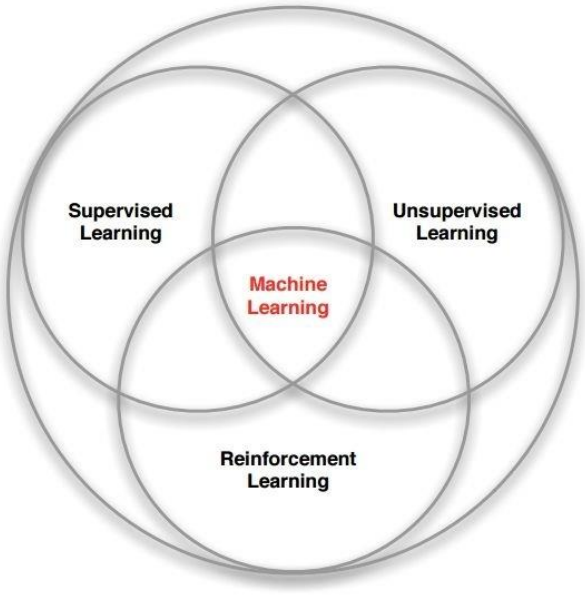


Fig. 1. Machine learning and its types.

Machine learning in general can be divided into three types. First one is supervised, or predictive, learning. In this approach, the goal is to map input *x* into output *y* in a given input pairs $D = \{(x_i, y_i)\}_{i=1}^{N}$, where *D* is a training set and *N* is the number of training examples. As a simple example, input x_i in each training set is a number of D-dimensional vector, and, say, it represents a man's height and weight. This is called features, attributes, or covariates. However, in general, x_i can be into a form of complicated structured instances, such an image, a sentence, e-mail message, sequential data, or even graphs. Similarly, the output form y_i, or response variable, theoretically can be in any format, but in most cases is limited to a set of categorical or nominal variables such that $y_i \in \{1, \cdots, C\}$. If y_i is in the form of scalar or categorical variable, the problem is defined to be *classification* or *pattern recognition* problem, where as it is considered as a *regression* problem if y_i is in a form of real numbers. As another variant form, *ordinal regression* is labelling given inputs into different ranks, such as A through F, within the label space Y that has a natural ordering property.

Second type of machine learning is known as unsupervised, or descriptive, learning. In this approach, only the input $D = \{x_i\}_{i=1}^{N}$ is given without any wanted outputs y_i, and the goal is to discover "interesting patterns" among the given input data. This can be referred as a *knowledge discovery*. Because the resulting pattern from unsupervised learning is often unpredictable and cannot be predefined, this area is still in the progress of being defined, and thus, no accurate measurement, or metric, for error exists yet (Unlike supervised learning, produced output y of given x cannot be mapped to predicted output to measure error). Unsupervised learning has not been used in this research, and thus will not be further discussed in this paper.

The last type of machine learning is reinforcement learning. This approach uses a method of giving reward or negative reward that is especially effective at making agent learn to take specific actions at its environment. The biggest difference between supervised learning is that there exists no correct input and output pairs nor corrections on sub-optimal actions that agent takes. The only attempt made in reinforcement learning is finding a policy that can map the environment states to actions that learning agent should take in those states that ultimately lead to maximization of future reward.

In the following sub-chapters, supervised learning, reinforcement learning, and deep learning will be discussed in detail as a prerequisite knowledge to help readers understand the rest of the paper regarding the main contribution of the paper, a self-learning approach in making agent learn to avoid collision using concept from both supervised and reinforcement learning.

2.2 Supervised Learning

According to S.B. Kotsiantis [4], the goal of supervised learning is to build a concise model of the distribution of class labels in terms of predictor features, where the term *supervised* means to have each instances of training data be labelled in contrast to *unsupervised learning* where instances are not labelled. It is the most common form of machine learning that is widely used and applied in many fields, and is divided into two problems; classification and regression.

In classification problem of supervised learning, the goal is to map input *x* to output *y*, $y \in \{1, \cdots, C\}$ where *C* indicates the number of classes. If C = 2, it is referred to as a binary classification problem, and often denoted as $y \in \{0,1\}$. If C > 2, it is called multiclass classification, or multi-label classification if the labels are mutually exclusive. The problem to classification is often approached using function approximation. For an unknown function *f*, we assume *y=f(x)*, and by using $\hat{y} = \hat{f}(x)$, the goal of learning is to predict *f*, a function of training set that contains all labels, Thus, we want a generalization to an input that has not been seen in training set through predictions on learnt training sets.

How is classification used in real life? Some of the applications to classification are categorizing documents and email spam filtering, classification of dogs according to its different body size, color, and breed, and recognition of hand written numbers, such as using MNIST (Modified National Institute of Standards). Recent advances in machine learning with the aid of deep learning techniques, which will be discuss in the proceeding subchapters, has allowed facial recognition which is accurate as 97.35% on the Labeled Faces in the Wild (LFW) dataset in 2014 [5].

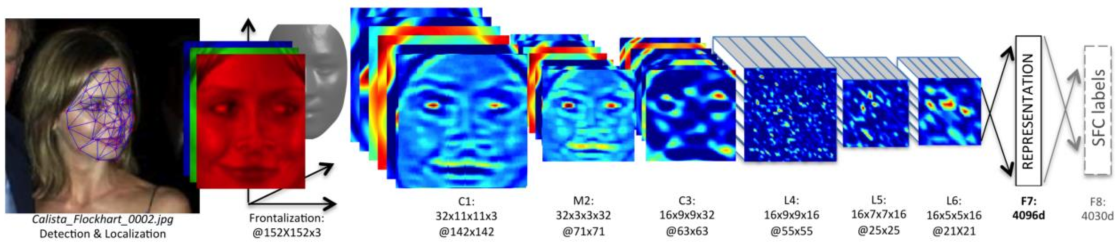


Fig. 2. An example of supervised learning. Wolf et al. [5] achieved 97.35% accuracy in facial recognition in 2014 using deep learning.

Regression, the other part of supervised learning, is similar to classification other than the continuity of response variable. Fig. 3 shows a diagram that shows a difference between different regressions where the input of a single real number is $x_i \in \mathbb{R}$, and the response to a single real number is $y_i \in \mathbb{R}$. These simple problems have multi-dimensional input, singular value, non-smooth response that can be expanded into various forms.

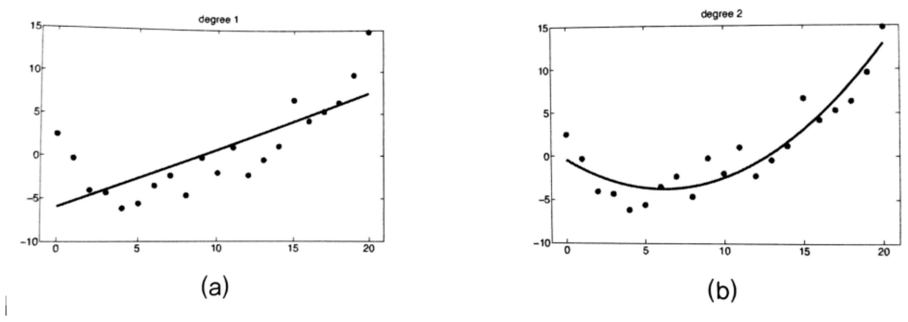


Fig. 3. (a) Linear regression of a 1-D data. (b) Polynomial regression on the same data.

The followings are real regression problems that occur in real life.

- Estimation of future stock price given current stock price along with side information.
- Estimation of probability of economic growth given past data regarding financial information and population.
- Estimation of average age of subscribed members of a specific Youtube channel given a portion of user information.
- Estimation of temperature within a building given weather information and number of people inside.

Advantages and disadvantages of supervised learning can be as follows.

Advantages:

- Specific definition of classes, that is the pairs of input and output, results into clear decision boundary that can distinguish different classes accurately after training process.

- Number of classes that can explicitly be determined by the user

Disadvantages:

- Overtraining that result into overfitting. Overfitting is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably .
- High dependency on training set. Training set must carefully be selected that can generalize the whole features of data including data that are not available in the training dataset, such as testset.
- Large amount of dataset required for training that must also be labelled. This makes creation of training dataset a very time-consuming and laborious work.

2.3 Reinforcement Learning

Reinforcement learning is another type of learning in which the training information provided to the learning system by the environment is in the form of a scalar signal (reinforcement signal) that measures how well the system operates through discovering actions that maximizes future reward without explicitly being told which actions to take [4]. Simply, reinforcement learning allows agents to determine the ideal behavior by itself within a specific context to maximize its performance through feedback from its interacting environment. This environment is typically formulated as MDP, which also known as Markov Decision Process, which can be described as a discrete time stochastic control process. In detail, MDP is a 5-tuple process (S, A, r, P, γ), and at time *t*, the system is in state $s_t \in S$, the agent chooses an action $a_t \in A$, resulting the system transfer to a new state s_{t+1}, and a corresponding reward $r(s_t, a_t)$ is given to the agent with new state s_{t+1} randomized according to a transition probability function, $P(\cdot | s_t, a_t)$ [6]. Thus, the agent's goal is to maximize cumulative reward $R_t = \sum_{t=0}^{n} \gamma^t r_{t+1}$, where $\gamma \in (0,1]$ is a discount factor, by choosing a policy π, a function that specifies the action π (*s*) that the agent will choose during state *s*.

Q-learning is one of a model-free reinforcement learning that can be used for finding an MDP's optimum policy, defined by,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)].$$

In Q-learning, a Q function that predicts the expected reward resulting from taking an action at current state is optimized thereby learning the optimal policy π , also known as action-value function denoted as $Q^*(s,a) = E_\pi[R_t \mid s_t = s, a_t = a] = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\}$. Table 1 shows an algorithm for Q-learning [7].

One advantage of this approach is that no model is required unlike utility learning, another type of reinforcement learning that uses utility function and selects actions that maximize expected utility. However, agent can only have shallow knowledge due to limited access to future data and its restriction on its ability to learn, which is only through *s*.

TABLE I

Algorithm 1 Q-learning.

```
Initialize $Q(s,a,)$, $\forall s \in S, a \in A(s)$, arbitrarily, and Q(terminal-state,·)=0
For each episode:
    Initialize $s$
    for each step of episode:
        Choose action $a$ from $s$ using policy derived from $Q$
        Take action $a$, observe $r$, $s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$
    Until $s$ is terminal
```

Reinforcement learning results into an automated procedure that allows a trial and error learning which selects actions based on its past experiences as well as making new decisions, thus achieving both exploitation and exploration throughout the learning process. One advantage over supervised learning is that learning process does not require user's involvement throughout learning process as opposed to supervised learning, and therefore requires less manual laborious work, such as gathering of training data and labelling work. Reinforcement Learning combined with deep learning techniques has been one of the most promising researches recently and will be discussed in the following chapter.

2.4 Deep Learning

Deep learning is a class of machine learning algorithms and is based on learning data representations. Deep learning can be defined with following characteristics [8].

- Cascade of multiple layers of nonlinear processing units, which is used for feature extraction and transformation, where each successive layer uses previous layer's output as input.
- Can learn through supervised, unsupervised, and reinforcement learning methods.
- Learn multiple levels of representations that correspond to different levels of abstraction.
- Training done through loss functions, such as Stochastic Gradient Descent, for backpropagation that calculates distance between ground truth and predicted value.

Deep learning architecture is a multilayer stack of simple modules which all constitute to learning and computing non-linear input-output mappings. With a collection of multiple non-linear layers, a system can implement extremely intricate functions from its inputs to distinguish even the extremely sensitive subjects [9].

Deep learning is unique in a way that it's performance depends on how a machine should change its internal parameters in the discovery process of intricate structure in large dataset through back-propagation process. Thus, layers of features are not designed by man, but learnt through general-purpose learning, or feature learning, procedure.

Founding father of convolutional nets, Yann LeCun, in his paper [9] states that deep learning is a technique that brought breakthroughs in processing images, videos, speech and audio using deep convolutional nets, and have shone light on sequential data such as text and speech using recurrent nets. In the following sections, convolutional neural network, deep q-learning network, and recurrent neural network will briefly be explained.

2.4.1 CNN

Convolutional Neural Network (CNN) [10] is a network inspired by the natural visual perception mechanism of the living creatures [11] designed to effectively process high dimensional data which can be expressed in the form of multiple array such as 3 color channeled RGB images. It is a multi-layer artificial neural network that can obtain effective representations of the original image and recognize patterns within directly via raw pixels through backpropagation algorithm [12]. Fig. 4 shows a typical example of a Convolutional Neural Network, LeNet-5, one of the first of its kind that could classify handwritten digits [13].

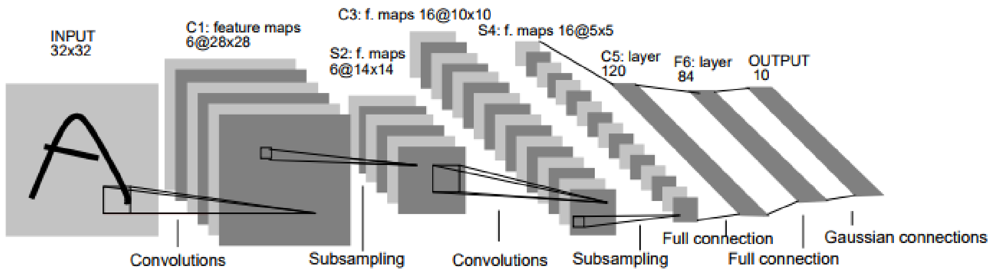


Fig. 4. Architecture of LeNet-5 proposed in 1998 by Yann LeCun, a Convolutional Neural Network for handwritten digit classification.

As shown in the figure, CNN is composed of three types of layers; convolutional, pooling, and fully-connected layers, and through these layers, the network learns feature representations of its input data, which are training sets, by repeatedly passing through convolution layer and pooling layer. The resulting feature maps starts first with basics such as representation of edges, followed through by arrangements of edges, and the third layer may assemble motifs into larger combinations that represent part of familiar objects. The point is those subsequent layers show combinations of parts that have been learnt in the previous layers, and that these are not engineered intentionally by human hands, but through input data's general-purpose learning [9]. AlexNet, proposed by Krizhevsky et al. in 2012 [14], is a simple structured CNN similar to that of LeNet-5 but with addition to using 2 GPUs, dropout technique, and ReLU. Their proposed architecture achieved breakthrough in the area of image classification at ImageNet Large-Scale Visual Recognition Challenge, ILSVRC-2012, with accuracy of top-5 error rate of 15.3%, bringing spotlight to the power of deep learning and convolutional neural network to the world. Fig. 5 is a visualization of AlexNet of each feature map after subsequent layers which are designed by the network. Note that there exists strong grouping within each feature map, greater invariance at higher layers, and exaggeration of discriminative parts of the image [15].

Since then, outstanding achievements using Convolutional Neural Networks have been proposed, setting new records every following ILSVRC challenges. Some examples include ZFNet, VGGNet [16], GoogLeNet [17], ResNet [18] that achieved performance of top-5 error rate of 11.7%, 7.3%, 6.7%, and 3.57% accordingly. As such, CNN have been developed to deeper and more complicated network structures that succeeds human's performance, approximated to be 5%, in 2015.

However, it must be noted that Convolutional Neural Networks have thousands to millions of free parameters to train and deep, complicated network structure. Therefore, it is necessary to have high-performance Graphical Processing Unit (GPU) to perform computation heavy training process and large training data that can be accommodated to fully train is required, not to mention the long period of time it takes to fully train them.

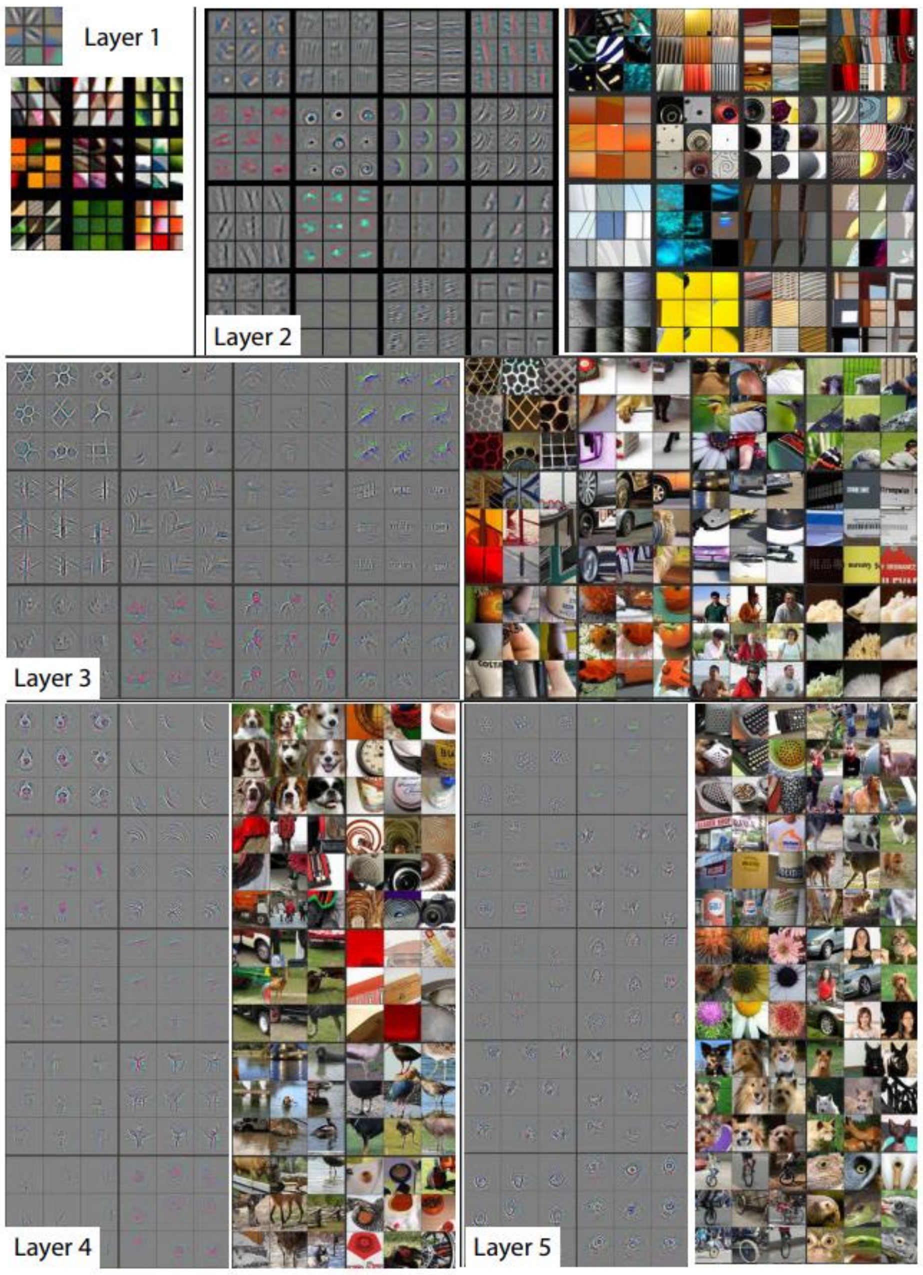


Fig. 5. Work of Matthew D. Zeiler et. al on visualization of feature maps of AlexNet.

2.4.2 DQN

Also known as Deep Q-Network (DQN), this new deep learning technique is a combination of Q-learning, mentioned in Section 2.3, and Convolutional Neural Network, and was proposed in the 2013 by Minh, V., et al regarding agents learning to play Atari 2600 games [19]. Using a single neural network agent, their goal was to learn successfully to play as many games possible through visual RGB input (210 x 160 x 3), without any game-specific information or hand-designed visual features given.

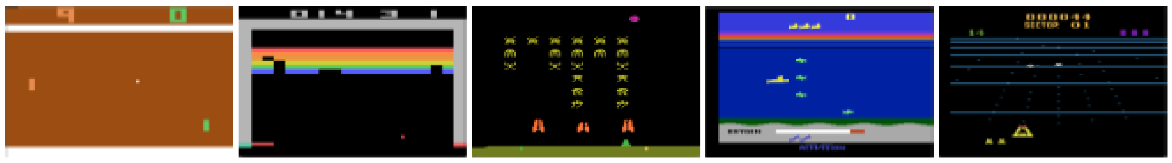


Fig. 6. Atari 2600 games for DQN testbed.

In their work, they introduce what is known as experience replay [20], where agent's experiences at each time-step, $e_t = (s_t, a_t, r_t, s_{t+1})$, is stored in a dataset $D = e_1, ..., e_N$ pooled over many episodes into a replay memory [19]. Table 2 shows the algorithm for DQN with experience relay. Through experience replay approach that utilizes historical information of fixed lengths, several advantages over standard online Q-learning [7] could be achieved. First is greater data efficiency that comes from multiple weight updates from each step of experiences. Second is effective sampling through random selection that breaks correlation between samples, reducing variance of updates. Third advantage is oscillation or parameter divergence avoidance through averaged multiple previous states which overcomes falling into local minimum.

TABLE II

Algorithm 2 Deep Q-learning with Experience Replay

```
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for epidsode=1, M do
    Initialize sequence $s_1=\{x_1\}$ and preprocessed sequenced $\phi_1=\phi(s_1)$
    for t=1, T do
        with probability ε select a random action $a_t$
        otherwise select $a_t=\max_a Q^*(\phi(s_1),a;\theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1}=s_t,a_t,x_{t+1}$ and preprocess $\phi_{t+1}=\phi(s_{t+1})$
        Store transition $(\phi_t,a_t,r_t,\phi_{t+1})$ in D
        Sample random minibatch of transitions $(\phi_j,a_j,r_j,\phi_{j+1})$ from D
        set $y_j=\begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j+\gamma\max_{a'} Q(\phi_{j+1},a';\theta) & \text{for non-terminal } \phi_{j+1}\end{cases}$
        Perform a gradient descent step on $(y_j-Q(\phi_{j+1},a';\theta))^2$ accroding to
            equation 3
    end for
end for
```

Where *equation 3* is

$$\nabla_{\theta_i} L_i(\theta_i)=\mathrm{E}_{s,a\sim p(\cdot);s'\sim\varepsilon}\left[\left(r+\gamma\max_{a'}Q(s',a';\theta_{i-1})-Q(s,a;\theta_i)\right)\nabla_{\theta_i}Q(s,a;\theta_i)\right].$$

Because 210 x 160 x 3 image data requires computation heavy process, input dimensionality has been made by converting the RGB to gray-scale and down-sampling to 110 x 84, with additional cropping into an 84 x 84 x 4 image that roughly summarizes the whole scene. Inputs to neural network were state representations and outputs were number of valid actions that an agent can perform within games, ranging from 4 to 18 depending on the games.

In 2015, Volodymyr Mnih et al. introduce DQN that further test its performance on 49 Atari games using the same network [21], and achieved more than 75% of the human score on 29 games. Fig. 7 shows the DQN architecture diagram. Unlike supervised learning where all labelled training data must be prepared beforehand, a single architecture of DQN required only very minimal prior knowledge i.e., pixel and the game score as inputs, and could achieve high performance on most of the environments using a single algorithm which was the result of interaction of agent with its environment by selecting actions that gives maximum future reward.

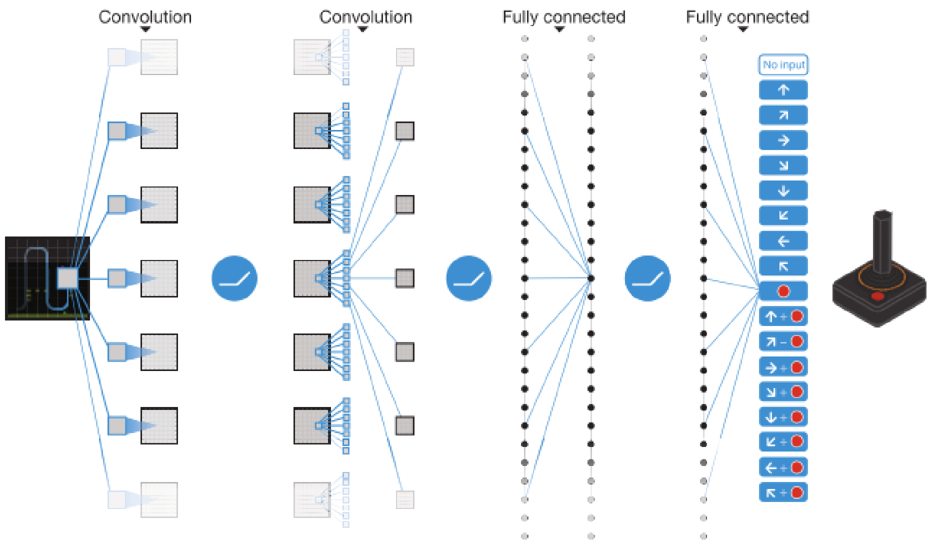


Fig. 7. DQN used for training agent in Atari 2600 based on a convolutional neural network.

2.4.3 RNN

Recurrent Neural Networks (RNN) are connectionist models with the ability to selectively pass information across sequence steps, while processing sequential data one element at a time allowing the network to input and/or output sequences of data that are not independent [22]. Fig. 8 is a simple diagram illustrating RNN structure. This structure allows RNN output for recognition, production, or prediction problem to be applicable in many fields, such as machine translation, speech recognition, image captioning, and so on. The word recurrent comes from the way how the networks perform the same task for every element of a sequence. RNNs have memories that allow abstraction of its past calculations that can, in theory, capture information in arbitrarily long sequences. However, in reality, only few time-steps backward, practically less than 5 [23], can give influence to the output making the network unsuitable for handling long sequence of information due to long-term dependency problem caused by vanishing gradient and exploding gradient during training process.

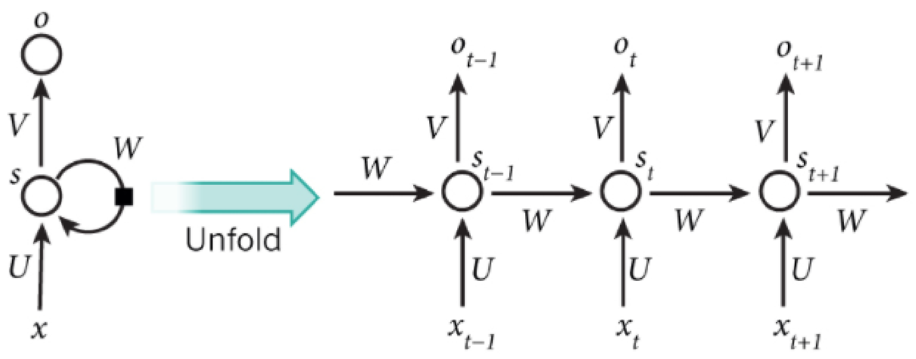


Fig. 8. RNN structure.

First proposed in 1997 by Sepp Hochreiter and Jurgen Schmidhuber [24], Long Short-Term Memory (LSTM) networks are different type of recurrent neural network that is able to learn from order dependencies in its input sequences, solving problem of long-term dependency. It is said that LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps [23]. The key to solving the problem was through internal structure of the units used in the model, the memory cells and three multiplicative units, the input, output and forget gates, within which only net can interact with cells, that provide continuous analogues of write, read, and reset operations for the cells [25]. Fig. 9 shows LSTM cell structure.

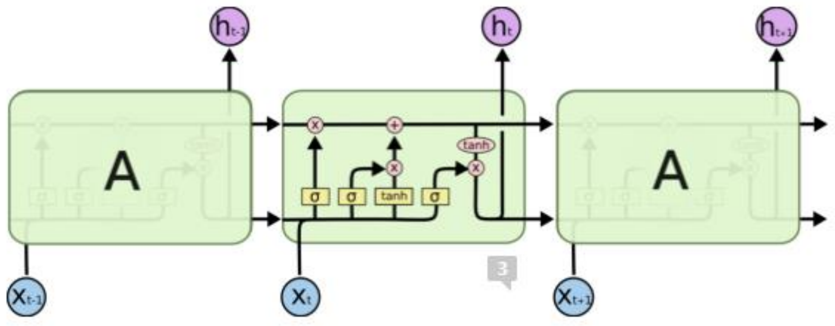


Fig. 9. LSTM memory block with one cell, with forget, input, output gates.

The *gates* within each cells allow learnt information from previous cells to be erased or added to current state, which are carefully controlled by the gates, where each of the gates are composed of sigmoid layer that outputs value between 0 and 1 and element multiplications. As for training LSTM, iterative gradient descent, such as backpropagation through time (BPTT), is used to change each weight through derivative with respect to an error.   More details to structures and operations within LSTM cell are out of scope of the paper and will not be discussed further.

As of now, LSTM network is actively being used as fundamental components within state-of-the-art technologies and markets from Good, Apple, and Microsoft for speech recognition, smart assistant, and language translations. In the following chapter, a collision avoidance system will be introduced in detail that combines the advantage of supervised learning and reinforcement learning through LSTM.

3. Related Works

In this chapter, recent technologies on collision avoidance using deep learning techniques will be introduced regarding two learning types; supervised learning and reinforcement learning.

3.1 Collision Avoidance using Supervised Learning

Among recent researches regarding autonomous flight using deep learning techniques, one common approach is the use of image data acquired by cameras and processing them through Convolutional Neural Network [10], a network designed to effectively process high dimensional data such as images. This method is often accompanied by supervised learning to train their networks and produce outputs that specify actions agent needs to take when certain input image is given [26]–[31]. Fig. 10 is an example of CNN used for controlling agent through image input [32]. However, this procedure faces challenges in construction of image dataset itself due to large number of training sets required for properly training the networks, not to mention a time-consuming task that necessitates each training data be labelled beforehand. To solve such data demanding issue, there have been researches regarding various methods of gathering or creating training data. Examples include self-supervised method in autonomously gathering collision image dataset [31] and use of synthetic 3D environment [33]. Still, large amount of dataset that must be pre-acquired and be labelled before training remains as a challenge yet to be solved in supervised learning. In addition, using CNN to process image inputs through on-board camera and calculating features on the fly requires heavy computation device on-board as well, such as Jetson board [26] that has mounted GPU processor, which is expensive in price as well.

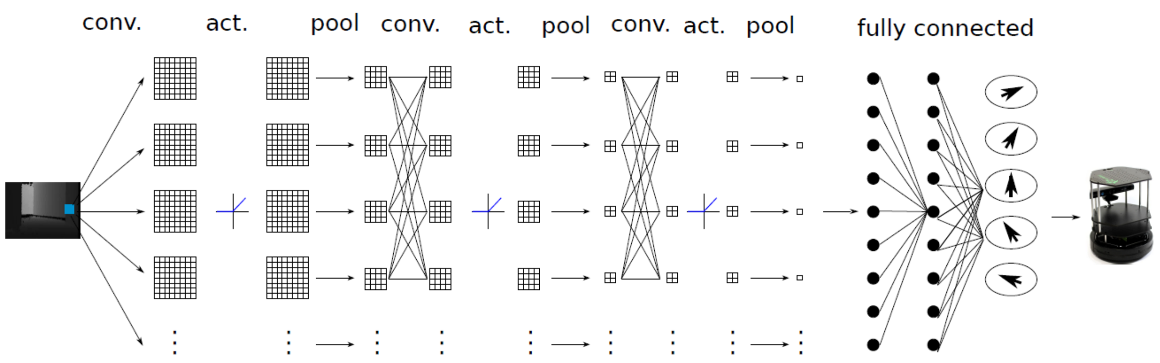


Fig. 10. Proposed model by Lei Tai et al., used for collision avoidance through CNN.

3.2 Collision Avoidance using Reinforcement Learning

Deep reinforcement learning is another method that is being researched in obstacle avoidance especially after recent progresses in reinforcement learning that have been proved to outperform human levels in certain fields [19], [21]. Ranging from use of depth sensors [34], synthetic 3D images [33], to direct raw image inputs from cameras [35], [36] as input data, DRL methods are based on trial-and-error procedures that optimize its performance in a given environment with proceeding iterations by taking actions that maximizes its future reward based on Q-value function. Thus, DRL is often data-intensive and time-consuming to train compared to supervised learning. However, the fact that no data labelling and pre-acquired training dataset are required gives DRL an advantage that supervised learning does not. Fig. 11 is a DRL based network that combines fully convolutional residual network (FCRN) and Q-learning network by Linhai Xie et al [37].

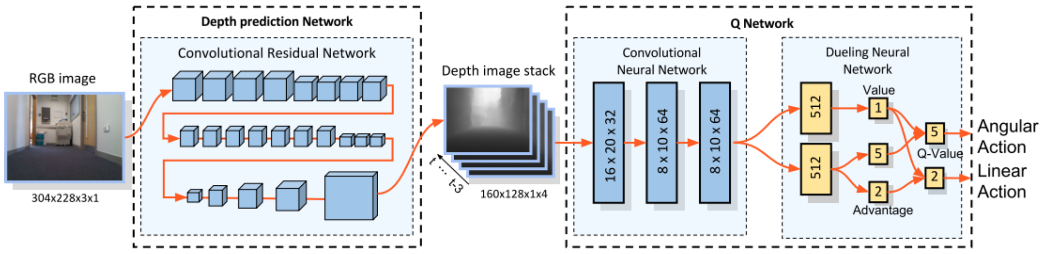


Fig. 11 Network architecture that uses deep reinforcement learning in combination with fully convolutional residual network for acquiring depth map.

4 Proposed Method

In this section, we give a detailed description about the neural network model along with its input and output data that has been used. Also, training methodology will be introduced that goes through the following phases; data gathering phase and training phase.

4.1 Problem Description

The goal of this research is to make a learning agent using neural network that can freely navigate within a 2D simulated grid environment avoiding any collisions with no access to dataset required for training.

Under limited computation capacity, the agent is given only a single distance indicating sensor to interact with the environment, where the sensor is presumed to return precise measurement without malfunction at any time input is required. Thus, we assume the agent has no knowledge on its surrounding environment. We assume that at every time-unit, the agent, or UAV, can only move forward, rotate left or right from its current position, and freely increase or decrease its travelling speed in a simulated grid environment.

4.2 Model Selection

Collision can be defined as an event that is dependent on time, because the same data containing distance information could be indicating either moving closer to or moving away from its facing obstacle depending on its previous data received. Thus, we adopted LSTM, or Long Short-Term Memory, which is a neural network architecture originated from recurrent neural network (RNN) that is specialized in handling sequential data [11].

The maximum number of time-steps in LSTM, denoted as n, determines the complexity of an event to be remembered. In our designed model, we used a short LSTM with n=4. This is because collision is an event that could be predicted and be avoided shortly before the collision and does not require data long before the impact.

4.3 Input and Output Data Processing

The key factor determining the event collision is the distance between the moving object and the obstacle. As such, we use simplest form of input data x_{ti} as a means to process distance information, i.e., x_{ti} is positive integer values for i=1, 2, $\cdots$, n where i represents each LSTM time-unit and n is the maximum time-step value with i=1 being oldest and i=n being the most recent data acquired. Hence, input data x_{ti} represents a distance between UAV and an obstacle along UAV's facing direction at time t_i, acquired by a single front facing depth sensor that returns distance value between itself and facing obstacle.

All experiences, or sequences of distance information acquired, will be stored into memory, or dynamic dataset, to be used for training as a supervised learning approach. However, when considering different travelling speeds of UAV, depending solely on the combinations of distance data to represent an event produces multiple identical sequences of input that may point out to different outputs due to the simplicity of input data, i.e., a single integer. Therefore, speed information x_s has been additionally appended to input sequence as $X = (x_{t1}, x_{t2}, \cdots, x_{tn-1}, x_s)$, allowing uniqueness to each input sequences regardless of flight speeds. As a result, among the 4 time-steps in our LSTM network, first three time-units are used for processing sensor inputs and the 4th time-step x_{tn} for inputting travelling speed obtained through distance difference between x_{tn-2} and x_{tn-1}.

When n numbers of input data from $(x_{t1}, x_{t2}, \cdots, x_{tn-1}, x_s)$ has been fed into an LSTM with maximum time-step of n, a single event X is formed. This event will then produce an output indicating collision or non-collision event depending on the state of agent, which is pre-defined by the validity of the current position. If current position of UAV after acquiring x_{tn} is out of bounds or has collided into an obstacle, state S_t is defined invalid and will produce output "collision". State is otherwise defined valid and will output non-collision event for all the other happenings. Fig. 12 illustrates the relationship between the agent and the environment, and Fig. 13 shows the overall LSTM model, input and output data processing.

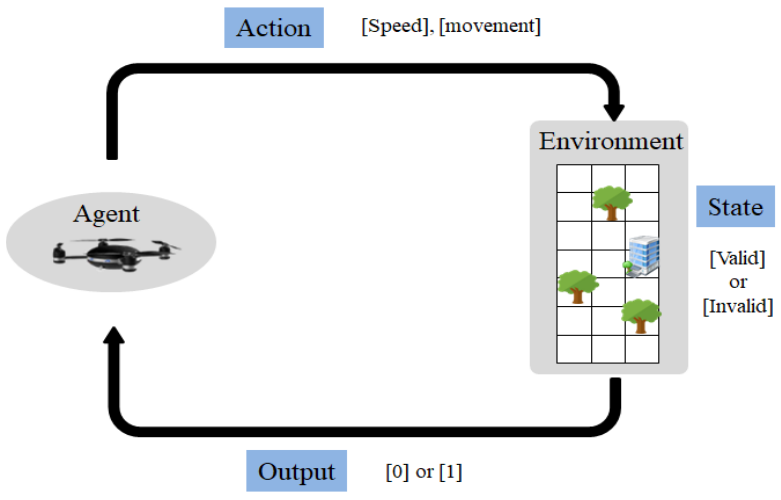


Fig. 12. Interaction between an agent and its environment.

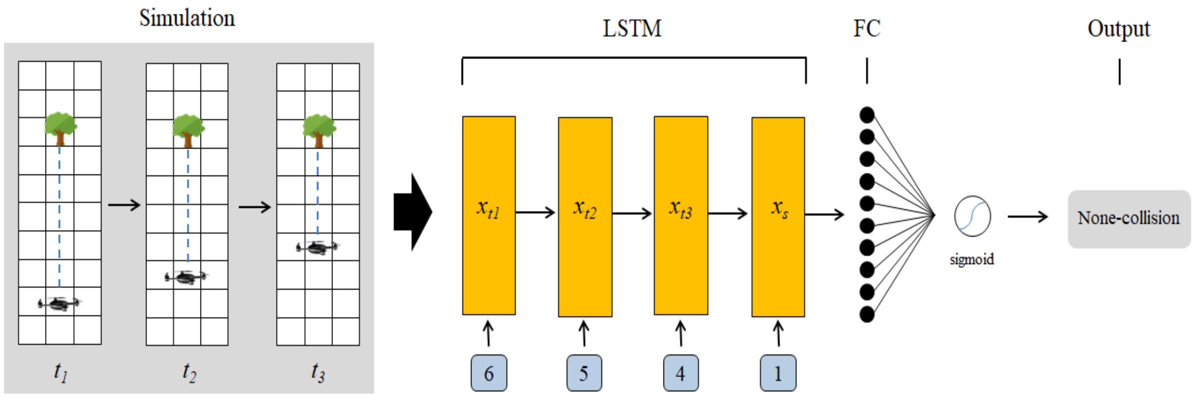


Fig. 13. Simulation running at time t1, t2, t3, and its corresponding input to LSTM. Each input data fed to an LSTM is a distance between UAV and an obstacle in its travelling direction at time t, with xt1 being the oldest and xt3 being the most recent input data. Output is a label indicating collision or non-collision event. After three consecutive inputs to the network, travelling speed xs is appended before producing a single output to uniquely define each experiences, i.e., $\{x_{t1}, x_{t2}, x_{t3}, x_s\}$.

4.4 Learning Procedure

The goal of training is to optimize the given model to distinguish collision events from any other events, where each event is comprised of distance data acquired at different time along with additional speed information. By combining supervised learning's train data labelling method and reinforcement learning's online trial-and-error learning approach, we have allowed the agent to dynamically gather data, store them, and learn through the accumulated data to train itself. The training procedure is composed of two phases, data gathering phase and training phase. Each phase switches back and forth between each other during the whole learning process.

1) Data Gathering Phase: In data gathering phase, the aim is to let agent experience events that represent either collision or non-collision through interaction with its environment based on iterative trial-and-error approach similar to reinforcement learning.

Starting with an empty memory, however, the agent will record all experiences into its memory, which we define as dynamic dataset similar to that of dataset used in supervised learning, along with definition on which experiences are good or bad depending on its state at the moment of experience.

In detail, for every input sequence *X* being formed throughout data gathering phase, duplication is checked from its dynamic dataset. If a sequence that has not previously been discovered is found, current state of the agent is checked immediately. If the state indicates valid, the new input sequence will be assigned with value 0, where 0 represents non-collision events. This set of input sequence and label will then be recorded into agent's memory, and the exploration within the map will be continued for further sequence acquisition. However, if the new input sequence results in invalid state, this implies a discovery of a new collision sequence, and label 1 will be assigned to the sequence. This new collision discovery shifts running process from data gathering phase into training phase.

On the other hand, if the inputted sequence already exists in the dataset, it is directly processed through LSTM for an output and action will be taken accordingly as programmed.

2) Training Phase: In training phase, optimization of LSTM model is processed with the dataset that has been accumulated at *its present memory*, and *this procedure happens* every time a new collision sequence is discovered.

Once the training is complete with the provided dataset, phase shifts back from training phase to data gathering phase to further proceed in data gathering. The position of the UAV must also be reset to the initial position. As a result of the training phase, UAV in data gathering phase will be able to bypass learnt colliding sequences through LSTM processing, and continue its exploration.

3) Overall Procedure - Algorithm: Pseudocode for detailed process of the algorithm can be summarized as algorithm shown in Table 3.

It must be noted that gathering of data is heavily dependent on the agent's action taken especially due to the randomness within agent's movement. As a result, numbers and types of training data gathered may be different when run again under the same parameters values.

TABLE III

ALGORITHM PSEUDO CODE

Algorithm 3 Adverse event based learning procedure.

```
initialize dataset $M$
$x_t$=do_random_movement()
**While** s_iter < s_max_iter **do**
    discard oldest input data $x_{t1}$ in sequence $X$
set $x_t$ as $x_{tn-1}$ of $S$
replace new speed $x_s$ as $\mathrm{x}_{\mathrm{tn}}$ of $S$
    **if** $X$ exists in $M$ **do**
        **if** LSTM($X$) is non-collision **do**
            $x_t$ = do_random_movement()
        **else**
            $x_t$= rotate()
    **if** $X$ does not exist in $M$ **do**
        $x_t$ **=** move_forward()
        **if** state $S_t$ is VALID **do**
            add $X$ to $M$ and label as non-collision
        **else if** $\mathrm{S}_t$ is INVALID **do**
            add $X$ to $M$ and label as collision
            **for** t_iter= 1,t_max_iter **do**
                run training with all data in $M$
            reset UAV position
            s_iter += 1
```

where

X = sequence of input data, $\{x_{t1}, x_{t2}, \ldots, x_{tn}\}$

M = dataset containing all sequences

x_t = distance data acquired at time t

n = maximum time-step length in LSTM model

s_iter = number of collision sequences learnt

s_max_iter = maximum number of iteration

t_iter = training iteration number with dataset M

t_max_iter = max number of training for optimization

5 Simulation and Results

5.1 Experimental Setups

All simulations and results were processed in Windows OS environment and implemented with Python using Tensorflow deep learning library. A single GPU system with Nvidia GTX 1060 has been used during the whole procedure.

5.2 Flight Environment Design

The flight environment for UAV has been designed as grid model, with a map size of 25 x 25 grids. Multiple obstacles of various sizes are randomly positioned within the grid map. Providing UAV a space to freely roam around and gather as many input sequences as possible is the primary purpose the flight environment must provide. Thus no complicated obstacle structures were additionally considered within the environmental settings. Fig. 14 illustrates a flight environment used for simulation.

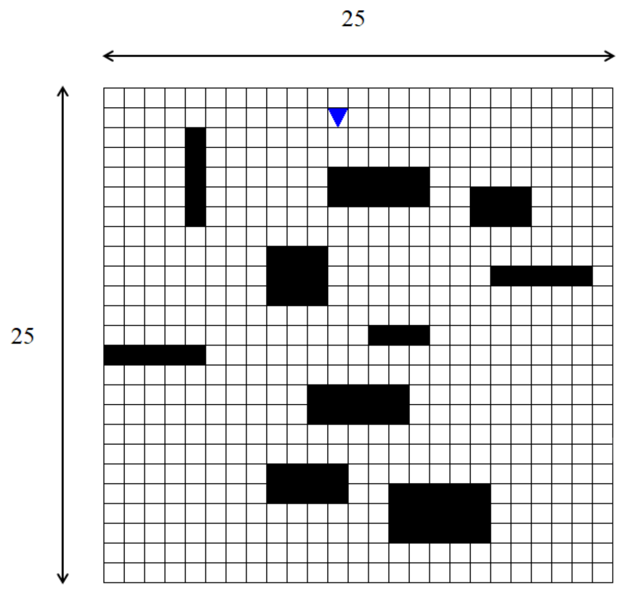


Fig. 14. A flight environment with multiple obstacles (black squares) for simulation. The UAV (blue triangle) starts gathering data once training starts.

5.3 UAV Setting

UAV can perform 3 actions while flying, i.e., rotate left, rotate right, and move forward. We have allowed diagonal flight within the environment by allowing each rotation to change the facing angle by 45 degrees, giving total of 8 different facing positions for UAV. A simulated depth sensor has been attached to the UAV's forward facing direction with the maximum distance measuring up to 20 grids. For every action UAV performs, distance measured from the distance indicator is read and used as an input to LSTM network. Lastly, the speed for the UAV is by default 1 grid per movement and may increase its speed up to 3 grids per movement, where each movement refers to a single frame within the simulation.

5.4 Training

Under the given setups, training has been performed as mentioned in algorithm 1 of table 1. Training parameters were configured as follows; 4 unit LSTM with 10 hidden states, Adam optimizer [38] with learning rate of 0.001 and full batch training. Maximum training iteration (*t_max_iter*), and sequence number (*s_max_iter*) were set to 100. We used many-to-one LSTM, where only the last sequence output of size [time-step, hidden state] was used as an input to fully connected network. Output of fully connected network was then passed through sigmoid function for binary classification of value between 0 and 1; 0 symbolized non-collision event and 1 represented collision. Threshold value of 0.5 was used for classifying output value.

Before the UAV's first random movement, we have programmed it to rotate on its starting position one complete round as a means to acquire input to fully occupy the sequence in order to prevent empty values from being processed through the network. Also, we have run the algorithm to increase its speed by 1 grid/movement after one complete run, allowing it to collect data at 3 different travelling speeds.

To show learning rate of our algorithm, we have first compared numbers of movements required before collision to number of iteration, i.e., number of collisions learned. Fig. 15 shows average number of movements at specific intervals of iterations, and Fig. 16 shows total accumulated number of movements versus number of collision events learned.

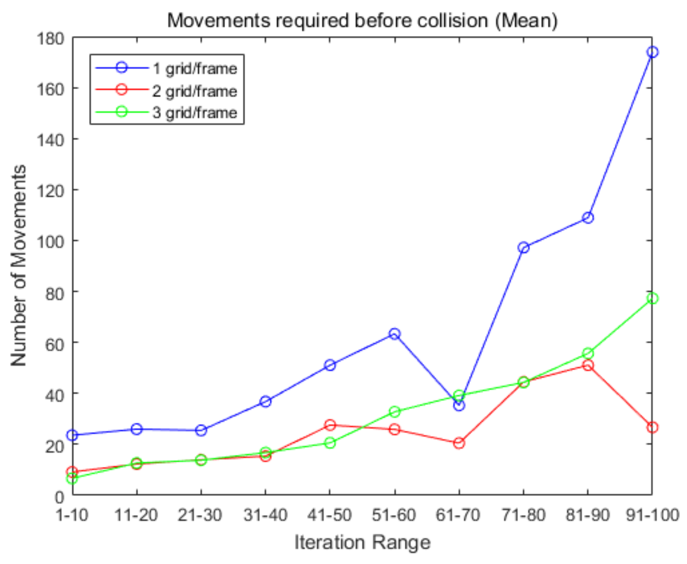


Fig. 15. A graph showing average number of movements required to discover new collision at different intervals of events learned.

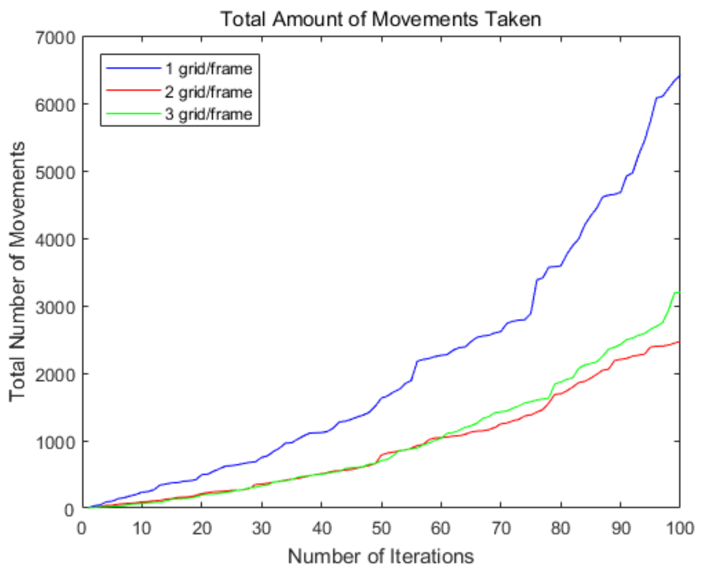


Fig. 16. The amount of total accumulated movements taken at learning moment of collision at certain index. An exponential increase in number of movements can be observed.

Despite randomness in learning agent's actions, the graph shows exponential increase in total movements required, and hence implies that more efforts to seek new "experience" are required with respect to the amount of already experienced collisions.

5.5 Testing and Results

To test how well our trained model can classify collision and non-collision events, we constructed a testset by calculating all possible combinations of sequences that are available in the simulated environment, which numbered out to be 24,000, where 21,600 sequences are non-collision events and 2,400 are collision events. Note that the test set consisted of the trained sequences that had been gathered in the training process as well due to different number of dataset at each number of iterations. The average number of trained sequences after 100 iterations numbered 1770 for non-collision events and 300 for collision events, totaling 2070 learnt experiences.

Using this test dataset, we first calculated precision (PPV), recall (TPR), and accuracy (ACC) for each iterations of learning processes. Precision, recall, and accuracy can be calculated as follows

$$\begin{aligned} PPV &= \frac{TP}{TP+FP} \\ TPR &= \frac{TP}{TP+FN} \\ ACC &= \frac{TP+TN}{TP+TN+FP+FN} \end{aligned} \tag{1}$$

where TP refers number of collisions correctly classified as collision, FN is collision incorrectly classified as non-collision, FP is number of non-collisions incorrectly classified as collision, and TN is number of non-collision events correctly classified as non-collisions. Fig. 17 shows results of precision, recall, and accuracy percentage of our training model throughout its training process accordingly under 3 different trials of training using the same learning parameters. Due to convergence, only iterations of up to 50 have been considered.

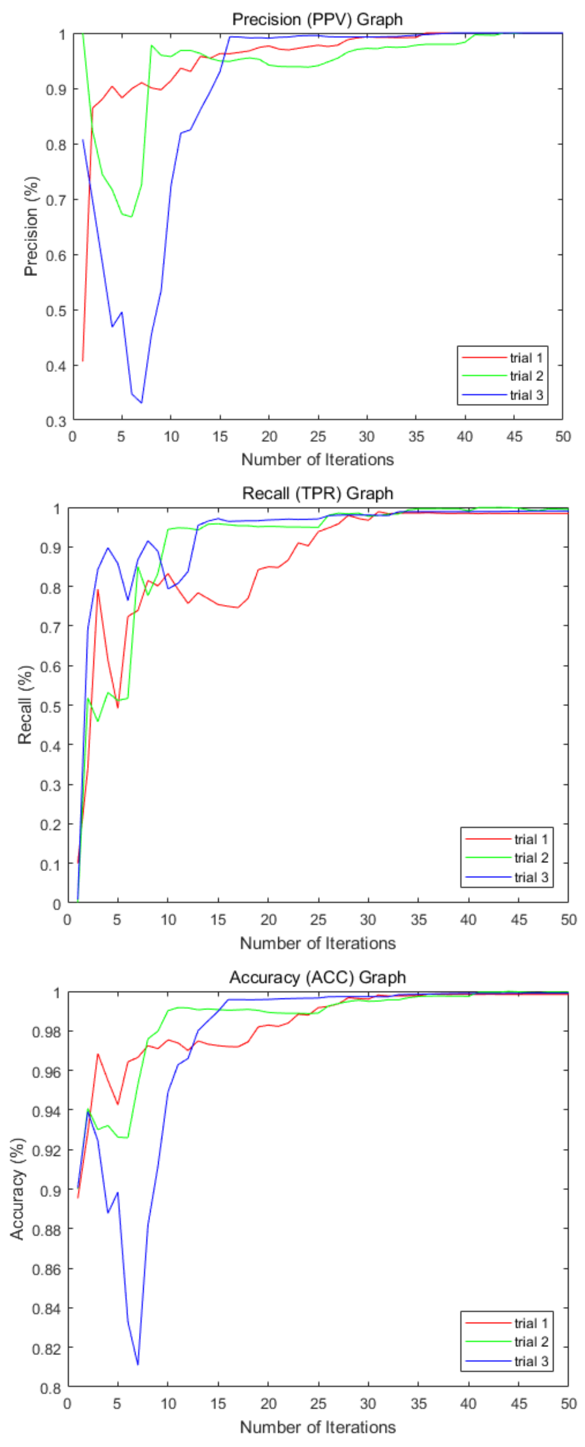


Fig. 17. Precision, recall, and accuracy of training model during its learning process. Due to random learning behavior of the agent, note that each learning trial shows different learning curve in early learning stages of iterations.

From Fig. 17, it can be noticed that each trials show very different learning curving especially in the early stages of iteration, i.e., only few collisions have been learnt. That is primarily due to dependency on learning agent's randomness in discovering collision sequences, where order of sequences and contents within dataset itself would differ from one another. Thus, each trial can be defined as each agent's unique memory. However, after 50 iterations at max, all trials within each graphs converged to nearly the same values. Table 4 shows the table with highest value for each metric within 50 iterations for trial 1, 2, and 3 and the iteration number it was achieved.

TABLE IV

RESULT (%) AFTER 50 ITERATIONS

Metric	Model trial number		
	1	2	3
Precision (iter. #)	100% (36)	100% (46)	100% (39)
Recall (iter. #)	98.88% (31)	99.96% (43)	99.04% (46)
Accuracy (iter. #)	99.86% (36)	99.99% (44)	99.90% (46)

The highest recall and accuracy rate was achieved with trial 2, achieving recall rate of 99.96% at 43rd iteration and accuracy of 99.99% at 44th iteration. In case of precision, each trial could achieve 100% at least by 46th iteration. This means that our system misclassified only 2 events out of 24000 total events at the best within the three trials. Fig. 18 is a graph showing total number of misclassifications, which is number of false negative (FN) and false positive (FP) combined, along the training process. The least number of misclassification was numbered 37, 2, and 21 accordingly for trial 1, 2 and 3 within the first 50 iterations.

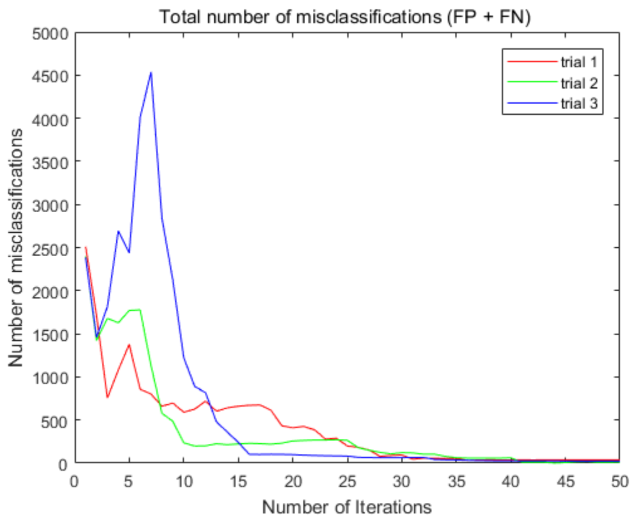
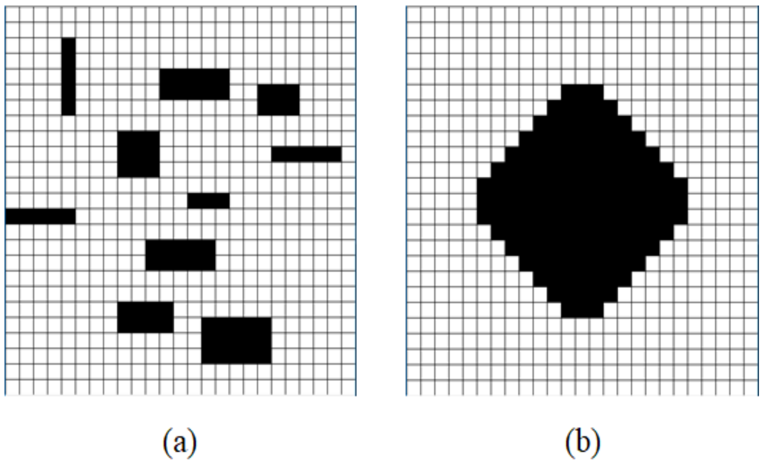


Fig. 18. Graph showing total number of misclassifications within each training iteration for different learning trials.

Lastly, we gave our trained UAV a simple destination reaching test. We assigned a destination point within given maps and provided simple guidelines on when to rotate left or right depending on the relative position and angle with the destination point. We assumed UAV knows its coordinates as well as that of destination. For this test, we have tried different maps, which are shown in Fig. 19, that are without dangers of falling into a local minimum. Fig. 20 shows the path UAV travelled within each maps under dynamically changing flight speed. The result shows that our system allows the agent to safely reach destination points without complex algorithms or high dimensional sensory inputs. However, it is expected that additional sensory inputs will give the agent ability to effectively reach its destination even in complex environments, which will be discussed as future work in Section 5



.

Fig. 19. Simple flight environemnts for testing destination reaching experiment.

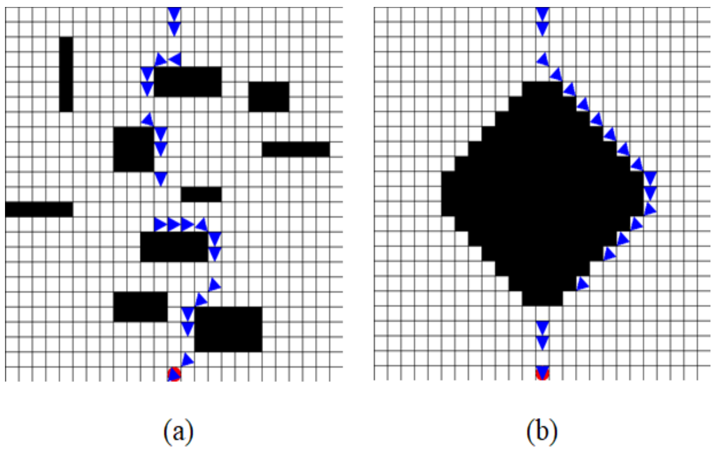


Fig. 20. Destination reaching test with manual instructions on rotation with respect to the destination point. Destination point is shown as a red dot, and each blue triangle indicates UAV’s position at different times along its flight path.

6. Conclusion

6.1 Summary

In this paper, we proposed a collision avoidance system that is based on learning events where each event is composed of sequences of distance data acquired from a single simulated distance indicator. We introduced a training algorithm that combines reinforcement learning's online trial-end-error learning method and supervised learning's labelling approach using an LSTM network that allows the agent to gather data and learn by itself with proceeding training iterations without pre-acquired training dataset.

The experiment result showed that the designed model can perform its purpose clearly, achieving minimum of 99.90% accuracy in identifying collision among all possible events that can happen within the simulated environment. The event based learning also has allowed UAV to fully adapt to and navigate in environments with no prior knowledge or additional training.

6.2 Future Works

Future researches may include adaptation into real world environment as well as multiple sensor attachments for wider field of view allowing detection of collision from multiple angles. Also, it is expected that our system combined with effective decision making system to be used at the moment of collision detection, such as CNN for effectively choosing travelling direction, will make the system applicable to path finding in complex environments. Such systems will provide UAVs a far more cost efficient autonomous flight system compared to other systems that only depend on computation heavy data such as images, which also requires large training dataset beforehand.

7. References

[1] A. Samuel, “Some Studies in Machine Learning Using the Game of Checkers,” *IBM J. Res. Dev.*, 1959.

[2] M. T. Michael, “Machine learning,” no. ISBN 0-07-042807-7, 1997.

[3] R. Kohavi, “Glossary of terms,” *Mach. Learn.*, no. 30, pp. 271–274, 1998.

[4] S. B. Kotsiantis, “Supervised Machine Learning: A Review of Classification Techniques,” *Informatica*, vol. 31, pp. 249–268, 2007.

[5] Y. Taigman, M. Yang, and M. A. Ranzato, “Deepface: Closing the gap to human -level performance in face verification,” *CVPR IEEE Conf.*, pp. 1701–1708, 2014.

[6] K. Y. Levy and N. Shimkin, “Unified Inter and Intra Options Learning Using Policy Gradient Methods,” *Recent Adv. Reinforecement Learn. 99th Eur. Work.*, pp. 153–164, 2012.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. 2016.

[8] L. Deng and D. Yu, “Deep Learning: Methods and Applications,” *Found. Trends® Signal Process.*, vol. 7, no. 3–4, pp. 199–200, 2013.

[9] Y. LeCun, Y. Bengio, and G. Hinton, “Deep Learning,” *Nature*, vol. 521, pp. 436–444, 2015.

[10] Y. Lecun and Y. Bengio, “Convolutional Networks for Images, Speech, and Time-Series,” *Handb. brain theory neural networks*, no. November, pp. 255–258, 1995.

[11] J. Gu *et al.*, “Recent Advances in Convolutional Neural Networks,” pp. 1–38, 2015.

[12] R. Hecht-Nielsen, “Theory of the backpropagation neural network,” *Neural Networks*, vol. 1, pp. 445–448, 1988.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2012.

[15] M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks arXiv:1311.2901v3 [cs.CV] 28 Nov 2013,” *Comput. Vision–ECCV 2014*, vol. 8689, pp. 818–833, 2014.

[16] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” pp. 1–14, 2014.

[17] C. Szegedy *et al.*, “Going Deeper with Convolutions,” pp. 1–9, 2014.

[18] S. Wu, S. Zhong, and Y. Liu, “Deep residual learning for image steganalysis,” *CVPR IEEE Conf.*, pp.

1–17, 2015.

[19] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al. Playing Atari with deep reinforcement learning. Technical report. Deepmind Technologies, arXiv:1312.5602, 2013

[20] L.-J. Lin, "Reinforcement learning for robots using neural networks," *Tech. report, DTIC Doc.*, 1993.

[21] V. Mnih *et al.*, "Human-level Control through Deep Reinforcement Learning," *Nature*, vol. 518, 2015.

[22] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A Critical Review of Recurrent Neural Networks for Sequence Learning," pp. 1–38, 2015.

[23] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Comput.*, vol. 12, no. 10, pp. 2451–2471, 2000.

[24] S. Hochreiter and J. Urgen Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[25] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM networks," in *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on*, 2005.

[26] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield, "Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness," 2017.

[27] P. Chakravarty, K. Kelchtermans, T. Roussel, S. Wellens, T. Tuytelaars, and L. Van Eycken, "CNN-based single image obstacle avoidance on a quadrotor," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 6369–6374, 2017.

[28] K. Sullivan and W. Lawson, "Deep Obstacle Avoidance ∗," 2013.

[29] D. K. Kim and T. Chen, "Deep Neural Network for Real-Time Autonomous Indoor Navigation," 2015.

[30] A. Guisti J. Guzzi, D. C. Ciresan, F.-L. He, J.P. Rodriguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, *et al* ., "A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots," *Robot. Autom. Lett.*, vol. 1, no. 2, pp. 661–667, 2016.

[31] D. Gandhi, L. Pinto, and A. Gupta, "Learning to Fly by Crashing," arXiv:1704.05588, 2017.

[32] L. Tai, S. Li, and M. Liu, "A deep-network solution towards model-less obstacle avoidance," *IEEE Int. Conf. Intell. Robot. Syst.*, vol. 2016–Novem, no. 6140021318, pp. 2759–2764, 2016.

[33] F. Sadeghi and S. Levine, "CAD2RL: Real Single-Image Flight without a Single Real Image," arXiv:1611.04201, 2016.

[34] L. Tai, G. Paolo, and M. Liu, “Virtual-to-real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation,”arXiv:1703.00420, 2017.

[35] A. Ueno, N. Kajihara, N. Fujii, and T. Takubo, “Vision-based path learning for home robots,” *Proc. - 2014 10th Int. Conf. Intell. Inf. Hiding Multimed. Signal Process. IIH-MSP 2014,* pp. 411–414, 2014.

[36] L. Tai and M. Liu, “Towards Cognitive Exploration through Deep Reinforcement Learning for Mobile Robots,” arXiv:1610.01733, 2016.

[37] L. Xie, S. Wang, A. Markham, and N. Trigoni, “Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning,” 2017.

[38] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” pp. 1–15, 2014.

8. Acknowledgements

걱정 반 기대 반으로 대학원 생활을 시작한지 2년이란 시간이 흘러 졸업을 하게 되었습니다. 석사과정 2년의 생활은 제 삶에 있어 빠르게 지나간 시간인 만큼 정말 귀한 시간이었습니다. 참 많은 일들이 있었지만, 지금 돌이켜보면 정말 행복한 시간들로 가득 차있는 것 같습니다. 이 모든 과정의 중심이 되셨던 이흥노 교수님께 깊은 감사의 말씀을 드리고 싶습니다. 저의 연구 분야에 있어 언제든지 성실하게 지도해주시고 후원을 아껴주시지 않으시며, 제 연구에 있어 책임감과 용기를 가지게 해주셔서 감사합니다. 인포넷 연구실의 선배, 후배, 그리고 동기들 한 명 한 명, 정말 감사합니다. 특히, 개인 연구에 있어 많은 지도를 해준 주성이형 감사합니다. 랩 멤버들로 인해 학문적으로 많은 도움을 받을 수 있었을 뿐만 아니라, 제 랩실의 삶이 즐거울 수 있었습니다. 함께 했던 아름다운 추억들과 경험들, 잊지 않겠습니다.

제가 힘들 때나 즐거울 때나, 뒤에서 늘 한결같이 저를 응원해주시고 기도해주시며 사랑을 아낌없이 주신 엄마, 아빠, 누나에게 정말 감사하다는 말 전하고 싶습니다. 가족들이 있었기에, 제가 대학원을 끝까지 잘 마칠 수 있었습니다. 그리고 매일 따뜻한 위로와 응원으로 저에게 항상 긍정적인 마인드를 잃지 않게 도와준 기쁨아, 정말 고마워. 또한, 뒤에서 저를 위해 기도해주신 과기원 교회 식구들, 정말 감사합니다.

마지막으로, 이 모든 길을 인도해주셨고 앞으로도 인도해 주실 하나님께 모든 영광과 감사를 드립니다.