ISTANBUL TECHNICAL UNIVERSITY ★ FACULTY OF AERONAUTICS AND ASTRONAUTICS

MODELLING AIRCRAFT FIGHTING MANEUVER DYNAMICS USING ARTIFICIAL INTELLIGENCE ALGORITHMS

GRADUATION PROJECT

Berkay SOYLUOĞLU

Department of Aeronautical Engineering

Thesis Advisor: Assist. Prof. Dr. İsmail BAYEZİT

JUNE, 2021

ISTANBUL TECHNICAL UNIVERSITY ★ FACULTY OF AERONAUTICS AND ASTRONAUTICS

MODELLING AIRCRAFT FIGHTING MANEUVER DYNAMICS USING ARTIFICIAL INTELLIGENCE ALGORITHMS

GRADUATION PROJECT

Berkay SOYLUOĞLU
(1101603031)

Department of Aeronautical Engineering

Thesis Advisor: Assist. Prof. Dr. İsmail BAYEZİT

JUNE, 2021

BERKAY SOYLUOĞLU, student of ITU Faculty of Aeronautics and Astronautics, ID 110160301, successfully defended the **graduation** entitled "modelling aircraft fighting maneuver dynamics using artificial intelligence algorithms", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor:	**Assist. Prof. Dr. İsmail BAYEZİT** Istanbul Technical University	.....

Jury Members: Nazım Kemal Üre
Istanbul Technical University

Cengiz Hacızade
Istanbul Technical University

Date of Submission: 14 June 2021
Date of Defense: 29 June 2021

To my family,

FOREWORD

This dissertation is the culmination of many years of perseverance and effort, and it is the work of many people, each of whom is equally crucial to its proper development.

To begin, I'd want to express my gratitude to Prof. İsmail Bayezit, my supervisor, for providing me with this incredible opportunity to learn under his direction at such a well-equipped aeronautics department at Istanbul Technical University. Thanks also to my TAI co-supervisors for guiding me and introducing me to the fascinating field of Reinforcement Learning.

I appreciate my parents and family's unwavering support, as well as their patience, dedication, support, and wisdom when I needed it most. They always believed in me and supported me, loving and supporting me as I grew into the person I am today.

Special thanks to my friends Batu, Buğra, Kerem, and Turgut, whose unwavering support and perseverance have enabled me to pursue my lofty life ambitions.

June 2021 BERKAY SOYLUOĞLU

TABLE OF CONTENTS

Page

ABBREVIATIONS

AA : Aspect angle
ANN : Artificial neural network
ATA : Antenna train angle
CNN : Convolutional neural network
DDPG : Deep deterministic policy gradient
DNN : Deep neural network
DQN : Deep Q network
DRL : Deep reinforcement learning
MDP : Markov decision process
MP : Markov process
NN : Neural network
PPO : Proximal policy optimization
RL : Reinforcement learning
RNN : Recurrent neural network
SGD : Stochastic gradiend descent
SOF : Surrogate objective function

LIST OF TABLES

Page

LIST OF FIGURES

Page

AIR MODELLING AIRCRAFT FIGHTING MANEUVER DYNAMICS USING ARTIFICIAL INTELLIGENCE

SUMMARY

The ability to quickly and accurately dominate aircraft during a dogfight scenario is critical to the safe and effective operation of Turkish Armed Forces. Modern technology and computer-aided decision-making tools offer a viable replacement to antiquated battle procedures. This thesis investigates the possibility for collaboration between the two by combining the state-of-the-art policy-based reinforcement learning algorithms' capabilities with battle identification approaches. This thesis examines the overall correctness of the constructed agent to established truths in order to determine the level of system learning after building a basic interface between agent and environment identification methods. While the breadth of this preliminary study is limited, the findings point to a significant upgrading of combat maneuverings. These findings can help future study to design a robust system that can replicate and/or improve the decision-making abilities of a human operator, in addition to proving proof of concept. While this research focuses on an air-based vehicle, jet fighter, the findings can be applied across the Department of Defense.

1. INTRODUCTION

Military jet fighters have gained a lot of interest because of their inexpensive cost, lengthy flying length, and willingness to sacrifice in comparison to human aircraft. The performance of military jet fighters has improved dramatically as sensor technology, computer technology, and communication technology have advanced, and the range of jobs that they can accomplish has expanded. Although a military jet fighter may execute surveillance and ground assault missions, the majority of mission tasks require human participation, and the choice is ultimately made by the ground station control officers. This ground-based remote-control option is primarily reliant on a data link that is susceptible to weather and electromagnetic interference. Because of the difficulties in adjusting to quick and varied air combat conditions, standard ground-based remote operations are difficult to command jet fighters to conduct air warfare. (Yang, 2019) As a result, allowing jet fighters to autonomously make control decisions based on the circumstance and achieving air combat independent from the jet fighter is an important jet fighter intelligence research path.

1.1. Purpose of Thesis

This thesis examines the use of reinforcement learning instead of entirely rule-based behavior to train an artificial pilot. The learning will be implemented at a higher level of decision-making, with lower layers relying on rules. Of course, it makes no difference how the lower layers are implemented; they can be replaced at any time with any other control approach, including hierarchical reinforcement learning. Although the learning will be done in Python, it may be done in any programming language. Constructing a state-action to value relationship requires a neural network to be employed as a function approximator. Rules define actions, but the algorithm chooses them on the fly. The activity with the highest expected benefit or least expected punishment is chosen. The method develops a form of long-term value estimate by rewarding or penalizing particular states, which is then learned by the value function. To address gaps in the existing state of the art, many goals were specified. The initial goal is to create a system that

could operate in real time, which is necessary for any prospect of practical deployment on real aerial vehicles. The second goal was to create a system that had a lengthy planning horizon. When an aircraft which is on a combat mission is manned, the pilot has to make near-term maneuvering choices within the context of longer-term objectives (McGrew et.al., 2012). When these objectives are taken into account, wise decisions may be made that provide a better long-term value, which is crucial for effective air warfare. The third goal is to reduce the requirement for extensive human engagement in the learning of encoded tactics by developing an algorithm that can learn suitable movements on its own. The ultimate goal is creating a robust, and stable algorithm that can be applied on multiple devices involved in different scenarios from offensive positions to defensive ones which means that the airplane will adapt to its current environment.

1.2. Literature Review

1.2.1 Machine learning

Gammon, which is a computer software that plays board games, is the first successful notable reinforcement learning application (Andriambololona & Lefeuvre, 2003). It developed a policy by training the model by itself. 300,000 episodes took place to reach the point where it could defeat another artificial intelligence bot or a player. More than a million episodes were run to improve its skills to the point where it could beat a grandmaster. However, after this first success, researchers seeking to use neural networks to address notable reinforcement learning issues encountered recurrent failures, as detailed in [2]. As a consequence, there was a move away from reinforcement learning networks before numerous strategies were presented that contributed to the development of the algorithms. Approaches that rely on a experience replay, that include DQN (Deep Q Network) and NFQ (Neural Fitted Q Iteration), as well as alternatives that are based on asynchronous parallel agents, for example A3C (asynchronous advantage actor-critic) implemented with ACER (actor-critic with experience replay) , were among the strategies used. Experience replay is a fascinating idea since it inspired many years later successful notable reinforcement learning examples (Dong, Ai & Liu, 2019). while performing an action, leading in a reward and the emergence of a new state. A way of presenting a set of events to a reinforcement learning algorithm in order to learn what the agent has experience based on what the agent decided which action to take experienced on repeat is known as experience replay. The expediency of the credit assignment procedure and the possibility so that the agent forgets what

it has learned and can experience new formations of states are two advantages of experience replay (Li, 2019). NFQ is a derivative of experience replay method for multiple layered perceptron algorithms. The basic concept behind NFQ is to gather all experience tuples during a single run time and display them at the conclusion of the show. MLP (multi-layered perception) trains the network for each and every episode. A cost function is used to produce targets. A specific and designated number of episodes which are already set, with each episode's experience collection including all previous history of the episodes (Changqiang et.al., 2018). One famous application for the experience replay resides in old game environments configured for DQN type reinforcement learning algorithms. These algorithms usually retrieve information from the environment and stores them in a local data in order to access them later to gather a small subset of them and use them to train the current state of the neural network by selecting a random selection (Dong, Ai & Liu, 2019). DQN has seen better improvements over the time such as the addition of another network to be integrated with the first one. This is done by copying the first neural network that targets the original neural network. Then, a selection of update interval is made. At the update intervals, second network generates directives to update the first neural network. This algorithm inspired many others by its unique design of the utilization of the second target neural network and training taking place inside a replay buffer. Many DQN algorithm types exist such as double deep Q network, D3QN (dueling double deep Q network), DDQN (deep deterministic Q network).

1.2.2 Air combat

Air combat simulations based on artificial intelligence algorithms are divided into 2 categories. These can be simplified down to 2D and 3D scenarios. 2D scenarios are overly simplified point-mass models that rely on unrealistic guidance algorithms. Same thing can be said about the 3D aircraft maneuvers as well, however the mathematical models varies greatly from the 2D scenarios. Earliest achievements in the field is from NASA scientists who used game theory to solve the problem (Austin et.al., 1990). This approach was crude and did not rely on modern designs due to its time’s limitations. In the following years, heuristic approaches were tried and neuro-dynamic algorithms were tested in the aircraft simulations (McGrew, 2008). Ground-breaking work was done by McGrew in which he applied dynamic programming and novel reinforcement learning algorithms (Mcgrew et.al., 2010). The foundatins were later improved in the fields of stability and simulation environment (Kong et.al., 2020). Other algorithms that

include actor-critic structure were also tested with greater stability and success rates than the ones that do not include such model. Although the models were quite responsive and successful in terms of robustness, physical reality aspect of the investigations were exaggerated greatly. Therefore, scenarios that take place in 3D environments became the main concern of this paper. The roots of this research environment dates back to game theory (Austin et.al., 1990). These methods solved the early problems of air combat maneuverings. Later on, more advanced methods have been applied. Bayesian inference and moving horizon optimization techniques are two of these techniques that improved the 3D air combat scenario (Changqiang et.al., 2018). Basic reinforcement learning algorithms were also applied with success but it lacked depth and a realistic aircraft model or environment. For smoother controlling, DDPG algorithm was deployed to the environment (Yang et.al., 2019). Accuracy of the aircraft was improved and the noise handling qualities were enhanced than the DQN's more simplistic versions. More comprehensive works with realistic aircraft models were introduced into the literature, however these resulted in specific scenarios and conditions that require the algorithm to comply with.

1.3. Hypotheses

The aircraft which is in the air warfare autonomous maneuver making model is executed in this research using reinforcement learning. The model will be created in 2D space on the first phase due to simplicity and then during the next phase 6-DOF aircraft motion model and air combat model is constructed. In addition, by integrating two aspects of circumstance and distance, an assessment model for air combat advantage is given.

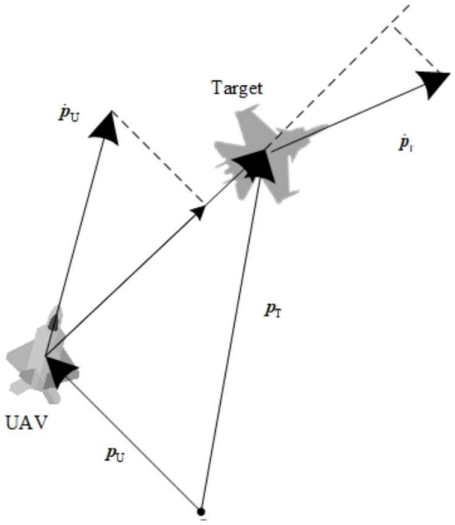

Figure 1: Jet fighter models, adapted from (Yang, 2020).

Aircraft model can robustly and efficiently represent the aircraft's advantages in every short-range air warfare scenario. Second, using the value-based (DQN) or policy-based (PPO) framework, a maneuver decision model is built, as well as a new maneuver library. The 7 traditional maneuver actions have been expanded greatly, resulting in a larger choice action area. Simultaneously, the reward function is developed to thoroughly represent the alteration of the behavior to the circumstance, based on the benefit assessment function. Finally, a training approach termed "basic confrontation" is presented to solve the issue of the maneuver choice model which bears high-dimensional spaces for both state and action which are 15 and 11 respectively, makes it difficult to train to converge conventionally. The maneuver decision model developed in this thesis has been shown to be able to learn the maneuver policy and behavior autonomously and thus gained the advantage that dominates the opposing air unit in air combat through a large number of trial-and-error based scenarios generated by the RL algorithm. These scenarios are made by agent to agent confrontation and manually controlled model to agent confrontation that are done in a simulation. What differs this thesis from other algorithm-based research papers is that the agents that has high-dimensional state and action spaces does not get stuck in the local optima and performs well under stochaistic scenarios which makes it kinda realistic for the real-life situations.

It is critical not just to develop new and improved ways, but also to comprehend these techniques for the relative study elements related to this field, in order to drive the field ahead. The goal of this project is to understand the differences and enhancements between two reinforcement learning algorithms i.e. DQN and PPO on simple problems in order to gain a better understanding of the advantages and disadvantages of applying reinforcement learning

algorithms, as well as the effects of various parameters and environments on performance. To do so, the algorithms are tested to see how they react in defensive and attacking situations. The program will be written in Python utilizing Python machine learning libraries such as Keras and Tensorflow or Pytorch (Montague et.al., 1999). These methods will be put to the test in the OpenAI Gym environment, which is part of a github project that uses dubins route planning (İşci, 2021).

1.4. Outline of Thesis

The techniques for calculating maneuvering choices for one-to-one air combat are presented in this paper (Qiu, 2020). These techniques are usually known as basic fighter maneuvering (BFM). The application of reinforcement learning architecture to the air combat issue is this thesis' main contribution. The end result is a model that can learn a maneuvering strategy and use it to make real-time air combat decisions.

Prior to the work using reinforcement learning algorithm development, many methods and techniques related to air combat field were investigated. During Chapter 2 of the thesis, the thesis describes the development of machine learning algorithms that can be used for the agent's model that will be used in aerial warfare. The utility of these equations is evaluated in an air warfare simulation by using deep neural networks. During Chapter 3, the thesis describes how to use reinforcement learning equations to develop a real-time route planner that works in real time. The aircraft is used in the OpenAI Gym's 2D simulation environment, where genuine jet fighter models are greatly exaggareted and simulated to fly in air combat missions to test their capabilities. In Chapter 3, you will also find a discussion of the creation of the algorithm type and model that were used in this simulation. The results of the flying tests are reported. The reinforcement learning models that was introduced in the previous chapters is further extended in the following sections. Chapter 4 discusses the measures that were used in order to frame the problem as a artificial intelligence program. In this methodology, basic flight maneuvering is characterized as a discrete time approximation that is solved by reinforcement learning algorithms. Policy decisions made in real time are easy to understand and to assess. The calibration of this approach is described in Chapter 5 together with the simulation performance data and flying outcomes. The thesis is concluded in Chapter 6 with a summary, discussion of the thesis aims, and research ideas for future study.

2. THEORY

This section will give necessary background information on the methodologies and technologies that were employed in this paper or that are linked to this field of study.

2.1. Machine Learning

The majority of computer agents nowadays are developed using rules; these systems are referred to as based on certain rules and are fairly simple to construct. Despite their obvious advantages, systems that are based on certain rules have a few flaws. For a variety of reasonsm, these systems cannot grow indefinitely. Firstly, because a person must express each policy manually and the policy database grows drastically as the scenario's model and its observed states get larger, as does the demand for more particular behavior. A human's ability to predict all probable events, much alone anticipate or compute ideal values for them, is likewise incredibly difficult. As the possible actions that must be drafted by hand grows, performance suffers. Decision policies may be a data tree or an array expressed by creating a data variable that holds a lengthy list of equal values. Because not all policies must be assessed, the latter considerably increases runtime speed, but it will need more pre-work. In the case of human performance, this is the polar opposite of how humans work. If a person has enriched experience over a certain topic, they can be quicker to act which means that they can make judgments fluently. Machine learning holds the key to solving this issue. If the program is able to learn the optimal method to accomplish a job, the heavy labor of developing rules for every conceivable event, condition, and action may be avoided, and a more optimum solution may be found.

2.2. Deep Learning

The artificial intelleginces applications in the form of neural networks assert to be very similar to that of biological neuron information exchange systems. Artificial neural networks are made up mostly of neurons. They send and receive messages in the same way that the human brain does.

The artificial neurons use related weights that are affected by functions that are non-linear due to provide complexity so that the system cannot be represented by a single transformation to link numerous inputs to one or more outputs. When data is fed into a neural network, it goes through a learning process using specialized algorithms to determine the right weights that characterize the output's behavior in relation to various inputs. Neural networks have a lot of promise for studying and examining vast historical datasets that are not usually employed in traditional modeling. To put it another way, neural network architecture is advised to be used when mathematical modeling is not feasible. The difficulties in the aeronautic and astronautic businesses are complicated. Unconventional approaches, such as artificial deep learning intelligence, may be used to address these difficulties.

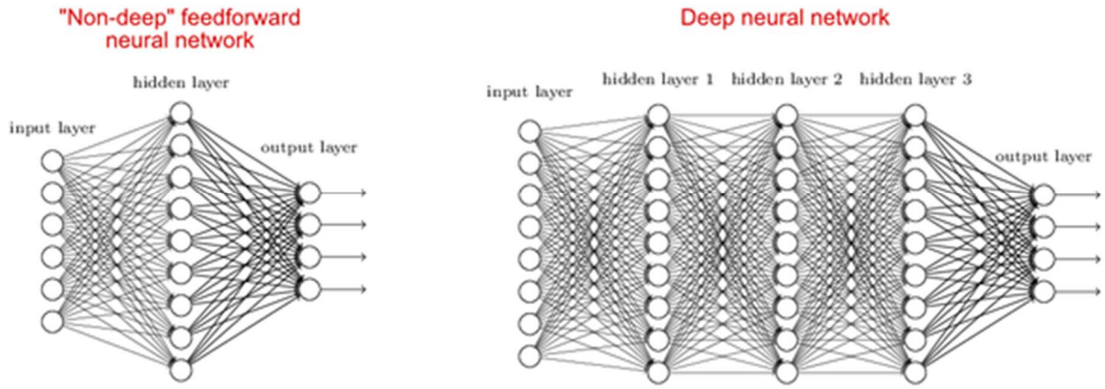

Figure 3: Difference of NN and deep NN architectures, adapted from (Sutton, 2018).

Nonlinear functions are applied to neurons that are between the first and the last neuron section. These neuron layers are called hidden layers. They are used to extract and alter features. A deep neural network has three layers. These layers can be categorized down to tree segments which are the input layer which is the initialization of the network, hidden layers which alters the transition from one neuron to another via non-linear functions, and output layer which is the output portion of the neural network, as shown in Figure 3. The neurons in the input layer are generalized from characteristics gathered by sensors as they see the surroundings. The hidden layers may have multiple layers, and the neurons on those levels are referred to as feature representations. The output layer holds the desired outcomes, such as the distribution of all potential actions. The output values received from an earlier layer is used as input for each subsequent layer of DNN. Through the network connections between neuron layers, weights are fully connected to each other, leaving no unconnected neuron connection behind the output layer.

Every node from a hidden layer takes weight values from an earlier node which can belong to a hidden layer or the input layer, then analyzes bias and generates an output value with a non-linear function applied. This non-linearity breaks the simple linear representation of the network and is crucial. This non-linear function is referred as activation function. Assume a certain number of input nodes to understand this procedure (right side of Figure 3). Let's annote the weights from *i*th node *j*th node as *wij*, the input signal as *I* and and bias as θ.

$$h_j = \sum_{i=1}^{N} w_{ij} I_i + \theta_j \qquad \textbf{(2.1)}$$

You may utilize multiple activition functions to address various difficulties, or you may configure a custom activation function to match your specific situation. There are also multiple well-defined, trusted, and widely used activation functions, such as ReLu, logistic, heaveside tanh, and sigmoid. We may calculate error derivatives backwards in order to optimize the loss of our neural network. This process is called backpropagation. The way this method applied is by finding a gradient called backpropagation gradient that calculates the derivatives from input to output. This guarantees that weights may be adjusted to maximize some loss function. Finding the appropriate weights and biases is a crucial step for learning process and should be put time into configuration of the current network

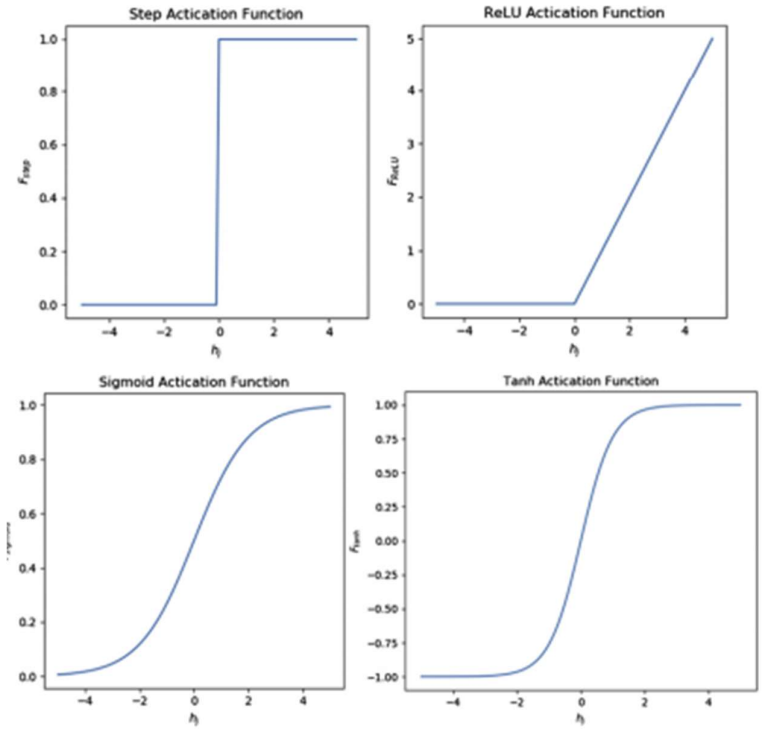

Figure 2: Brief overview of activation functions, adapted from (Lapan, 2020).

There are over 15 different kinds of deep learning algorithms. Each algorithm may be appropriate for one or more of the aforementioned uses. Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs) are the most extensively used deep learning techniques. Aside from the several other techniques are also valid for example, Self Organizing Map (SOM), Generative Adversarial Networks (GANs), and Autoencoders. CNNs are very good at image related databases, but RNNs are not. RNNs are good at text and voice-based data recognition.

2.2.1 Convolutional neural network (CNN)

CNNs were created particularly to perform image recognition for a data that consists of handwritten digits. The picture is decomposed into three scales, and sorted, using CNNs. Initial layers of the network architecture may extract minor details or local aspects of the picture, such as corners, paths, and curves brutely. The following layers would then put these characteristics together, and output layer would recreate the whole picture. Different procedures that involves activation functions, and convulution are all part of the CNN's training segment for image databases.

2.2.1.1 Convolution layer

The initial stage is discretization of the picture into pixels. This construct bases of in image recognition and classification. Recognition and classification depend on the form and color, each pixel may have a different RGB value. Let's imagine a simple case: discretization of a plus sign picture. The number one may be used to represent black pixels, while the number zero may be used to represent white pixels (Table 1).

Table 1: Discreatization of a plus sign

0	0	1	0	0
0	0	1	0	0
1	1	1	1	1
0	0	1	0	0
0	0	1	0	0

The major portion of the CNN is the convolution process which detects local characteristics by applying certain filters or functions to a particular area of the picture. To put it another way,

convolution allows you to concentrate on a single aspect of an image by using certain filters. The filter scans the picture for particular patterns associated with each characteristic such as corners, paths, and curves. Kernel function delivers huge positive numbers when the pattern is discovered in a certain location. The convolution process may be thought of as the summation of the dot products of the selected sections and the applied filter. Consider a three-by-three matrix as a detection filter that should detect the plus sign. Since plus sign consists of a horizontal and a vertical segment. Two vector types can be constructed for this image. First is a 1x3 and the other 3x1 sized vector of ones. The rest of the matrix is set to 0. The matrix moved to the top left corner to the image and the values are dot producted. Then, a column is shifted towards right and the same procedure is applied. By doing this for every row a matrix smaller than the size of the image will be produced. The output is called a feature map. To extract a range of feature maps from an image, several filters may be utilized. The convolutional layer is made up of a data array consisting of these feature maps.

2.2.1.2 Pooling layer

Pooling, which reduces the dimension of the extracted feature maps by removing the dominating features, is the next stage in the CNN. The decrease of feature map's size preserves critical information while decreasing computing resource requirements, which is very useful for processing huge photos. Kernel function covers a region of the picture during the pooling action. The filter (kernel function) is obligated to return values based on the portion it resembles. These can be both maximum (maximum pooling) or average (average pooling).

2.2.1.3 Fully connected layer

Following the creation of the feature maps, outputted from the pooling layer, the feature maps are reduced to their vectoral forms. These are then going to be used in deep learning networks by feeding them into the input layer of the neural network. Finally, deep learning algorithm trains itself to recognize and classify the image at hand.

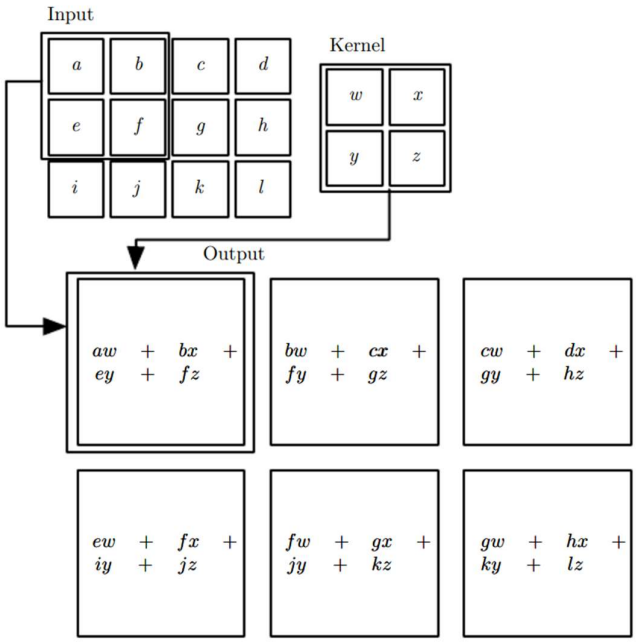

Figure 3: Basic convolutional neural network architecture, adapted from (Sutton, 2018).

2.2.2 Recurrent neural network (RNN)

Deep learning algorithms varies from data type to data type. A database dealing with sequential data, such as text sentences, is required to run RNN algorithms. When confronted with a text sequence recurrent neural network would output the desired solution as the network is configured for this specific use case. For aerial combat this may be used for estimating future air combat confrantations. Commnication data can be retrieved and analyzed to understand when the confrontation should begin. Furthermore, unknown environment specifications can be guessed by analyzing information such as weather forecasting, time, etc. RNNs are built to process data in a sequential order. The RNN's fundamental difference from the others is that the earlier episode's reward value is taken into account for the determination of the next episode's output. Recurrent neural network is depicted in Figure 4.

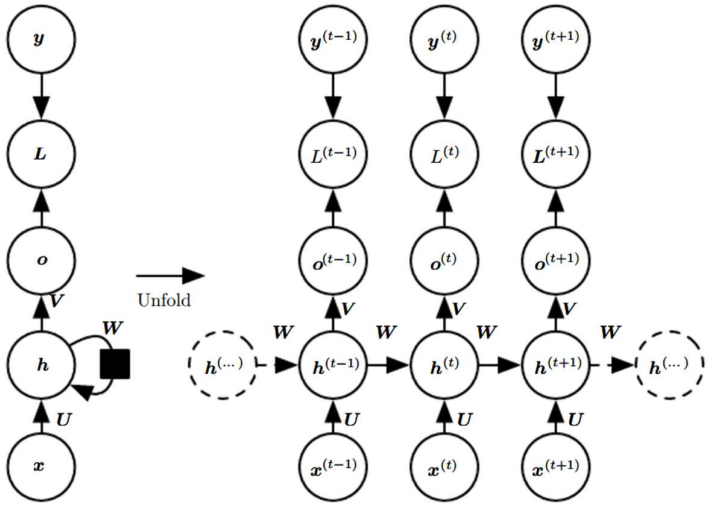

Figure 4: Overview for the recurrent neural network, adapted from (Sutton, 2018).

The state vector of the hidden layer from an earlier layer is multiplied by the respective weight vector denoted as *Waa*. Then, using the equation below, you can determine the hidden layer and output states:

$$A^t = F(W_{ai}I^t + W_{aa}A^{(t-1)} + \theta_a)$$
$$O^t = F(W_{ao}A^t + \theta_o)$$

W is the weight matrix between layers and θ is the bias. a in the subscript represents the intial layer, while i representing hidden, and o representing output layer.

Table 2: Important hyperparameters to pay attention to, adapted from (Chen, 2018)

Neural Networks	**CNN**	**RNN**
Number of layers	Number of layers	Hidden state initialization
Number of hidden-layers' neurons	Number of FC layers	Design of long short-term memory
Sparsity	Patch/filter size	
Initial weights	Padding types	
Activation functions	Pooling types	
Loss functions	Layers' connection types	
Optimization types	Stride length	

Clipping		
Regularization		
Batch sizes		
Learning rate		

2.3. Reinforcement Learning

We need to understand more about Reinforcement Learning in order to attain the desired behavior of an agent that learns from its errors and improves its performance (RL). RL is a sort of machine learning that enables us to design AI agents that learn from their surroundings by interacting with them in order to optimize their overall benefit. Agents in RL algorithms are motivated by penalties for poor behaviors and rewards for successful ones, similar to how humans learn to ride a bicycle by trial and error. The agent gets feedback after each action. The reward and the next state of the environment make up the feedback. In most cases, a person determines the award. Using the bicycle as an example, reward may be defined as the distance traveled from the initial starting place.

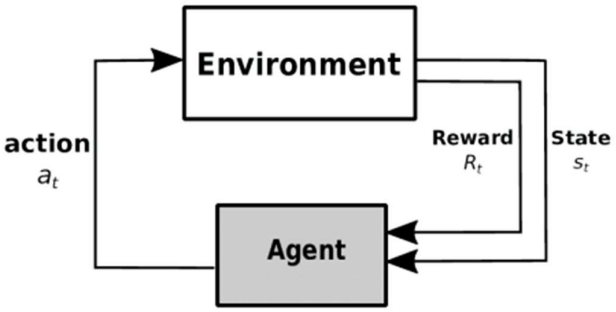

Figure 5: Representation of interactions between the agent and the environment in reinforcement learning, adapted from (Goodfellow, 2016).

The first point to remember is that observation is dependent on an agent's activity. If the agent does wrong actions, the feedback will all be negative observations. Thus, it will be harder to understand which action was relatively better to take. If the agent is obstinate and continues to make errors, the observations may provide the erroneous believes and think that no other benefical policy can be taken, which is completely inaccurate. The second factor that makes the

agent's life more difficult is that the agent does not apply the ideal policy and take correct actions, but also actively investigate the environment in order to see whether taking other actions can considerably enhance the result. The concern is that too much exploration might reduce the reward (agent's ability to remember what it has learnt before), so we must strike a balance between the two activities. Reinforcement learning, despite all of these challenges and complexities, has made significant progress recently, therefore it is becoming more and more popular in both academia and industry. Figure 6 represents the many approaches that can be taken from the basic reinforcement learning algortihms.

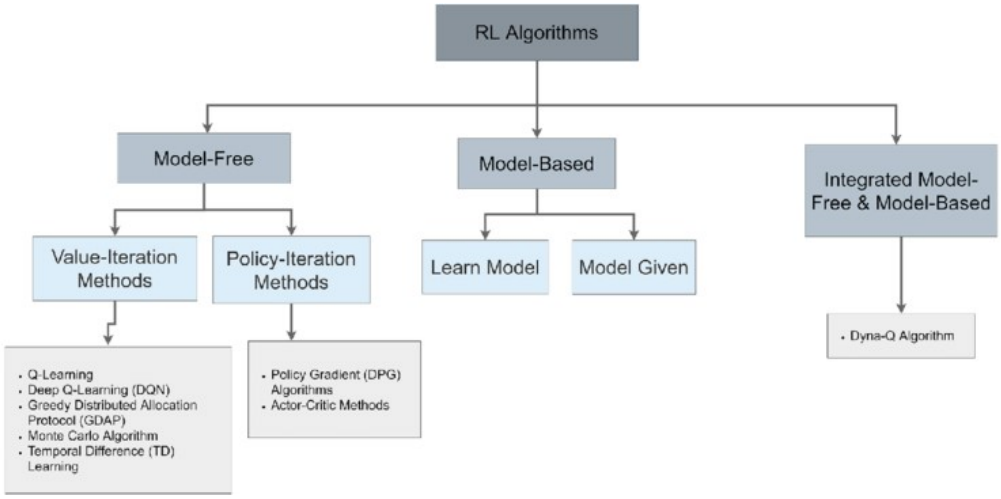

Figure 6: Overview of RL algorithms, adapted from (Shyalika, Silva & Karunananda., 2020).

Reinforcemnet learning algorithms can be explained in terms of Markov Decision Processes. The state space is a collection of all potential states for a system. We need this collection of states to be limited for Markov Processes. Your observations establish a chain or a series of states. Markov Processes are sometimes known as Markov chains because of this. State history is the combination of states formed by a series of observations at each step. To be classified as an MP, a system must satisfy the Markov property, which states that each individual state must be solely dependent on current state. The Markov property's major goal is converting each and every state independently contained from one another to characterize the system's next state. To put it another way, the Markov property demands that the system's states be distinct and distinct from one another. In this situation, just one state is needed to predict the next state, rather the entire history of the states.

To add reward functionality, Markov Process should be tweaked a little. First and foremost, we must offer value to our movement from one stage to the next. Probability

distribution is already created, however it is being utilized to get the state of the system, so we now have an additional value without adding any further weight to the system. A new constant called discount factor is introduced and it is between 0 and 1. The value of the state is expressed by equation 2.2:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad \textbf{(2.2)}$$

Reinforcemnet learning algorithms can be classified as follow (Lapan, 2020):

- Value-basedorpolicy-based
- Model-free or model-based
- On-policy or off-policy

In value-based systems, instead of calculating the probability distribution of the action space, the agent assesses the value of each potential action and selects the action with the highest value.

2.3.1 Tabular Q-learning

The creation of an off-policy control method known as Q-learning is probably the fundamental approach to reinforcement learning algorithms because it dates back to 1990s and inspired many research fields and algorithms (Watkins, 1989), as defined by:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right] \quad \textbf{(2.3)}$$

In this scenario, regardless of the policy used, Q approximates the ideal action-value function. (Sutton, 2018) This greatly simplifies the method's algorithmic configuration and allows for quick and stable responses relatively earlier. In that it decides which state–action pairings are visited and altered, the policy still has an impact. All links should be updated nonstop so that the the algorithm does not get stuck in a local optimum. It seems that the Q value does

not get stuck in the local optima when the probability of 1 to the value of the optimum action-value function. Below is a procedural represent

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 7: Q-learning algorithm, adapted from (Sutton, 2018)

Figure 6 represents the general scheme of the algorithm. The algorithms begin by relating states to actions in a tabular table. Then acquires previous state, reward, action, and new state by acting on the model's environment. Following that, a new action is taken by the agent and the user can see its effect on the environment observations. The implementation of this can varies greatly from environment to environment. Finally, the Q value is updated by using the Bellman approximation with equation 2.2.

2.3.2 Policy gradients

The policy gradient indicates which way we should adjust the hyperparameter values of our neural network to better the policy based on the reward the environment delivers. The gradient's derivative can be regarded as the action's outcome because the action's probabilistic value is taken into calculation. This implies we're attempting to raise the likelihood of behaviors that result in a positive overall reward while decreasing the likelihood of activities that result in negative ultimate outcomes. The term "expectation" denoted by the letter E in the formula, simply indicates the mean average of the steps the agent took in the environment. From a practical perspective, policy gradient approaches might be integrated by optimizing this loss function. The negative sign is significant because of tradition in the optimization mathematics. The defined loss function drops lower and lower while the stochastic gradient descent takes place, however on the contrary the gradient should be increased.

2.3.2.1 REINFORCE method

When calculating the Q value action and state is taken into account. There are several advantages to REINFORCE method. A smaller division of episodes is the solution. Between each episode the reward value will affect the system greatly in this method. For example, instead of having a reward of either 0 or 1, other numeric values can be achieved and this would improve on the stability of the system. The second reason to use Q(s, a) for REINFORCE method is to boost the probability of positive actions at the start of the episode and lower them as the episode progresses because unknown nature of the action space is predicted by the discount factor. That is precisely the point of the REINFORCE approach. Firstly, the neural network should be initialized with random weights. Then, episodes should be played out while recording state, action and reward values. Then, at every designated step discounted reward should be calculated accordingly to the formula:

$$Q_{k,t} = \sum_{i=0} \gamma^i r_i \tag{2.4}$$

Loss function should be found with the following formula:

$$\mathcal{L} = -\sum_{k,t} Q_{k,t} \log\left(\pi\left(s_{k,t}, a_{k,t}\right)\right) \tag{2.5}$$

Lastly, the loss function should be aimed to be minimized by updating stochastic gradient descent.

In many crucial ways, the previous method differs from Q-learning (Lapan, 2020):

- There is no need for explicit examination. We employed an epsilon-greedy approach in Q-learning to observe the environment and escaping agent from being trapped with a suboptimal policy. The exploration is now carried out automatically using the probabilities supplied by the network. The network is set up with random weights at the start, and the system outputs a uniform probability distribution. This distribution represents the behavior of a random agent.
- There is no usage of a replay buffer. This results in stochastic behavior for the agent. On-policy reinforcement learning algorithms include policy gradient methods, which implies that the data belonging to older segments are no longer valid for the current system at hand. The good news is that such approaches tend to converge more quickly. The

disadvantage is that they generally need a lot more contact with the model's environment than off-policy approaches like DQN.

- There is no requirement for a target network. We utilize Q-values in this case, but they are based on our expertise in the setting. We utilized the target network in DQN to lose contact with the approximation of Q-values, however approximations are no longer valid.

Approaches based on policy versus value-based methods yield different results and can be summarized like the following (Lapan, 2020):

- Policy strategies immediately improve our behavior, which is what we care about. Value approaches like DQN achieve the same thing in an indirect way, by learning the value first and then giving us with a policy based on that value.
- Policy approaches are on-policy and need new environmental samples. Prior data from old policies, human demonstrations, and other sources may boost value approaches.
- Policy approaches are often less sample-efficient, implying that greater engagement with the environment is required. Big replay buffers are beneficial to value techniques. Sample efficiency, on the other hand, does not always imply that value approaches are more computationally efficient; in fact, it is often the case.

2.4. Deep Reinforcement Learning

Value function approaches and policy search techniques, as explained in the reinforcement learning section, brings about their own advantages and disadvantages. These methods vary from use case to use case as the state and action definitions change. Reinforcement learning have the same concerns with regard to complexity. Inefficient feature vectors are what brings down reinforcement learning because of the size of states and actions. It also performs poorly at continuous action spaces. Consequently, desired result might take longer than usual to be formed. To get around this problem new techniques called deep reinforcement learning algorithms were invented. Q-learning has to create each and every state individually. An observation for every time step and tabulation of these observations drastically effects computer performance. Instead, deep reinforcement learning constructs ways to express the state of the environment by inhibiting value function methods and combining it with a neural network.

A policy specifies how the learning agent should act at any given moment. A policy, in general, is relating environmental conditions to actions to be done while in those situations. The policy might be as basic as a function in some circumstances, while it may be as complex as a search procedure in others. Because it is sufficient to decide behavior, the policy is probably the most critical element of reinforcement learning. Policies have the potential to be completely random, thus this would address a probability value for each action.

The main objective of the reinforcement learning algorithm is defined in terms of its response which is simply called as reward. For every step that the application took, the environments responds with a reward related to the action that the agent took during that time step. The agent's main goal is to increase the overall payment it earns over time. The agent's favorable and negative occurrences are therefore defined by the reward signal. The reward the environments responds with is the key foundation for changing the policy; if a policy-selected action is faced with a relatively less reward, the policy may be adjusted for the next time step in order to pick a different action in that circumstances. Rewards can be completely randomized functions of the environment and the consequences the actions bring to the environment.

A value function explains what is beneficial in the general sense, but a reward response describes what is beneficial in the short term. State’s current value is close to the entire reward the agent may anticipate to accrue. While rewards can be seen as an immediate response about what type of the situation the agent faces in the environment or what has been observed from it, values correspond to the overall likeliness of how good the agent will act in the environment. Relatively less reward responses do not corrolote to bad actions. Other states might assert that the action that the agent is taking will result in a better situation even in the vicinity of lesser rewards. Therefore, in a sense it can be said that we seek relatively high state values which might not corrolate to high rewards. A downside to this is that reward configuration is much faster and accurate to the implementation than state values.

The environment primarily provides rewards; however, state values have to be evaluated and re-evaluated based on the history of the agent’s previous observations. Over the recent years, the key function of value estimate has perhaps been the most significant item founded in the field of reinforcement learning studies.

The model is anything that replicates environmental behavior, or more broadly, anything that permits predictions about the environment’s behavior. The model might anticipate the future state and reward based on a given state and activity. Models predict the future states and acts based on those predictions, which is defined as any method that acts on the environment by examining potential future scenarios before they occur. Model-based approaches employ models

and predictions to solve reinforcement learning issues, in contrast to their simpler model-free counterparts where the approaches are specifically based on trials that the agent makes. In the earlier example, we only needed to consult our NN once during training to get the probabilities of actions. In the Bellman upgrade, we need to process two batches of states in In DQN, we need to process two batches of states: one for the current state and another for the next state in the Bellman update.

In other scenarios, such as continuous control challenges or circumstances where access to the environment is inexpensive and quick, policy techniques will be the more logical option. Value approaches, on the other hand, will shine in a variety of settings.

2.4.1 Deep Q network

Although the Q-learning approach we just discussed overcomes the problem of iteration through the whole set of states, it may still suffer in circumstances when the number of observable states is quite big. To give an example, old video games might have a wide range of displays, so if we utilize raw pixels as separate states, we'll rapidly run out of states to store and estimate values for. The number of possible observable states in certain situations might be almost limitless. In CartPole, for example, the environment provides us with a state of four floating point values. The number of possible value combinations is limited (it is expressed in bits), yet it is enormous. We could use lines, curves and corner detections to discrete the image segment, but it typically causes more difficulties than it solves: we'd have to figure out which parameter ranges are crucial to differentiate as various states and which ranges may be grouped together. Because a single RGB variation within a pixel differens on the display of this game does not make much of a difference, it is more economical to handle both pictures as a single state. In spite of this, certain states must still be distinguished. We may utilize nonlinear functions to solve this issue, which translates state and action arrays to a value. This is known as a "regression issue" in machine learning (Lapan, 2020). The specifics of how to express and train such this technique can differ from implementation to implementation, however deep neural network integration is one of the most common approaches, particularly when the agent comes across similar yet different observations. With this in mind, consider the following changes to the standard Q-learning technique (Lapan, 2020):

- With some initial estimate, start Q(s, a).
- Get the state, action, and reward information from the environment

- Determine the loss with the following formula if the episode has ended

$$\mathcal{L} = (Q(s,a) - r)^2 \quad \textbf{(2.5)}$$

Otherwise, use the following to calculate the loss function:

$$\mathcal{L} = \left(Q(s,a) - \left(r + \gamma \max_{a' \in A} Q_{s',a'}\right)\right)^2 \quad \textbf{(2.5)}$$

- Using the stochastic gradient descent (SGD) technique, update Q(s, a) by minimizing the loss in terms of the model parameters.
- Continue to getting new state, reward and action values from the environment until you reach a point of convergence.

2.4.1.1 Epsilon-greedy algorithm

When the Q estimate is poor at the start of the training, random behavior is preferable since it offers us more consistently dispersed status of the model's environment. If the stochastic model becomes wasteful as the training proceeds, and then it would be favorable to return to the Q-value approximation to determine how to operate. An epsilon-greedy technique, which simply implies alternating between stochastic and policy based on Q-learning while taking the probability hyperparameter into account, is a technique that accomplishes such a blend of two very different behaviors. We may choose the scale of the scohastic level of actions by changing the epsilon value. The standard procedure is to begin with a value of = 1.0 (100 percent random activities) and gradually reduce it to a smaller number. Using an epsilon-greedy strategy allows us to discover the environment at the start of the training and adhere to excellent policy at the conclusion. This is one of the most basic unanswered topics in RL, as well as an active study topic that is nowhere near being fully answered. An example algorithm is presented below (Zhang, 2018):

Algorithm 1. ε – *greedy* strategy.

Input: control parameter ε
Process:
1: **if** random() < ε
2: action←random from set A
3: **else**
4: action←$\underset{a}{\text{argmax}} Q(s,a)$
5: **end if**

2.4.1.2 Replay buffer

Q-learning approach takes its foundation from supervised learning. Indeed, we're using a neural networks to approximate a complicated, nonlinear function called Q(s, a). To do so, we'll use the Bellman equation to compute goals for this function, then pretend we're working on a supervised learning issue. One of the most important conditions for stochastic gradient descent optimization is that the training data be uniformly distributed and independent. In our scenario, the data we'll utilize for the stochastic gradient descent update does not meet these requirements. Even if we collect a great number of observations from the environment and produce states from it, they will all be quite similar since they will all be from the same episode. Moreover, the training data will not be distributed in the same way as the samples supplied by the optimum strategy we wish to learn. We'll get data as a consequence of another policy; however stochastic behavior should not be adopted by the agent; the agent should act on the optimal policy when it receives a relatively high reward instead. Instead of utilizing our most recent experience, we normally need to employ a big buffer of previous state and a selection of the training set from it to cope with this annoyance. Replay buffer is the term for this technology. The most basic solution is a fixed-size buffer with fresh data pushed to the last place of the said buffer. By doing this the older states in the history should be extracted from the array. We can on different data using the replay buffer, however the aforementioned states can still integrate into newly trained state samples created by the current strategy.

The algorithm for DQN is presented in the figure below (Winberg and Ohrstam, 2020):

Algorithm 2 DQN

initialize experience memory D with capacity N
initialize ϵ
initialize network, state-action function Q with weights θ
for $episode = 1$ to $total\ episodes$ **do**
  $s_t \leftarrow$ initialize random state
  for $step = 1$ to T **do**
    with probability ϵ choose a_t to random action
    Otherwise $a_t \leftarrow \max_{a'} Q(s_t, a'; \theta_t)$
    execute action a_t and observe $r_t, termin, s_{t+1}$
    $[s_t, a_t, r_t, s_{t+1}, termin]$ store to D
    $s_t \leftarrow s_{t+1}$
    sample random minibatch $(s_j, a_j, r_j, s_{j+1}, termin)$ from D

$$y_j^* = \begin{cases} r_j \text{ if episode terminates in step } j+1 \\ r_t + \gamma \cdot \max_a Q(s_{j+1}, a; \theta_t) \text{ otherwise} \end{cases}$$

    preform gradient descent step on $(y_j^* - Q(s_j, a_j; \theta_t))^2$
    to update weights
    if termin **then**
      terminate episode
    end if
  end for
end for

Figure 8: DQN algorithm

2.4.2 Actor-Critic

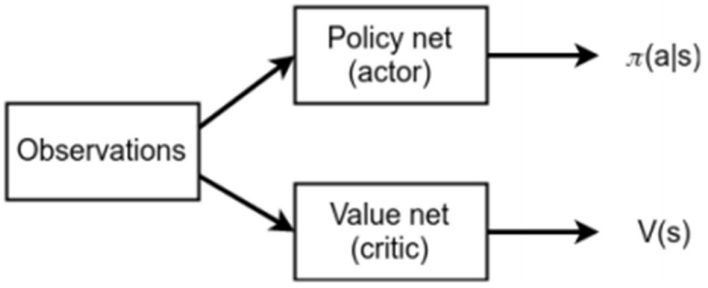

Figure 9: Actor-critic architecture

Despite the fact that the REINFORCE approach can comprehend both value of state and the policy, it is still not a proper actor–critic technique since the value of the state is only utilized as a baseline. That is, it is only utilized as the main network of the state, instead of bootstrapping which is a term used for updating the state estimate based on the succesivity of the previous

states. This is an important difference since it is only via bootstrapping that bias and an asymptotic dependency on the approximation are introduced. As we've shown, the bias generated by bootstrapping and dependence on the state can be advantageous since it decreases variance and speeds up learning. In this situation, with the absence of actor-critic, REINFORCE algorithm is bound to be stuck in the local optima point of the optimization process, however it comprehends the environment gradually and is cumbersome to use online or for ongoing situations, like other Monte Carlo techniques (Sutton, 2018). We may avoid these inconvenients using temporal-difference approaches, and we may pick the bootstrapping frequency with multi-step approaches. In the instance of policy gradient approaches, we combine actor–critic approaches with a critic to achieve these benefits. One-step techniques are appealing since they are totally online and incremental while avoiding the complexity of eligibility traces. The entire return of REINFORCE is replaced with a learnt state-value function in one-step actor–critic approaches which can be represented as follows:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\Big(G_{t:t+1} - \hat{v}(S_t,\mathbf{w})\Big)\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)} \qquad \textbf{(2.5)}$$

The following is the pseudocode for the actor-critic algorithm (Sutton, 2018):

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\nabla\hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I\delta\nabla\ln\pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Figure 10: Algorithm for the actor-critic method

2.4.3 Proximal policy optimization

The OpenAI team suggested the PPO approach which was proposed in 2015 by OpenAI team (Schulman et al., 2017). PPO, on the other hand, is considerably easier to learn than some of the newer approaches like TRPO, therefore it'll be applied in the context of this thesis. John Schulman et al. suggested it in a work titled Proximal Policy Optimization Algorithms, which was published in 2017 (arXiv:1707.06347). This is a technique that requires a lot of hyperparameter tuning. The main difference between the previous actor-critic model and this one is the formula for calculating policy gradients. The PPO technique employs a different viewpoint that takes the policy ratios into account. These parameters are affected by benefits, instead of rather than the gradient of logarithm likelihood of the action adopted. The main objective of the PPO algorithm is represented like the following:

$$J_\theta = E_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\, A_t\right] \qquad \textbf{(2.5)}$$

Sampling is the main reason why the difference in the objective function has changed from the other methods. Maximization part of this function might lead to drastic results as it grows exponentionally. Therefore, clipped function is used to restrict the update. The trimmed aim might be stated as

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \qquad \textbf{(2.5)}$$

if the ratio between the new and old policies is stated as

$$J_\theta^{clip} = \mathbb{E}_t[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t)] \qquad \textbf{(2.5)}$$

The algorithm can be expressed like the following (Schulman, 2017):

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Figure 11: PPO algorithm

3. METHODOLOGY

3.1. Tools

For the main programming language, Python is used due to its popularity, community sport and rich machine learning modules. Deep reinforcement learning modules are scarce in Python and mainly dominated by OpenAI Gym's Python module. *Baselines* is an important extension to Gym framework. Moreover, simulation environment is maintained with Pyglet module. The reason of this choice resides within the OpenGL capabilities of the module as it is fast and reliable.

3.1.1 OpenAI Gym

OpenAI Gym, as a non-profit organizatio was established in 2015 with the mission to develop robust and reliable artificial intelligence algorithms and ensuring that artificial intelligence's advantages are as broadly and equally dispersed as feasible. In addition to researching a wide range of challenges related to artificial general intelligence, OpenAI made significant contributions to the reinforcement learning community by building the Gym Python module. There are various toolkits available for the construction of reinforcement learning environments. OpenAI Gym is a framework for creating reinforcement learning environments as well as a collection of standards for algorithm structure that may be utilized in research articles. It is possible to test and create algorithms that utilizes RL methods in Gym, which is a collection of settings developed for this purpose. Gym is mainly used for benchmarking purposes for different environments it has or the custom environments that researchers created for their projects. It also built a common coding structure as well as a community where other engineers can review articles, projects, and algorithms. They also offer the comparison of different reinforcement learning frameworks which is one of the main focus of this thesis.

In addition to relieving the user from the need to build up complex settings, Gym aids to generalization and standardization, allowing researchers to operate from a similar set of guesses.

Gym is developed in the Python programming language. Its diverse contexts include robotics and old-fashioned games (due to small pixel sizes), to name a few examples. Within the context of this thesis OpenGL rendering protocols are used which is supported by *Pyglet* module.

3.1.2 Keras and Tensorflow module

TensorFlow was founded for scientific computing taking performance in mind which is also available as an open source project. With its highly adaptable architecture, it is simple to deploy computing over a wide range of platforms. Supported platforms supports CPUs but GPUs are the main area where this module shines. There also TPUs as a new technology that can lead the way of artificial integelligence. Tensor flow can be applied from desktop computers to high performance computing clusters, to mobile devices, among other things. Main authors of this piece of software belongs to a team from Google called Google Brain. This team utilizes artificial intelligence algorithm in favor of Google. Therefore, it has a fundamental support for many areas of machine learning including DL, and RL. Moreover, at the core of the system high-performance, and complex scientific computations that are used across a wide range of scientific fields, including aeronautical physics are rich within research papers. Keras is a neural networks program that is developed in Python language and it may be used on top of neural network frameworks such as TensorFlow. With the help of the Keras, you can create your models as in declarative programming languages, which is a time-saving feature. Many standard neural network building pieces including as layers, goals, activation functions, and optimizers are implemented in Keras as well as several tools to make dealing with picture and text data simpler. Keras is available as a free download from the official website. In this study, we utilize Keras to create the model, using Tensorflow and Keras python modules.

3.2. Environment

3.2.1 Aircraft maneuver modelling

First of all, aerial vehicle's combat model needs to be modelled. There are famous working environments that can handle difficult aircraft combat situations (Ernest et al., 2016). The emphasis of this thesis' study is on deciding maneuvering policy, which primarily takes into account the positional connection and velocity orientation and strength for both of the aircrafts

in 3D space. As a result, the aircraft manuevering model of the airplane in thesis is built upon a three-degree-of-freedom framework based on a particle model. By taking into account that the velocity orientation and strength corresponds with the body axis, other angles related to the aircraft maneuvering decision are simply ignored. The lateral axis represents east, the longitudinal axis represents north, and the vertica axis represents altitude's direction in the ground coordinate system. The fixed wing aircraft maneuvering model is described by the following formula (Yang et al., 2020):

$$\begin{cases} \dot{x} = v\cos\gamma\sin\psi \\ \dot{y} = v\cos\gamma\cos\psi \\ \dot{z} = v\sin\gamma. \end{cases} \tag{3.1}$$

Attitude change of the aircraft is modelled like the following (Yang et al., 2020):

$$\begin{cases} \dot{v} = g\,(n_x - \sin\gamma) \\ \dot{\gamma} = \dfrac{g}{v}\,(n_z\cos\mu - \cos\gamma) \\ \dot{\psi} = \dfrac{g n_z \sin\mu}{v\cos\gamma}. \end{cases} \tag{3.2}$$

Where the position coordinates are denoted as x, y, and z for lateral, longitudinal and vertical axes respectively. v represents the speed of the aircraft. $\dot{x}, \dot{y}, \dot{z}$ are simply the derivatives of the position of the aircraft and represents the velocity of the aircraft for their respective coordinate axes. γ is the flight path angle which is the angle between the 2D plane of the environment and the velocity direction. ϕ is the heading angle which is the angle between the velocity vector's projection on the ground and the true north. μ is the bank angle of the aircraft which is the angle of the wing and the airplane's lateral axis. g is the gravitational force and equals to 9.81. The positional vector is denoted as $p = [x, y, z]$ and velocity vector is defined as $\dot{p} = [\dot{x}, \dot{y}, \dot{z}]$. These variables are modelled in the figure following (Wang, 2020):

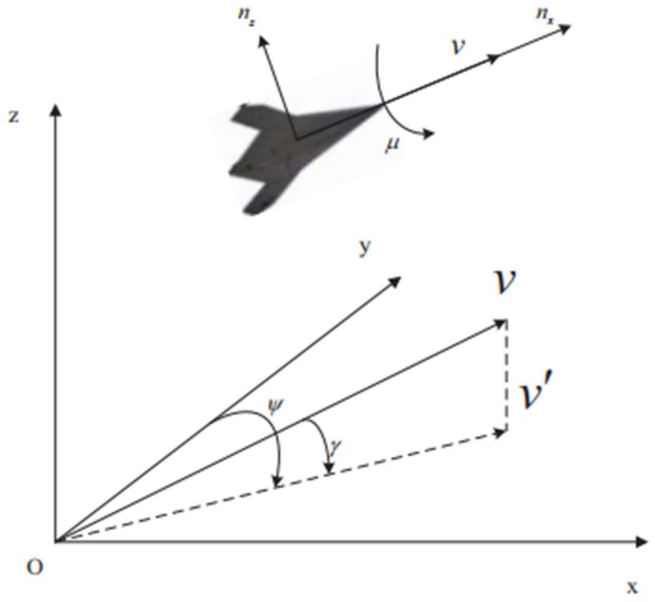

Figure 12: Aircraft three degree of freedom maneuvering model, adapted from (Yang, 2019).

n_z, n_x, μ are the parameters that design the maneuvering decision the aircraft. n_x is the longitudinal motion of the aircraft. It resembles an engine and it is responsible for the propulsion effect for the aircraft. n_x is called as overload velocity. The effect that decides the altitude change is defined by n_z and called as normal overload for the pitching direction. Overload velocity is responsible for the pitching motion for the aircraft. μ is the bank angle and it is responsible for the rolling motion. These parameters are visualized in Figure 12.

Dogfighting is another name for these types of air combat. The purpose of this aerial battle is to use guidance to allow the jet fighter to trail the opponent's tail while avoiding the enemy entering its attack zone. In a typical aerial warfare scenario like the one shown in Figure 1, the jet fighter should attempt to change its orientation towards the center of the enemy aircraft target, and pursue it. During air combat scnearios important parameters comes forth such as distance. Distance is an important consideration in addition to the circumstance. The weapon employed in close air warfare has a very short attack range compared to other weapons. It is unable to assault the target if it is outside of the range. The shorter the distance between the target and the shooter within the available shooting range, the higher the likelihood of eliminating the enemy aircraft.

3.2.2 Maneuvering decision modelling

Figure 13 depicts the environment's working scheme of reinforcement learning for the close distance air warfare maneuver choice made by a jet fighter such as F16 in accordance with the interactive process. The states of the jet fighter and the enemy are combined and computed to generate a state vector based on the model's environment statement for the aerial warfare orientation, that is then sent to the model's agent. The aerial warfare framework's model generates the jet fighter's benefit assessment value depending on the present scenario for every jet fighter, and it directs this value in the form of reward vector that is feed into the model's agent. This reward is important because it aims to better the learning state of the agent.

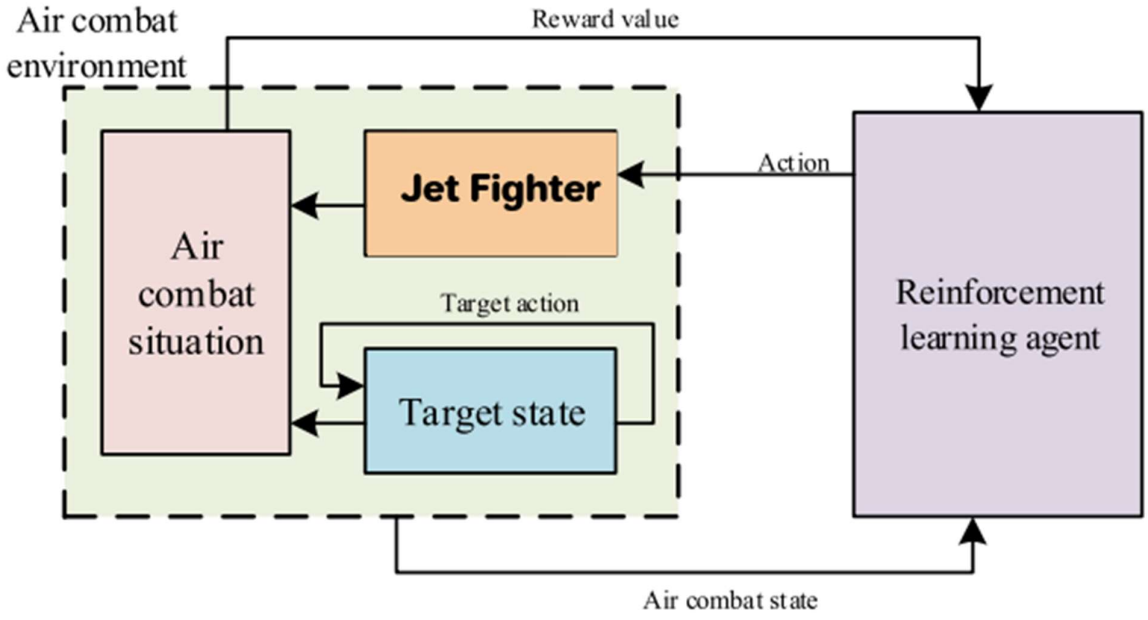

Figure 13: RL framework for dogfight scenario, adapted from (Zhang, 2018).

During the action phase, the agent sends the action vector to the framework's model's environment, which changes the jet fighter's state vector, and the target network is updated based on the state vector generated with its own maneuver strategy, as shown in Figure 13. The agent is bounded to the environment for receiving information, state, reward, action vectors based on air combat scenario to update the policy of agent's network. The maneuver policy of the jet fighter is constantly modified in response to the changes, by doing this the action is becomes idealistic, therefore implementing the reinforcement learning policy of the jet fighter's air warfare maneuvering decision. Because action and state spaces are large and discrete, DQN and their derivative algorithms are not preferable. The reason for this will become clear in the

following chapters. Therefore, as an alternative Proximal Policy Optimization policy is also used for the environment. PPO entails gathering a modest number of experiences from engaging with the environment and utilizing that collection to update the organization's decision-making policy over time. As soon as the selected experiences are worked on to change the current policy, the previous experiences are removed to be replaced by another selection of experiences generated by means of the policy that has just been updated. This is why it is called as on-policy technique, in which the selection of experiences acquired are valuable when the present policy is updated, rather than developing new policies. The underlying concept is that, after an update, the new policy should not be too dissimilar from the previous policy. PPO employs clipping to do this in order to prevent making too many massive modifications. This results in less variation in training at the expense of some bias, but it also provides smoother training and prevents the agent from going down an unrecoverable road of doing stupid acts.

3.3. State Modelling

In order to access the relative orientation of an aircraft, new angles must be defined. These are named ATA (antenna train angle) and AA (aspect angle) and visualized in Figure 14.

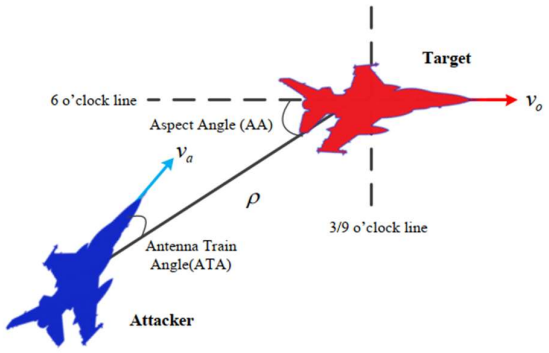

Figure 14: Angle definitions

The line of sight is the path between the center of the two aircrafts. The LOS also corrolates to the distance between the aircrafts It is the angle formed between the LOS and the tail of the enemy jet fighter that is known as the AA. It is the angular distance between the pursuer and the pursued aircraft's tail, expressed as an angle of inclination. The ATA angle is defined as the angle formed between the longitidunal axis the attacking jet fighter and the LOS between the two aircraft (Kurniawan, 2020).

Rather than simply injecting the state vector into the network, we create features that indicate the critical status information of the instances that are sharing the score. The fundamental properties of the state are determined by the relative orientation of the two aircrafts. Although these phrases may be used to convey information regarding velocity and attitude, they are not capable of conveying this information. Additionally, ATA and AA angles are significant for agent's judgment, as they indicate the level of enemy's aggressiveness (Ma, 2018). Taken these into account the following state vector is formed:

$$S = \{\Delta p, ATA, AA, ATA_r, AA_r, q\,, q_r, \psi, \gamma, \phi, v\} \quad \textbf{(3.3)}$$

3.4. Action Modelling

When it comes to the choice issue of air confrontation maneuvers, developing a realistic maneuver framework is a critical step in the design of the environment. Several action models have been established that represent such definitions (Hu, Gao & Wang, 2019). Generally speaking, the design of an air combat maneuver framework is separated into two categories: First of this categorization is based on familiar air warfare guidance techniques, such as maximum overload, acceleration, climb rate, and etc. Second element in the list consists of 25 most common maneuvering techniques on typical pilot maneuvering for aerial warfare such as straight-flat, fixed-height, slow-speed Yo-Yo (Zhang et al., 2018). Accelerating, slowing down, uniform flight, turning left and right, ascending, and diving are the seven main types of maneuvering movements defined by NASA standard (Kong et al., 2020). Each maneuvering action is made up of a unique mix of overload forces which are as follows: n_x, n_z, μ

Table 3: Action and control definitions

v, μ and n_z	μ	n_z	V_{cmd}
Constant speed	0	0	v
Decrease speed	0	0	$v-3$
Increase speed	0	0	$v+3$
Turn right	1	0	v
Turn left	-1	0	v
Climb at const speed	0	1	v

Descend at const speed	0	-1	v

3.5. Reward Function

Reward function is the numerical result of taking an action in a state, and outputting a value based on the policy defined by the environment (Sewak, 2019). One of the most important aspects of reinforcement learning is the use of rewards, which has an important effect on the overall speed of the algorithm.

There are fundamental reinforcement learning algorithms that inspired many other algorithms in the field (McGrew, 2008). The blue aircraft's primary objective is to gain and try to get behind the enemy aircraft for missile engagement, which is their primary objective. This is achieved by McGrew from the following equation: (2010)

$$S = \left(\frac{\left[\left(1 - \frac{\mathtt{AA}}{180°}\right) + \left(1 - \frac{\mathtt{ATA}}{180°}\right) \right]}{2} \right) \exp\left(\frac{-|R - R_d|}{180° k} \right) \qquad \textbf{(3.4)}$$

Based on Equation 3.4 McGrew developed an algorithm that utilizes the equation to achieve a reward configuration. This algorithm is represented by Figure

Input: $\{x\}$
R =Euclidean distance between aircraft
if $(0.1\ m < R < 3.0\ m)$
& $(|\mathtt{AA}| < 60°)$
& $(|\mathtt{ATA}| < 30°)$ **then**
$g_{pa}(x) = 1.0$
else
$g_{pa}(x) = 0.0$
end if
Output Reward: (g_{pa})

Figure 15: Example reward configuration, adapted from (Mcgrew et.al., 2010)

Many more advancements have been made over the reward configuration since Equation 3.4 came out. One such example would be the following (Kong et.al., 2020):

$$\Phi_o(s) = \frac{180° - (|\lambda_b(s)| + |\epsilon_b(s)|)}{180°}$$

$$\Phi_d(s) = \Phi_o(s) \exp^{-k_d|D_e - D(s)|}$$

$$\Phi_v(s) = \Phi_o(s) \exp^{-k_v|v_b - v_r|} \quad \textbf{(3.5)}$$

$$\Phi(s) = \Phi_o(s) + \Phi_d(s) + \Phi_v(s)$$

The dimensionality of the model's environment affects the reward configuration heavily. 2D air combat environments are heavily inspired by Equation 3.4 due to its reliable, robus, stable nature and the fact that it has been used in multiple other research articles. However, this is ineffective for environments that takes place in 3D space because of the additional angle definitions that Equation 3.4 lacks. This lack of definition is satisfied by other reward functions (Yang, 2019).

```
if (Distance) < 0
    Reward += 0.5
    if (Attitude difference) < 0
            Reward += 4
        if (q angle change) < 0
            Reward += 15
if 20 < 3D Distance < 40 & |q| < 20 & |qr| < 20
        Reward += 2000
        Info: "Win"
elif 3D Distance < 20
        Reward -= 3000
        Info: "Collision"
elif 20 < 3D Distance < 40 & |qr| > 160 & |q| >
160
        Reward -= 2000
        Info: "Collision"
```

4. RESULTS

4.1. Setup

Aircraft will learn which policy to apply using inside a simulation program. Therefore, as a simulation tool OpenAI Gym's simulation environment which is the backend for Pyglet module is used as a source of visualization and manual control of the enemy aircraft. The simulation environment is presented in the figure below:

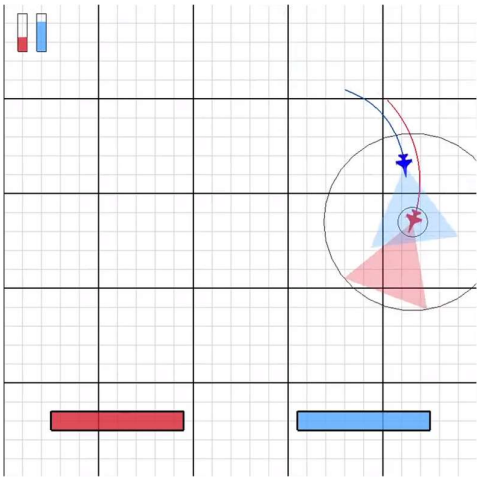

Figure 16: Main simulation environment.

Algorithms, states, rewards and other configurations should be applied according to their relative respective positions in Chapter 3—methodology. However, using different deep reinforcement learning algorithms differs how the agent's code should be implemented greatly. Therefore, initial planning with which common hyperparameters will be used for both of the algorithms and noting them based on their effects on the algorithm is important for future steps. Following the generation of a variable by the neural network, bank angle is determined for the agent's aircraft which will be addressed as μ. The simulation inputs this variable and takes an action based on the current state. After this time step, new position, attitude and relative geometric variables are created and some of them are passed in a state to be served to the neural network algorithm in return. The simulation was firstly based on a simple guidance algorithm as in path finding. The

agent tries to find the enemy aircraft's position which is fixed in space by setting the velocity of the aircraft to 0. If the agent fails to find the location of the enemy aircraft the algorithm is determined to be falsely build and revised for hyperparameter configuration, bug tracking and additional improvements such as soft update. Another detection without visualization resides of the geometric location of the agent relative to the other aircraft because knowing how much offensive position the agent gets during an episode of training is an important to know parameter. Therefore 3 configurations were designed with ATA and AA angles in mind. If the agent is located near-tail section of the enemy aircraft, it is said that the agent is at an offensive position. Opposed to this, if the agent is in front of the enemy aircraft and is continued to be pursued, defensive points are taken. Otherwise, the algorithm returns neutral behavior. These states are set to percentages and displayed in Table 6 and Table 7. Algorithm design is based mainly on the DQN algorithm since it was going to be the first algorithm to be tested.

4.1.1 DQN setup

DQN algorithm did not perform well out of the box. Several modifications were applied for it to perform better for our custom environment. However, not all of these responded well with the model. Here are some of the few that actually performed quite well. First of all, the gradient acting on the loss function is clipped for the stability of the network. Addition to this, a target network is also applied for further stabilization. As mentioned before, replay buffers have distinct advantages for Q-learning applications, which is the reason why it has been used in this thesis as well. Finally, epsilon-greedy algorithm is also applied to control the stochastic nature of the model as it is preferable to have a more randomness agent at the beginning of the testing and less randomness as the testing goes on. The neural network architure along with parameters related to the algorithm structure is presented in Table 5.

Table 4: Algorithm parameters for DQN.

	DQN
Hidden layers	5
Hidden layer size	64-128-256-256-128
Activation function	ReLu
Loss function	MSE
Optimizer	Adam

Learning rate	5e-4
Episodes	10,000
Memory size	100,000
Discount rate	.999
Epsilon decay	9e-5 (lower limit: .1)
Batch size	128
Buffer size	10^6

Later on, based on the given figures in the table above the algorithm is designed, connected to the environment through OpenAI Gym and set ready for training.

4.1.2 PPO setup

For an alternate view on the model at hand environment is observed and learned by means of another deep reinforcement learning algorithm called PPO. However, certain models are established and integrated into the current PPO algorithm in order to enhance the end results. What worked with PPO in regards of the current algorithm was still unknown due to the poor scoring retrieved by the agent that used DQN architecture. By means of testing, actor-critic model looked promising and became involved in the algorithm. In addition to actor-critic model, SOF was preferable because the relative wellness of the current policy constrat to the earlier policy was an essential parameter that can be used in a policy-search based deep reinforcement learning algorithm. Hence, SOF was employed to understand whether the current state resulted from the action retrieved from the NN was truly benefiticial regarding the previous action's results. To conclude, important parameters to be noted are presented in Table 5.

Table 5: Algorithm parameters of PPO.

	PPO
Hidden layers	5
Hidden layer size	64-128-256-256-128
Activation function	ReLu
Loss function	PPO with SOF
Optimizer	Adam
Learning rate	5e-4

Episodes	10,000
Discount rate	.999
∈ (SOF)	.2
Learning decay	.01%

By using these numbers, the current algorithm is designed for an altered code that relies on the DQN algorithm to reflect PPO with most of the code which functions as a utilization tool between different structures and have no effect on how the algorithm functions stood intact.

4.2. Training

Table 6: Air combat behavior of the agent.

	DQN	**PPO**
Offensive	7%	77%
Defensive	3%	12%
Neutral	91%	11%

In addition to this, final results are also recorded and reviewed by the development team for further optimizations. DQN and PPO algorithms' performance on the environment are clearly indicated by Table 6 and Table 7. PPO seems to outperform DQN with a significant degree. The reason for this difference can be summarized by the definitions explained in Section 2.3.2.1 because the greates drawback the algorithm faced comes from the expansion of action space. DQN does not perform well under high dimensional action and state spaces. Adding the third dimension to the already designed 2-dimensional aircraft model (for testing purposes) complicates the algorithm greatly. In 2 dimensional models DQN did not perform too bad, however for three-dimensional space and additional angle definitions that affected reward affected the algorithm drastically.

Table 7: Air combat results

	DQN	**PPO**
Win	7%	77%
Lose	3%	12%
Draw	91%	11%

4.3. Testing

Certain scenarios are tested for the environment to test the stability of the algorithm. First of all is the speed difference. In this scenario, the speed variations for the models was observed based on how different situations they put themselves into. These categorizations are in the form of Table 6. Moreover, whether the near-tail engagement done by the agent is accomplished or not is an important criterion for the determination of the stability. Moving on to the second phase, constant movement types are analyzed. Whether they perform well under strict motions are measured. These motions include constant circular motion, linear path and custom maneuvering which means that a person is manually controlling the enemy aircraft with custom control inputs which needs to be designed. For the final scenario, both agents are given artificial intelligence and observation took place. A stable algorithm results in a circular motion acted by both of the agents because the same neural network is applied to both of the aircrafts. Aside from these, multiple configurations of starting points as in coordinates and aircraft attidues are also tested to see if the agent really performs well under stochastic scenarios.

4.3.1 DQN

After the specified number of episodes took place or the reward criteria is breached, the training section is paused and next section, i.e., testing, took action. Training results for the DQN algorithm is presented in Figure 17.

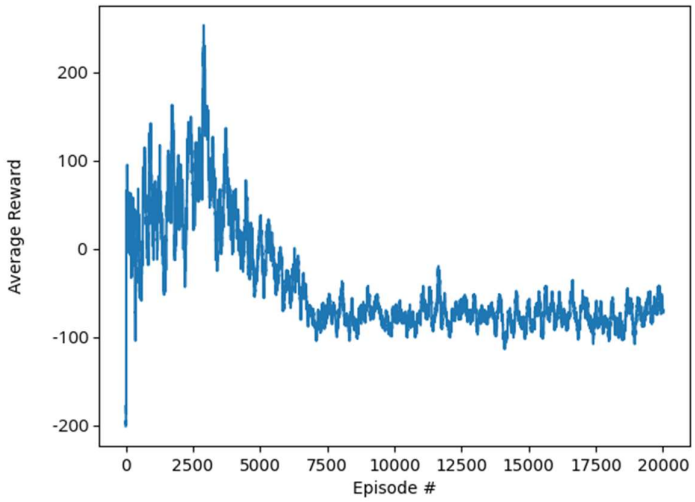

Figure 17: DQN Training result

As can be seen, just like it is mentioned earlier, it does not perform well for 3D environments involving aircraft maneuverings. Visualization done under testing shows that the action determined by the algorithm gets stuck in a local optimum. This behavior is best seen from the circular path in Figure 18.

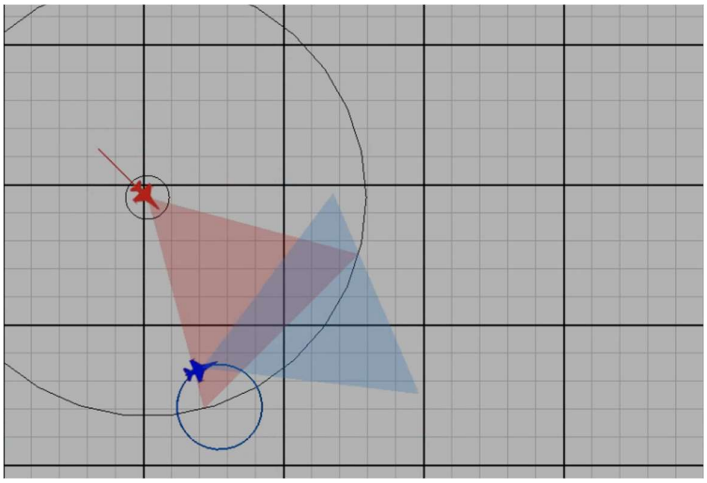

Figure 18: Circular maneuver done by the agent.

The reason why this failure is because of a phenomenon called catastrophic forgetting (Goodfellow et.al., 2014). For projects involving Q-learning practices and deep learning architectures, catastrophic forgetting is a significant issue. Many projects involving deep neural learning architectures forget how to execute the initial task after being trained on only one assignment. According to this, it is commonly assumed that NNs are struggling to remember. While forgettability of memories is a challenge for NNs, there has been methods to move around this problem (Goodfellow et.al., 2014). During the initial stage of training, when one job is learned by the agent, it is possible that the agent can forget the policy for completing the initial job. To elaborate, the deep learning architecture would end up in similar situations where the agent forgets its initial optimal for the second training session regardless of the configuration of the first training route as long as it were initiated with a convex objective. Because the two different aims were used while training the machine learning model, it has forgotten the policy regarding the first aim fully. The accuracy of the machine learning system is solely owing to random resemblances between the jobs.

Due to the lack of success of the algorithm testing scenarios that can be presented for this algorithm is scarce and discarded from the thesis.

4.3.2 PPO

PPO adapted quickly to the environment. The result of the training is visualized in Figure 19.

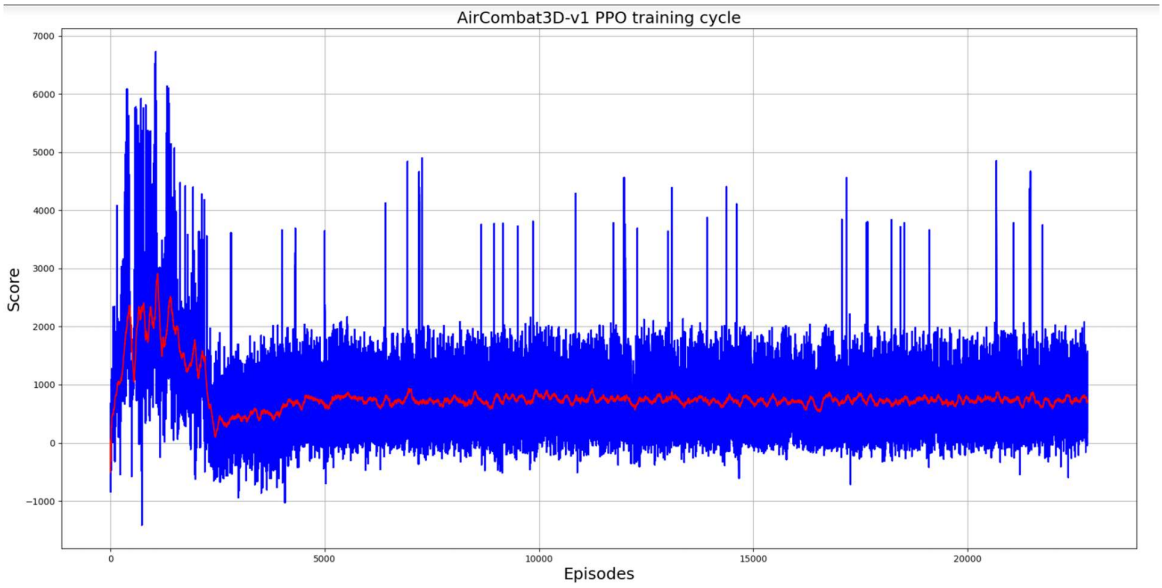

Figure 19: PPO scores

In Figure 19, blue lines indicate the score gained from the environment while the redline denotes the average score. As can be seen the environment is quite stable even in random starting locations and attitude orientations. The visualization that Figure 19 belongs to Figure 20 in which the fighting maneuvers are clearly visible.

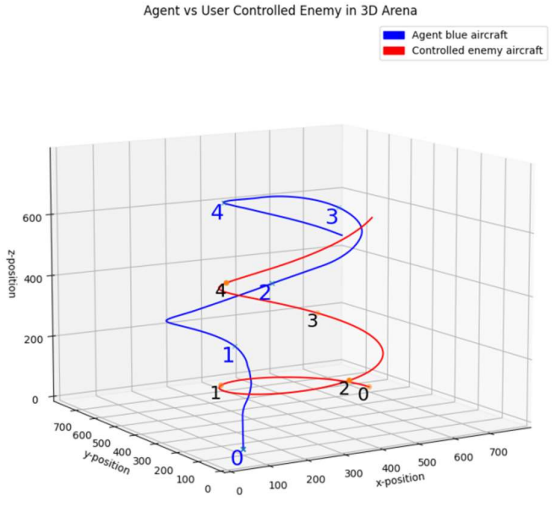

Figure 20: Testing of PPO algorithm

Moreover, the scenarios mentioned in Chapter 4.3 are also tested to see the stability of the algorithm. The end result of these scenarios performed quite well and can be best visualized for the scenario involving multiple agents. This scenario is visualized in Figure 21.

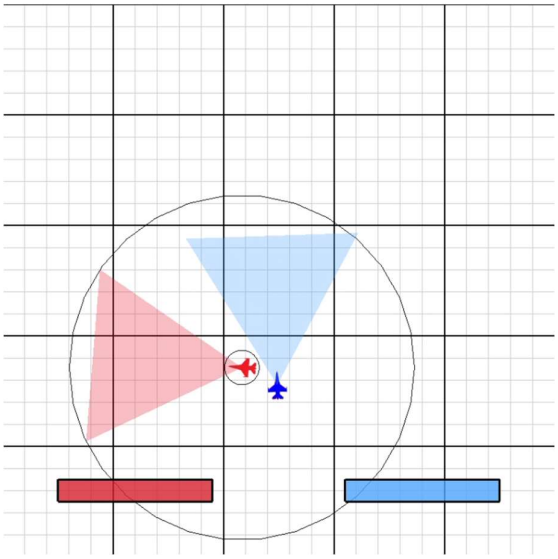

Figure 21: Multi-agent state of the environment.

The agents select compatible actions and are in constant circular loop indicating that the artificial intelligence algorithm that has been developed is stable for use.

4.4. Animation

Testing also took place within the main simulation environment which can be visualized in Figure 16, however a prettier simulation is desired to show the final results. External software that has a steep learning curve such as Unreal Engine or other programs are discarded from the considireation list and simplier modelling and animating softwares started to shine. Because of this, a low-poly model is designed and run within Blender by using the position and attitude history recorded with Python and applied within a script that is integrated in a Blender file. The last result of the environment is presented in Figure 17.

Figure 22: A frame from the blender animation.

5. CONCLUSION

In order to address the issue of poor learning performance and getting stuck on local optimum caused by the big state and action spaces, this research developed a training approach based on the idea of basic and simplified training in air combat. In this thesis, proximal policy optimization is utilized to create an artificially intelligent aerial warfare environment that let's the aircraft decide which motion to apply. The constructed actor-critical network may generate discrete actions for the aircraft to do basic flight maneuvers more accurately and smoothly. The training approach significantly improved the jet fighter's maneuvering effectiveness while training in scenarios using a jet fighter's basic flight maneuvering capabilities. The aerial warfare model represented is more logical, and the reward values that are generated by the model is more fluent when relative position and orientation indicators combined. The addition of missile attack region (the region the agent can attack at) also enhances the overall reward value gained from the environment. In order to show the aerial battle situation, the DRL can easily capture the jet fighter's best maneuver to apply using location and attitude indications.

During the development of this thesis, two distinguishing methods to compare the algorithms have come out. The most important thing in this research was the hyperparameter tuning. We performed a comparison for the selection of hyperparameters, which compared each technique individually. We chose the most optimal hyperparameter setting that gave us the maximum efficiency. Later, we tested algorithms with the selected hyperparameters across diverse environments to see which gave the best results. To analyze the approaches further, we subdivided the results of the training into sections: aircraft behavior, reward gained, and the end reason of the episode. Our findings suggest that only if the environment model is simple, the DQN approach performs better. This environment could have been defined in action and state spaces in lower dimensions at the cost of lower precision and more uncertainty. It is safe to infer that, in situations requiring basic flight maneuverings, when we can construct low-dimensional training models in the form of DQN, we have the best option for training. Proximal policy optimization technique had better overall success during the late stages of the development cycle, and notably when the model was at the near completion stage. Therefore, algorithms like

proximal policy optimization should be considered for high-dimensional reinforcement learning models since it can be seen that it counters them better than DQN.

Optimal maneuvering of a jetfighter based on an aerial warfare environment using proximal policy optimization has been proven to think and therefore comply, in the absence of a significant quantity of artificially created dataset, by using a discrete aerial battle management plan learned from stochastic movement patterns.

5.1. Further Research

This research is restricted by the simplistic and idealistic characteristics of the model written (to exemplify, non-linearity, and simulation of avionic equipment) and the time the RL model has been trained for. The NN is not versatile which means that it is not advisable to apply it to different environments with proper control but it is described theoretically only on the modelling, training, and testing level. Although algorithms may produce solid outcomes, many deep reinforcement learning techniques could not have been verified. This research is not able to undertake more comprehensive simulation and learning model study on several concerns, including the influence of the dimension of the action space as it directly effects the efficacy of the model. An increase in the dimension of the action space may enhance jet fighter's understanding of the environment and the precision of the optimal action, but doing so would degrade the machine learning capability of the DNN. As a result, finding the ideal dimension for the action space in order to identify the network topology may be done by conducting a significant amount of trials, which is a yet to be done challenge for study.

Thus, further research intends to create a network that can be applied to multiple environments that are related to air combat and supply reliable, stable rewards. Another improvement would be the enhanced aerial motion capabilities such as pitchback, cobra turn, and kulbit. On top of all these, the problems in the previous paragraph needs to be addressed as well. As a final work, the only work left to be done is the fine-tuning hyperparameter values.

6. REFERENCES

Andriambololona, M., & Lefeuvre, P. (2003). *Implementing A Dogfight Artificial Pilot*. *July*, 94.

Austin, F., Carbone, G., Falco, M., Hinz, H., & Lewis, M. (1990). Game theory for automated maneuvering during air-to-air combat. *Journal of Guidance, Control, and Dynamics*, *13*(6), 1143–1149. https://doi.org/10.2514/3.20590

Changqiang, H., Kangsheng, D., Hanqiao, H., Shangqin, T., & Zhuoran, Z. (2018). Autonomous air combat maneuver decision using Bayesian inference and moving horizon optimization. *Journal of Systems Engineering and Electronics*, *29*(1), 86–97. https://doi.org/10.21629/JSEE.2018.01.09

Chen, S., Gabriel, M. S., & Durand, C. (2018). *Comparing Deep Reinforcement Learning Methods for Engineering Applications*.

Dong, Y., Ai, J., & Liu, J. (2019). Guidance and control for own aircraft in the autonomous air combat: A historical review and future prospects. In *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* (Vol. 233, Issue 16). https://doi.org/10.1177/0954410019889447

Ernest, N., & Carroll, D. (2016). Genetic Fuzzy based Artificial Intelligence for Unmanned Combat Aerial Vehicle Control in Simulated Air Combat Missions. *Journal of Defense Management*, *06*(01), 1–7. https://doi.org/10.4172/2167-0374.1000144

Goodfellow, B. and C. (2016). Deep learning - Goodfellow. *Nature*.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., & Bengio, Y. (2014). An empirical investigation of catastrophic forgetting in gradient-based neural networks. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*.

Hu, Z., Gao, P., & Wang, F. (n.d.). *Based on Approximate Dynamic Programming*.

İşçi, H. https://github.com/hasanisci/gym-dubins-ac, retrieved at 13.06.2021

Kong, W., Zhou, D., Yang, Z., Zhao, Y., & Zhang, K. (2020). UAV autonomous aerial combat maneuver strategy generation with observation error based on state-adversarial deep deterministic policy gradient and inverse reinforcement learning. *Electronics (Switzerland)*, *9*(7), 1–24. https://doi.org/10.3390/electronics9071121

Kong, W., Zhou, D., Zhang, K., & Yang, Z. (2020). Air combat autonomous maneuver decision for one-on-one within visual range engagement base on robust multi-agent reinforcement learning. *IEEE International Conference on Control and Automation, ICCA*, *2020-Octob*(2), 506–512. https://doi.org/10.1109/ICCA51439.2020.9264567

Kurniawan, B., Vamplew, P., Papasimeon, M., Dazeley, R., & Foale, C. (2020). *Discrete-to-Deep Supervised Policy Learning*. http://arxiv.org/abs/2005.02057

Li, C. (2019). Deep Reinforcement Learning. In *Reinforcement Learning for Cyber-Physical Systems*. https://doi.org/10.1201/9781351006620-6

Ma, X., Xia, L., & Zhao, Q. (2019). Air-Combat Strategy Using Deep Q-Learning. *Proceedings 2018 Chinese Automation Congress, CAC 2018*, 3952–3957. https://doi.org/10.1109/CAC.2018.8623434

Ma, Y., Ma, X., & Song, X. (2014). A case study on air combat decision using approximated dynamic programming. *Mathematical Problems in Engineering*, *2014*. https://doi.org/10.1155/2014/183401

Maxim Lapan. (2018). Deep Reinforcement Learning Hands-On 2nd Edition. In *arXiv*.

Mcgrew, J. S. (2008). *Real-Time Maneuvering Decisions for Autonomous Air Combat*.

McGrew, J. S., How, J. P., Williams, B., & Roy, N. (2010). Air-combat strategy using approximate dynamic programming. *Journal of Guidance, Control, and Dynamics*, *33*(5), 1641–1654. https://doi.org/10.2514/1.46815

Montague, P. R. (1999). Reinforcement Learning: An Introduction, by Sutton, R.S. and Barto, A.G. *Trends in Cognitive Sciences*. https://doi.org/10.1016/s1364-6613(99)01331-5

Qiu, X., Yao, Z., Tan, F., Zhu, Z., & Lu, J. G. (2020). One-to-one Air-combat Maneuver Strategy Based on Improved TD3 Algorithm. *Proceedings - 2020 Chinese Automation Congress, CAC 2020*. https://doi.org/10.1109/CAC51589.2020.9327310

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms*. 1–12. http://arxiv.org/abs/1707.06347

Sewak, M. (2019). Introduction to Reinforcement Learning. In *Deep Reinforcement Learning*. https://doi.org/10.1007/978-981-13-8285-7_1

Shyalika, C., Silva, T., & Karunananda, A. (2020). Reinforcement Learning in Dynamic Task Scheduling: A Review. *SN Computer Science*. https://doi.org/10.1007/s42979-020-00326-5

Sutton, R. S., Barto, A. G., & Book, A. B. (2018). *Reinforcement Learning, 2nd Ed.* https://libgen.is/book/index.php?md5=6FE0E48B8EE6D46EAB49814D846242E7

Wang, Z., Li, H., Wu, H., & Wu, Z. (2020). Improving Maneuver Strategy in Air Combat by Alternate Freeze Games with a Deep Reinforcement Learning Algorithm. *Mathematical Problems in Engineering*, *2020*. https://doi.org/10.1155/2020/7180639

Winberg, A., & Öhrstam Lindström, O. (2020). Reinforcement Learning Methods for OpenAI Environments (Dissertation). Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-293855

Yang, Q., Zhang, J., Shi, G., Hu, J., & Wu, Y. (2020). Maneuver Decision of UAV in Short-Range Air Combat Based on Deep Reinforcement Learning. *IEEE Access*. https://doi.org/10.1109/ACCESS.2019.2961426

Yang, Q., Zhu, Y., Zhang, J., Qiao, S., & Liu, J. (2019). UAV Air Combat Autonomous Maneuver Decision Based on DDPG Algorithm. *IEEE International Conference on Control and Automation, ICCA*, *2019-July*, 37–42. https://doi.org/10.1109/ICCA.2019.8899703

Zhang, X., Liu, G., Yang, C., & Wu, J. (2018). Research on air combat maneuver decision-making method based on reinforcement learning. *Electronics (Switzerland)*, *7*(11). https://doi.org/10.3390/electronics7110279

Zhang, Y., Zu, W., Gao, Y., & Chang, H. (2018). Research on autonomous maneuvering decision of UCAV based on deep reinforcement learning. *Proceedings of the 30th Chinese Control and Decision Conference, CCDC 2018*, 230–235. https://doi.org/10.1109/CCDC.2018.8407136

7. CURRICULUM VITAE

Berkay **Soyluoglu**

Esentepe Mahallesi, Kartal, Istanbul, 34870, Turkey.

soyluoglu16@itu.edu.tr | (+90) 534-674-7635 | Personal website | LinkedIn profile

Institutions

Bachelor of Science
Istanbul Technical University
Sep 2016 – July 2021 · Istanbul

Faculty of Aeronautics and Astronautics
Department of Aeronautics

Undergraduate Aeronautical Engineering Student

I'm currently studying aeronautical engineering as a senior undergraduate student with a GPA of 3.62 at Istanbul Technical University. My branch is control and avionics. I specialize in deep learning applications on aerial vehicles as in robotics.

Languages

Programming Languages

Numerical analysis	Python, MATLAB & Simulink, C/C++, MPI/OpenMPI, Octave, Fortran
Version control	Git, Fossil
Markup	LaTeX, Org, Markdown, HTML
Synchronization	rsync
Vector graphics language	Asymptote, Tikz/PGFPlots
Web development	HTML, CSS
Prototyping	Javascript, Typescript
Personal Finance	(h)ledger

Engineering Programs

Modelling	Blender
CAD	AutoCAD, CATIA
Video	Sony Vegas Pro, Hitfilm, Adobe Premiere, DaVinci Resolve
Image	Inkscape, GIMP

Natural Languages

English	**Level B**, Measured by the Language Proficiency Test administered in Turkey
English	**C1**, Measured by International Test of English Proficiency (ITEP)
Turkish	**First language**, I was born and raised in Tukey, hence can speak the language fluently

Experience

Internships

2021 **Turkish Aerospace Industries**, Model-based design (SIL and HIL) for FELIX (a fixed-wing aircraft)

Projects

2020 **Turkish Aerospace Industries**, Reinforcement learning development for a dogfight scenario between two jet fighters. Consultants: Uğur Çakın, Hasan İşçi

Unpublished Works

2020 **Deformation in Truss Systems**
Deformation length is compared for theoretic, simulative, and experimentative results.

2020 **Experiment of the Buckling of a Cylinder**
This article demonstrates the basics of buckling for cylindrical section.

2020 | **Experiment of the Deflection of a Beam**
Deformation characteristics of a beam is analysed in this experimental article.

2020 | **Homework Assignment | Mathematical Modelling of an RC/RLC Circuit**
Computational analysis for the response of an RC/RLC circuit.

2019 | **Computational Laminar Boundary Layer Analysis**
This work contains analysis via MATLAB for an inviscid flow past a flat plate.

2019 | **Charpy and Izod Experimentation**
Basic experiments for Charpy and Izod tests that I did as an university assignment.

2019 | **Computational Analysis of a Flow Around a Cylinder**
This is an experiment for determining the correlation between theoretical results and experimental data.

Certificates

- **Quality Awareness and Information Training**, 07-09 October 2020
- **Process Management Training**, 13-16 October 2020