Master's Thesis of Aerospace Engineering

Aerodynamic Shape Optimization of a Guided Missile Using Reinforcement Learning with a Neural Network Environment

신경망 환경에서의 강화학습을 이용한 유도탄의 공기역학적 형상 최적화

02 2023

Graduate School of Aerospace Engineering
Seoul National University
Aerospace Engineering Major

Rawdha Khalaf Abdulla Rahma Alhammadi

Aerodynamic Shape Optimization of a Guided Missile Using Reinforcement Learning with a Neural Network Environment

이복직

Submitting a master's thesis of engineering

12 2022

Graduate School of Aerospace Engineering
Seoul National University
Aerospace Engineering Major

Rawdha Khalaf Alhammadi

Confirming the master's thesis written by
Rawdha Khalaf Alhammadi
12 2022

Chair	이관중	(Seal)
Vice Chair	이복직	(Seal)
Examiner	김규홍	(Seal)

학위논문 원문제공 서비스에 대한 동의서

본인의 학위논문에 대하여 서울대학교가 아래와 같이 학위논문 제공하는 것에 동의합니다.

1. 동의사항

① 본인의 논문을 보존이나 인터넷 등을 통한 온라인 서비스 목적으로 복제할 경우 저작물의 내용을 변경하지 않는 범위 내에서의 복제를 허용합니다.

② 본인의 논문을 디지털화하여 인터넷 등 정보통신망을 통한 논문의 일부 또는 전부의 복제.배포 및 전송 시 무료로 제공하는 것에 동의합니다.

2. 개인(저작자)의 의무

본 논문의 저작권을 타인에게 양도하거나 또는 출판을 허락하는 등 동의 내용을 변경하고자 할 때는 소속대학(원)에 공개의 유보 또는 해지를 즉시 통보하겠습니다.

3. 서울대학교의 의무

① 서울대학교는 본 논문을 외부에 제공할 경우 저작권 보호장치(DRM)를 사용하여야 합니다.

② 서울대학교는 본 논문에 대한 공개의 유보나 해지 신청 시 즉시 처리해야 합니다.

논문제목 : Aerodynamic Shape Optimization of a Guided Missile Using Reinforcement Learning with a Neural Network Environment

학위구분 : **석사** ☑ · **박사** ☐

학 과 : 항공우주공학과

학 번 : 2021-21088

연 락 처 : +971553192121

저 작 자 : 알함마디 인)

제 출 일 : 200 23 년 01 월 30 일

서울대학교총장 귀하

Abstract

Rawdha Khalaf Abdulla Rahma Alhammadi

Aerospace Engineering Major

Graduate School of Aerospace Engineering

Seoul National University

This thesis optimizes the aerodynamic shape of a guided missile using deep deterministic policy gradient (DDPG). The optimization goal is to maximize the lift-drag ratio by adjusting the missile's fin geometry. The DDPG agent interacts with a neural network environment to predict the aerodynamic coefficient and hence calculate the reward for each configuration modification. The agent learns from the positive and negative rewards how to optimize the shape. The optimization follows some geometric constraints. The research studies the effect of treating the geometric constraints as soft constraints by including them in the reward function and allowing the agent to learn to avoid them compared to the commonly used method of treating the geometry constraints as hard constraints and checking them before running the optimization at each time step. Results have shown that DDPG is able to optimize the aerodynamic shape to achieve more than three times the baseline configuration. Treating the geometric constraints as soft optimizes the shape faster than the hard geometry constraints.

Keyword : Aerodynamic Shape Optimization, Reinforcement Learning, Artificial Intelligence, Deep Deterministic Policy Gradient, Objective Function, Hard and Soft Constraints

Student Number : 2021-21088

Table of Contents

List of Tables

List of Figures

Chapter 1. Introduction

1.1. Motivation

The aerodynamic optimization process takes place during the project's concept design and preliminary design stages. The aerodynamic coefficient of various design variables and flight conditions are calculated to determine the aerodynamic shape, which is considered numerically intensive. It is expensive and time-consuming to produce precise aerodynamic coefficients using computational fluid dynamics (CFD). While replacing CFD with a semi-empirical method such as Missile DATCOM (MD) saves time and money, it is constrained by the potential innovative configurations.

Using CFD or MD with a primitive aerodynamic shape optimization cycle that requires human input to plug in, test, and evaluate thousands of potential configurations is inefficient. Figure 1 illustrates the optimization model block, evaluation workflow block, and optimizer that make up the current aerodynamic design cycle described in [1]. The aerodynamic optimization of multiple design variables is accelerated further by using neural network methods instead of numerical simulations to calculate the aerodynamic coefficients.

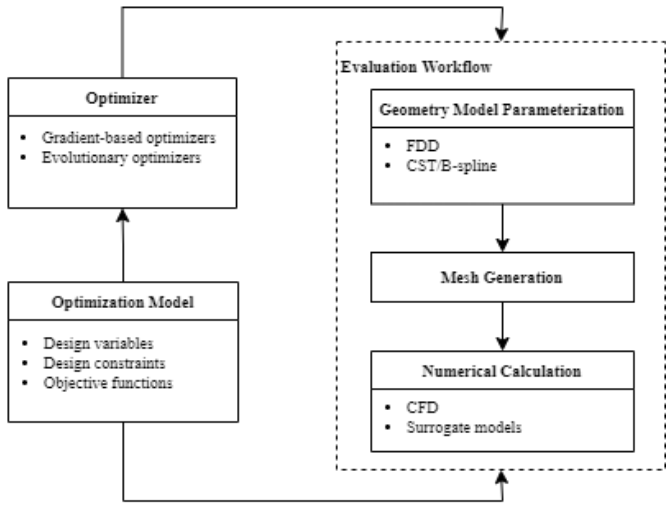

Figure 1: Aerodynamic optimization cycle

1.2. Aerodynamic Shape Optimization Cycle

The optimization model block, which is composed of the objective functions, design variables, and design constraints, is responsible of defining the optimization process. Aerodynamic coefficients are generated in the evaluation workflow block using numerical calculation, mesh generation, and geometric model parameterization. The typical optimizer uses evolution or gradient-based optimization. The finite difference and adjoint methods are examples of gradient-based optimization methods that converge on local optimum. For large design variable sets, the evolutionary method suffer from slow convergence rates and high dimensionality; examples of such methods include the multi-objective geometric algorithm (MOGA) and the multi-objective particle swarm optimization (MOPSO). [1]

Recent reinforcement learning (RL) developments provides a new optimizer method for aerodynamic optimization. By incorporating RL into the present aerodynamic design cycle, it is possible to decide which design

variable variations to apply to accomplish an aerodynamic improvement within the design space through learning from the experiences.

This study employs deep deterministic policy gradient (DDPG) as an RL technique for aerodynamic shape optimization. Instead of working directly with CFD or MD software, a neural network environment was built for DDPG optimization. The variable variation was limitless as long as it followed certain geometry constraints. This research is not meant to replace the precise CFD and semi-empirical approaches; rather, it is meant to illustrate the possibilities of using neural networks in creating an environment for the aerodynamic optimization that takes place in the concept design and early preliminary design stages.

1.3. Problem Statement

The goal of the aerodynamic fin optimization is to satisfy the geometrical and aerodynamic constraints and provide a geometry with the highest lift-drag ratio under cruising conditions. The optimization problem's objective function is to maximize the lift-drag ratio (CL/CD), which is a measure of aerodynamic performance. Optimization constraint over the geometry, static margin and drag coefficients (CD) are employed. The variable values should be limited by the geometry constraints so that they don't exceed the missile's length or cross over one another. The static margin, which measures the space between the missile's center of gravity and center of pressure, is responsible of the missile's stability. The XCG is the center of the missile's body, does not change by the fin optimization because the mass of the fins is negligible in comparison to that of the main body. Thus, the XCP controls the static stability. To avoid a decline in flight

performance, the drag coefficient is also limited to be less than the baseline configuration. The optimization model is described as: [1]

Flow condition:	Mach number of 2.0 Altitude of 5000 m Angle of attack ($ALPHA$) of 4 degrees
Objective Functions:	Maximize lift-drag ratio coefficient
Constraints:	Design parameters within geometric constraints XCP within [-0.605,-2.1] CD should be less than baseline value

Where XCP is measured from the center of gravity and divided by the missile's diameter.

The problem's design variables are presented in Table 1, and some of these variables visualized in Figure 2. The problem's design variables are presented in Table 1, and some of these variables visualized in Figure 2. The angles are in degrees, and the variable lengths are in centimeters. The variables are used to generate different configuration combinations to be executed in MD and produce training data for the neural network. The baseline model features a cone-shaped nose, a cylindrical body, trapezoid wings, and tails with a hexagonal airfoil. In this study, just the optimization of fin geometry and placement is taken into account. The parameters of the hexagonal airfoil and missile body are thus fixed. The fins' trailing edges are arranged in a "+" pattern and are perpendicular to the body axis. As a result, the wings and tails are right trapezoids, and each is controlled by four factors: the span length ($SSPAN\#\_2$), the root chord length ($CHORD\#\_1$), the tip chord length ($CHORD\#\_2$), and the location of the fin ($XLE\#$). Therefore, during the optimization process, a total of 8

independent design variables are adjusted. [1]

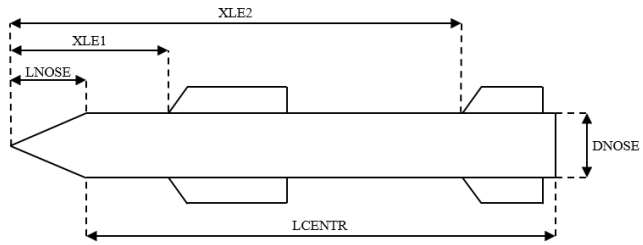

Figure 2: Missile design variables

Table 1: Design variables details

Design variable	Definition	Lower bound	Upper bound	Baseline
TNOSE	Nose type	-	-	Cone
LNOSE	Nose length	-	-	49
DNOSE	Nose diameter	-	-	18
LCENTR	Missile body length	-	-	320
DCENTR	Missile body diameter	-	-	=DNOSE
XLE1	Nose tip to wing leading-edge distance	125	175	172
SWEEP1	Wing trailing-edge sweep angle	-	-	0
CHORD1_1	Wing root chord	10	40	29
CHORD1_2	Wing tip chord	0	9	6
SSPAN1_1	Semi-span location of wing root chord	-	-	=DNOSE
SSPAN1_2	Semi-span location of wing tip chord	10	30	23
ZUPPER	Wing and tail thickness to chord ratio of upper surface	-	-	0.025
ZLOWER	Wing and tail thickness to chord ration of lower surface	-	-	=ZUPPER
LMAXU	Wing and tail fraction of chord from section leading-edge to maximum thickness of upper surface	-	-	0.25
LMAXL	Wing and tail section of the chord from leading-edge to maximum thickness of lower surface	-	-	=LMAXL
LFLATU	Wing and tail section of constant thickness section of upper surface	-	-	0.25
LFLATL	Wing and tail section of constant thickness section of lower surface	-	-	=LFLATU
XLE2	Nose tip to tail leading-edge distance	300	320	320
SWEEP2	Tail trailing-edge sweep angle	-	-	0
CHORD2_1	Tail root chord	10	40	38
CHORD2_2	Tail tip chord	0	25	19
SSPAN1_2	Semi-span location of tail root chord	-	-	=DNOSE
SSPAN2_2	Semi-span location of tail tip chord	10	30	22

Chapter 2. Artificial Neural Network

2.1. Introduction

Machines having brain-like abilities are referred to as artificial neural networks (ANNs), and they are able to model and predict complex nonlinear mechanisms without precise background knowledge [4]. We can utilize ANN to approximate the nonlinear relationship between the geometric shape and the aerodynamic coefficients because of this benefit [5]. Prediction accuracy is increased by training the ANN with a large amount of precise wind tunnel and flight test data.

It is crucial to use a surrogate model to approximate aerodynamic coefficients rather than CFD since the optimization of multiple objective problem is computationally expensive and slow. The surrogate model produces good results while reducing the optimization computation time. [5]

In this study, the lift, drag, lift-drag, and center of pressure coefficients are predicted using a surrogate model based on ANN. A full model is used to do the prediction because it gives accurate results [6]. The surrogate model is incorporated into the environment during the DDPG training process. It provides the predicted aerodynamic coefficients and uses them to calculate the reward for a specific state and action. [5]

2.2. Model Building

Figure 5 shows the structure of the neural network. The aerodynamic coefficients (CL, CD, CL/CD, and XCP) are the outputs, and the 8 independent geometrical variables are the inputs. The network consists of 7 layers, 5 of which are hidden layers. The number of input and output layer nodes corresponds to the number of input and output variables. There are 64, 32, 16, 8, and 4 nodes in each of the hidden layers, respectively. To minimize the possibility that large values would dominate the training, the generated data using MD is scaled to [0, 1] before starting the training. The data underwent some preprocessing in order to reduce the impact of outliers on the prediction [6]. After extensive studies on several activation functions, the sigmoid function, which has the expression: $\sigma(x) = \frac{1}{1+e^{-x}}$, was found to be the best accurate activation function for our model. The data is divided into a training set (655238), a validation set (163810), and a testing set (1642). Adam is the network optimizer, and equation (1) states that the learning rate (lr) exhibits exponential decay.

$$lr = lr_i \exp(-dr * epoch) \quad (1)$$

Where lr_i is the initial learning rate and dr is the decay rate.

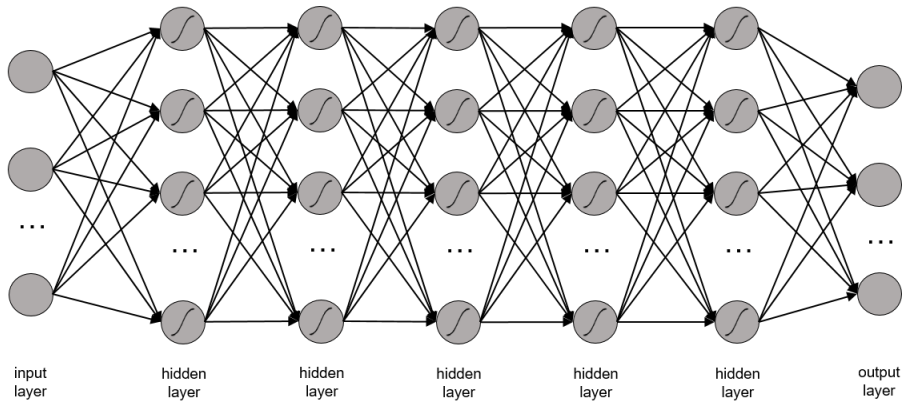

Figure 3: Artificial neural network structure

2.3. Results

How well ANNs predict the expected outcome could be evaluated using some loss metrics, such as the mean square error (MSE). During the training and validation phases, MSE is calculated and shown in Figure 4. The MSE for training and validation decreases across numerous epochs until it reaches a value of roughly 10^{-5}, indicating that the model is accurate and reliable.

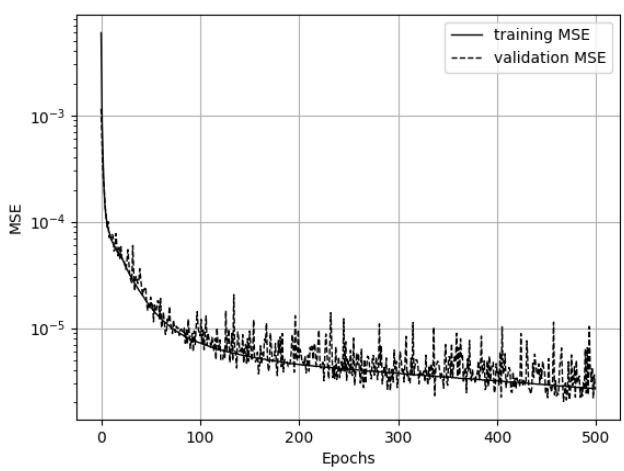

Figure 4: Mean square error vs. epochs

The test data's prediction accuracy can be shown using scatter plots.

Although the projected CL values and the actual value often agreed, values over 8 indicated a slight divergence, as can be seen in Figure 5a. There is a minor difference at the greatest angle of attacks between CD's predicted values and their actual values, as seen in Figure 5b. For CL/CD and XCP, Figures 5c and 5d, respectively, yield precise predictions.

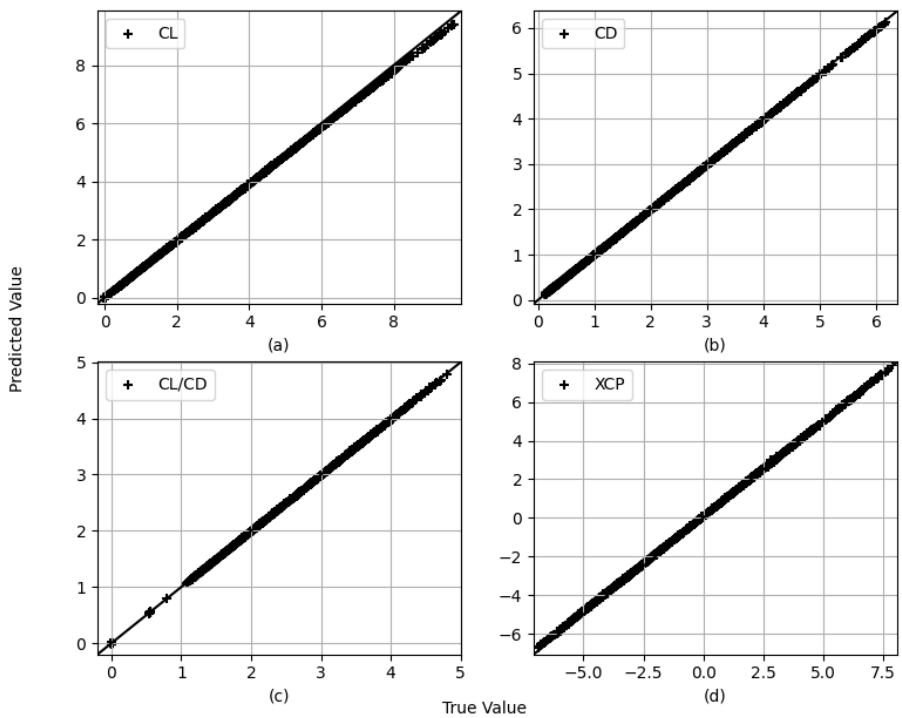

Figure 5: Coefficient Prediction accuracy

Figure 6 compares the ANN predictions and the MD values for the aerodynamic coefficient. The setup parameters for this study are as follows: $XLE1 = 130$, $CHORD1\_1 = 12$, $CHORD1\_2 = 6$, $SSPAN1\_2 = 22$, $XLE2 = 300$, $CHORD2\_1 = 27$, $CHORD2\_2 = 19$, and $SSPAN2\_2 = 22$. Figure 6a shows that CL predicted values are accurate at alpha numbers less than 32. At higher alpha values, the predicted values do, however, marginally deviate. As can be shown in Figs. 6b and 6c, respectively, CD and CL/CD predictions accurately agree with MD values.

Although they are not an exact match, the ANN XCP predictions in Fig. 6d are quite close to the MD values. Since the observed deviation from MD values is small and appropriate for our goals, it is deemed as insignificant. This could be mitigated by using a different neural network model to predict XCP values.

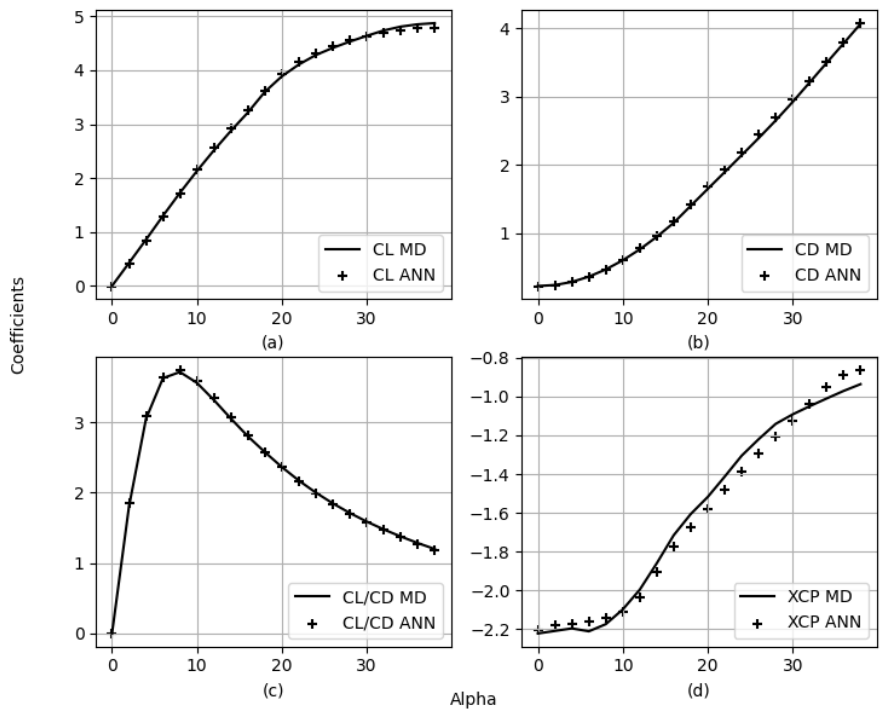

Figure 6: Predicted (a) CL, (b) CD, (c) CL/CD, and (d) XCP vs. $alpha$

Chapter 3. Reinforcement Learning

3.1. Introduction

A decision-maker is referred to as an agent in RL. Anything that an agent interacts with that is external to it is considered the environment. In order to learn, the agent constantly engages with the environment through a series of time steps. At each time step t, the agent starts out in the state $s_t \in S$, where S is the set of potential states. It then interacts with its environment by taking the action $a_t \in A$, where A is the set of available actions at state s_t. In response, the environment gives a new state $s_{t+1} \in S$ and a reward $r_{t+1} \in R$, where R is the set of rewards. [3]

The agent maps the states to the probabilities of choosing each of the possible actions at each time step. This mapping is known as the agent's policy π_t, where $\pi_t(s,a)$ is a stochastic policy's probability of taking $a_t = a$ if $s_t = s$. The agent's objective function $J = \mathrm{E}[R_t]$ is to maximize expected discounted return, which is the total discounted future reward and is expressed in the equation (2). [7, 8]

$$R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i) \quad (2)$$

Where T is the terminal time-step and $\gamma \in [0,1]$ is the discount rate.

3.2. Q-Learning

Q-learning is a model-free value-based off-policy temporal difference algorithm, which is an example of RL algorithms. Model-free refers to the absence of an environment model, whereas off-policy describes the agent's use of behavior policy μ to choose what actions to take in a given state to discover policy π. The Bellman equation is used to describe the Q values with a deterministic target policy $\mu : S \leftarrow A$ as shown in equation (3). Equation (4) state that Q-learning employs greedy policy $\mu(s) = \arg\max_a Q(s,a)$. [10]

$$Q^{\mu}(s_t, a_t) = \mathrm{E}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))] \qquad (3)$$

Q values are initially assigned to a low random value in Figure 7, and they are then adjusted as the episodes proceed. The agent observes its current state s_t at the beginning of an episode, chooses an action a_t from μ and executes it, observes the new state s_{t+1}, receives an immediate reward r_{t+1}, and then uses π to choose the action for the new state to roughly approximate the Q value according to equation (4). This sequence is repeated until termination. As seen in Figure 7, estimates of Q value are arranged in a look-up table, where the agent learns both the value function and the policy. [7, 8]

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \qquad (4)$$

Q-learning learning objective is to find the optimal policy π that that maximizes the total expected discounted reward. The problem with Q-learning is that it can only be used for discrete, low dimensional state and

action spaces since at each iteration, the Q value is adjusted by looking up the table. [9, 10]

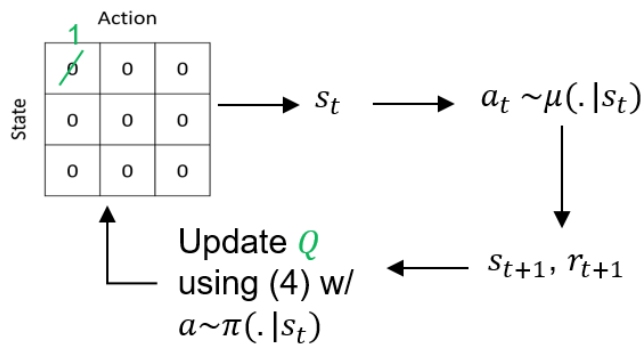

Figure 7: Q-learning algorithm

3.3. Deep Q-Learning

In order to extract features from input in a high dimensional state space, deep reinforcement learning (DRL) combines the capabilities of deep learning (DL) and the RL algorithm. In order to construct deep Q-learning (DQN), the lookup table is replaced by a neural network that is used as a function approximator of the action-value function, $Q(s,a;\theta) \approx Q^*(s,a)$ with parameter θ^Q. Additionally, DQN combines the stochastic minibatch updates with a replay buffer to ensure the least possible correlation between samples. In order to sample certain experiences for learning, the replay buffer is employed. It is made up of experience tuples $(s_t, a_t, r_{t+1}, s_{t+1})$ that gradually grow in size as the agent interacts with the environment. The oldest experience is discarded once the buffer reaches a predetermined size to create room for the new ones. Moreover, DQN employs a target network that is a replica of our function approximator neural network and serves as a fixed target during temporal-

difference backup. DQN seeks to minimize the loss described in equation (5) in order to optimize the parameter Q. DQN is currently limited to cases with discrete actions and low-dimensional state and action spaces since obtaining the action of the maximum action-value function for continuous domains requires an iterative optimization process. [6, 10]

$$L(\theta^Q) = \mathrm{E}[(Q(s_t, a_t \mid \theta^Q) - y_t)^2] \quad (5)$$

Where the target y_t :

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1}) \mid \theta^Q) \quad (6)$$

3.4. Deterministic Policy Gradient

Many model-free RL algorithms are based on policy iteration that consists of policy evaluation and policy improvement. Policy evaluation uses temporal-difference learning to estimates the action-value function. Policy improvement methods, such as greedy policy $\mu^{k+1}(s) = \arg\max_a Q^{\mu^k}(s, a)$, update the policy with respect to the estimated action-value function. Greedy policy requires global maximization at every step, which is problematic. Therefore, instead of using greedy policy, the policy is moved in the direction of the gradient of Q and that is known as the deterministic policy gradient (DPG). Specifically, the policy parameters θ^{k+1} are updated in the direction of the gradient $\nabla_\theta Q^{\mu^k}(s, \mu_\theta(s))$ as expressed in equation (7). And by applying the chain rule, the gradient in equation (7) is decomposed to gradient of the policy with respect to the policy parameter and gradient of action-value with respect to the actions as shown in equation (8). [3, 5]

$$\theta^{k+1} = \theta^k + \alpha \mathrm{E}[\nabla_\theta Q^{\mu^k}(s, \mu_\theta(s))] \quad (7)$$

$$\theta^{k+1} = \theta^k + \alpha \mathrm{E}[\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu^k}(s, a)|_{a=\mu_\theta(s)}] \quad (8)$$

DPG is ideal for continuous action problems with high dimensions. DPG is policy-based, unlike Q-learning and DQN which are value-based. It uses gradient-based methods to optimize deterministic policies as neural network parameters. Combining DQN with DPG, deep deterministic policy gradient (DDPG) can be used for high-dimensional continuous actions which is suitable for our problem. [3, 7]

Chapter 4. Deep Deterministic Policy Gradient

4.1. Introduction

DQN and DPG theorem provide the foundation of the DDPG algorithm. Similarly to DQN, DDPG is an actor-critical algorithm that employs replay buffers and target networks. The DPG algorithm parameterizes the actor function $\mu(s \mid \theta^{\mu})$, which defines the current policy that deterministically maps state to an action. The critic judges the action chosen by the actor, its function $Q(s, a \mid \theta^{Q})$ known as the action-value function is learned through the Bellman equation just like in Q-learning. Neural networks with parameters θ^{μ} and θ^{Q}, respectively, represent both the actor and critic functions. By reducing the loss described in equation (5), the parameter Q is optimized (5). The actor's parameter θ^{μ} is updated using the objective function gradient $\nabla_{\theta^{\mu}} J$ in equation (9). [3, 5]

$$\nabla_{\theta^{\mu}} J = [\nabla_{a} Q(s, a \mid \theta^{Q}) |_{s=s_t, a=\mu(s_t)} \nabla_{\theta^{\mu}} \mu(s, \theta^{\mu}) |_{s=s_t}] \quad (9)$$

The tuples ($s_t, a_t, r_{t+1}, s_{t+1}$) are the transitions samples from the environment while following an exploration policy, are stored in the replay buffer, which is a finite sized cache. The oldest sample are removed once the replay buffer is full. At each time step, the actor and critic are updated by sampling a minibatch from the buffer. The replay buffer can be large because of the DDPG's off-policy property, which permits learning across a set of uncorrelated transitions. [5]

The divergence problem of Q-learning is due to updating $Q(s, a \mid \theta^{Q})$ network that is also used in the target value equation

(6). By using soft target updates rather than an exact copy of the network weights, DDPG addresses this problem. Calculating the target values involves making a duplicate of the actor and critic networks. The learning is more stable because the weights of the neural networks are updated gradually toward the learned networks' parameters: $\theta' \leftarrow \tau\theta - (1-\tau)\theta'$ where $\tau << 1$. [3, 5]

When the observation value varies across the environment, the network is unable to learn effectively. To address this, DDPG uses batch normalization. Using this technique, each dimension is normalized across all samples in a minibatch to have a unit mean and variance. It also maintains a running average of the mean and variance for testing normalization. [10]

In order to account for exploration in the continuous action space, an exploration policy is created. Equation (10) shows how this policy integrates noise from a noise process N into our actor policy. [10]

$$\mu'(s_t) = \mu(s_t \mid \theta_t^{\mu}) + \mathrm{N}_t \quad (10)$$

Where N_t is an Ornstein-Uhlenbeck process used for effective exploration.

4.2. Algorithm

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s,a \mid \theta^Q)$ with weight θ^Q and actor $\mu(s \mid \theta^\mu)$ with weight θ^μ

Initialize target networks Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, and $\theta^{\mu'} \leftarrow \theta^\mu$ respectively

Initialize replay buffer R

for episode=1, M **do**

Initialize a random process N for exploration

Observe initial observation state s_t

for t=1, T **do**

Select action a_t using equation 10 according to the current policy $\mu(s_t \mid \theta_t^\mu)$ and exploration noise N

Clip the action $a = clip(a_t, a_{Low}, a_{High})$ where a_{Low} and a_{High} are defined in the environment

Execute a_t in the environment and observe reward r_t and new state s_{t+1}

Store (s_t, a_t, r_t, s_{t+1}) in R

Sample a random minibatch of N transitions (s_t, a_t, r_t, s_{t+1}) from R

Set targets as $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} \mid \theta^{\mu'}) \mid \theta^{Q'})$

Update the critic by gradient descent to minimize the loss: $L = \frac{1}{N}\sum_i (Q(s_i, a_i \mid \theta^Q) - y_i)^2$

Update the actor policy by gradient ascent: $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a \mid \theta^Q)\mid_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s \mid \theta^\mu)\mid_{s_i}$

Update the target networks: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$ and $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$

end

end

Chapter 5. Methodology

Starting from a baseline design as shown in Table 1, the optimal aerodynamic configuration is learned using DDPG. In equation (11), the agent chooses an action at a specific state based on a deterministic policy. [1]

$$a_t = \mu(s_t \mid \theta_t^{\mu}) \qquad (11)$$

Where s_t is the current configuration that is made up of the 8 independent design variables and a_t is the value added to s_t resulting in the new configuration s_{t+1}. The action a_t has positive and negative values, and it has different values for each design variable represented in state s_t.

As previously discussed, the training goal is the find the optimal network parameters under a specific reward function. Reward function design is crucial for adequate, stable learning. According to equation (12), the reward function rewards lift-drag ratio increases after each interaction with the environment while penalizing excessive XCP and CD fluctuations. [1]

$$r(s_t, a_t) = \begin{cases} 10.(f(s_t) - f(s_0)), \text{ if } XCP \in [-0.605, -2.1] \ \& \ CD < \text{baseline} \\ f(s_t) - f(s_0) - 10.\Delta XCP - 10.\Delta CD, \text{ otherwise} \end{cases} \qquad (12)$$

Where f refers to the lift-drag ratio value predicted using the ANN, s_0 is the baseline design configuration that we aim to optimize and s_t is the current design configuration. The ΔXCP and ΔCD are the XCP and CD variation from the baseline. This reward is used in the sum of discounted future reward equation (2).

There are different types of optimization constraints, some influence

the objective function known as soft constraints where they are included in the reward function. The other type of constraint is known as hard constraint, and it ensures that all optimization solutions are meeting the hard constraint, hence they are checked before doing the optimization. An example of the different constraint types are: 1) the XCP and CD constraints discussed earlier in the problem statement section are soft constraints and they are included in the reward function in Equation (12), 2) the geometry constraints are commonly considered hard constraints. In this research we observe the effect of treating the geometry constraints as soft constraints by including them in the reward function.

5.1. Geometry Constraints Study 1

The first study constraining method treats the geometry constraints as soft constraints by including them in the reward function and the DDPG optimization is carried on regardless if the geometry constraints are met or not as shown in Figure 8. The DDPG agent is expected to learn that defying the geometry constraints is not desirable and receives negative rewards for that, and therefore the agent would avoid the actions that lead to these constraints. This method will be referred to as “method 1”

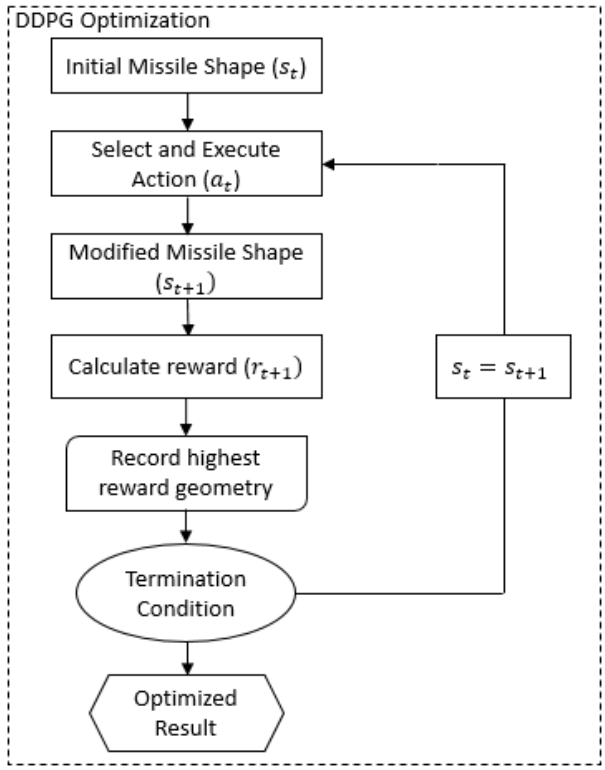

Figure 8: Method 1 of applying geometry constraints (Soft)

After understanding the method of constraining used in this study, we will focus now on two types of geometry constraints. The first type called "constraint 1" limits the 8 independent variables with upper and lower bounds as found in Table 1. The reward function is modified to reflect the variables limitations as expressed in equation (13).

$$r(s_t, a_t) = \begin{cases} 10.(\text{largest negative difference from the bounds}) \\ 10.(f(s_t) - f(s_0) \text{ if } XCP \in [-0.605, -2.1] \text{ and } CD < CD_0 \\ f(s_t) - f(s_0) - 10.\Delta XCP - 10.\Delta CD \text{ otherwise} \end{cases} \quad (13)$$

The second constraint type called "constraint 2" is looser, therefore it is able to explore values beyond the defined bounds of the variables. The second constraint tries to maintain a sensible geometrical combinations conditions such as: 1) positive geometry variables, 2) $XLE1 > LNOSE$, 3) $XLE1 + CHORD1\_1 < XLE2$, 4) $SSPAN1\_2 \leq 30$, 5) $SSPAN2\_2 \leq 30$

and 6) $XLE3+CHORD2\_1<LNOSE+LCENTR$. The second geometry constraint reward function is modified to include the geometric conditions as expressed in equation (14).

$$r(s_t,a_t)=\begin{cases}\text{Largest negative variable if cond. 1)}\\ LCENTR-XLE1 \text{ if cond. 2)}\\ XLE2-(XLE1+CHORD1\_1) \text{ if cond. 3)}\\ 10.(30-SSPAN1\_2) \text{ if cond. 4)}\\ 10.(30-SSPAN2\_2) \text{ if cond. 5)}\\ 10.(LNOSE+LCENTR-(XLE2+CHORD2\_1)) \text{ if cond. 6)}\\ \text{other.,}\begin{cases}10(f(s_t)-f(s_0))\text{if } XCP\in[-0.605,-2.1]\&CD<CD_0\\ f(s_t)-f(s_0)-10.\Delta XCP-10.\Delta CD \text{ otherwise}\end{cases}\end{cases} \quad (14)$$

Where CD_0 is the baseline configuration drag coefficient.

5.2. Geometry Constraints Study 2

The second study's compares two different constraining methods with fixed geometry constraints type. The common method shown in Figure 9 treats the geometry constraints as hard constraints and checks them before running the DDPG optimization, this method is called "method 2". This method is compared with the method 1 from the geometry constraints study 1 shown in Figure 8, where the constraints are soft constraints and the DDPG optimization is always running. The two methods being compared in this study uses constraint 2, which is a looser geometry constraints expressed in equation (14). Method 2 reward function is given by equation (12).

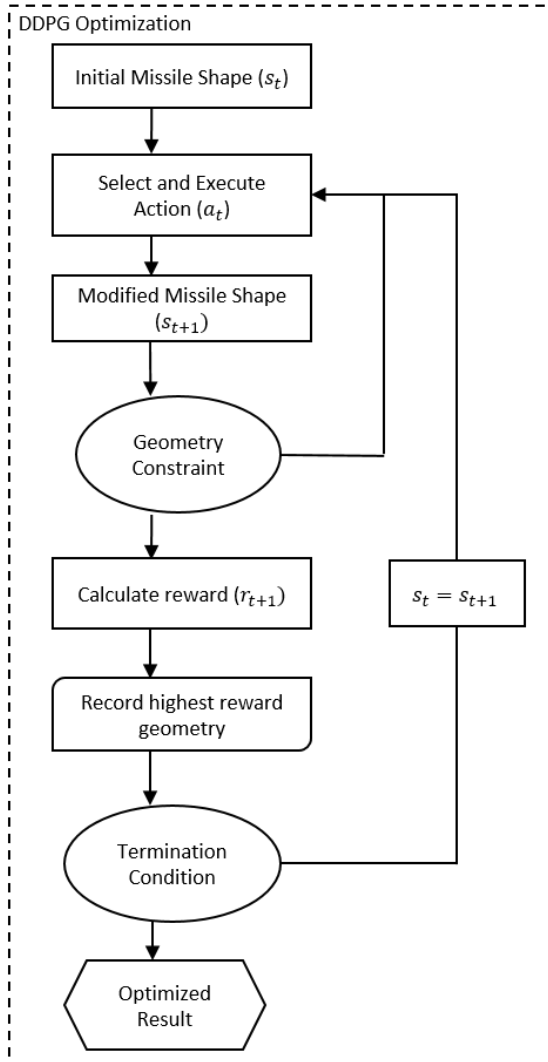

Figure 9: Method 2 of applying geometry constraints (Hard)

Chapter 6. Numerical Setup

The network is based on TensorFlow and the environment is custom built using OpenAI Gym with ANN to calculate the rewards. The learning rates for the actor and critic are 0.0001 and 0.001, respectively, whereas the reward discount rate is 0.99. The actor's and critic's network are composed of three layers with 256 nodes each. The activation function of the hidden layers is $\sigma(X_i) = \max(0, X_i)$, the critic's output layer's activation function is linear, and the actor's output layer's activation function is $\psi(X_i) = \sinh(X_i) / \cosh(X_i)$. The size of the replay buffer is 50000, and the minibatch size is 64. The maximum number of episodes was 500, and each episode terminates after 20 steps. Different action value range is used for different studies. At each step in the episode, the action is restricted to the range [-0.1, 0.1].

Chapter 7. Results and Discussion

7.1. Geometry Constraints Study 1

In this first study, the DDPG is always running and learns to avoid the geometric constraints by receiving negative rewards for them. The reward function of constraint 1 if given by equation (13) and it limits the variables to upper and lower bounds. The results of this geometry constraint is shown in Figure 10. A good configuration was recorded in less than 100 episodes, and this indicates that DDPG is able to rapidly optimize the aerodynamic configuration. The loss described in equation (5) steeps at the same region the reward escalates.

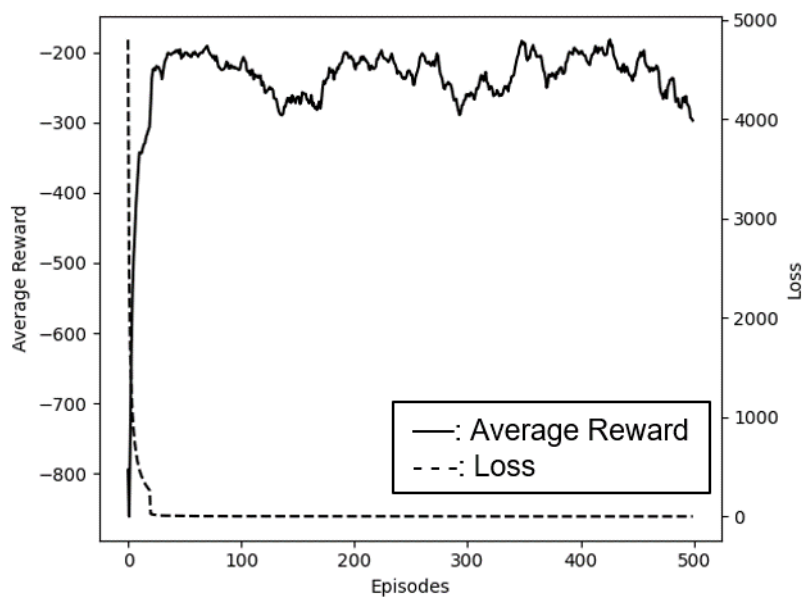

Figure 10: Method 1, constraint 1 average reward and loss vs. episodes

Figure 11 displays the reward and loss curves of constraint 2 that is expressed in equation (14). The average reward increases and the loss

decreases to reach an optimal configuration in less than 100 episodes as well.

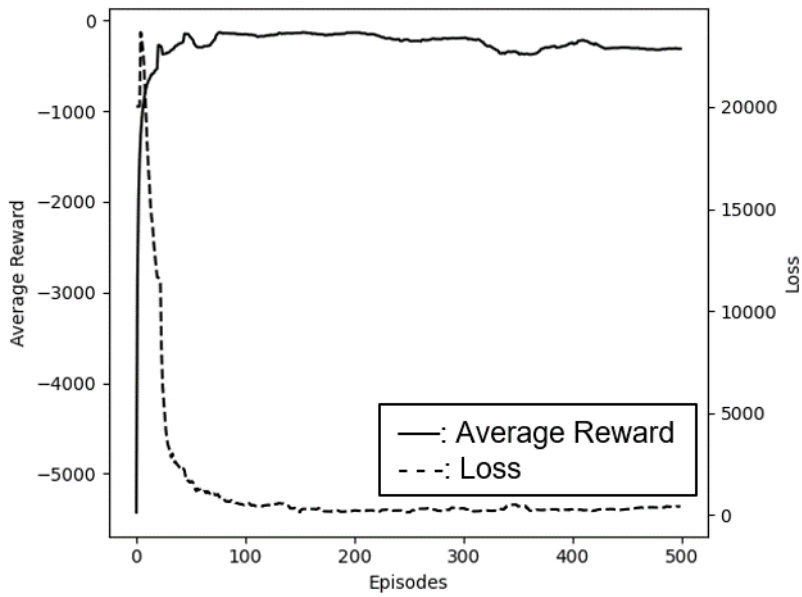

Figure 11: Method 1, constraint 2 average reward and loss vs. episodes

Table 2 compares the two aerodynamic configuration proposed by the two different geometry constraint types. Both constraint types increased the lift-drag ratio and decreased the drag coefficient. Constraint 2 is better than the limited constraint 1 because it increased the CL/CD 5% more, decreased the CD 16% less and found a configuration with XCP within the specified bound of [-0.605,-2.1]. Constraint 1 struggles to find a configuration that follows the XCP limitation because of its tight bounds, and this showcases the limitation of treating the geometry constraint as soft.

Table 2: Comparison table of geometry study 1

Variables \ Constraints	Constraint 1	Constraint 2	Baseline
$XLE1$	167	157	172
$CHORD1\_1$	26.5998	20	29
$CHORD1\_2$	5.1	3.3	6
$SSPAN1\_2$	25	29	23
$XLE2$	318.4098	314	320
$CHORD2\_1$	35	29	38
$CHORD2\_2$	21.5	26.5	19
$SSPAN2\_2$	20.850	16	22
CL/CD	3.794	4.003	1.051
CD	0.339	0.284	3.132
XCP	-3.311	-2.04	-2.422

7.2 Geometry Constraints Study 2

Two different constraining methods are compared here. The two methods use the same constraint which is constraint 2, since it gave the desired results. Method 1 is already observed in the geometry constraints study 1, and it is shown in Figure 11. Method 2 treats the geometry constraints as hard constraints and always checks them before DDPG optimization. The result of method 2 is shown in Figure 12.

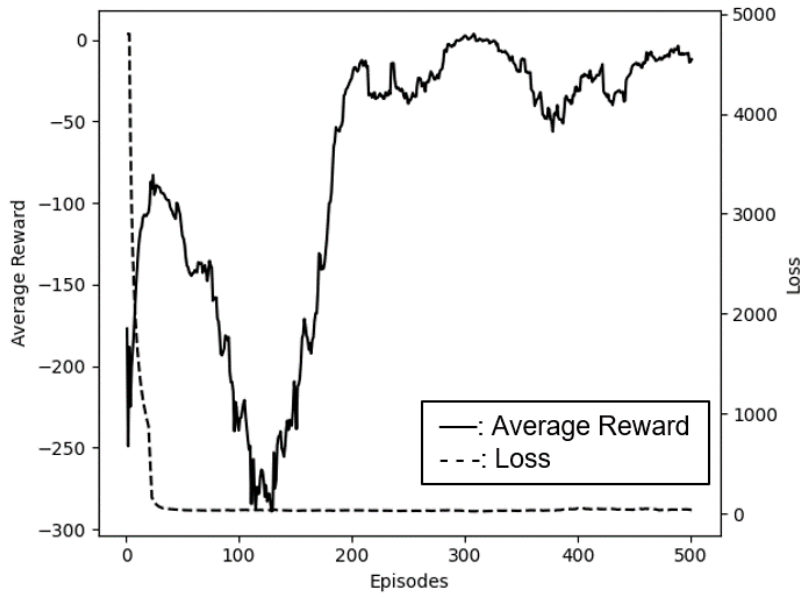

Figure 12: Method 2, constraint 2 average reward and loss vs. epochs

Figure 12 shows that the average reward increases until 40 episodes and then drops, this is due to the agent exploration because it discovered new states that gave negative rewards. After the agent has observed these negative reward states, it learns to avoid them, which is why the average reward increases back around 120 episodes to reach a higher value than the first increase.

Method 1 and method 2 both increased the average reward and decreased the loss. The proposed method in this thesis, method 1, finds the optimized aerodynamic configuration in 100 episodes whereas the common constraining method, method 2, finds it in 300 episodes. The state space of method 1 is larger since it considers states that defy the geometric constraints, which is why its average reward stabilizes around -200. Method 2's state space is smaller because the geometric constraints must be met before running the optimization, and this is reflected on the average reward value as it stabilizes around -50.

The aerodynamic configuration of method 1 and method 2 look exactly the same as shown in Table 3, except the wing's root chord length. Method 1 increased the lift-drag ratio by 280% of the baseline configuration, and method 2 increased it by 309%.

Table 3: Comparison table of geometry study 2

Variables \ Constraints	Method 1	Method 2	Baseline
$XLE1$	157	157	172
$CHORD1\_1$	20	38	29
$CHORD1\_2$	3.3	3.3	6
$SSPAN1\_2$	29	29	23
$XLE2$	314	314	320
$CHORD2\_1$	29	29	38
$CHORD2\_2$	26.5	26.5	19
$SSPAN2\_2$	16	16	22
CL/CD	4.003	4.305	1.051
CD	0.284	0.325	3.132
XCP	-2.04	-1.954	-2.422

Chapter 8. Conclusion

In conclusion, the aerodynamic shape optimization is essential for the concept design stage to minimize future changes and hence reduce the cost. The advancement in the artificial intelligence and especially reinforcement learning have introduced aerodynamic shape designers to a new method for optimization. Using DDPG for optimization is useful in situations where continuous state and action are necessary. DDPG has optimized the aerodynamic shape for higher lift-drag ratio and increased it to 3 times its baseline configuration value. In this paper, method 1 proved its rapid optimization capability especially when the geometry constraints are not too limited. The common constraining method, method 2, has higher chance of finding the global optimal in case of a large state space.

Bibliography

[1] Yan X, Zhu J, Kuang M, Wang X (2019) Aerodynamic shape optimization using a novel optimizer based on machine learning techniques. Aerospace Science and Technology: 826-835. https://doi.org/10.1016/j.ast.2019.02.003

[2] Chang K (2015) Chaptere 4-Structural Design Sensitivity Analysis. In: Design theory and methods using CAD/CAE. Elsevier Inc, pp 213

[3] Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M (2014) Deterministic Policy Gradient Algorithms. Proceedings of the 31st International Conference on Machine Learning: 387-395

[4] Gurney K (1997) An Introduction to Neural Network. CRC Press

[5] Qin S, Wang S, Wang L, Wang C, Sun G, Zhong Y (2020) Multi-objective optimization of Cascade Blade profile based on reinforcement learning. Applied Sciences: 1-27. https://doi.org/10.3390/app11010106

[6] Ritz S, Hartfield R, Dehlen J, Burkhalter J, Woltosz W (2015) Rapid calculation of missile aerodynamic coefficients using artificial neural networks. IEEE Aerospace Conference: 1-19. https://doi.org/10.1109/AERO.2015.7119031

[7] Sutton S, Wang L, Sun G, Zhong Y (2018) Reinforcement learning: An introduction. MIT Press Ltd, Massachusetts

[8] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antogolou I, Wiersta D, Riedmiller M (2013) Playing Atari with Deep Reinforcement Learning. DeepMind Technologies: 1-9. https://doi.org/10.48550/arXiv.1312.5602

[9] Watkins J, Dayan P (1992) Q-learning. Machine Learning 8: 279-292. https://doi.org/10.1007/BF00992698

[10] Lillicrap T, Hunt J, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2016) Continuous Control With Deep Reinforcement Learning. International Conference on Learning Representations: 1-14. https://doi.org/10.48550/arXiv.1509.02971

초 록

이 논문는 심층 결정론적 정책 그레이디언트(DDPG)를 사용하여 유도 미사일의 공기역학적 형태를 최적화에 관한 연구입니다. 최적화 목표는 미사일의 핀 형상을 조정하여 리프트-드래그 비율을 최대화하는 것입니다. DDPG 에이전트는 신경망 환경과 상호 작용하여 공력 계수를 예측하여 미사일 형상 수정에 대한 보상을 합니다. 에이전트는 긍정적인 보상과 부정적인 보상으로부터 형상을 최적화하는 방법을 배울니다. 이 최적화에는 몇 가지 기하학적 제약이 있고, 이 연구는 기하학적 제약 조건을 보상 함수에 포함하여 소프트 제약 (soft constraints)으로 처리하고, 또한 이 연구에서는 기하학 제약을 하드 제약 (hard constraints)으로 취급하여 각 시간 단계에서 최적화를 수행하기 전에 기하학 제약을 점검하는 통상적으로 사용되는 방법과 비교하여 DDPG 에이전트가 소프트 제약을 피하는 방법을 배울 수 있습니다. 이 연구 결과는 DDPG가 공력 형상을 기본 구성의 3배 이상을 달성하기 위해 최적화 되었음을 확인하였다. 본 논문은 기하학적 제약을 소프트 제약 (soft constraints)으로 처리하여 하드 제약 (hard constraints)보다 빠르게 미사일 형상을 최적화할 수 있음을 보여줍니다.

주요어 : 공기역학적 형상 최적화, 강화 학습, 인공지능, 심층 결정론적 정책 그레이디언트, 목적 함수, 하드 및 소프트 제약

학 번 : 2021-21088