Extended Abstract

Motivation In this study we tackle the problem of designing optimal ‘airfoils’ which are fundamental shapes that underpin aircraft wing design. A measure of an airfoil’s quality is its lift-to-drag ratio, which is independent of size and only depends on its shape. In this study, we work towards reaching the optimal shape that achieves the highest lift-to-drag ratio airfoil. In particular, we employ a Deep RL approach to learn policies to modify shapes sequentially. Also, we utilize NIGnets for our geometry representation which give us a hard guarantee on representing only non-self-intersecting geometry. This helps us solve the two major issues created by self-intersecting geometry in RL based optimization: design space blow-up and divergence in optimization.

Method We formulate aerodynamic shape optimization as a Deep RL problem. To tackle the problems caused by self-intersection produced during optimization we use NIGnets, a new neural architecture that we developed that gives a hard guarantee on non-self-intersecting geometry representation. A NIGnet as a whole with a particular setting of its weights and biases (denoted by ϕ) represents a particular shape. Our shape optimization approach can be summarized as follows:

1. Represent state (shape) using NIGnet parameters ϕ.
2. An action corresponds to outputting $\Delta\phi$ that perturbs the NIGnet parameters to $\phi' = \phi + \Delta\phi$.
3. A reward is produced at each step that corresponds to the $\frac{L}{D}$ ratio of the current shape.

Since the NIGnet for any set of parameters ϕ always represents non-self-intersecting geometry, it will constrain the design space to only the set of physically reasonable shapes and therefore allow an RL agent to perform ambitious shape optimization. We also utilize Hindsight Experience Replay (HER) to train more general policies that can design for any target lift-to-drag ratio.

Implementation We instantiate the RL problem as a problem with a fixed time horizon T. Where at each stage the agent can take an action on each network parameter value in the clipped interval $[-\sigma, \sigma]$ where we fix $\sigma = 0.01$. We experiment with $T = 5, 15$ and total training steps. We use PPO as our DeepRL algorithm with an MLP policy of 2 layers with a hidden dimension of 64. We use the stablebaselines3 Raffin et al. (2021) library’s implementation of PPO to run our experiments. For our Environment we use XFoil to compute the lift-to-drag ratio. In cases when the simulation does not converge we use a negative reward of -50. We also perform experiments with similar hyperparameters to train general policies that design for specified lift-to-drag using HER.

Results We observe intuitively interesting results that produce shapes with high lift-to-drag ratios. In particular we observe that for short time horizon $T = 5$ and smaller number of training steps an increase in L/D comes mainly from an increase in angle of attack with barely any shape change. Increasing only the time horizon $T = 15$ leads to increase in angle of attack till stagnation and some shape morphing. Further increasing the total training epochs leads to significant shape morphing with a high angle of attack. We are able to produce shapes with lift-to-drag ratios as high as 61.64!

Discussion We find that initial experiments with Deep RL for shape optimization give interesting results. One major limitation of our pipeline is the CFD solver, XFoil. XFoil is extremely fast but has major downsides such as: Non-convergence and lack of physics information. Using a more high-fidelity RANS solver will theoretically provide us more physics information and also alleviate the non-convergence problem. Another limitation is the size of the NIGnet. To move towards larger NIGnets we will have to warm start our reward model. Thus supervised pretraining of a reward model will be crucial in scaling up design optimization using Deep RL.

Conclusion We find that Deep RL based shape optimization works really well when the agent can perform unconstrained shape optimization over the geometry parameters without running into self-intersection related issues. NIGnets not only allow a hard guarantee on providing such a geometry representation but also morph fast and stably enough to work well even with small horizon tasks. This approach to shape optimization is exponentially more scalable than past approaches that utilized splines. The exponential benefit comes from the fact that larger number of shape parameters can lead to exponentially larger number of possible self-intersecting shapes. Directions for further research include comparison to other non-self-intersecting geometry representations like Neural ODEs.

Robust Aerodynamic Shape Optimization on NIGnets using Deep Reinforcement Learning

Atharva Aalok
Department of Aeronautics and Astronautics
Stanford University
`atharva1@stanford.edu`

Abstract

We present a Deep Reinforcement Learning framework for aerodynamic airfoil design that directly maximizes lift-to-drag (L/D) ratio while circumventing the challenges created by self-intersecting geometries. Shapes are parameterized with NIGnets, a neural architecture that guarantees non-self-intersecting geometry representation for any set of network parameters, thereby constraining the search space to physically admissible airfoils. The environment state is the current NIGnet parameter vector ϕ, an action is a bounded perturbation $\Delta\phi$, and the reward is the simulated L/D obtained via XFoil; negative reward is assigned when the solver fails to converge. Policies are learned with Proximal Policy Optimization over fixed horizons of $T = 5, 15$ and enhanced with Hindsight Experience Replay to generalize across target L/D goals.

We observe intuitively interesting results that show that short horizons and limited training primarily exploit angle-of-attack adjustments, whereas longer horizons and extended training enable substantial shape morphing that, together with AoA changes, yields airfoils achieving L/D values up to 61.64. The study highlights the promise of Deep RL coupled with non-self-intersecting geometry representations for scalable shape optimization, notes current bottlenecks arising from XFoil's fidelity and NIGnet size, and outlines future work toward higher-fidelity CFD rewards and larger, pretrained reward models.

1 Introduction

Shape optimization is the study of designing shapes that minimize or maximize some quantity of interest Jameson (1988); He et al. (2019). In particular, we will be looking at the lift-to-drag ratio as our quantity of interest. A body immersed in a flow experiences two forces a vertical force called Lift and a force opposing motion called Drag. An efficient design has high lift while having low drag therefore having a high overall lift-to-drag ratio. An optimization problem of interest for example would be designing the wing of an airplane that maximizes the lift-to-drag ratio.

A typical shape optimization process loop consists of three steps Li et al. (2022):

Shape Representation Using a parameterization method to represent the shape. The parameters ϕ are the design variables. E.g. splines with their control points as parameters.

Shape Evaluation Evaluating some characteristic of the shape that is to be minimized or maximized. E.g. lift-to-drag ratio of a wing.

Shape Improvement Changing the parameters ϕ to design shapes that achieve better characteristics. E.g. gradient descent to optimize ϕ.

In a large class of shape optimization problems the shapes of interest are simple closed curves. Simple closed curves, also called Jordan curves are curves that are closed, i.e. they form loops and are simple, i.e. they do not self-intersect. A distinction between different types of curves in 2D is shown in Figure 1.

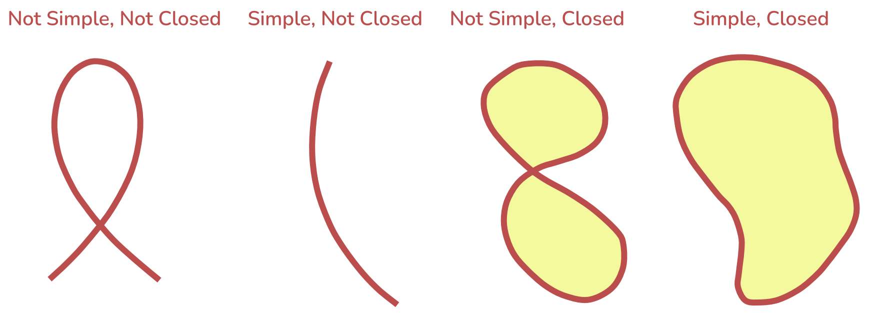

Figure 1: We will be concerned with simple closed curves, i.e. curves that loop back and do not self-intersect.

To understand why an optimization problem might be concerned only with simple closed curves consider the following problem: Imagine that there is a flow going left $\rightarrow$ right around the body shown in Figure 2, and consider the lift-to-drag ratio of this shape. The flow only sees the boundary of this shape, the complicated internal representation does not have any effect whatsoever on the properties of the body. This is because the flow cannot penetrate the boundary wall and never interacts with the geometry inside. All the representation used to describe the inner curves is wasteful. This is especially crucial when doing optimization. Shape representation methods that can potentially represent self-intersecting shapes would cause the optimization algorithm to search a bigger design space than needed. This leads to the design space blow-up problem. As a concrete example, consider an RL agent trying to modify the geometry to improve the lift-to-drag ratio. The agent would keep modifying the internal curves to study how those changes have an impact on the reward but get no useful reward signal as any change to internal geometry has no effect on the lift or drag. Therefore in effect, the agent would be searching through a design space bigger than it should.

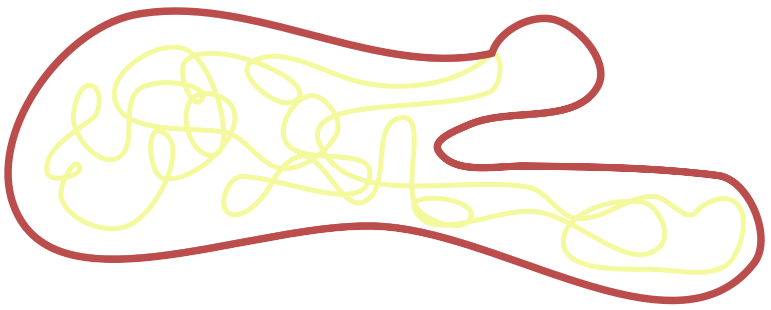

Figure 2: An incoming flow would only interact with the boundary. The complicated internal mess is wasteful representation.

Also, self-intersecting shapes might be physically undesirable or problematic to deal with in downstream tasks such as shape evaluation. Thus, such a shape parameterization would require manual tuning during optimization. As an e.g. consider the optimization happening in Figure 3. We represent the shape using 12 spline control points which are then fed into a neural network that predicts the shape's lift-to-drag ratio. Essentially, we have trained a surrogate model to mimic a Computational Fluid Dynamics (CFD) solver. During training we used non-self-intersecting shapes as they are the ones of interest, but during optimization, if after a gradient step the spline starts intersecting the network struggles to predict its lift-to-drag ratio and steers the optimization in an even worse direction. This is the classical distributional shift problem and leads to a divergence in optimization.

When using shape parameterization that can represent self-intersecting shapes the optimization algorithm's search has to be made limited, or someone has to sit and manually tune and check for

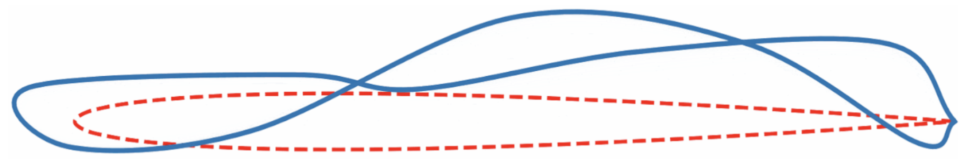

Figure 3: Optimization starts from the dashed red initial airfoil shape which is iteratively modified. We see that the optimization process suffers from distributional shift. Once a self-intersecting shape is reached it is iteratively made even worse.

self-intersection. Sometimes additional losses are added to the objective to prevent self-intersection Li et al. (2022); He et al. (2019). This is a hassle. In effect prevents an automated and aggressive shape space exploration. In an ideal setting, one would like to leave gradient descent running and go to sleep and wake up to find the optimal shape. Thus we need a shape parameterization that has the quality of non-self-intersection in-built and obeys this constraint for any set of parameters.

We apply Deep Reinforcement Learning to the shape optimization problem. Our novelty is in using a new neural shape representation, NIGnets, that will give us a hard guarantee on only representing non-self-intersecting shapes allowing our agent to search the design space in an unconstrained way without suffering from the design space blow-up or the divergence in optimization problems.

2 Related Work

Shape optimization has been studied using various optimization techniques. The most successfull techniques have been adjoint-based shape optimization Jameson (2003) and level-set-based shape optimization Allaire et al. (2002). While extremely powerful, both these methods suffer from two major issues:

Expensive high-fidelity computations These methods use gradient-based optimization where at each step of the shape optimization gradient step a full-scale Computation Fluid Dynamics (CFD) solver is called to compute the entire flow field around the current shape. This is an extremely costly computation and is the bottleneck in the optimization process Li et al. (2022).

Single-use computations After every gradient step is taken, the entire flow field computations that were performed are now thrown away as they have no use in further steps of the optimization. This leads to a huge waste of useful information that has a lot of physics encoded in it Yan et al. (2019).

Machine learning based optimization techniques can improve upon both these issues as they in effect use data to build surrogates for the high-fidelity CFD solver that can then be queried much faster during optimization. Also, all the flow field computations performed using a CFD solver can be used to train the surrogate further to predict the physics of the flow better.

Deep Reinforcement Learning based shape optimization is a nascent research area. The RL framework of actions to modify future states while trying to improve some reward fits naturally with shape optimization. The current shape during optimization can be thought of as the state, actions are modifications to the state that manifest as deformations on top of the current shape and rewards are the lift-to-drag ratios of the next state produced. Once an agent learns a policy that maps current shapes to deformations that should be produced to improve the shape, then it can be queried at great speed later to optimize shapes.

The few research studies that have explored deep reinforcement learning for shape optimization Viquerat et al. (2021); Dussauge et al. (2023) follow a typical procedure as shown in Figure 4. All these studies have one major issue in common: they deal with specifically constrained shape representations which are deeply based on domain intuition. These typically include constraining the shape to be represented by an upper and a lower loop that can never self-intersect. Using regularization to prevent shape self-intersection does not produce physically infeasible shapes but the optimization procedure as a whole still suffers from the two major problems that we discussed before: design space blow-up and divergence in optimization.

General robust shape optimization which can deal with arbitrary shaped curves have not been explored. This is because more general shape representation methods such as splines can lead to self-intersecting shapes during optimization Berzins et al. (2024). The usage of constrained shape representations has prevented aggressive design space exploration. This is because studies have restricted shapes to the set of designs that we now know through past experiments to be efficient. This prevents the design of unconventional designs that improve over previously designed shapes through human experimentation Lyu and Martins (2014).

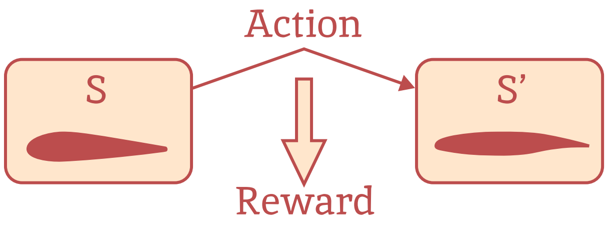

Figure 4: State-Action-Reward diagram for the deep RL based shape optimization procedure.

In our study the novelty is in using a new neural architecture that we have developed that can represent arbitrary simple closed curves. This architecture called NIGNets Aalok (2025), gives a hard guarantee on only representing non-self-intersecting geometry. A NIGnet is used quite differently from the way neural networks are used traditionally. There is no 'prediction' in a sense, the network as a whole represents a shape. That is, given some setting of weights and biases, a given network represents a particular shape, changing the parameters would lead to the network representing a different shape. But the guarantee that NIGnets provide us is that no matter what parameters we choose, the shapes we get are always non-self-intersecting.

Therefore, in our Deep Reinforcement Learning based shape optimization pipeline we will perform unconstrained shape optimization over NIGnet parameters to modify our geometry. This will help us tackle both the design space blow-up and divergence in optimization problems at the same time.

3 Method

The core part of our setup are Neural Injective Geometry networks or NIGnets. NIGnets are a specifically designed neural architecture that give hard guarantees on only representing non-self-intersecting geometry in any dimension. In particular for our current use case we will consider NIGnets in 2 dimensions. A typical NIGnet architecture is shown in Figure 5.

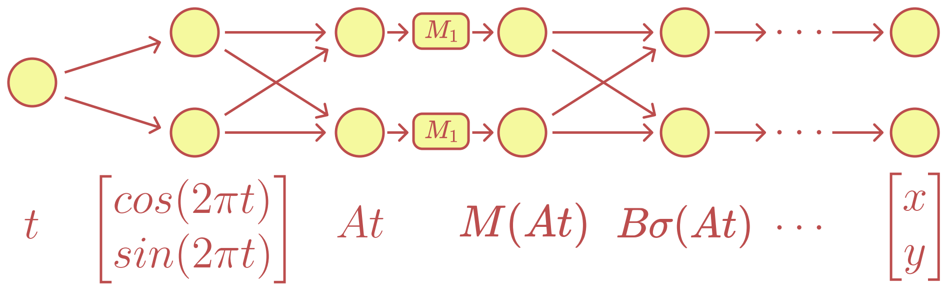
M_1
M_1
t
$\begin{bmatrix} cos(2\pi t) \\ sin(2\pi t) \end{bmatrix}$
At
$M(At)$
$B\sigma(At)$
$\cdots$
$\begin{bmatrix} x \\ y \end{bmatrix}$

Figure 5: General Neural Injective Geometry network Architecture.

The input to a NIGnet is a scalar $t \in [0, 1]$ that gets mapped to a vector of point coordinates $[x, y]$. As t varies from $[0, 1]$ the curve from the starting point to the ending point is traced out. This is visualized in Figure 6.

A NIGnet as a whole with a particular setting of its weights and biases represents a particular shape. We denote the parameters of the network by ϕ. Therefore, a ϕ represents a shape and modifying to a different ϕ' would lead to representing a different shape. NIGnets provide the guarantee that any ϕ would only represent non-self-intersecting geometry.

Our Deep Reinforcement Learning based shape optimization can be summarized as follows:

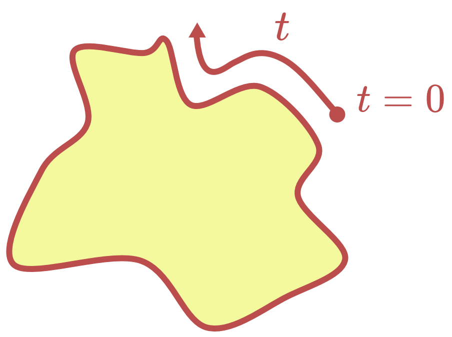

Figure 6: General Neural Injective Geometry network Architecture.

1. Represent state (shape) using NIGnet parameters ϕ. Where ϕ denotes the weights and biases of the network.
2. An action corresponds to outputting $\Delta\phi$ that perturbs the NIGnet parameters to $\phi' = \phi + \Delta\phi$.
3. A reward is produced at each step that corresponds to the $\frac{L}{D}$ ratio of the shape represented by the NIGnet with parameters ϕ'.

This is represented visually in Figure 7.

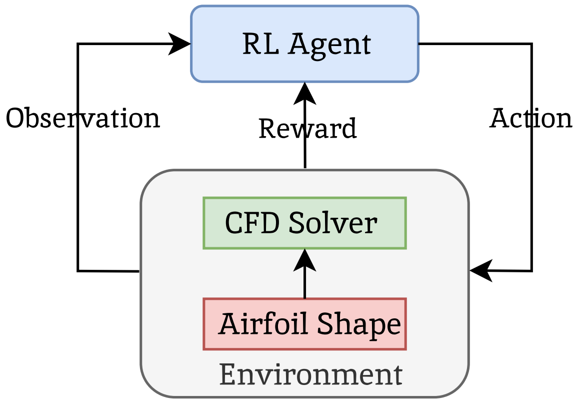

Figure 7: Reinforcement learning architecture for airfoil design.

The reward comes from the Computational Fluid Dynamics (CFD) solver that given a shape computes the fluid flow field around it, in particular the pressure distribution and uses that to compute separately the total lift and drag force on the body. The lift-to-drag ratio is then used as a reward for our agent.

In particular, for the CFD solver we use XFoil Drela (1989). XFoil is a low-fidelity panel-based external aerodynamics solver that is written in Fortran and is extremely fast. This allows us to query XFoil (and therefore the environment) very quickly, allowing the use of a large training dataset and the use of stable on-policy algorithms. Each XFoil query takes in the current shape and outputs the lift-to-drag ratio, taking on average 0.2 seconds.

We setup the reinforcement learning task as a fixed horizon task of length T. Where, in this fixed horizon setting to make sure that the policy is time-stationary we pass in the time step appended to the state as well.

At each stage the agent can produce a network parameter perturbation $\Delta\phi$ in the clipped interval $[\sigma, \sigma]$. We use a clipped interval to allow only small perturbations to the network parameters, this allows us to learn a policy that can learn to make fine-grained geometry changes to influence the lift-to-drag ratio. To allow for large overall changes to the shape we can instead increase the time horizon T while keeping the per-step perturbation $\Delta\phi$ small.

Another reason for small per-step perturbations is related to the particular environment choice. XFoil is extremely fast, but that comes at the trade-off of it being low-fidelity and non-convergent for highly complex shapes. Large perturbations to the network parameters can lead to shapes that are non-physical and fail to converge in XFoil therefore, leading to no output lift-to-drag ratio. In such cases we have to provide a negative reward, which we choose to be -50 (empirically found to work well). Since large perturbations would lead to mostly complex-shapes giving negative rewards the agent does not accumulate enough meaningful state, action, reward pairs for effective learning. Therefore, smaller steps allowing for fine-grained geometry changes allow the agent to avoid non-physical shapes and accumulate better reward trajectories.

In all our experiments we fix the per-step interval range to $\sigma = 0.01$ and vary the time horizon of the task T. We start off with a lower T of 5, once we have sufficiently tuned our hyperparameters for learning we experiment with larger T of 10 and 15 that allow us larger changes in geometry overall. We also experiment with the total number of learning time steps that is, the total environment queries that our agent can make. We test our time horizons of length $T = 5, 15$ first with a total time step budget of $100,000$ and then perform experiments with a much larger budget of $1,000,000$ steps.

3.1 Hindsight Experience Replay for more General Shape Design

The approach defined above learns a policy to design shapes that maximize lift-to-drag ratio. But several real-world scenarios require us to design airfoils with a pre-specified lift-to-drag ratio. This could be required for specific aircraft maneuvers that do not target efficiency but for e.g. may target agile maneuverability. A policy learnt through the previous procedure can given a shape, optimize it to maximize the lift-to-drag ratio but cannot be used to design shapes with target ratios.

We can learn more general policies that can perform shape design to produce shapes with specified target lift-to-drag ratios by using Hindsight Experience Replay. Suppose in a particular the agent produces a set of actions to modify the initial shape and produces a final shape. The lift-to-drag ratio of this final shape produced could be relabelled to be the 'right' goal for this sequence of actions taken. This requires us to use the target and current lift-to-drag ratio in our state. As before we experiment with a fixed time horizon $T = 5$ of fixed per-step parameter perturbation σ.

4 Experimental Setup

For our experiments we use the stablebaselines3 library Raffin et al. (2021). As discussed before we initialize the optimization as a fixed time horizon task with deformations per-step to the NIGnet parameters restricted to the clipped range $[-\sigma, \sigma]$. We experiment with different T values while keeping σ fixed at 0.01. In particular we observed during initial experimentation that with this setting of sigma $T \geq 20$ lead to largely non-convergence and default negative rewards. Therefore, we perform initial testing with $T = 5$ and then move on to $T = 15$ for larger overall shape deformations. We construct a custom gym environment where the parameters of the NIGnet are flattened into a vector to form the state. Actions which are also vectors of the same dimension are then used to explicitly set the new parameters of the network.

A custom lift-to-drag ratio calculation script is written that uses a python interface to the XFoil fortran code. Care is taken to make sure that the memory explosions do not happen when using vectorized environments from within stablebaselines3, especially due to creation of copies of the compiled fortran files during XFoil calls.

On non-convergence a negative reward of -50.0 is given. We experimented with a few different penalty values and found this to be a good value empirically through initial experimentation.

Also, to consider only the effects of shape on the lift-to-drag ratio and not the scaling of the shape we further modify NIGnets specifically for our task. Specifically, we add an additional layer to the network that moves the centroid of the shape to $(0, 0)$ and scales the shape to make its largest dimension in x or y to lie exactly in the range $[-1, 1]$. These two steps are essentially mean removal and scaling of shapes.

For lift-to-drag ratio maximization we used the Proximal Policy Optimization (PPO) algorithm as implemented in stablebaselines3.

4.1 Considerations for Hindsight Experience Replay

When using HER for learning more general policies we experimented similarly with the time horizon T. We additionally also experimented with the reward formulation. We found learning general policies to design for arbitrarily specified target lift-to-drag ratios to be much harder than simply maximizing the ratio.

We got good results with $T = 5, 10$, especially $T = 10$ gave us really good reward values on longer training. We tried the following different loss formulations:

- Negative L2 norm: We used the negative L2 norm between the desired lift-to-drag ratio and the lift-to-drag ratio at the current time step and state.
- Inverse L2 norm: We also experimented with using the inverse of the L2 norm between the desired and the current lift-to-drag ratio.

We found that even though inverse L2 norm had slow initial training it performed much better when trained for longer. This is probably because when the shape produced is very close to the desired lift-to-drag ratio then the reward is large and the actions in the sequence are highly weighted. Whereas, in the case of negative L2 norm the rewards were always negative and smaller distances to the target ratio was weighted lower the closer it was.

For general lift-to-drag ratio design policy learning with HER we use the Soft Actor Critic (SAC) algorithm as implemented in stablebaselines3.

5 Results

We trained agents for the lift-to-drag maximization task using the following different settings of our hyperparameters:

- T: 5 and 15.
- Total timesteps: $100,000$ and $1,000,000$.

After the training is done we freeze the agent and perform an evaluation. This happens by starting from a NIGnet that is fit to a symmetric airfoil that has a lift-to-drag ratio of 0 as it is symmetric and produces no lift (flow above and below it is the same). We allow the agent as many steps to modify the shape as it was trained on i.e. 5 or 15. We then report and visualize the lift-to-drag ratio achieved by the final shape produced by our agent.

Similarly for HER we consider:

- T: 5 and 10.
- Total timesteps: $100,000$ and $1,000,000$.

5.1 Quantitative Evaluation

We perform experiments for progressively increasing T first and then the total timestep budget for training. Result summary for our experiments on the lift-to-drag maximization task are shown in Table 1 and for learning generalized policies using HER are shown in Table 2. T represents the task horizon and TS represents the total training time steps i.e. total environment steps taken. The training reward shown is the sum of all rewards in the trajectory i.e. sum of all lift-to-drag ratios for all shapes produced during a training episode.

Table 1: Performance Comparison

Case	Training Full Trajectory Reward	L/D at Evaluation
$T = 5, TS = 100k$	151.4	37.53
$T = 5, TS = 1M$	180.6	45.21
$T = 15, TS = 100k$	524.3	52.84
$T = 15, TS = 1M$	686.8	61.64

Table 2: Performance Comparison

Case	Training Full Trajectory Reward	L/D at Evaluation for target 30
$T = 5, TS = 100k$	-70.1	0.01
$T = 5, TS = 1M$	54.2	11.26
$T = 10, TS = 100k$	-94.5	-3.96
$T = 10, TS = 1M$	234.2	32.46

We observe that we are able to achieve a decent lift-to-drag ratio of 37.53 with just $T = 5$ and total time steps = $100,000$. This entire training takes around 30 minutes with 16 vectorized environments. As we explain in the next section through visualizations, this lift-to-drag ratio comes mainly from an angle of attack increase.

On keeping the time horizon fixed and increasing the total training time steps we get a boost on the lift-to-drag ratio to 45.21. This comes mainly from a further increase in angle of attack.

We observe an interesting change in the optimization results when we increase the time horizon T to 15 while keeping the total training time steps at the lower budget of $100,000$. During evaluation we observe that the angle of attack increase stagnates and some shape morphing starts to appear. We get a boost in the lift-to-drag ratio achieved during evaluation up to 52.84.

Major shape morphing occurs on increasing the total training time step budget to $1,000,000$ with a simultaneous stagnation of the increase in angle of attack. We are able to achieve a high lift-to-drag ratio of 61.64.

An example mean reward training run is shown in Figure 8 for the case $T = 15$ and total timesteps $100,000$.

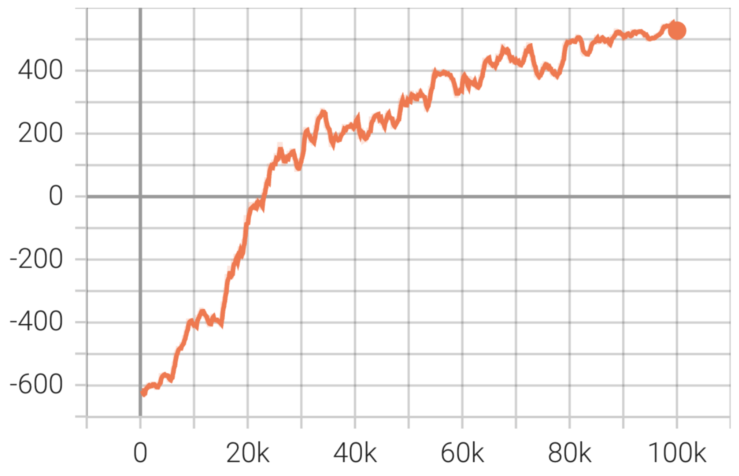

Figure 8: Sample training run showing mean reward during training for the case of $T = 15$ and total timesteps $100,000$.

5.2 HER results

With HER we observed that training is much more unstable due to non-convergence issues in XFoil. We were able to achieve good results by using the inverse L2 norm as our reward function with a non-convergence penalty of -50 as before.

Training results are shown in Table 2. Our best performance came from a trained agent with $T = 10$ and total training time steps $1,000,000$. During evaluation we design for a target L/D ratio of 30. The final design is shown in Figure 9 which achieved an L/D ratio of 32.46.

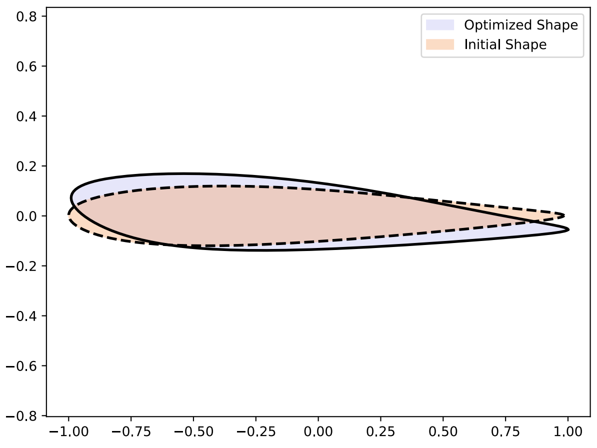

Figure 9: Optimization Results for HER with target lift-to-drag ratio of 30.

5.3 Qualitative Analysis

The visualizations of the final produced airfoils show very interesting intuitive effects. The most interesting observation is that for a short horizon $T = 5$ and for small number of training steps $100,000$ an increase in the lift-to-drag ratio comes almost completely from an increase in the angle of attack with barely any shape change. This is a physically intuitive result familar to most practitioners of aerodynamics where the easiest way to produce lift is to align a body at an angle to the flow. This in effect turns the flow downwards and in effect creates an upward force by Newton's 3^{rd} law.

On further increasing the time horizon to $T = 15$ while keeping the total timesteps fixed at $100,000$ we observe a further increase in angle of attack but now we also start observing some shape morphing. This shows that the increase in lift-to-drag ratio that could be produced from just an increase in angle of attack has already been exploited and that further increase has to involve shape deformation.

On further cranking up the total timesteps to $1,000,000$ while keeping the time horizon constant at $T = 15$ we witness significant shape morphing while the angle of attack seems to have stagnated to the highest possible value. Any further angle of attack would lead to flow separation with massive increments in drag and therefore further increase in the lift-to-drag ratio cannot come from an increase in angle of attack and comes from geometry deformation.

We also experimented with $T = 20$ but in that case the final few shapes are so morphed that they do not converge in XFoil and lead to continuous large negative rewards that hinder good training.

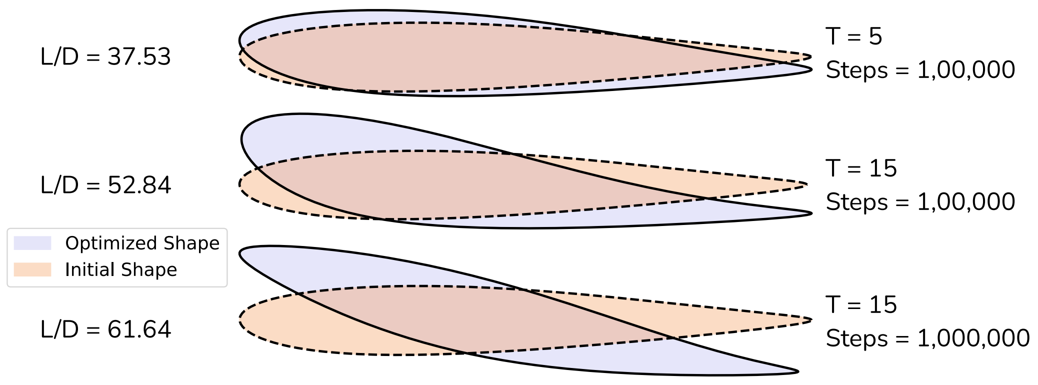

Figure 10: A clear distinction in change in angle of attack and shape morphing with increasing T and total training steps is observed.

Overall we observe optimization results that parallel intuition built from physical experiments further solidifying the use case of our Deep Reinforcement Learning based shape optimization on top of NIGnets.

6 Discussion

We find that initial experiments with Deep Reinforcement Learning for shape optimization have proved very fruitful with interesting results.

One major limitation of our pipeline is the environment, in particular the CFD solver, XFoil that we use. We use XFoil because it is extremely fast. But has a few major downsides:

1. **Non-convergence:** For complex shapes XFoil fails to converge and therefore we have to provide a default negative reward to our agent. This means that these parts of the design space are like dark-matter that the agent does not understand and only learns to avoid. Also these default negative rewards reduces the amount of information that the agent receives. This we observe to be especially problematic during the initial learning phase where the agent runs into shapes that do not converge in XFoil quite often making the initial learning slow.
2. **Lack of Physics information:** XFoil solves directly for lift-to-drag ratio and does not provide a pressure field. A more high-fidelity RANS solver would provide us with an entire pressure distribution field around our shapes which contains a lot of physics information about how flows behave.

Using a more high-fidelity RANS solver will theoretically provide us with much more physics information and also alleviate the non-convergence problem to a large extent. But this would come at the cost of much more compute per environment query. Therefore, to move to using a RANS solver we will also have to move to a more powerful reward prediction network. This can be done best with a Neural Operator in the arbitrary shape resolution setting Kovachki et al. (2023).

The usage of Hindsight Experience Replay for training more general policies will also be quite useful when using compute intensive RANS solvers as that would make our entire approach more data efficient.

Finally, another major limitation is the size of the NIGnet itself. Since we start learning a policy on top of NIGnet parameters from scratch, we need to limit the NIGnet parameter count to a low number otherwise our RL agent will suffer from the curse of dimensionality. To move towards larger NIGnets that are more expressive, we will have to warm start our reward model after some pre-training using an available dataset of shapes, pressure field values. Thus supervised pretraining of a reward model will be crucial in scaling up design optimization using Deep RL.

7 Conclusion

We find that Deep Reinforcement Learning based shape optimization works really well when the agent can perform unconstrained shape optimization over the geometry parameters without running into self-intersection related issues. NIGnets not only allow a hard guarantee on providing such a geometry representation but also morph fast and stably enough to work well for aerodynamic shape optimization even with small horizon tasks.

This approach to applying Deep Reinforcement Learning for shape optimization is exponentially more scalable than past approaches that utilized splines that could lead to self-intersecting geometry during optimization. The exponential benefit comes from the fact that larger shape parameters can lead to exponentially larger number of possible self-intersecting shapes.

We strongly recommend building Deep RL based approaches in this direction using NIGnets or by developing other shape representations that provide similar guarantees.

Interesting future avenues for research include:

1. **Further Experimentation:** Scale up the policy network and the critic network and train for longer. Make no changes to the environment and the algorithm. Compare results with spline based representation results Dussauge et al. (2023).

2. **Algorithm Refinement:** Stick with PPO as the training is very stable even without warm starting the model with any flow information. Increase the time horizon to a much larger value to allow the agent to make more aggressive overall perturbations to the shape while also tuning the action range σ.
3. **Evaluation & Analysis:** Further, compare the results with those of Dussauge et al. (2023) especially the training stability characteristics. Establish whether NIGnets perform better than spline based representations.

8 Team Contributions

- **Atharva Aalok:** Complete project end-to-end.

Changes from Proposal Our original hypothesis of shape optimization using iterative perturbations to NIGnet parameters worked really well and the results are very interesting! We do not produce any changes to the original proposal in the central pipeline. We do extend our proposal to also include Hindsight Experience Replay allowing us to learn more general policies that can not just maximize lift-to-drag ratio but also design for any specified target value.

References

Atharva Aalok. 2025. NIGnets. `https://github.com/atharvaaalok/NIGnets`

Grégoire Allaire, François Jouve, and Anca-Maria Toader. 2002. A level-set method for shape optimization. *Comptes rendus. Mathématique* 334, 12 (2002), 1125–1130.

Arturs Berzins, Andreas Radler, Eric Volkmann, Sebastian Sanokowski, Sepp Hochreiter, and Johannes Brandstetter. 2024. Geometry-informed neural networks. *arXiv preprint arXiv:2402.14009* (2024).

Mark Drela. 1989. XFOIL: An analysis and design system for low Reynolds number airfoils. In *Low Reynolds Number Aerodynamics: Proceedings of the Conference Notre Dame, Indiana, USA, 5–7 June 1989*. Springer, 1–12.

Thomas P Dussauge, Woong Je Sung, Olivia J Pinon Fischer, and Dimitri N Mavris. 2023. A reinforcement learning approach to airfoil shape optimization. *Scientific Reports* 13, 1 (2023), 9753.

Xiaolong He, Jichao Li, Charles A Mader, Anil Yildirim, and Joaquim RRA Martins. 2019. Robust aerodynamic shape optimization—from a circle to an airfoil. *Aerospace Science and Technology* 87 (2019), 48–61.

Antony Jameson. 1988. Aerodynamic design via control theory. *Journal of scientific computing* 3 (1988), 233–260.

Antony Jameson. 2003. Aerodynamic shape optimization using the adjoint method. *Lectures at the Von Karman Institute, Brussels* 6 (2003).

Hadi Keramati, Feridun Hamdullahpur, and Mojtaba Barzegari. 2022. Deep reinforcement learning for heat exchanger shape optimization. *International Journal of Heat and Mass Transfer* 194 (2022), 123112.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. 2023. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research* 24, 89 (2023), 1–97.

Jichao Li, Xiaosong Du, and Joaquim RRA Martins. 2022. Machine learning in aerodynamic shape optimization. *Progress in Aerospace Sciences* 134 (2022), 100849.

Shaopeng Li, Reda Snaiki, and Teng Wu. 2021. A knowledge-enhanced deep reinforcement learning-based shape optimizer for aerodynamic mitigation of wind-sensitive structures. *Computer-Aided Civil and Infrastructure Engineering* 36, 6 (2021), 733–746.

Zhoujie Lyu and Joaquim RRA Martins. 2014. Aerodynamic design optimization studies of a blended-wing-body aircraft. *Journal of Aircraft* 51, 5 (2014), 1604–1617.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research* 22, 268 (2021), 1–8. `http://jmlr.org/papers/v22/20-1364.html`

Jamshid Samareh. 2004. Aerodynamic shape optimization based on free-form deformation. In *10th AIAA/ISSMO multidisciplinary analysis and optimization conference*. 4630.

Jonathan Viquerat, Jean Rabault, Alexander Kuhnle, Hassan Ghraieb, Aurélien Larcher, and Elie Hachem. 2021. Direct shape optimization through deep reinforcement learning. *J. Comput. Phys.* 428 (2021), 110080.

Xinghui Yan, Jihong Zhu, Minchi Kuang, and Xiangyang Wang. 2019. Aerodynamic shape optimization using a novel optimizer based on machine learning techniques. *Aerospace Science and Technology* 86 (2019), 826–835.