Exploring Reinforcement Learning for Constrained Wing Shape Optimization

Noël van Putten

Delft University of Technology

TU Delft

Exploring Reinforcement Learning for Constrained Wing Shape Optimization

by

Noël van Putten

In partial fulfillment of the requirements for the degree of

Master of Science
in Aerospace Engineering

at the Delft University of Technology,
to be defended publicly on Monday June 6, 2024 at 14:00.

Student number: 4431782
Project duration: Feb 6, 2023 – June 6, 2024
Thesis committee:

Dr. ir. C. Varriale,	TU Delft, Supervisor
Ir. K. Swannet,	TU Delft, Supervisor
Dr. ir. G. la Rocca,	TU Delft, Chair
Dr. ir. M. Ribeiro,	TU Delft, Examiner

Cover: AI generated image of an Aircraft, obtained from vecteezy.com

TU Delft

Acknowledgements

By handing in of my thesis, I will finally conclude my student time. Throughout my studies I gained an increasing interest in Machine Learning and I'm happy I was able to combine this with aerospace in this thesis. I will look back fondly on my time at the TU as I'm very proud of the knowledge I was able to acquire.

First I want to thank my family for always encouraging me to continue my academic career and telling me its alright to slow down sometimes. I would also like to thank my friends, who made the remote parts of the studying a bit more bearable. Particularly Kevin, I can't say I enjoyed those late nights working on the assignments but we got through them nonetheless. Couldn't have done it without you, pal!

Finally, I would like to thank my supervisors Carmine and Kilian. I am very thankful for your continued guidance throughout my thesis. You have allowed me the freedom to explore what really motivated me and together we came up with an assignment that really interested me. I am definitely going to miss our biweekly discussions. Sadly, I wasn't able to complete everything that I set out to at the start of the thesis, but I am still proud of what I have accomplished. That being said, it is definitely time for me to conclude my research and start applying my newly gathered skills. I look forward to whatever will come next.

Noël van Putten
Spijkenisse, May 2024

Contents

List of Figures

List of Acronyms

Acronyms

ACA Advisor Critic Architecture. 33

AI Artificial Intelligence. 25

ANN Artificial Neural Network. 28, 30–33, 41, 42

ASO aerodynamic shape optimizations. 41, 42

AVL Athena Vortex Lattice. 39

BO Bayesian Optimization. 37, 47

CFD Computational Fluid Dynamic. 39, 42, 43

DEN Dynamically Expandable Network. 32

DQN Deep Q-Network. 31, 41

DRL Deep Reinforcement Learning. 31

FFD Free-Form Deformation. 42, 43

GA Genetic Algorithm. 37, 42, 47

HLD High-Lift Devices. 39, 48

KBE Knowledge Based Engineering. 32, 34, 36, 43

MARL Multi-Agent Reinforcement Learning. 33

MDAO Multi-Disciplinary Analysis and Optimization. 25–27, 34–37, 43, 44

ML Machine Learning. 25–34, 36, 37, 42–45, 47–49, 51

OOP Object-Oriented Programming. 36

PINN Physics Informed Neural Network. 33

PPO Proximal Policy Optimization. 42

PSO Particle Swarm Optimization. 37, 38, 41, 47

RL Reinforcement Learning. vii, 30–33, 41, 42, 45, 48, 49, 51

VLM Vortex Lattice Method. 39

XDSM eXtended Design Structure Matrix. 34

Introduction

With the recent advancements in Artificial Intelligence (AI), innovative applications are being developed for all branches of science and technology. This thesis started with the goal in mind of finding advantageous ways of combining AI with Aerospace Engineering. This topic was chosen as the current methods of automated aerospace design struggle to be widely adopted professionally, despite their effectiveness [1, 2]. The addition of AI to aerospace design could help uncover innovative aircraft designs.

In this thesis, a ML algorithm is selected to construct an optimization framework. This ML framework is then tasked to perform the constrained optimization of a wing shape for a specific aircraft at the conceptual design phase level. The adaptation of the ML framework to the design problem is quite complex and is a trial and error process. This adaptation is not only dependent on the specific ML algorithm selected, but also requires knowledge of the design space of the particular design problem. Once the framework is created, the ML model is trained to improve itself by repeatedly performing the same optimization and learning from its results. The performance of the model throughout its training phase is studied in order to select the best model. The performance of the final ML framework is then compared to a traditional optimization algorithm commonly found in automated aerospace design frameworks. Performance parameters like the reached objective value, consistency and time per optimization to solve the design problem are evaluated.

The training of a ML algorithm can be computationally costly. In order for the ML framework to be competitive with the traditional optimization algorithms, the training time has to be justified. The trained ML frameworks are able to be repurposed through a technique called Transfer Learning (TL). The purpose of TL is to reduce the overall computational costs of the optimization process and allows the framework to optimize the wings of different aircraft.

This report consists of three parts. In part I, a scientific paper is presented that details the final results and conclusion of the research. In part II, the literature study discusses the research done on the relevant topics to formalize the thesis objective. This objective has shifted since the literature study has concluded, so the conclusion of this study diverges slightly from the research paper. The research paper is written with the assumption that the reader possesses some knowledge on the basics of ML and is advised to refer to the literature study if additional information is required. This report concludes with part III. Here, additional results and insights are given to support the findings of the research paper.

I

Scientific paper

Exploring Reinforcement Learning for Constrained Wing Shape Optimization

Noël van Putten [1], Carmine Varriale [2] and Kilian Swannet[3]

[1]MSc. Student, TU Delft
[2]Supervisor, Assistant Professor TU Delft
[3]Supervisor, Phd Candidate TU Delft

Abstract

In this paper, the Proximal Policy Optimization (PPO) algorithm is used to perform a constrained wing shape optimization. The PPO algorithm is a Machine Learning (ML) algorithm that improves itself by repeatedly performing the same optimization and learning from its results. The complete adaptation of the PPO framework to the design problem is detailed and evaluated. Not only was the PPO framework able to consistently optimize the wing 4% further than the Particle Swarm Optimization (PSO) algorithm, it was able to do so 35 times faster once the model is fully trained. The PPO framework was able to find more efficient wing shapes than the PSO framework. The trained PPO model was able to optimize the wing of other similar aircraft, even without direct retraining. These results illustrate that PPO could be a promising technique for automated aerospace design problems. Due to the significant training time of the ML approach, the PPO algorithm is not an effective replacement of traditional optimization algorithms for design problems where only a single optimization is required.

Nomenclature

C_{D_f}	Friction drag coefficient
C_{D_p}	Profile drag coefficient
FF	Form factor
ϵ	Twist angle
Γ	Dihedral angle
λ	Taper ratio
$\Lambda_{C/4}$	Quarter chord sweep angle
Λ_{LE}	Leading edge sweep angle
σ	Standard deviation
$\vec{dv}$	Design vector
a_i	Action at step i
AR	Aspect ratio
b	Wing span
C_D	Drag coefficient
C_f	Friction Coefficient
C_L	Lift coefficient
c_{kink}	Kink chord length
c_{root}	Root chord length
c_{tip}	Tip chord length
f_i	Objective value at step i
g_n	Constraint n
j	Design variable from the design vector
j_{LB}	Lower bound value of design variable j
j_{UB}	Upper bound value of design variable j
M	Mach number
m_{fuel}	Fuel mass
m_{tot}	Total aircraft mass
n_z	Ultimate load factor
P_n	Penalty value n
q	Dynamic pressure
Re	Reynolds number
S_i	State at step i
S_{wet}	Wetted surface area
S_{wing}	Wing surface area
t/c	Chord thickness
t_{train}	Time required to train the DRL framework
V_{wing}	Volume of wing
W/S	Wing loading
x/c_{max}	Chord-wise location of maximum chord thickness
X_{tr}	Transition point
y_{kink}	Span-wise location of the kink chord
z_{tip}	Z-location of the tip chord

1 Introduction

One of the main points of focus of the aviation industry is reducing the environmental impact of aircraft [1]. In order to achieve this, the efficiency and performance of designed aircraft have to be optimized as much as possible. The optimization of an aircraft design is a very complex process as many fields are dependent on each other. This leads to a lot of time spent on inter-department communication which results in many small changes during the conceptual design phase. The field that specializes in addressing this problem is called Multi-Disciplinary Design Optimization (MDAO) [2]. The main focus of MDAO is to develop techniques and tools that allow engineers to simultaneously design and optimize all components of the aircraft. However, MDAO techniques are currently not widely utilized professionally as the set-up and expertise required prove to be too great of a hurdle [3, 4].

The aim of the thesis is to investigate the possible

advantages and limitations of Machine Learning (ML) applications to automated aerospace design. ML applications have inherent synergies with automated processes and its application to automated design problems could prove advantageous. However, ML applications for aerospace problems are limited, especially regarding automated design. ML algorithms can be used as optimization frameworks similar to those employed in MDAO. As ML employs a fundamentally different strategy from traditional optimization algorithms, exploring this could uncover potential advantages. A traditional optimization algorithm refers to an algorithm that is used for automated design optimization and can commonly be found in MDAO applications.

In this paper, the Proximal Policy Optimization (PPO) algorithm [5] is selected to perform the constrained optimization of a wing shape at the conceptual design phase level. Due to limited computational resources, the design problem is kept as simple as possible. The performance of the RL framework is then compared to the performance of the Particle Swarm Optimization (PSO) algorithm [6]. Performance parameters like the reached objective value, time per optimization and ratio of converged to failed optimizations are evaluated. Then, the trained ML model is repurposed to solve the design problem for an altered design space with the goal of gaining computational advantages.

2 Literature Review

ML techniques offer many opportunities to improve the efficiency of automated aerospace design while simultaneously improving the performance of the designed aircraft components [7, 8]. Within aerospace, the data-driven ML frameworks are most common and are often combined with Aerodynamic Shape Optimization (ASO). While these ML frameworks offer great advantages, a large data-set is required to effectively train these models. Reliable and high quality data is not always available for design projects. Deep Reinforcement Learning (DRL) is an alternative ML method that has been successfully implemented as an optimization framework [9, 10, 11]. DRL differs from the data-driven ML approach by directly interacting with physics based solvers and optimizing a design on the basis of the results. A key advantage of the DRL framework is that its results are based on the output from the solvers. This makes the results from the DRL framework match the accuracy of those solvers, while the data-driven frameworks simply predict their results.

Y. Azabi and collegues have created a framework using the PSO algorithm with an interactive approach to perform an ASO for a UAV [12]. In their framework, a separate 'Decision Maker' interacts with both the aerodynamic solver and the optimization algorithm. The Decision Maker guides the optimization algorithm to advantageous locations in the design space, similarly to the DRL frameworks. This framework resulted in computational costs being halved to reach similar results as the traditional optimization approach. The DRL framework iteratively learns how to best solve a problem and can be set-up to directly comply with constraints. Because the DRL algorithms has learned how to most effectively explore a design space, it is able to significantly reduce the required computational costs. The DRL frameworks are able to perform ASO in only a fraction of the time compared to traditional optimization frameworks [13, 14]. These efforts illustrate the increase in computational efficiency that DRL frameworks can offer. Discovering more efficient design techniques could aid in creating an interactive design tool for an engineer to rapidly iterate and generate new designs [15, 16].

A big hurdle for the use of all ML framework types is that they require a costly training phase before they exhibit adequate performance [17]. ML frameworks are typically designed to solve a single problem. This means that an entirely new ML framework has to be trained when changes are made the design problem. This severely off-sets the computational advantages gained from a trained ML framework. Transfer learning (TL) is a technique that tries to alleviate this problem. The goal of TL is to leverage the knowledge of a trained framework to solve similar but new problems [17, 18, 19]. This has the potential to greatly increase the computational efficiency of any optimization framework. DRL techniques have shown some resilience to small changes to the design space. These DRL frameworks can generalize their design strategy to solve design problems without the need for direct retraining [9, 10, 20].

A DRL algorithm that has already shown great promise in aerospace optimization problems is called PPO [5]. For example, PPO was successfully used for a constrained optimization problem of an airfoil shape and was able to do this more efficiently than a traditional optimization framework [21]. The PPO algorithm also shows great results when used in conjunction with transfer learning techniques [22].

Data-driven approaches are quite popular applications of ML frameworks, where large data sets are leveraged to train ML models. This allows a ML model to replace high-fidelity CFD solvers to predict the aerodynamic performance of an aerodynamic body [23, 24, 25]. A key advantage of using a ML framework to achieve this is that the evaluation can be completed almost instantaneously, allowing an engineer to rapidly explore a design space [24, 25]. These ML frameworks can also be used as an optimization framework for aerospace design. A model can be trained to generate new airfoil or wing shapes that exhibit certain aerodynamic performance parameters [26, 27]. This methodology is able to significantly decrease computational costs compared to the traditional optimization strategies [28]. While the data-driven ML approaches have been extensively explored, the implementation of DRL methods to aerospace problems remain somewhat undefined. The general adap-

tation of a DRL framework to a design problem has not been formalized as the DRL applications described above all employ their own unique strategies to solve their respective design problems. In this paper, the adaptation of the DRL framework to the optimization problem is documented and evaluated.

3 Methodology

An optimization problem is solved with both a DRL and the PSO algorithm. The available computational resources greatly determined the fidelity of the solvers used in this project. All computations performed during this project are done using a Windows PC equipped with an i9-13900F CPU and a Nvidia 3060 GPU.

3.1 The basics of Deep Reinforcement Learning

In DRL, an agent continually takes actions and reacts to the response of its environment on a trial and error basis [29]. The agent is the main component of a DRL model and determines what actions a_i should be performed on the basis of the policy function and the current state. The state S_i is typically an array comprised of the design vector at the current step i. The environment consists of a structure of physics based solvers and calculations that use the design variables as input and generates an output. This output is then evaluated by generating a reward signal R_i. The reward is a scalar variable that determines whether the response of the environment to the selected action of the agent is favorable. The main goal of the agent is to maximize the total reward it receives during the training phase and update its policy function accordingly. The agent will continue to take actions until a terminal state is reached.

3.2 The Proximal Policy Optimization algorithm

The PPO algorithm is selected as the DRL algorithm used to solve the design problem. The PPO algorithm utilizes two separate artificial neural networks (ANN) to represent its policy function. In the context of an optimization problem, the policy network is concerned with taking the best action to increase the objective value at the current step. The value network tries to guide the agent through the design space to find the best final objective value. The input for both networks are the state vector, action and reward. The policy network tries to predict what action would be best, given the current state. A policy evaluation is then performed to improve the policy network. The value network determines the expected cumulative reward given the current state, this is known as the state value. The state value is then evaluated using an advantage function. The output of the advantage function is used to update and improve both networks, see Fig 1. The PPO algorithm limits the maximum policy-updates made to these networks at each step. This increases the stability during the training phase.

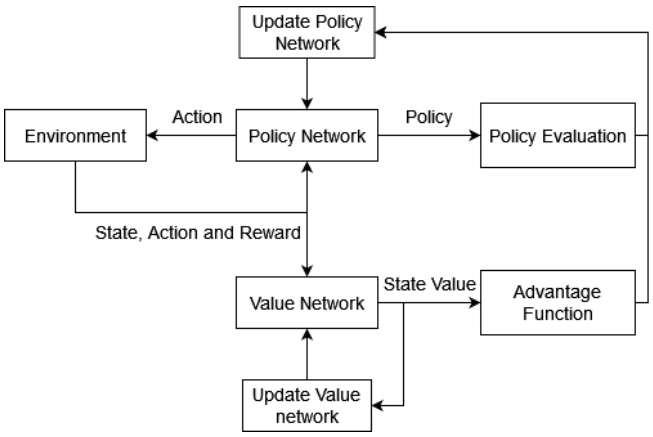

Figure 1: *PPO diagram showing the interaction between the components of the algorithm, adapted from [30].*

For this project, a python package called Stable Baselines[1] is used. This package has made improvements to the PPO algorithm presented in the original paper [5] to increase consistency and performance. For this project the hyper parameters suggested by Stable Baselines were unaltered except for the size of the networks. An additional hidden layer of 64 neurons was added to both the policy and value networks, as this appeared to have favourable performance while not significantly increasing computational time of the training phase. The hyper parameters can be found in Table 7 in Appendix A.

3.3 The Particle Swarm Optimization algorithm

The gradient-free PSO is selected as a baseline comparison to the DRL framework results. This is chosen as its capabilities more closely match those of the DRL framework than gradient-based algorithms [12, 14, 20]. The gradient-free optimization algorithms are well suited for the optimization of complex design problems and are good at finding global optima.

PSO distributes a number of random design vectors, called particles, across the entire design space [6]. The total number of particles is referred to as the swarm size. The design variables of these particles are then adjusted to look for local optimum solutions distributed across the entire design space. These particles share information with each other to attempt to look for a global optimum together. PSO is known to be a robust method but can become computationally expensive, depending on the number of particles and design vector length.

[1] `https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html`, accessed on 26-02-2024

A python package called indago[2] is used to perform the PSO optimization. This package contains an integrated method for incorporating constraints into the optimization algorithm. Most of the hyper-parameters are unaltered from the recommended setting as this delivered adequate performance. These hyper parameters can be found in Table 8 in Appendix A. The optimization was set to a maximum of 7500 evaluations and the swarm size was set to 100.

3.4 Design problem setup

The design problem concerns a simplified wing shape optimization of a Cirrus SR22 wing. The Cirrus SR22 is a single-engine 4-seat aircraft with a maximum take-off mass of 1633 kg. A full overview of the parameters of this aircraft can be found in Table 13 in Appendix B. The design problem is setup to be able to be solved using low fidelity solvers and empirical formulas appropriate for the conceptual design phase.

Design problem definition The objective of the design problem is to maximize the $\frac{C_L}{C_D}$ for the cruise condition. The optimization objective is given in Eq. 1, its constraints g_i are discussed in an upcoming section.

$$\begin{aligned} &\max\ f(x) = [\frac{C_L}{C_D}]_{cruise} \\ &s.t.\ g_i(x)\ \leq\ 0,\ i = 1, 2, .., 9 \end{aligned} \tag{1}$$

The design vector $\vec{dv}$ consists of the following parameters: wing span b, root chord c_{root}, kink chord c_{kink}, tip chord c_{tip}, sweep angle Λ_{LE}, span-wise kink location y_{kink}, Z-location of the tip chord z_{tip}, twist angle ϵ and chord thickness t/c, see Eq. 2. In Fig. 2, a diagram is shown of the isolated wing planform. The wing of the SR22 aircraft is used as the design vector at the start of the optimization. The parameters of this wing can be found in Table 9 in Appendix B.

$$\vec{dv} = [b, c_{root}, c_{kink}, c_{tip}, \Lambda_{LE}, y_{kink}, z_{tip}, \epsilon, t/c] \tag{2}$$

Calculations During the optimization, the maximum take-off mass of the reference aircraft minus the weight of its wings is kept constant. During the optimization, the mass of the wing is calculated using the Raymer wing weight equation [31], see Eq. 3. This equation is calculated using the ultimate load factor n_z, the dynamic pressure at cruise flight q, the total mass of the aircraft m_{tot} and the fuel mass m_{fuel} of the SR22 aircraft. An equivalent trapezoidal wing is created on the basis of the design vector that is used solely for the empirical weight calculations [31]. The following parameters are determined for the equivalent wing: surface area S_{eq}, aspect ratio AR_{eq}, taper ratio λ_{eq}, quarter chord sweep $\Lambda_{C/4}$ and chord thickness t/c_{eq}.

[2] `https://pypi.org/project/Indago/0.2.2/`, accessed on 23/10/2023

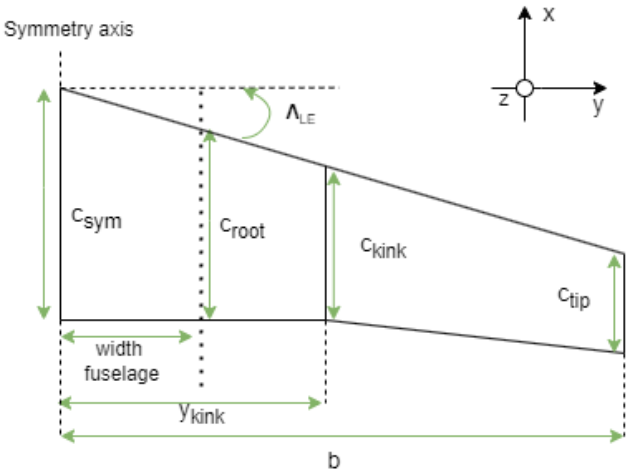

Figure 2: *Planform of the isolated wing used in the design problem setup, including the fuselage-covered section.*

$$\begin{aligned} m_{wing} = 0.036\, S_{eq}^{0.758} m_{fuel}^{0.0035} (\frac{AR_{eq}}{cos^2(\Lambda_{C/4})})^{0.6}\, q^{0.006} \\ \times \lambda_{eq}^{0.04} (\frac{100\,(t/c)_{eq}}{cos(\Lambda_{c/4})})^{-0.3} (n_z m_{tot})^{0.49} \end{aligned} \tag{3}$$

The cruise lift coefficient $C_{L_{cruise}}$ is determined using the mass of the reference aircraft combined with the estimated mass of the wing, see Eq.4. The Athena Vortex Lattice (AVL) solver is used to determine the induced drag C_{D_i} at this lift condition [32]. In order to strike a balance between computational efforts and accuracy, the wing is divided into 8 chord-wise and 22 span-wise elements.

$$C_{L_{cruise}} = \frac{m_{tot}}{q\, S_{wing}} \tag{4}$$

An empirical equation by Raymer is used to determine the profile drag coefficient C_{D_p} [31]. The profile drag is calculated using the friction drag coefficient C_{D_f} and the form factor FF, see Eq. 5.

$$C_{D_p} = C_{D_f} * FF \tag{5}$$

The form factor is calculated to determine the pressure drag. The form factor is dependent on the chord thickness, cruise Mach number M, the point of maximum chord thickness x/c_{max} and the sweep angle at maximum chord thickness $\Lambda_{t_{max}}$, see Eq. 6.

$$\begin{aligned} FF = [1 + \frac{0.6}{x/c_{max}} t/c + 100(t/c)^4] \\ \times [1.34 M^{0.18}\, cos(\Lambda_{t_{max}})^{0.28}] \end{aligned} \tag{6}$$

Due to the presence of the kink, the wing is span-wise divided into two sections in order to calculate the friction drag coefficient of each wing section. The inboard section consists of the wing section from the symmetry axis up to the kink chord. The outboard section consists of wing section from the kink chord to the tip chord. The

friction drag coefficient is determined using the friction coefficient $C_{f_{section}}$ and wetted surface area $S_{wet,section}$ of both sections of the wing, see Eq. 7.

$$C_{D_f} = \frac{S_{wet}}{S_{wing}} \frac{C_{f_{inboard}} S_{wet,inboard} + C_{f_{outboard}} S_{wet,outboard}}{S_{wet}} \tag{7}$$

The friction coefficient of both wing sections are calculated using the friction coefficient of each chord $C_{f,chord}$, see Eq. 8. The chord friction coefficients are determined using the transition points X_{tr}, the chord length c and the Reynolds number Re at cruise condition, see Eq. 9. The transition point is set at a fixed location on the airfoil. On the basis of the average angle of attack at the cruise condition, the transition point was set to 30 percent of the top chord and 60 percent of the bottom chord.

$$\begin{aligned} C_{f,inboard} &= \frac{C_{f,sym} + C_{f,kink}}{2} \\ C_{f,outboard} &= \frac{C_{f,kink} + C_{f,tip}}{2} \end{aligned} \tag{8}$$

$$\begin{aligned} \frac{X_0}{c} &= 36.9\,(\frac{X_{tr}}{c})^{0.625}(\frac{1}{Re})^{0.375} \\ C_{f,chord} &= \frac{0.074}{Re^{0.2}}\,(1 - \frac{X_{tr} - X_0}{c})^{0.8} \end{aligned} \tag{9}$$

The total drag C_D is calculated by adding the profile drag to the induced drag calculated by AVL, see Eq. 7. The objective value is determined using this drag coefficient and lift coefficient at cruise condition previously discussed.

$$C_D = C_{D_f}\, FF + C_{D_i} \tag{10}$$

Geometry constraints The constraints are divided into geometric and design constraints. The geometric constraints are chosen to reject unacceptable wing shapes, like reverse tapered wing sections, see Eq. 11. In order to speed up the optimization, the aerodynamic evaluation is skipped when any of the constraints are not met and the wing design is rejected.

$$\begin{aligned} g_1 &= c_{kink} - c_{root} \\ g_2 &= c_{tip} - c_{kink} \\ g_3 &= 0.5\, c_{root} - c_{kink} \\ g_4 &= y_{kink} - 0.75\, b/2 \\ g_5 &= 0.25\, b/2 - y_{kink} \end{aligned} \tag{11}$$

Design constraints The design constraints are chosen to limit the resulting wing shapes to be more realistic. The constraints are picked with the low fidelity of the optimizer in mind. The mass of the wing is constrained to be within a range of 13 to 27 percent of the empty weight m_{empty} of the reference aircraft [33].

For the volume constraint, the fuselage-covered section of the wing is excluded from the wing volume. This is done to ensure that the wing volume is able to store the required fuel and flight systems in the wings. The wing volume V_{wing} is estimated using the volume of frustum formula. This formula determines the volume of a body with two parallel planes. The volume of each wing section is calculated individually using the area of the relevant chords A_{chord}, and the span wise length of both sections, see Eq.12. The span wise length of the inboard section b_1 extends from the root chord to the kink chord. The span wise length of the outboard section b_2 extends from the kink chord to the tip chord. The volume constraint condition scales with the total mass of the aircraft relative to the reference aircraft. This constraint serves as a penalty for increasing the wing weight.

$$\begin{aligned} V_{inboard} &= \frac{2\,b_1}{3}[A_{root} + A_{kink} + \sqrt{A_{root}\,A_{kink}}] \\ V_{outboard} &= \frac{2\,b_2}{3}[A_{kink} + A_{tip} + \sqrt{A_{kink}\,A_{tip}}] \\ V_{wing} &= V_{inboard} + V_{outboard} \end{aligned} \tag{12}$$

The wing loading (W/S) constraint is created to ensure the designed wing belongs to the same aircraft category as the reference wing. For this constraint, the wing is able to increase the wing loading by 10 percent relative to the reference aircraft. The design constraint equations are listed in Eq. 13.

$$\begin{aligned} g_6 &= m_{wing} - 0.27\, m_{empty} \\ g_7 &= 0.13\, m_{empty} - m_{wing} \\ g_8 &= \frac{m_{tot}}{m_{tot,ref}}\, V_{wing,ref} - V_{wing} \\ g_9 &= (W/S) - 1.1\,(W/S)_{ref} \end{aligned} \tag{13}$$

3.5 PPO environment design

The DRL framework is setup to solve the optimization problem in a gradual manner. Complex optimization problems are typically setup to be solved iteratively. The gradual exploration of the design space encourages the DRL framework to avoid violating the constraints and work towards finding the global optimum [12, 13, 34]. For the selected design problem, the objective function can be evaluated at each step. This means that many reward signals can be determined which should increase the learning performance [35]. If the DRL framework is trained to complete the optimization within a single step, it could exclusively generate constraint violating designs when a shift to the design space occurs. An iterative optimization framework could therefore also exhibit increased TL capabilities.

It is important to understand that the PPO framework will not maximize the objective value directly but instead tries to maximize the cumulative reward it collects within the environment. The shaping of the rewards is arguably the most important part of the DRL framework [35]. Reward shaping is typically a trial and error process [36]. The rewards structure within the environment

should be constructed to reflect the desired optimization behaviour. If this is not done properly, the DRL algorithm could learn to exploit the environment. This has the potential to degrade the generated results [37, 38]. There are many different strategies to shape the rewards, as a consensus has not been established [39].

Termination condition The agent is only able to end the optimization when it has collected a certain number of termination counters. The environment includes 3 separate termination conditions for each design constraint and a termination condition for failing to improve the objective value. During the optimization, termination counters are collected when a termination condition occurs. These counters only increase when each subsequent step of the optimization triggers the same termination condition. The termination counters are reset once a new state complies with its respective condition. The termination counter limit for the design constraints is 25. The termination counter limit for not improving the objective value is 50. These counters are chosen to help the agent avoid the constraints and generate better designs.

Reward shaping The rewards of the environment are split into gains and penalties. The gains have a positive value and are generated to encourage good behaviour by the agent. The penalties have a negative value and are used to deter the agent from exhibiting bad behaviour. The gain R_1 is generated when the agent is able to find a new maximum objective value. As for most optimization problems, the optimization problem becomes increasingly more difficult to improve, the closer it gets to the global optimum. To alleviate this, the gain scales exponentially on the basis of the difference between the objective value at the current step f_i and the starting objective value f_{start}, see Eq.14. This encourages the agent to continue finding higher objective values.

$$R_1 = [20\,\frac{f_i - f_{start}}{f_{start}} + 1]^3 \tag{14}$$

This gain does not scale on the basis of how much the objective value has increased. This could have negative effects as the agent could decide to farm small rewards by improving the design as little as possible, thus generating a high cumulative reward. However, the agent can also learn how to exploit this. The agent can learn to increase the objective value while violating the design constraints. This means it will not collect a gain until it has reached its maximum objective value. This results in the agent exclusively collecting one disproportionately large gain.

Once the agent terminates the optimization, it collects the final gain R_2. This final gain also scales exponentially in a similar manner to R_1. However, now the maximum objective value reached during the optimization f_{max} is compared with the starting condition, see Eq.15.

$$R_2 = [\frac{f_{max} - f_{start}}{f_{start}}\,28 + 1]^4 \tag{15}$$

The collected penalties have a negative value and are generated when the agent is unable to find a higher objective value. When the design at a step of the optimization would violate the design constraints, a penalty is awarded. These penalties should train the agent to comply with all constraints as it searches for higher objective values [39]. P_1 and P_2 are the penalties generated when the wing weight is either too high or too low, respectively. P_3 is the penalty generated when the volume of the wing is too low. P_4 is the penalty generated when the wing loading exceeds the upper bound , see Eq.16.

$$\begin{aligned}
P_1 &= -[6\frac{m_{wing,i} - 0.27\,m_{empty}}{0.27\,m_{empty}} + 1]^2 \\
P_2 &= -[3\frac{m_{wing,i} - 0.13\,m_{empty}}{0.13\,m_{empty}} + 1]^2 \\
P_3 &= -[5\frac{\frac{m_{tot,i}}{m_{tot,ref}}\,V_{wing,ref} - V_{wing,i}}{\frac{m_{tot,i}}{m_{tot,ref}}\,V_{wing,ref}} + 1]^2 \\
P_4 &= -[5\frac{(W/S)_i - 1.1\,(W/S)_{ref}}{1.1\,(W/S)_{ref}} + 1]^2
\end{aligned} \tag{16}$$

A set of additional penalties is created for when the design complies with all constraints but was unable to find a new maximum objective value. These penalties use the objective value of the previous optimization step to tailor the penalty signal, this technique is known as reward saltation [40]. These penalties are constructed to help the agent understand if the action it has taken at the current step is advantageous, relative to the state of the previous step. The penalty P_5 is collected when the objective value has decreased, relative to the previous step. Penalty P_6 is collected when the objective value at the current step has not changed. Penalty P_7 is collected when the objective value has increased relative to the previous step. The equations for these penalties can be found in Eq. 17 together with the corresponding parameters in Table 1. A relatively small penalty P_8 of -0.05 is awarded when the current design is stuck at the optimum design. This is done to prevent the agent from unnecessarily delaying the termination of the optimization.

$$P_{5,6,7} = [\frac{f_i - f_{i-1}}{f_{start}}X_1] + X_2 \tag{17}$$

-	X1	X2
P5	10	-0.4
P6	5	-0.2
P7	1.5	-0.1

Table 1: *Failure penalty parameters*

The coefficients in all the reward equations are selected by estimating the cumulative gain collected during a single optimization. The cumulative gain is then balanced with the expected cumulative penalties to ensure that the agent will be properly rewarded on the basis of the reached objective value. Thus the setup of

the reward signals requires knowledge of the design space.

Interaction between the agent and environment At the start of the optimization the environment is reset. This includes setting the state S_i to the starting design vector, and setting all termination counters to zero. The first thing the agent does in the environment is determine an action a_i. The action vector generated by the model at each step is a vector equal to the length of the design vector and contains values between 1 and -1. Eq. 18 shows how each design variable j is altered at each step, on the basis of the suggested action from the agent. The action is calculated using the upper bound j_{UB} and lower bound j_{LB} of that design variables. The design variable changes are then dividing by a factor of 75. This should increase the stability of optimization, as the agent is limited in how much it can change the design vector at each step.

$$S_{i+1}[j] = S_i[j] + a_{i+1}[j] \frac{j_{UB} - j_{LB}}{75} \tag{18}$$

The new state is evaluated to determine if it complies with all geometry constraints. If the wing design does not comply with these constraints, the design vector is adjusted by setting the design variables equal to the bounds of the violated constraints.

Subsequently, the design is evaluated on the basis of the design constraints. If the state does not comply with all constraints, it ends the optimization step and generates a penalty (P_1, P_2, P_3, P_4) and a termination counter is awarded on the basis of the violated constraint. When the state complies with all constraints, the new objective value is calculated. If the new objective value is not higher than the maximum objective value it has currently reached, a penalty (P_5, P_6, P_7, P_8) is generated, a termination counter is awarded and the agent exists the optimization step. When a new maximum objective is reached a gain R_1 is generated and the agent exits the optimization step. The agent will continue to find better wing designs until enough termination counters are collected and terminates the optimization process. Finally, the sum of all rewards and penalties are used for the model to evaluate its performance and improve its policy networks. A flow chart is created of the setup of the environment, see Fig. 3.

3.6 Transfer Learning techniques

In this paper three TL techniques are compared. The first technique takes the best trained model used to optimize the wing of the SR22 aircraft and optimizes the wing of a different aircraft. This technique requires no retraining and is therefore computationally efficient. The second technique involves copying the best trained SR22 model and directly training it for 500 000 training steps to design a wing for a new aircraft. This technique is referred to as continual learning (CL). The third technique is a more traditional TL technique where the best

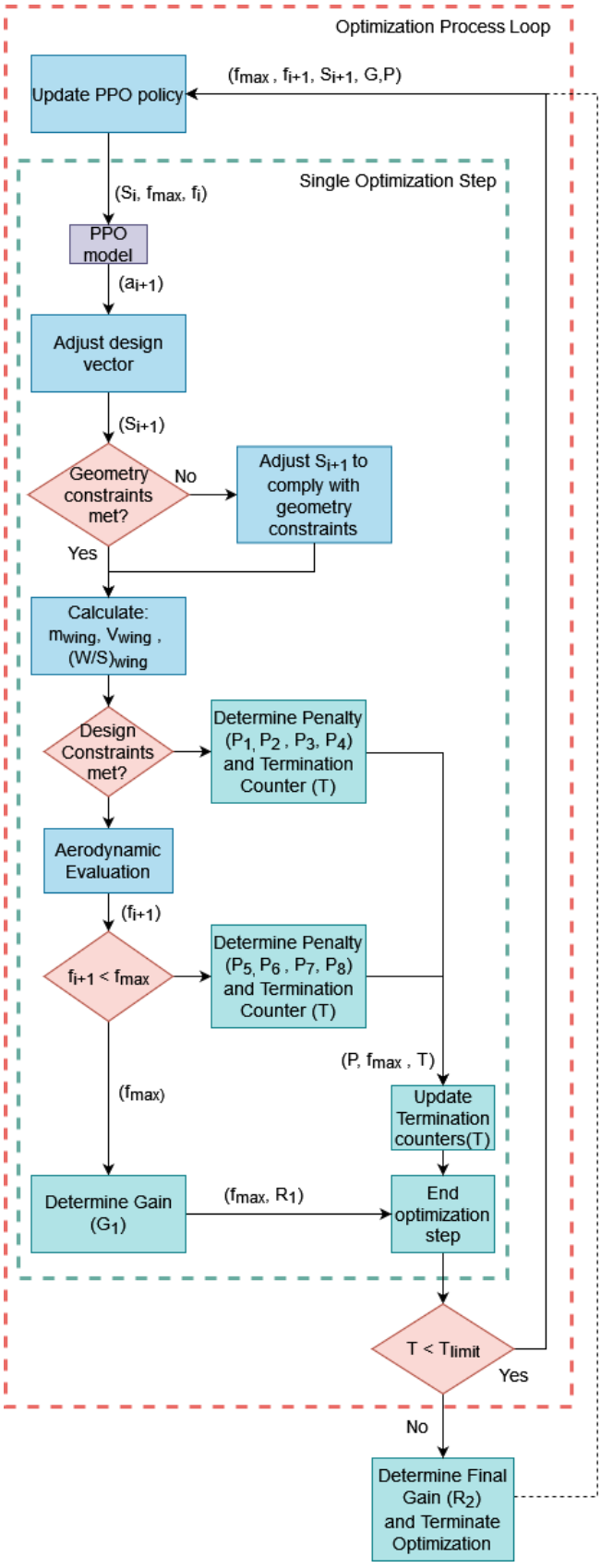

Figure 3: *Flow chart of the PPO environment.*

trained SR22 model is copied and the values of the original layers of both ANNs are locked during training. A new hidden layer of 64 neurons is inserted into the policy and value network of the model. This model will then be trained for 500 000 training steps to optimize the wing for the new aircraft, see Fig. 4. The results of all three techniques are evaluated to determine if the TL techniques were able to increase the computational efficiency of the PPO framework.

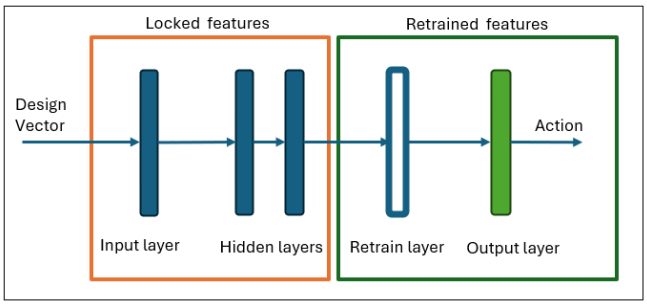

Figure 4: *Diagram showing the application of transfer learning to an ANN architecture.*

4 Results

The results are divided into three sections. The first section studies the training phase of the PPO model, selects the best performing model and evaluates the environment set-up. The second section evaluates and compares the performance of both optimization frameworks. Lastly, the TL capabilities of the model are discussed.

4.1 Training performance

The training phase is studied in order to select the best performing model. The environment set-up is evaluated by analyzing the training performance of the PPO framework. The generated rewards are studied to determine if they are well calibrated to the design problem.

General training progress First, we investigate the average total rewards collected during a single optimization of the training phase, see Fig. 5. The performance climbs quite rapidly at the start of the training phase. This is because there is most room for improvement here. Past this initial climb, a constant increase in performance can be observed. Up until 1 270 000 training steps the rewards steadily increase, until it starts to stabilize. It is estimated that the performance will degrade when trained further, as this is quite common for RL algorithms [29]. This model is selected as the best performing PPO model and will be referred to as the final model from this point onwards. The results in the previous and upcoming sections are generated using this final model.

Performance at progressive training steps The model is evaluated at various training steps to study the progression of its performance. In Fig. 6, the performance

Figure 5: *The training phase performance of the PPO model showing the mean rewards per optimization vs training steps. The selected model is indicated by the blue line.*

of 500 optimizations by the PPO model is plotted at increasing training steps. The standard deviation is indicated by the shaded parts. At a low number of training steps, the model is able to reach respectable objective values. However, at 10 000 training steps the consistency of the model is quite poor. The time required to train the model for 10 000 training steps is roughly equivalent to a single PSO optimization. As the training phase progresses, both the consistency and objective values steadily increase. It takes about 20 hours of training time for the PPO model to reach the same average objective value as the PSO framework.

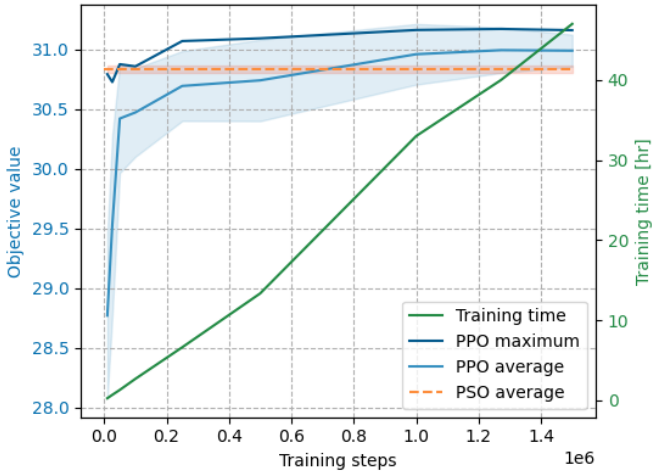

Figure 6: *PPO framework performance at an increasing number of training steps for the SR22 wing, starting objective = 28.08.*

Fig. 7 shows the planforms with the highest objective values made by the PPO framework at a progressive number of training steps. At 10 000 training steps the model has already learned to maximize the wing span. Interestingly, the model first tries to create a tapered wing planform. However, at around 250 000 training steps the optimized wing starts to progressively become more rectangular. This is further explored in the upcoming sections.

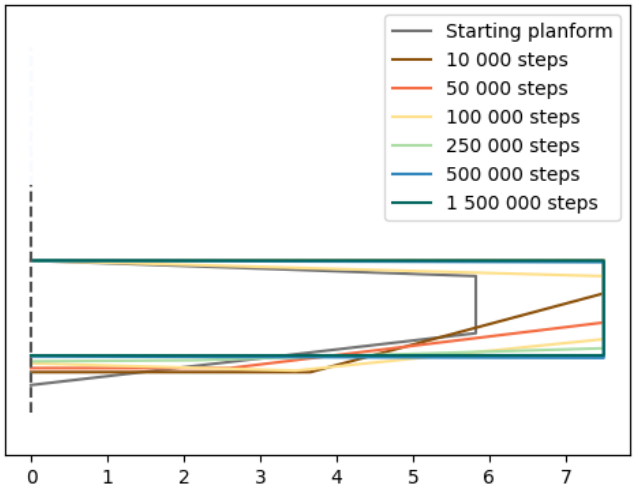

Figure 7: *The best semi-planforms generated by the PPO framework at progressive levels of training.*

Reward evaluation of the environment The performance of the PPO optimization is intrinsically tied to the reward generation of the environment. Fig. 8 illustrates the relation between the cumulative rewards collected by the fully trained agent and the objective values reached. In this graph, the gains and penalties collected during a single optimization are plotted together with the final objective value it reached. The graph shows that a higher cumulative reward scales exponentially with the reached objective value. The environment is well calibrated to the design problem as reaching a higher objective should result in higher rewards. This graph also illustrates how the results are denser at the higher objective values. This demonstrates the importance of a reward system that scales with the optimization values. The model is encouraged to reach the highest objective value possible at every optimization.

Figure 8: *Rewards obtained from the environment vs objective value reached by the final model.*

For the final model, the penalties collected do not increase with the higher objective values reached. The reason why the penalties do not increase is because most of the penalties are collected when the model terminates. This means that the model is able to consistently optimize the wing while avoiding to violate the constraints. However, this is not the case when the model is not fully trained yet. This is further explored in the upcoming sections.

A similar graph is created for the model trained for 10 000 training steps, see Fig. 9. We can see that at 10 000 steps the agent has not been sufficiently trained. The agent has a hard time optimizing the wing as the results are spread out over the entire range of the reached objective values.

Figure 9: *Rewards obtained from the environment vs objective value reached at 10 000 training steps.*

At 100 000 steps the agent seems to have learned how to consistently optimize the tapered wing planform, see Fig. 10. This can be seen by the density of the results around the highest reached objective values. Interestingly, the PPO framework still tried to look for more favourable wing shapes, even while trying to optimize the tapered wing shape as much as possible.

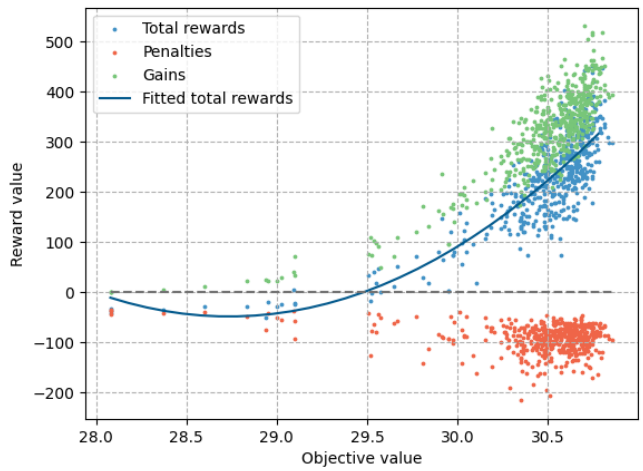

Figure 10: *Rewards obtained from the environment vs objective value reached at 100 000 training steps.*

Progression of a single optimization Fig. 11 shows the gains and penalties collected throughout one optimization. These rewards are normalized with respect

to the total rewards collected. The collected gains exponentially increase at increasing objective values. The flat parts of the objective value and gain graphs start at the same step and are due to the model not increasing the objective value. The spike at the end of the gain graph is due to the gain R_2 obtained for reaching the final objective value. The agent seems to explore the environment in a focused manner in order to reach the optimal wing design. It does not wander into sections of the design space that would not generate a favourable wing. The behaviour shown in the graphs seems consistent and suggest that the model has not learned to exploit the environment in an understandable way.

These graphs also illustrate how the model terminates the optimization. At some point the agent is unable to optimize the wing further, indicated by the dashed grey line. At this point it starts to collect termination counters until the optimization concludes. It is unclear if the agent is aware that it cannot optimize the wing further and tries to terminate itself as efficiently as possible. In the environment, the total collected gains are about 10 times larger than the penalties. In order to reach a high objective value consistently, the positive rewards are more important and should significantly outweigh the penalties.

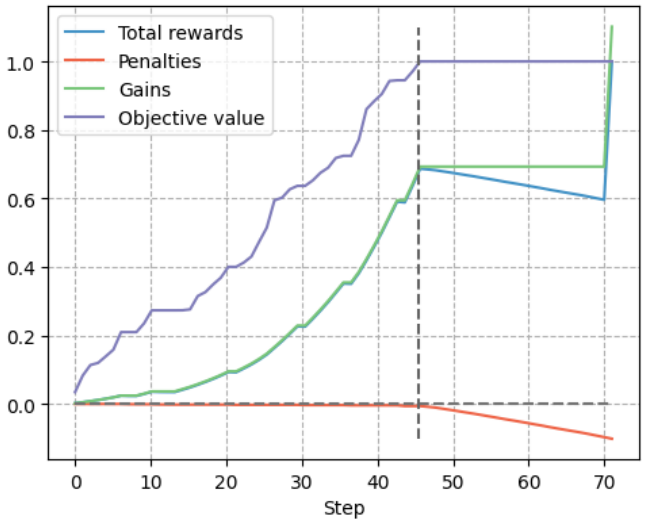

Figure 11: *Normalized rewards obtained from the environment during a single optimization by the final model, objective value reached: 31.02.*

The results from the agent trained for only 10 000 training steps show quite poor performance, see Fig. 12. We see that the cumulative penalties and gains are almost equal. The penalties are collected throughout the entire optimization, thus the agent has not learned how to efficiently optimize the wing. This can also be seen by the optimization step at which the model has found its optimum wing. This model took 17 more steps to reach the final wing design compared to the final model. The final gain R_2 helps the agent reach a positive cumulative reward, encouraging it to optimize the wing despite the struggle.

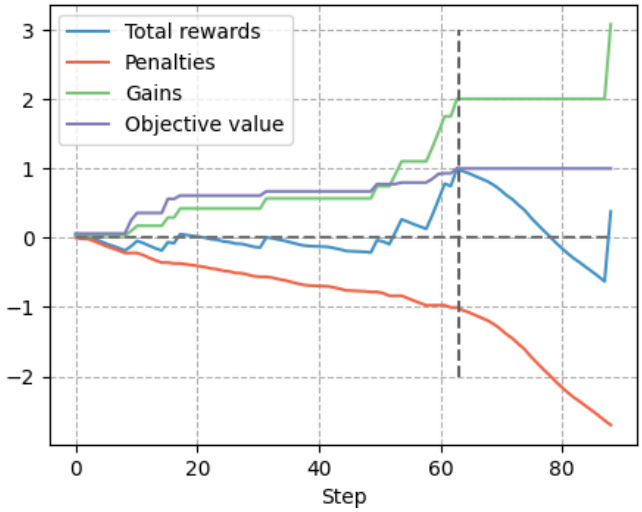

Figure 12: *Normalized rewards obtained from the environment during a single optimization at 10 000 training steps, objective value reached: 29.52.*

The results from the agent trained for 100 000 training steps show better behavior, see Fig. 13. Here the agent has learned how to mostly avoid the penalties at the start of the optimization. However, it still is not very efficient, as it struggles to find a better wing at around 30 steps. This model finds the optimum wing at optimization step 56. This is to be expected, as it finds the wing in fewer optimization steps than the model with fewer training steps but requires more steps than the final model. At 100 000 training steps, the cumulative gains start to outweigh the penalties more.

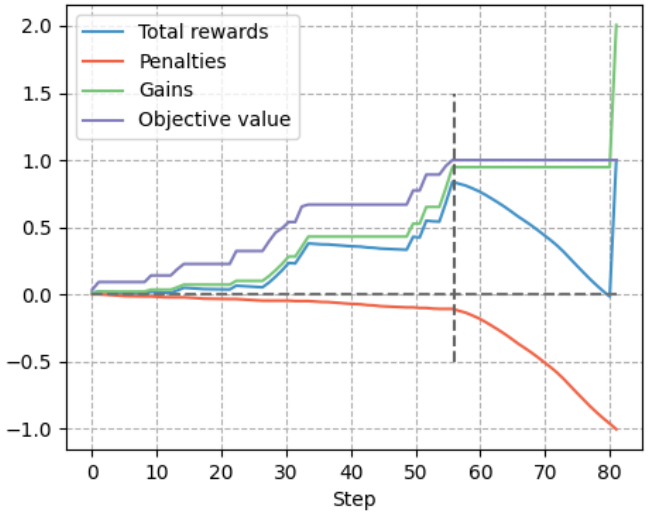

Figure 13: *Normalized rewards obtained from the environment during a single optimization at 100 000 training steps, objective value reached: 30.55.*

From these results we conclude that the penalties are more important when the model first starts the training phase. These penalties will help the agent learn how to navigate the design space to avoid violating the constraints and find better wing shapes. However, once the model is fully trained, it has learned how to avoid the

penalties all together and only collects penalties during termination.

Constraint progression of the optimization In Fig. 14, the constraint values from the same optimization as the previous section are plotted. In this graph, it is clear that the volume constraint is guiding the optimization. While the other two design constraints are not violated at all, the volume constraint is what caused the optimization to terminate. At the start of the optimization, the volume constraint is almost violated, as this constraint is related to the starting condition. At around optimization step 45, the framework fails to improve the wing and starts to accumulate penalties by violating the volume constraint. This is also the step at which the agent has found the optimal wing design. Thus the model has learned to avoid the constraints until the optimal wing is found. The performance of the PPO framework could be further improved by lowering the termination counter limit. This could reduce the time per optimization of the trained model.

Figure 14: *Constraint values obtained during a single optimization by the final model, objective value reached: 31.02.*

Similar graphs are created for the model trained for 10 000 and 100 000 training steps respectively, see Fig. 15 and Fig. 16. At 10 000 training steps, the model clearly has not learned how to properly avoid the constraints as they are violated at various optimization steps. At 100 000 training steps, the agent has learned how to properly avoid the volume constraint much like the final model.

4.2 Optimization performance

In this section the performance of the PPO framework is compared with the PSO algorithm. The main performance parameters that will be focused on are the reached objective values, the consistency of the results and the time per optimization.

General optimization performance In Table 2, the results of 500 PPO and 10 PSO optimizations can be found. Once fully trained, the PPO algorithm was able to reach

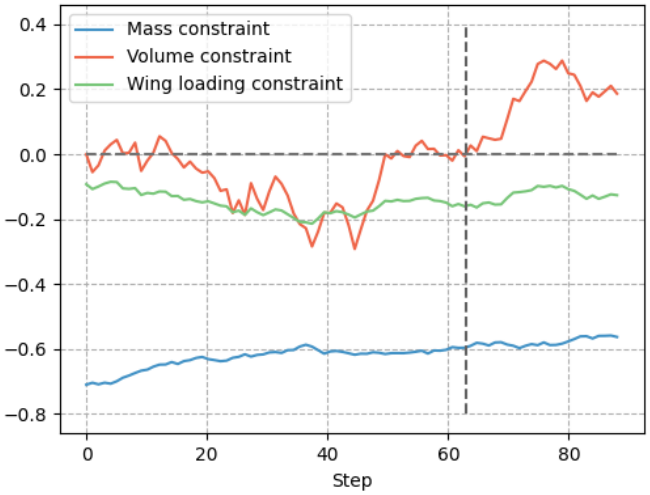

Figure 15: *Constraint values obtained during a single optimization at 10 000 training steps, objective value reached: 29.52.*

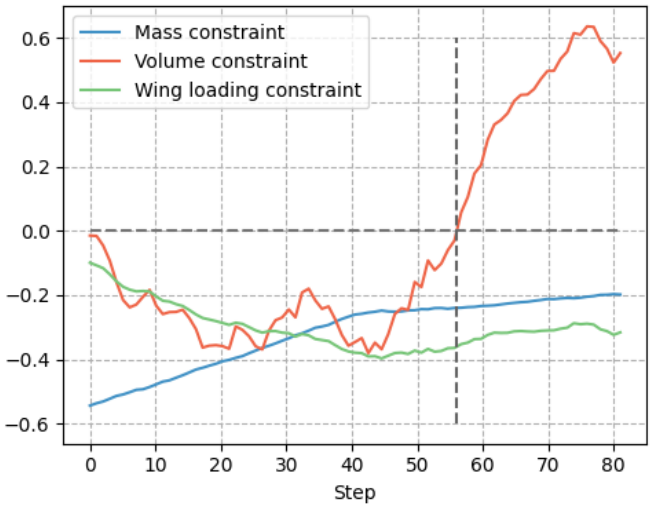

Figure 16: *Constraint values obtained during a single optimization at 100 000 training steps, objective value reached: 30.55.*

a higher average and maximum objective value than the PSO algorithm. The results of the PPO framework were less consistent, as indicated by the standard deviation σ and the difference between the maximum and average objective values. The average time per optimization t_{optim} of the PPO model is 9.45 seconds, while each PSO optimization took over 16 minutes. However, the PPO model required a training time of 1 day and 16 hours to achieve this. In order for RL to become a valid optimization algorithm, its costly training time has to be justified. Due to the variance in the PPO results, multiple optimizations will have to be done to ensure a good result is obtained, this is further explored in the upcoming sections.

Algo	f_{max}	f_{ave}	σ	t_{optim}
PPO	31.173	30.995	0.190	9.45 sec
PSO	30.900	30.840	0.031	16 min 32 sec

Table 2: *Optimization performance of 500 PPO and 10 PSO optimizations for the SR22 aircraft, starting objective = 28.08.*

The solutions from both frameworks are all unique. This means that the design problem is prone to local maxima. Because of this, the PSO framework was unable to fully converge. Increasing the number of iterations of the PSO algorithm was computationally costly, while very minimal improvements to the objective value were gained. To prevent a costly PSO optimization to be stuck in a local optima, the decision was made to generate multiple PSO optimizations.

In Fig. 17, the best semi-wing planforms created by the PPO framework are plotted and compared with the best PSO planform. For this comparison the t/c , ϵ and z_{tip} could not be visualized in the planforms. These variables are studied in the upcoming sections. From Fig. 17, it can be seen that both algorithms generate completely different shapes. The PPO framework consistently makes a rectangular wing, while the PSO framework makes a wing with a tapered outboard section. The reason why the PPO algorithm is most likely able to outperform the PSO algorithm is because it consistently finds a more efficient wing shape. The reason why such a planform generates a higher objective value is most likely due to the aerodynamic solvers and the interaction between the weight estimation formula and volume constraint. The PSO optimization should theoretically be able to create a rectangular wing. It is presumably unable to do so due to the nature of how it explores the design space and alters its particles. This illustrates the direct impact the optimization strategy can have on the results.

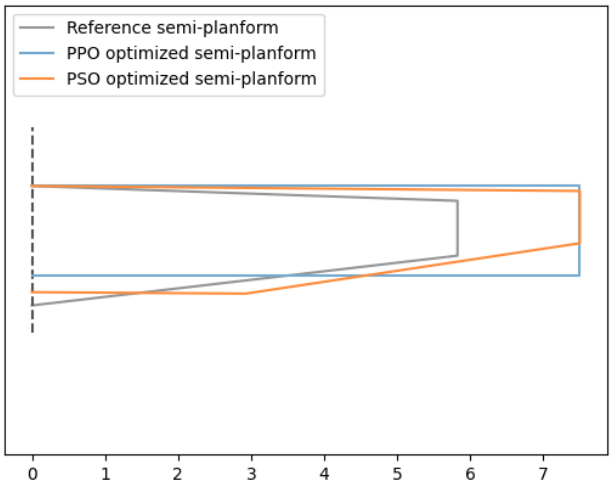

Figure 17: *Comparison of the optimized semi-wing planforms shapes with the highest objective values for the SR22 wing.*

Variance in results The consistency of the PPO results differs from that of PSO. The PPO algorithm makes predictions on how to improve the wing, while the PSO framework directly calculates how to most effectively alter its particles at each step. This means that a higher variance in the results from the PPO framework is to be expected.

As the results of the PPO algorithm can vary quite a bit, a distribution is made of the objective values including the average PSO objective value, see Fig. 18. The distribution shows that the PPO framework outperforms the PSO framework quite consistently. Again, this is most likely due to the fact that it is able to find a more favourable planform shape. The figure also shows the variety of results that the PPO algorithm produces. Due to the stochastic nature of the PPO algorithm, it sometimes fails to avoid the constraints and terminates the optimization at quite an early step. This means that in order to utilize the PPO algorithm for a framework, multiple optimizations would have to be performed to ensure good results. Out of the 500 PPO results, 445 were higher than the average PSO result. The PPO framework has a 89% chance to reach a higher objective value than the PSO. Running the PPO optimization three times will have a 99.9 % chance to reach a higher objective value than the PSO. This means the PPO framework can generate comparable results to the PSO framework in just 28.35 seconds, which is almost 35 times faster. In situations where the design space includes many local optima, the PPO framework could prove useful, as it is able to rapidly generate unique solutions.

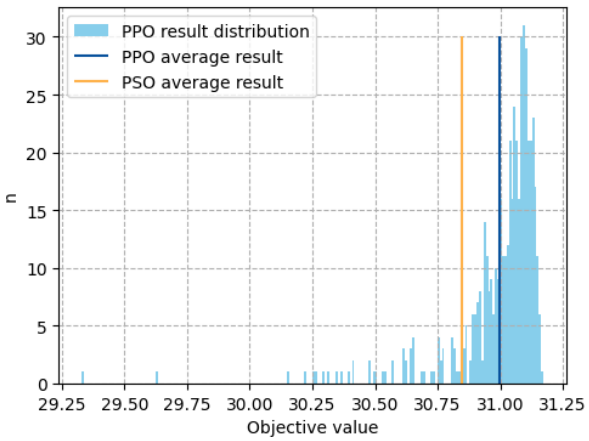

Figure 18: *Distribution of the PPO objective values (blue) compared with the PSO average objective value (orange) for the SR22 wing.*

A box plot is created of each individual design variable, as can be seen in Fig. 19. The first figure shows the variables from the 500 PPO optimized wings, while the figure below shows the results from 10 PSO optimized wings. The box plots are normalized with respect to the boundary values of each respective design variable. This illustrates how both frameworks changed the design variables relative to the starting wing planform, indicated with green lines in the figure. These box plots show which design variables are more prone to variation, and could illustrates which variables give the frameworks the most difficulty. There is some commonality between the frameworks. Both frameworks consistently maximized the wing span and removed the Λ_{LE} from the wing. As indicated by the planforms, the chord lengths between frameworks differ signifi-

cantly. Both frameworks lowered the t/c and z_{tip}. When looking at the box plot of the t/c optimized by PPO, it appears that there is quite a bit of variance in this. This is quite interesting, as the chord thickness heavily influences the wing weight and the wing volume, which directly appear in the constraints. The PPO framework effectively removes the kink in the wing, thus the actual span-wise location of the kink will have no impact on the shape of the wing. The y_{kink} variable has therefore many outliers.

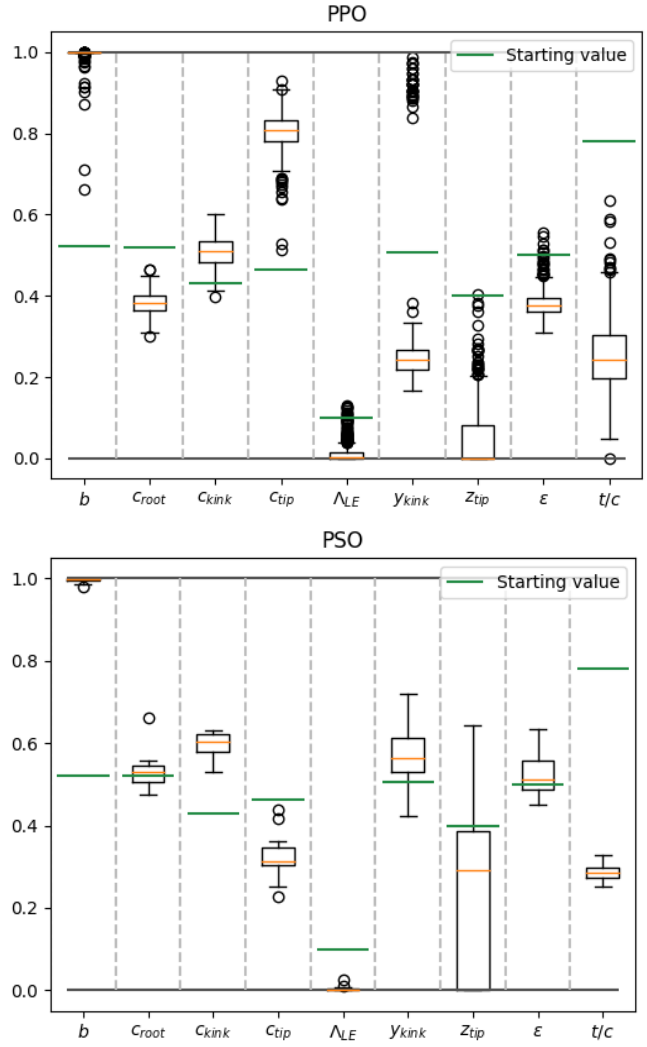

Figure 19: *Box plot of each design variable for the PPO and PSO optimized wings, normalized with respect to the bounds of each variable. The starting variables are indicated in green.*

4.3 Transfer Learning

In this section the ability of the model to adapt to a new design space is measured. Additionally, the TL and CL techniques are applied and evaluated. The aim of this section is to explore techniques that allow the reuse of a model to circumvent the costly training phase.

Resilience of the trained model In this section we measure the ability of the model to adapt to small changes to the design space. To achieve this, the total weight of the reference aircraft is incrementally changed. The model is then used to optimize the wing without performing any retraining. In Fig. 20, the results of 200 optimizations at each weight increment are plotted. PSO results are also generated as a baseline, indicated in orange. The results show that the PPO model is able to outperform the PSO algorithm for most of the weight increments. The performance of the model only starts to degrade when the weight is decreased significantly. The starting and optimized objective value changes more when the total weight is decreased. This indicates a more significant shift in the design space. The trained model can be repurposed to solve new design problems, as long as the changes to the design space are limited.

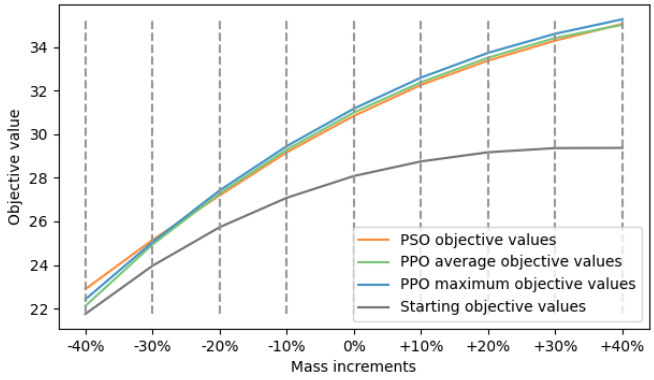

Figure 20: *Resilience of the model to alterations in design space by incrementally changing the total weight of the reference aircraft, compared with PSO results in orange.*

These results are quite promising, as this could prove useful for the conceptual design phase. As with interdisciplinary problems, small changes to the design are frequently made by each department, which means that each department has to re-evaluate the design and possibly start a new optimization process. A pre-trained model could be used to rapidly generate new solution for an interdisciplinary design problem. Using a RL model could significantly improve the efficiency of any problem that contains multiple separate optimization processes in its design problem.

General performance The optimization performance is highly dependent on both the created environment including its calibrated rewards, and the inherent design space of the design problem. In order to further test the adaptability of the model, the design space is altered by using the parameters of different aircraft types.

First, the wings are optimized for these different aircraft using the final SR22 model. For the application of the TL and CL techniques, the final SR22 model is retrained for 500 000 training steps. In order to evaluate the results, a completely new model is trained for 1 000 000 training steps for each aircraft. Additionally, PSO optimization is used to optimize the wings of each aircraft for comparison.

The selection of aircraft is limited by the aircraft that can be analyzed using AVL. This means that transonic

aircraft and unconventional wing shapes are avoided. First the Douglass DA50 is selected as this aircraft has a similar size and shape to the Cirrus SR22 reference aircraft. The Fokker F50 is selected as this aircraft is significantly larger and faster than the reference aircraft, thus significantly changing the design space. Lastly, the ASK21 glider is selected. This aircraft is very light and flies at a low speed, thus it should change the design space in a different way than the F50. The parameters used for these aircraft can be found in Table 13 in Appendix B. The environment rewards are not re-calibrated to better fit these new design spaces. This is done to better measure the total adaptability of the original framework setup and would require no knowledge of the adapted design space.

In Table 3, the results of 500 DA50 wing optimizations using the TL models are listed. Due to the variance of the models, the time to obtain an average or greater objective value t_{ave} is given. This is calculated by determining the number of results required before the probability is greater than 0.99 that one of those results is at least as good as its average value.

The performance of all models is quite good. The maximum objective each model can generate is higher than that of the PSO algorithm. The model that was not re-trained has quite poor consistency compared to the other models. The newly trained DA50 model showed good results, even without re-calibration of the environment. This shows that the environment set-up is somewhat flexible. The results of the models retrained with the TL and CL techniques are quite similar. Interestingly, they both are consistently able to reach a similar objective value as the newly trained model, although the TL model takes significantly longer to reach an average solution. The DA50 model was trained for 27 hours, the CL model was retrained for 15 hours and 15 minutes while the TL model was retrained for 16 hours and 45 minutes. When comparing the training times it shows it is more efficient to perform TL when possible, rather than training a completely new model.

TL method	f_{max}	f_{ave}	σ	t_{ave} [s]
SR22 model	35.22	34.39	0.40	60.6
DA50 model	35.25	35.10	0.11	29.9
DA50 CL model	35.29	35.10	0.16	48.44
DA50 TL model	35.36	35.12	0.17	81
PSO	35.11	35.03	0.037	-

Table 3: *TL results of 500 DA50 wing optimizations, starting objective: 33.20. Average PSO time per optimization: 15 min and 39 sec.*

In Fig. 21, the best DA50 planforms of each model are plotted. The planforms look quite similar to the optimized wings of the SR22. The rectangular PPO planforms almost completely overlap while the PSO framework created a tapered wing again. This could explain why the model performed so well, even without any retraining or re-calibration.

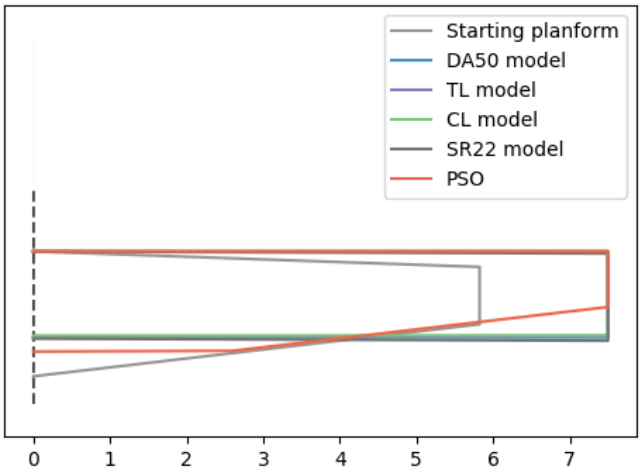

Figure 21: *Optimized DA50 wing planforms using various models and PSO.*

In Fig. 22, the average changes each framework has made to the design variables relative to the starting wing are plotted. The results are taken from 500 optimizations per model. As expected, the PSO results differs the most from the other results. However, the changes it has made to the design vector are more similar to the PPO results than the planform plot would initially imply. The biggest deviation between the models are the z_{tip} and ϵ. The DA50 model even slightly increased z_{tip}, while all other models decreased this. The PPO models all seem to agree on how the chord lengths should be changed, although there is some deviation in the tip chord. This graph illustrates the solutions of the PPO models are somewhat consistent, as all models independently generate similar wing shapes, even the SR22 model.

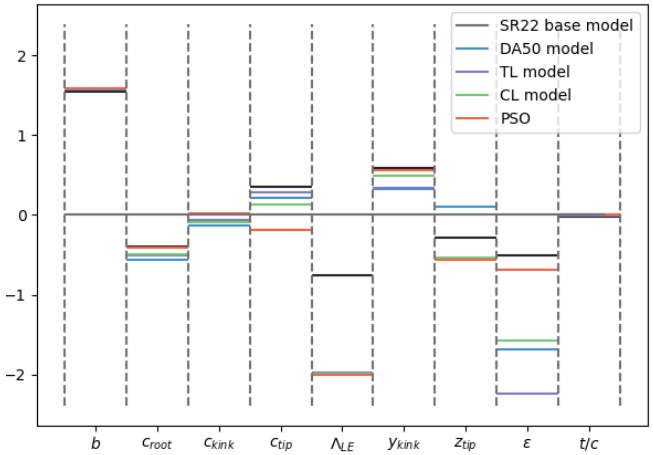

Figure 22: *Design variable changes by the different optimization strategies for the DA50 wing.*

Like with the DA50 results, the models of the F50 show good performance, see Table 4. The F50 model was trained for 26 hours, the CL model was retrained for 20 hours and 36 minutes while the TL model was retrained for 14 hours. The model that did not receive any additional training was able to optimize the wing

significantly, although it takes quite some time to reach an average objective value. The performance of the CL model was rather disappointing, especially when considering the training time. The performance is actually worse than that of the original SR22 model, meaning the continual training actually degraded the model. However, this did not happen with the TL model. Comparing its training time and with the newly trained model, the TL technique is more time efficient than training a completely new model as the time it takes to reach an average objective value is comparable.

TL method	f_{max}	f_{ave}	σ	t_{ave} [s]
SR22 model	38.96	37.95	0.38	137.8
F50 model	39.09	38.83	0.24	34.6
F50 CL model	36.92	36.44	0.11	105.3
F50 TL model	38.98	38.84	0.10	40.2
PSO	39.07	38.90	0.09	-

Table 4: *TL results of 500 F50 wing optimizations, starting objective: 34.91. Average PSO time per optimization: 15 min 39 sec.*

Fig. 23 shows the generated F50 wing planforms with the highest objective values. Here we see that the shapes vary quite a bit. The PSO framework has removed the kink and creates a tapered wing. The SR22 and newly trained F50 model both generated rectangular wing shapes. The SR22 model also increased the wing span to 36 meters, while the original upper bound for the SR22 wing span is 15 meters. The SR22 model has therefore learned to increase the wing span beyond its upper bound, regardless of what the actual length of the wing span is. Interestingly, the TL and CL models both reintroduce the kink into the wing shape.

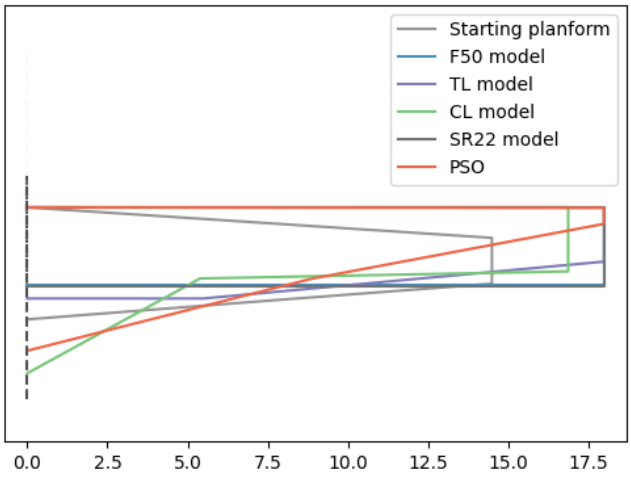

Figure 23: *Optimized F50 wing planforms using various models and PSO.*

The ASK21 model was trained for 30 hours, the CL model was retrained for 3 hours and 46 minutes and the TL model was retrained for 11 hours. The ASK21 models performed quite poorly, see Table 5. This can be explained by inspecting the design space. The PSO algorithm was able to optimize the CL/CD to 27.41, while the objective value starts at 26.50. This means that the total possible increase in objective value is quite low. These values are directly used by the environment to generate the gains. This causes the importance of penalties to outweigh the gains. The model now focuses on minimizing its penalties, rather than collecting positive rewards. If the poor performance was solely due to the change in design space, the newly trained model should have performed comparably to the PSO algorithm. In order to improve the performance of the model, the gain and penalty equations would have to be re-calibrated to better suit this new design space. The ASK21 model was able to perform better than both the CL and TL models, illustrating that the TL methods can actually have a detrimental effect on the training when the design space is altered too much. The time to obtain an average value vary drastically between the TL and CL models. This is because almost every optimization of the TL model failed and thus it takes a long time to reach an average objective value. The CL model only took 9.5 seconds to reach an average objective value. This means that this model more consistently optimized the wing, although not by much looking at the average objective value reached.

TL method	f_{max}	f_{ave}	σ	t_{ave} [s]
SR22 model	26.55	26.50	0.006	15.4
ASK21 model	26.97	26.90	0.037	34.1
ASK21 CL model	26.66	26.58	0.038	9.5
ASK21 TL model	26.54	26.50	0.003	450.8
PSO	27.41	27.21	0.12	-

Table 5: *TL results of 500 ASK21 wing optimizations, starting objective: 26.50. Average PSO time per optimization: 15 min 35 sec.*

Training phase In this section, the training phase of the TL methods for the DA50 wing optimization are studied. In Fig. 24, the average rewards collected during the training phase are plotted. The newly trained model shows expected behaviour, as it steadily improves with the increasing training steps. The performance of the TL model actually degrades at the start. This makes sense as the model now contains a completely new untrained layer. At around 30 000 training steps it starts to drastically improve until it reaches the same performance as the other two models. The CL and TL illustrate how the TL methods could be computationally advantageous as they reach a higher average reward at about half the training steps.

The training performance of the DA50 TL model is evaluated at an increasing number of training steps, starting from the 1 272 000 steps of the final SR22 model. At intervals of 100 000 steps, 100 wing optimizations are performed and the results are listed in Table 6. At 1 300 000 steps, the model performs quite poorly as discussed in the previous section. At 1 400 000 steps, the performance has increased drastically. After 228 000 steps, the model shows performance similar to the fully

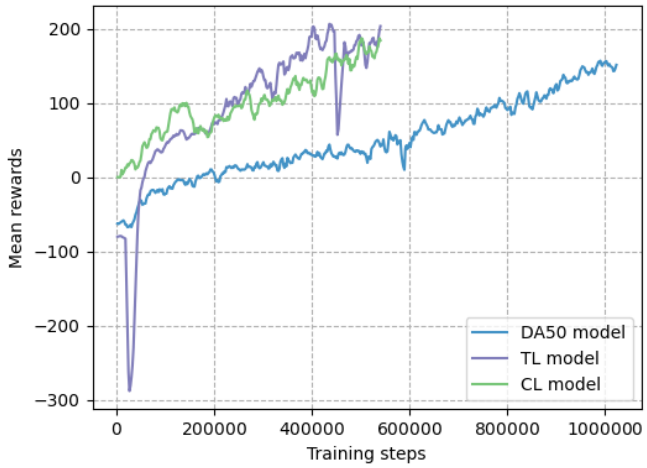

Figure 24: *Training performance of the TL techniques for the optimization of the DA50 wing.*

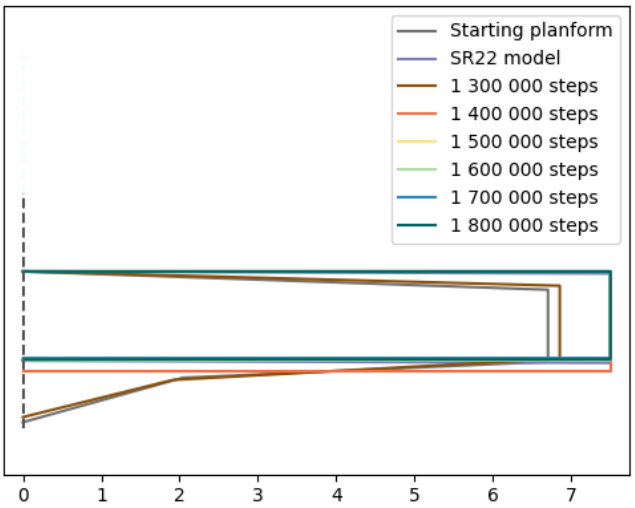

Figure 25: *DA50 planforms using TL with progressive training steps.*

trained model and its results are comparable to the PSO algorithm results.

training steps	f_{max}	f_{ave}	σ	t_{train}
1 272 000	35.22	34.39	0.40	-
1 300 000	33.51	33.27	0.069	52m
1 400 000	34.97	34.74	0.118	3h46m
1 500 000	35.34	35.07	0.238	6h40m
1 600 000	35.34	35.03	0.187	9h54m
1 700 000	35.35	35.12	0.144	13h15m
1 800 000	35.36	35.12	0.17	16h45m

Table 6: *TL progress for the DA50 aircraft, starting objective: 33.20.*

In Fig. 25, the best planforms at the various training steps are plotted. The planforms reflect what has been discussed so far. The planform created by the model at 1 300 000 steps barely changed relative to the starting condition. At 1 400 000 steps, the wing starts to progressively converge to the rectangular shape again.

5 Conclusions and future work

The results presented in the previous section show that DRL frameworks could be a promising technique for MDAO problems where automation and efficiency take priority over the simplicity of application. Not only was it able to consistently optimize the wing 4% further than the traditional optimization algorithm, it was able to do so 35 times faster. The RL algorithm was able to find a planform shape that the PSO algorithm could not. This demonstrates that the DRL framework could be an useful alternative to traditional optimization algorithms to find unique solutions, as it explores the design space in a unique way. When small changes are made to the design space, the model is still able to optimize design problem without the need for any retraining. To solve similar design problems, using the TL techniques is more efficient than training a new model entirely. When changes to the design spaces become too large, the environment will most likely need to be re-calibrated to see enhanced performance, as the TL methods could have detrimental effects on the results. The consistency of the DRL framework should be evaluated in order to produce reliable results. The significant training time of the PPO algorithm limits its utility. It is not an effective replacement of traditional optimization algorithms for design problems where only a single optimization is required.

On the basis of these results, the following recommendations are made. The framework set-up described in this paper was severely limited by the available computational resources. With significantly increased computational resources, more complex design problems can be solved using higher fidelity physics based solvers. This could help uncover innovative wing shapes. A more complex framework structure can be constructed that takes the design spaces of multiple aircraft into account during the initial training phase. This creates a generalised optimization framework that is inherently able to optimize the wings for a large range of aircraft. This would allow the model to be reused between multiple design problems without the need for any retraining.

References

[1] Tim Ryley, Stefan Baumeister, and Liese Coulter. "Climate change influences on aviation: A literature review". In: *Transport Policy* 92 (2020), pp. 55–64. ISSN: 0967-070X. DOI: https://doi.org/10.1016/j.tranpol.2020.04.010. URL: https://www.sciencedirect.com/science/article/pii/S0967070X20302870.

[2] Joaquim R. R. A. Martins and Andrew Ning. *Engineering Design Optimization*. Cambridge University Press, 2021. ISBN: 781108833417.

[3] Maurice Frederik Maria Hoogreef. "Advise, Formalize AND Integrate MDO Architectures". PhD thesis. Delft, 2017. DOI: https://doi.org/10.4233/uuid:cc2af611-6d78-4439-9b10-7e62ae579029.

[4] Imco van Gent. "Agile MDAO Systems - A Graph-based Methodology to Enhance Collaborative Multidisciplinary Design". PhD thesis. Delft, 2019. DOI: https://doi.org/10.4233/uuid:c42b30ba-2ba7-4fff-bf1c-f81f85e890af.

[5] John Schulman et al. "Proximal Policy optimization Algorithms". In: *arXiv (Cornell University)* (Jan. 2017). DOI: 10.48550/arxiv.1707.06347. URL: https://arxiv.org/abs/1707.06347.

[6] Dan Simon. *Evolutionary Optimization Algorithms*. John Wiley Sons, Inc., 2013. ISBN: 978-0-470-93741-9.

[7] Soledad Le Clainche et al. "Improving aircraft performance using machine learning: A review". In: *Aerospace Science and Technology* 138 (2023), p. 108354. ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2023.108354. URL: https://www.sciencedirect.com/science/article/pii/S1270963823002511.

[8] Zhehan Ni. "Reframe the Field of Aerospace Engineering Via Machine Learning: Application and Comparison". In: *Journal of Physics: Conference Series* 2386 (Dec. 2022), p. 012031. DOI: 10.1088/1742-6596/2386/1/012031.

[9] Kazuo Yonekura and Hitoshi Hattori. "Framework for design optimization using deep reinforcement learning". In: *Structural and Multidisciplinary Optimization* 60 (May 2019). DOI: 10.1007/s00158-019-02276-w.

[10] Nathan K. Brown et al. ""Deep reinforcement learning for engineering design through topology optimization of elementally discretized design domains"". In: *Materials Design* 218 (2022), p. 110672. ISSN: 0264-1275. DOI: https://doi.org/10.1016/j.matdes.2022.110672. URL: https://www.sciencedirect.com/science/article/pii/S0264127522002933.

[11] Jingfeng Wang and Guanghui Hu. *A mechanism-informed reinforcement learning framework for shape optimization of airfoils*. 2024. DOI: https://doi.org/10.48550/arXiv.2403.04329. arXiv: 2403.04329 [math.NA].

[12] Yousef Azabi, Al Savvaris, and Timoleon Kipouros. "The Interactive Design Approach for Aerodynamic Shape Design Optimisation of the Aegis UAV". In: *Aerospace* 6.4 (2019). ISSN: 2226-4310. DOI: 10.3390/aerospace6040042. URL: https://www.mdpi.com/2226-4310/6/4/42.

[13] Xinyu Hui et al. "Multi-object aerodynamic design optimization using deep reinforcement learning". In: *AIP Advances* 11 (2021). DOI: https://doi.org/10.1063/5.0058088.

[14] Thomas P Dussauge et al. "A reinforcement learning approach to airfoil shape optimization". In: *Scientific Reports* 13 (2023). DOI: https://doi.org/10.1038/s41598-023-36560-z. URL: https://api.semanticscholar.org/CorpusID:259184329.

[15] Jichao Li, Xiaosong Du, and Joaquim R.R.A. Martins. "Machine learning in aerodynamic shape optimization". In: *Elsevier, Progress in Aerospace Science* 134 (2022). DOI: https://doi.org/10.1016/j.paerosci.2022.100849.

[16] Isaksson Ola et al. "Model Based Decision Support for Value and Sustainability in Product Development". In: July 2015. URL: https://api.semanticscholar.org/CorpusID:38248828.

[17] Jindong Wang and Yiqiang Chen. *Introduction to Transfer Learning*. second. Springer, 2023. ISBN: 978-981-19-7583-7.

[18] Marco Wiering and Martijn van Otterlo. *Reinforcement Learning : State-of-the-Art*. Springer, 2012. ISBN: 978-3-642-27644-6.

[19] Zhuangdi Zhu et al. "Transfer Learning in Deep Reinforcement Learning: A Survey". In: *arXiv e-prints*, arXiv:2009.07888 (Sept. 2020), arXiv:2009.07888. DOI: 10.48550/arXiv.2009.07888. arXiv: 2009.07888 [cs.LG].

[20] Kazuki Hayashi and Makoto Ohsaki. "Reinforcement learning for optimum design of a plane frame under static loads". In: *Engineering with Computers* 37 (2021). DOI: https://doi.org/10.1007/s00366-019-00926-7.

[21] Xinyu Hui et al. "Multi-object aerodynamic design optimization using deep reinforcement learning". In: *AIP Advances* 11.8 (Aug. 2021), p. 085311. ISSN: 2158-3226. DOI: 10.1063/5.0058088. eprint: https://pubs.aip.org/aip/adv/article-pdf/doi/10.1063/5.0058088/12904930/085311_1_online.pdf. URL: https://doi.org/10.1063/5.0058088.

[22] An Guo, Lianghua Song, and Xiong Chen. "Learning Similar Tasks Based On PPO By Transferring Trajectory". In: May 2019, pp. 126–131. DOI: 10.1109/ICNSC.2019.8743298.

[23] Samuel Barnhart, Barath Narayanan, and Sidaard Gunasekaran. "Blown Wing Aerodynamic Coefficient Predictions Using Traditional Machine Learning and Data Science Approaches". In: Jan. 2021. DOI: 10.2514/6.2021-0616.

[24] Qiuwan Du et al. "Airfoil design and surrogate modeling for performance prediction based on deep learning method". In: *Physics of Fluids* 34.1, 015111 (Jan. 2022), p. 015111. DOI: 10.1063/5.0075784.

[25] Jichao Li and Mengqi Zhang. "Data-based approach for wing shape design optimization". In: *Aerospace Science and Technology* 112 (2021). DOI: http://dx.doi.org/10.1016/j.ast.2021.106639.

[26] Gang Sun, Yanjie Sun, and Shuyue Wang. "Artificial neural network based inverse design: Airfoils and wings". In: *Aerospace Science and Technology* 42 (2015), pp. 415–428. ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2015.01.030. URL: https://www.sciencedirect.com/science/article/pii/S1270963815000644.

[27] Shuyue Wang et al. "Database self-expansion based on artificial neural network: An approach in aircraft design". In: *Elsevier, Aerospace Science and Technology* 72 (2017). DOI: https://doi.org/10.1016/j.ast.2017.10.037.

[28] Zilan Zhang et al. "An adaptive machine learning-based optimization method in the aerodynamic analysis of a finite wing under various cruise conditions". In: *Theoretical and Applied Mechanics Letters* 14.1 (2024), p. 100489. ISSN: 2095-0349. DOI: https://doi.org/10.1016/j.taml.2023.100489. URL: https://www.sciencedirect.com/science/article/pii/S2095034923000600.

[29] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. second. MIT Press, 2020. ISBN: 9780262039246.

[30] Wei Shi et al. "Efficient hierarchical policy network with fuzzy rules". In: *International Journal of Machine Learning and Cybernetics* 13 (Feb. 2022). DOI: 10.1007/s13042-021-01417-2.

[31] Snorri Gudmundsson. *General Aviation Aircraft Design : Applied Methods and Procedures*. Nov. 2021. URL: https://www.amazon.com/General-Aviation-Aircraft-Design-Procedures/dp/0128184655.

[32] Mark Drela and Harold Youngren. *MIT AVL User Primer*. 2022. URL: https://web.mit.edu/drela/Public/web/avl/AVL_User_Primer.pdf (visited on 04/22/2023).

[33] Omran Al-Shamma and Rashid Ali. *AIRCRAFT WEIGHT ESTIMATION IN INTERACTIVE DESIGN PROCESS*. Apr. 2014.

[34] Pouria Razzaghi et al. "A Survey on Reinforcement Learning in Aviation Applications". In: (Nov. 2022). DOI: https://doi.org/10.48550/arXiv.2211.02147.

[35] Henry Sowerby, Zhi-Hua Zhou, and Michael L. Littman. "Designing Rewards for Fast Learning". In: *ArXiv* abs/2205.15400 (2022). DOI: https://doi.org/10.48550/arXiv.2205.15400. URL: https://api.semanticscholar.org/CorpusID:249210156.

[36] Serena Booth et al. "The perils of trial-and-error reward design: misdesign through overfitting and invalid task specifications". In: *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023. ISBN: 978-1-57735-880-0. DOI: 10.1609/aaai.v37i5.25733. URL: https://doi.org/10.1609/aaai.v37i5.25733.

[37] Adam Gleave et al. "Quantifying Differences in Reward Functions". In: *CoRR* abs/2006.13900 (2020). DOI: https://doi.org/10.48550/arXiv.2006.13900. arXiv: 2006.13900. URL: https://arxiv.org/abs/2006.13900.

[38] Yujing Hu et al. "Learning to Utilize Shaping Rewards: A New Approach of Reward Shaping". In: *CoRR* abs/2011.02669 (2020). DOI: https://doi.org/10.48550/arXiv.2011.02669. arXiv: 2011.02669. URL: https://arxiv.org/abs/2011.02669.

[39] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. "Reward Constrained Policy Optimization". In: *CoRR* abs/1805.11074 (2018). DOI: https://doi.org/10.48550/arXiv.1805.11074. arXiv: 1805.11074. URL: http://arxiv.org/abs/1805.11074.

[40] Zijian Hu et al. "A Dynamic Adjusting Reward Function Method for Deep Reinforcement Learning with Adjustable Parameters". In: *Mathematical Problems in Engineering* 2019 (Nov. 2019), pp. 1–10. DOI: 10.1155/2019/7619483.

A Optimization Algorithm Hyperparameters

Hyper-parameter	Value
Learning rate	3 e-4
N steps	2048
Minibatch size	64
Discount factor	0.99
Trade-off factor	0.95
Entropy coefficient	0.0
Value function coefficient	0.5
Clipping parameter	0.2
Optimizer	Adam
Activation function	ReLU
Policy	MlpPolicy

Table 7: *Hyper-parameters used for the PPO algorithm*

Hyper-parameter	Value
Swarm size	100
Cognitive rate	1.0
Social rate	1.0
Max evaluation	7500
Target fitness	-31.3
Variant	Vanilla
Intertia	Anakatabatic
Akb model	Languid

Table 8: *Hyper-parameters used for the PSO algorithm*

B Aircraft Reference Data

-	b	c_{root}	c_{kink}	c_{tip}	Λ_{LE}	y_{kink}	z_{tip}	ϵ	t/c	fuselage width
Starting condition	11.65	1.54	1.145	0.75	2	3.225	0.4	0	0.196	1.24
Upper bound	15	2.5	2	1.5	20	5	1	5	0.22	-
Lower bound	8	0.5	0.5	0.1	0	1.4	0	-5	0.11	

Table 9: *Aircraft geometry data used for the optimization of the SR22 wing.*

-	b	c_{root}	c_{kink}	c_{tip}	Λ_{LE}	y_{kink}	z_{tip}	ϵ	t/c	fuselage width
Starting condition	13.41	1.724	1.287	0.92	2	2.024	0.564	0	0.137	1.29
Upper bound	15	2.5	2	1.5	20	5	1	5	0.22	-
Lower bound	8	0.5	0.5	0.1	0	1.4	0	-5	0.11	

Table 10: *Aircraft geometry data used for the optimization of the Da50 wing*

-	b	c_{root}	c_{kink}	c_{tip}	Λ_{LE}	y_{kink}	z_{tip}	ϵ	t/c	fuselage width
Starting condition	29	3.3	2.36	1.43	3.75	7.92	0.433	0	0.1795	2.7
Upper bound	36	6	6	3	20	12.5	5	5	0.22	-
Lower bound	15	1	1	0.5	0	1.35	0	-5	0.08	-

Table 11: *Aircraft geometry data used for the optimization of the F50 wing.*

-	b	c_{root}	c_{kink}	c_{tip}	Λ_{LE}	y_{kink}	z_{tip}	ϵ	t/c	fuselage width
Starting condition	17.0	1.50	1.0	0.50	0	5.16	0.594	0	0.161	0.70
Upper bound	24	2.5	2	1.5	20	10	1	5	0.20	-
Lower bound	8	0.5	0.5	0.1	0	1.4	0	-5	0.11	-

Table 12: *Aircraft geometry data used for the optimization of the ASK21 wing.*

-	m_{tot} [kg]	m_{empty} [kg]	m_{wing} [kg]	m_{fuel} [kg]	n_z	V_{wing} [m^3]	W/S [$\frac{kg}{m^2}$]	f_0	T_{cruise}[K]	ρ_{cruise} [$\frac{kg}{m^3}$]	v_{cruise} [$\frac{m}{s}$]
SR22	1633	1076.8	156.4	220	5.7	1.90	136	28.08	253.6	0.711	92.6
DA50	1999	1450	234.5	148	5.7	1.67	137	33.20	260.41	0.797	88
F50	18990	12570	2514	4080	3.75	19.0	305	34.91	238.68	0.55	147
ASK21	600	402	198	0	6.5	2.11	35	26.50	278.24	1.056	49.90

Table 13: *Data of the reference aircraft.*

II

Literature Study

Previously graded under AE4020.

1 Introduction

1.1. Background

Currently, one of the main points of focus of the aviation industry is reducing the environmental impact of aircraft. In order to achieve this, the efficiency and performance of newly designed aircraft have to be optimized as much as possible. The optimization of an aircraft design is a very complex process as many fields are dependent on each other. This leads to a lot of time spent on inter-department communication which results in many small changes during the conceptual design phase. The field that specializes in addressing this problem is called Multi-Disciplinary Analysis and Optimization (MDAO) [3]. The main focus of MDAO is to use techniques and tools that allow engineers to simultaneously design and optimize all components of the aircraft. However, MDAO techniques are currently not widely utilized professionally as the set-up and expertise required prove to be too great of a hurdle [2].

Recently, Artificial Intelligence (AI) applications are becoming increasingly popular, as it appears new developments are being made every week with projects like StableDiffusion[1] and ChatGPT[2]. However, AI applications for aerospace problems are limited, especially regarding automated design. As AI employs a fundamentally different strategy from other optimization algorithms, exploring this could uncover potential advantages. In Chapter 2, AI is explained in further detail and various applications are explored to get an understanding of how ML is currently utilized.

1.2. Research Objective

This literature study aims to provide an overview of the current state of automated design together with its limitations. The basics of ML algorithms are then discussed together with design-related applications. A conclusion is then drawn on which ML techniques could be beneficial to automated aerospace design.

The aim of the thesis is to highlight the possible advantages and limitations of ML applications to automated aerospace design. During this project, a ML algorithm is selected and its performance as an optimizer is compared to a traditional optimization algorithm. A traditional optimization algorithm refers to an optimization algorithm that is used for automated design optimization and can commonly be found in MDAO applications. The term optimizer refers to the entire framework surrounding the optimization process.

ML algorithms can be used as optimizers similar to methods performed in MDAO. However, ML algorithms have fundamental differences in operation, which have the potential to address some of the weaknesses currently found in automated design. Several comparisons will be made, to evaluate the capabilities of the ML algorithms. Furthermore, the following aspects of ML algorithms will be explored: the ability to make predictions on categorical variables, retain knowledge from data, and perform multi-objective optimization.

[1] `https://stablediffusionweb.com/`, accessed on 26-04-2023
[2] `https://openai.com/blog/chatgpt`, accessed on 26-04-2023

The optimizations will be performed for aircraft design problems at the conceptual design phase level. This is done by creating 2 design problems of different complexity. The first design problem consists of the optimization of a wing geometry for aerodynamic cruise performance. The second design problem concerns the selection and sizing of control surfaces for a specified wing configuration, optimized for cruise, take-off, and landing performance.

Important to note is that the aim of the thesis project is not to ease the application of MDAO techniques to a professional setting, as this problem is far too complex to be addressed in a master thesis. The main aim is to evaluate the application of ML to automated design and not to solve a complex design optimization problem. The thesis will therefore focus on the analysis of the optimizers and their response while the design problem to evaluate this is kept as simple as possible.

This report is outlined in the following way. In Chapter 2, background information is given on ML and the current state of relevant fields is discussed. Concluding this chapter is a discussion about the ML techniques that could be beneficial for automated design. Chapter 3 lists the research questions that are set up to accomplish the objective of the thesis. Chapter 4 gives the methodology of the thesis research. Here the actions performed during the thesis to answer the research questions are discussed. Finally, a conclusion on the literature study is drawn in Chapter 5.

2

Literature Review

The aim of this chapter is to identify any limitations or gaps in the current knowledge surrounding automated aerospace design. In order to find these gaps, the current state of all relevant research areas is explored, including ML, automated design, and applicable software. Once this is established, the state-of-the-art is explored by looking at several relevant projects. This gives the reader an overview of the methods currently applied to both automated design and ML. A conclusion is then formed on how ML techniques could be utilized to solve automated aerospace design problems. These techniques are then applied to a design problem, discussed in Chapter 3, in order to accomplish the research objective.

2.1. Machine Learning

A good understanding of the machine learning fundamentals is required to fully utilize its potential in an optimization problem. The main goal of a ML algorithm is to learn patterns from data in order to make predictions on new data. The two most common tasks of ML algorithms are classification, where input data is categorized, and regression, where an output value is predicted based on an input. While ML algorithms can be used as optimizers similar to methods performed in MDAO, they do have fundamental differences in operation. These differences could be exploited to potentially address some of the weaknesses currently found in automated design. This literature review aims to identify the differences and find ML techniques that could benefit automated design processes the most. The following sections briefly describe key ML methods and techniques relevant to the research objective. For a more extensive overview of a great number of ML algorithms together with their applications, please refer to the journal paper by R.M. Jichao Li and colleagues [4].

2.1.1. The Basics of a ML algorithm

The main component of a ML algorithm is its model, which can drastically vary based on the algorithm. In general, a model consists of an algebraic equation that makes predictions based on a certain input. Most ML algorithms contain two types of variables called weights and biases. The available data is split into training and test data. A ML algorithm improves its predictions by processing training data and using an optimization algorithm to improve its weights and biases. The step size of this optimization algorithm can be adjusted using a parameter called the learning rate. The processing of training data is referred to as the training phase. This can be quite computationally expensive, depending on the amount of training data. However, once a model is trained, it is able to rapidly make new predictions. The test data is used for the evaluation of the trained ML model.

The results of the ML model can be altered by adjusting the hyperparameters [5]. Hyperparameters are specific parameters selected by the engineer prior to the training phase and have a big impact on the generated results. The hyperparameters that the engineer is able to tune can differ based on the ML algorithm. Examples of hyperparameters include the test-train data split ratio, the selection of the optimization algorithm, and its learning rate.

In general, the design problem determines what type of data is available. This will in turn determine

what category of ML algorithm should be chosen. If the available data includes both the input and output, a supervised learning algorithm should be selected. Popular examples of this are: Logistic Regression, Support Vector Machines, and Decision trees [5]. When the outputs of the training data are unavailable, the ML algorithm will try to find certain patterns in this data and subsequently classify new data. Examples of this are K-means Clustering, Gaussian Mixture Model, and Principal Component Analysis [5]. Finally, there is a third type of ML called semi-supervised learning, where data is generated on the basis of the interaction with a certain environment. The most common algorithm that fits this criterion is called reinforcement learning. This will be further explained in the upcoming section.

2.1.2. Neural Network

Currently, one of the most popular ML algorithms used to solve complex problems is called deep learning, where the ML model consists of an Artificial Neural Network (ANN) [6]. A neural network consists of stacked layers that all contain a various number of neurons. Each neuron consists of a number of inputs X_n, that are multiplied by an equal number of weights W_n. A bias b is then added to the weighted sum. The bias is used to influence the impact each neuron has on the next layer. Both the values of the weights and bias are typically unconstrained. However, regularization techniques can be implemented if these values become too large relative to other neurons. This prevents one neuron from dominating the entire prediction [6]. Additionally, an activation function ϕ can be added to each neuron. This allows the ML algorithm to make non-linear predictions. Finally, the output of the neuron h is calculated using Equation 2.1 [5]. The complete neuron and the relations between each parameter are shown in Fig. 2.1.

$$h(\underline{X}) = \phi(\underline{X}^T \underline{W} + b) \tag{2.1}$$

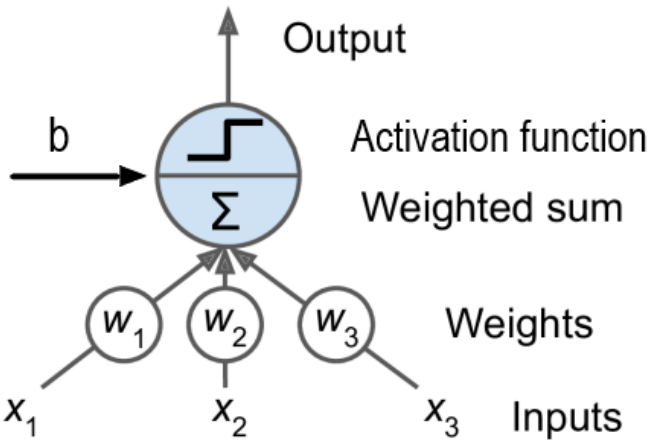

Figure 2.1: Example of a neuron, showing the interaction between the inputs, weights, bias, and the activation function. Adapted from [5].

An ANN is comprised of multiple layers, which can be categorized into three groups: the input, hidden, and output layers [6]. The input layer consists of a number of neurons equal to the size of the input vector. The purpose of the input layer is to create an interface between the input vector and the ML model. Therefore, the neurons of the input layer do not contain any weight or bias terms. The input layer is directly connected to the neurons of the first hidden layer. The output of each input layer neuron h_i now serves as the input for each neuron of this hidden layer. The output of the first hidden layer, indicated with i+1, can now be calculated using Equation 2.2. The neurons of subsequent hidden layers are connected in a similar fashion, as indicated in Fig. 2.2. Both the number of hidden layers and the number of neurons contained within each layer can be freely chosen by the engineer. Finally,

the output layer serves as a representation of the predictions made by the ML model. For example, if the ML model is a classifier that has to predict whether some input data belongs to one out of three classes, the output layer will consist of three neurons. The output of each neuron then consists of a probability of the input data belonging to that class.

$$h_{i+1}(\underline{h}_i) = \phi(\underline{h}_i^T \underline{W}_{i+1} + b_{i+1}) \tag{2.2}$$

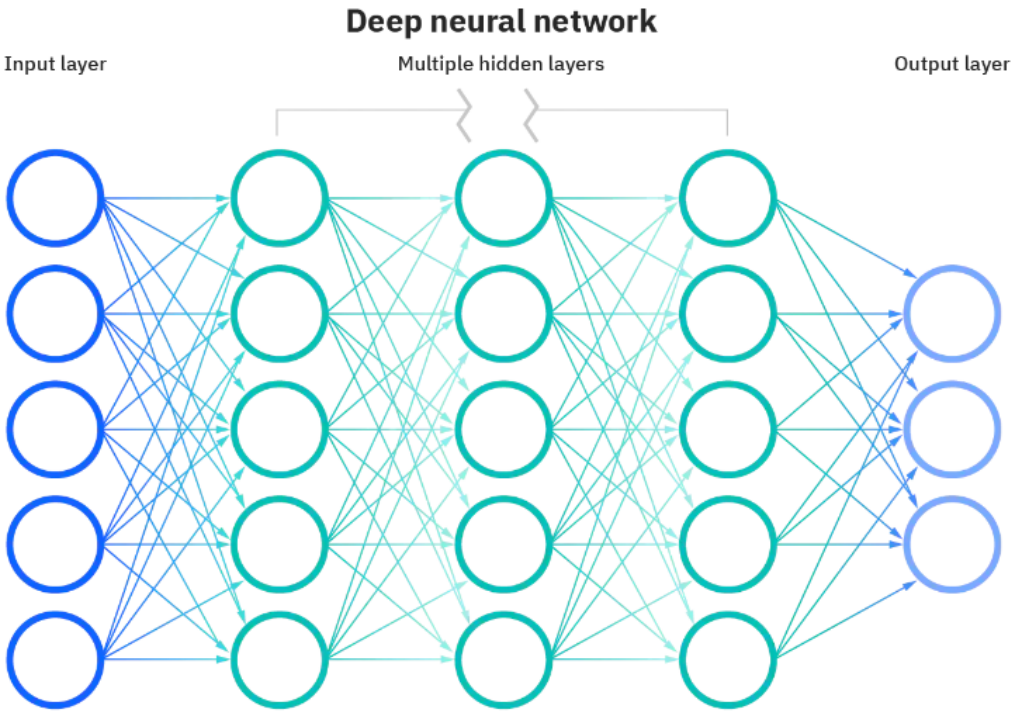

Figure 2.2: Basic neural network structure showing various layer types and how they are connected[1].

When the ML model is first constructed, the weights and biases have to be randomized. Using techniques called ”forward pass” and ”backpropagation”, the neurons are able to update the weights and bias values when the model is processing training data [5]. A trained model is stored as a tensor containing matrices and vectors. This tensor contains weight matrices and bias vectors for each hidden layer and the output layer. The number of columns of a weight matrix is equal to the number of neurons in the previous layer, while the number of rows is determined by the number of neurons in the current layer, as can be seen in Fig. 2.3. The length of the bias vector of a layer is equal to its number of neurons.

[1] Obtained from: `https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks`, accessed on 22-04-2023

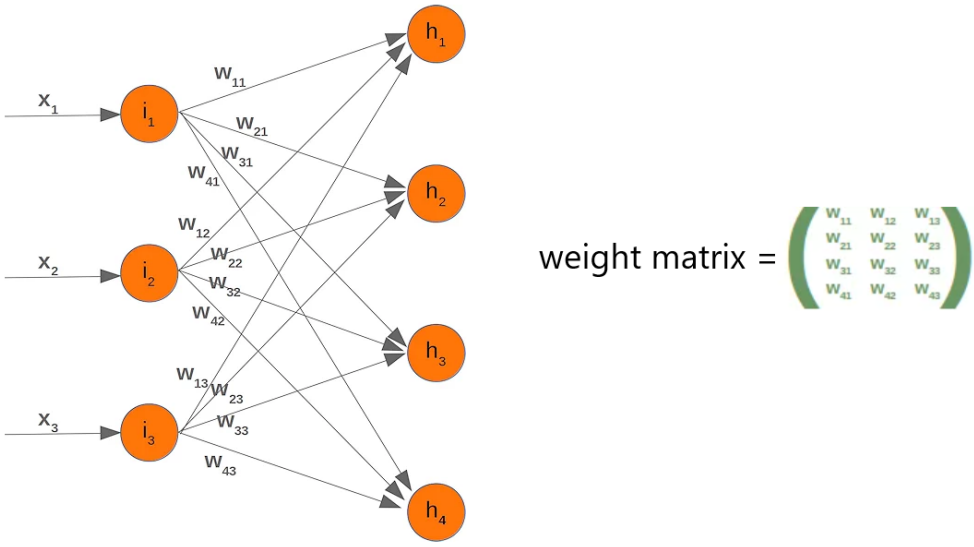

Figure 2.3: The construction of the weight matrix for a given a hidden layer[2].

A neural network gives a ML model the ability to make complex predictions, as it is able to recognize complex trends in the training data [6]. ANN are quite flexible as they are scalable by changing the number of hidden layers and neurons. This allows the neural network to adapt its complexity based on the problem at hand. The main drawback of neural networks is that they take a lot of computational effort to train, as each neuron has to be updated. A neural network should therefore only be used if the design problem requires a complex ML algorithm.

2.1.3. Reinforcement Learning

Reinforcement Learning (RL) is a semi-supervised learning model where an agent continually makes decisions and reacts to the response of its environment on a trial and error basis [7]. The goal of the model is to iteratively make better decisions that will improve its results. The agent is the main component of a RL model and determines what actions should be performed based on the policy function. The possible actions are stored in an array and describe how the agent can alter the design variables. This action list is predefined during the setup phase. The state S_t is typically an array comprised of the current state of the design vector combined with the response from the environment. In the context of automated aerospace design, the state could be comprised of the aircraft design variables, together with the performance parameters that the design is being optimized for. The environment consists of a structure of simulation software and calculations that use the design variables as input and generates an output. This output is then evaluated by generating a reward signal R_t. The reward is a scalar variable that determines whether the response of the environment to the selected actions is favorable. The main goal of the agent is to maximize the total reward it has received during the training phase.

The policy is created to relate the current state with the possible actions and is used to determine which actions can lead to the best results. The policy can be constructed in a variety of ways, as it is dependent on both the selected type of RL algorithm and the variable types of the design vector and environment response. Typically, it can be a simple function or a generated table that is updated each iteration, based on the generated reward signal. The policy function also allows the agent to either focus on short or long-term results.

Upon initialization, the decisions the agent makes are all random [7]. This is referred to as ”exploration”. Once the model has completed a sufficient number of iterations, it will try to utilize the data it has collected to improve its decision. This is called ”exploitation”. A good balance has to be found between the exploration and exploitation throughout the training phase. If a model performs too much exploitation

[2]Adapted from: `https://python-course.eu/machine-learning/neural-networks-structure-weights-and-matrices.php`, accessed on 15-05-2023

it may get stuck at a local optimum. Allowing some randomness in the selection of actions will help the agent find new ways of improving the results. The agent will continue to make decisions until either it has reached a set goal, enough iterations have been performed, or the state is stuck at a sub-optimal value. The interaction between the components of a RL model is illustrated in Fig. 2.4.

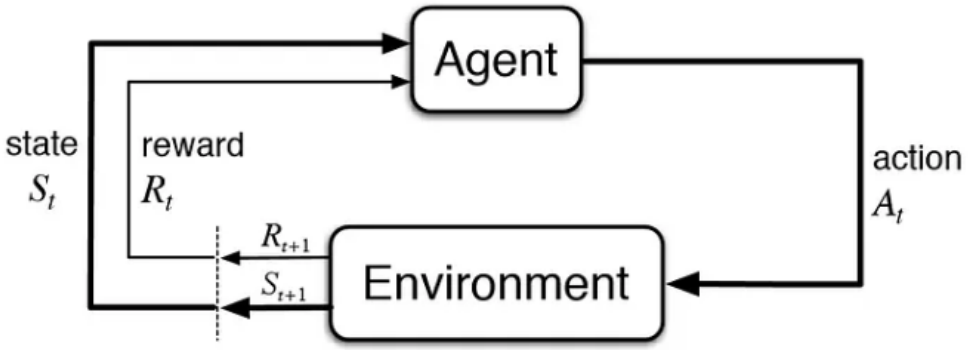

Figure 2.4: Basic diagram showing all components of a RL algorithm and how the agent interacts with the environment [7].

RL is thus ideal for problems where data is not available at the start of a project and must be calculated or generated through various simulations. There are a great number of unique RL algorithms. Additionally, ANNs can be utilized by most RL algorithms, this is known as Deep Reinforcement Learning (DRL).

Q-learning is a very popular algorithm, that utilized a Q-table[7]. The rows of the Q-table consist of the potential states while the columns represent all the possible actions the agent is able to take. The algorithm generates an expected reward for each combination of states and actions. This reward is called the Q-value. The policy function then selects an action from the table based on the current state of the agent. This Q-table is iteratively updated throughout the training phase.

A big problem with Q-learning is that the Q-table is not very scalable. When the number of actions and possible states are very high, the table becomes computationally expensive to update at each iteration. The Q-table can be replaced by an ANN, which is known as Deep Q-Network (DQN) [8, 9]. The input for this ANN is the state vector while the output is a vector with estimated Q-values for each action. The RL model can then select the action, based on the Q-value and its policy. The ANN is significantly less expensive to evaluate and can be scaled more efficiently than a large Q-table.

Another benefit of the DQN is that it more efficiently explores new areas of the design space. As the DQN uses an ANN, it is able to make predictions on the Q-values on new states the agent has not experienced before [8]. However, as the ANN is simply an approximation of the Q-table, DQN generally performs worse than Q-learning for simple problems with a low number of possible actions.

RL models can be adapted to solve multi-objective problems, where multiple variables need to be optimized [10]. Hayes and colleagues have created an extensive guide for the use of various multi-objective optimization techniques [11]. RL algorithms are very flexible and their utility can be increased using various techniques, which are explored in the upcoming sections.

2.1.4. Transfer Learning

ML models are typically trained on fixed data sets and are unable to make reliable predictions on new data that diverges too much from the original data set. A very powerful technique of ML that tries to circumvent this problem is called transfer learning. The goal of transfer learning is to leverage the knowledge of a trained model to solve new but similar problems [12, 13]. This also works well for ANN-based ML models, where new layers are added after the hidden layers of the original ANN. The values of the original hidden layers are kept constant while the additional hidden layers are used to retrain the model to fit the new problem. The main advantage of using transfer learning is that it greatly reduces the training time of the ML model as only the additional hidden layers are trained, thus greatly increasing

the computational efficiency of any optimization problem. Additionally, this can be a great technique when little data is available. Using transfer learning, the knowledge gained during the training phase can therefore be transferred to a new problem in order to continually expand its knowledge [13]. The application of reusing knowledge without ML is still being explored by a field called Knowledge Based Engineering (KBE), this is further discussed in section 2.2.2.

2.1.5. Progressive Learning

Training a ML can be error-prone, especially when they are designed to solve complex tasks. Additionally, the training time increases exponentially the more complex the ML model becomes. This makes the training of a large and complex ML model quite risky, as a lot of computational resources are wasted if the performance of the ML model is subpar. Progressive learning is a form of transfer learning where a ML model is sequentially retrained to solve increasingly complex tasks [14]. Progressive learning allows the engineer to carefully monitor the training phase and performance of the ML model at various stages of complexity. Progressive learning can significantly reduce the training time and subsequently the required computational resources [14, 15].

An application of progressive learning used a simple Q-learning algorithm to solve subsequent mazes that increased in size [16]. In this maze, the agent is represented in gray and must exit the maze without running into the black dot, see Fig. 2.5. The agent is able to move in all directions or wait. This project used an interesting RL scheme where the agent first acquires experience by solving a simple maze without any prior knowledge. The agent is able to adapt to the new mazes by continually updating its policy algorithm. The progressive learning algorithm was able to solve the mazes in about 30-40% fewer iterations than the standard RL algorithm.

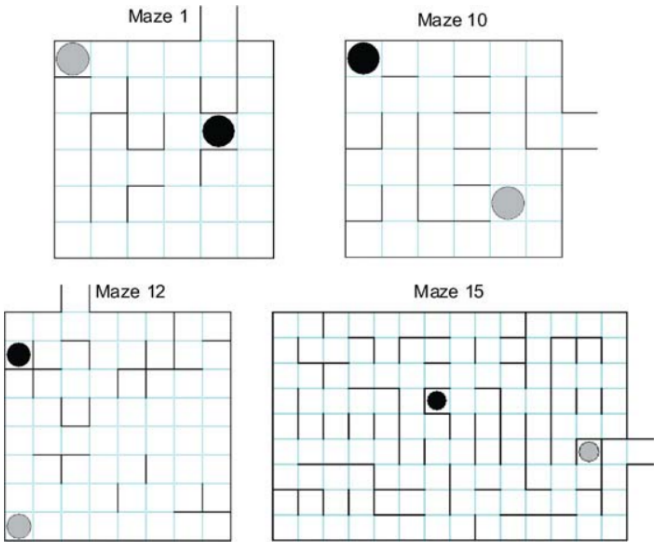

Figure 2.5: Overview of the various environments with the mazes increasing in size [16].

A related technique to progressive learning is known as continual or lifelong learning. The goal of this technique is to create a ML model that is able to expand its knowledge, while still retaining the knowledge it has previously gathered [14]. This allows the ML model to adapt its knowledge to solve new tasks while simultaneously being able to solve the tasks it was originally trained for. This creates a flexible ML model that is able to solve multiple tasks using its collective knowledge. Jaehong Yoon and colleagues created a complex deep network architecture called Dynamically Expandable Network (DEN) [17]. The DEN was able to dynamically expand the structure of its ANN by retraining the network on a variety of tasks. This resulted in a model that was able to create a ”compact overlapping knowledge-sharing structure”. The DEN was able to generate more accurate results than other lifelong learning methods while using a smaller ANN.

2.1.6. Advisor Agents

The concept of advisor agents is quite new and new methodologies are still being explored. Recently, the use of multiple agents, referred to as Multi-Agent Reinforcement Learning (MARL) has become quite popular. Advisor agents are trained agents that are able to support the training of a new primary agent [18, 19]. The advisor agents are trained on similar tasks as the primary agent. Advisor agents are used to decrease required computational resources and improve the stability and efficiency of the training phase of a new model. Advisor agent frameworks are quite flexible and can vary significantly in architecture as the number of advisors and their interactions with the primary agent can be left up to the engineer. Currently, advisor agent frameworks tend to struggle with high problem complexity and slow convergence of policies [20].

A popular scheme is the Advisor Critic Architecture (ACA), where the advisor is a pre-trained model and the primary agent is known as the critic [7]. Here, the advisor will suggest actions to the critic based on the current state. The critic will then evaluate these suggestions based on the rewards it has previously received. The main goal of this architecture is to help the RL model avoid less lucrative areas of the design space.

Subramanian and colleagues have built an interesting architecture including multiple advisor agents [21]. The framework of their architecture can be found in Fig. 2.6 and is called ADMIRAL (ADvising Multiple Intelligent Reinforcement Agents). For this project, two complex Q-learning algorithms were constructed to complement each other. Here, one traditional advisor framework has a number of advisor-agent pairs to improve the training phase, called ”ADMIRAL-DM”. The second model is quite similar to the first model, however, now only a single advisor is connected to each agent, called ”ADMIRAL-AE”. The purpose of this algorithm is to evaluate the performance of each individual advisor. In their experiments, they found that model could be adapted to a variety of environments and was resistant to bad suggestions from the advisor agent.

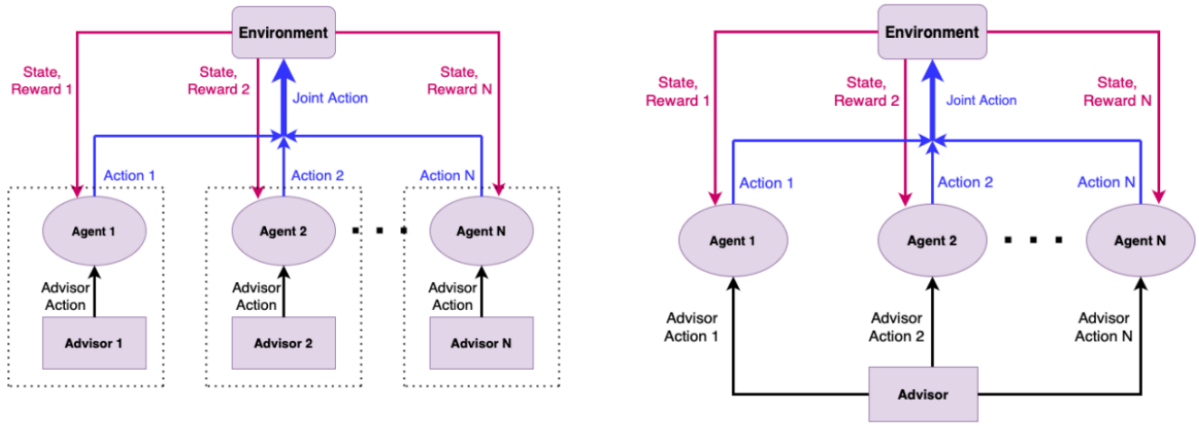

Figure 2.6: Example architecture of a multi-agent RL algorithm, shown here are the ADMIRAL-DM (left) and ADMIRAL-AE (right) algorithms [21].

2.1.7. Surrogate Models

Surrogate models are currently quite popular in engineering applications. A surrogate model aims to replace high-fidelity analysis methods with either simple algebraic equations or low-fidelity simulations [22, 23]. This means that for each iteration of an optimization process, the results of a simulation can be generated in quicker succession. This is therefore a very useful technique for optimization strategies where a low computational effort is prioritized over the accuracy of simulation results. A surrogate model can also be created using a supervised ML algorithm. This ML model tries to find certain trends in large amounts of training data. This data can either be obtained from databases or generated using high-fidelity simulations. When the trends generated by the underlying high-fidelity simulation are too complex, an ANN based ML algorithm can be selected. This is known as a Physics Informed Neural Network (PINN) and some applications will be discussed in section 2.4.2.

2.2. Automated Design

The main goal of this report is to find out what could be gained from the addition of Machine Learning to the automated aircraft design process. The primary fields relating to automated aerospace design are MDAO and KBE. In the upcoming sections, the techniques and methods that these fields employ are discussed to get an overview of the current state of automated design. The strengths and weaknesses of both fields are then evaluated, in order to find opportunities on how ML could aid automated design. Finally, the optimization algorithms suitable for the design problems of this thesis are discussed.

2.2.1. MDAO

The process of designing aircraft is a complex task as various disciplines such as aerodynamics, structures, dynamics, and performance are all interconnected. MDAO is a methodology that aims to create a framework to connect and oversee the interactions between each discipline during the conceptual design phase [3]. This allows engineers to explore a large design space by rapidly generating a large number of designs including unconventional designs.

Three limitations of current non-multidisciplinary design techniques are mentioned by van Gent [1]. When the analyses of all disciplines are uncoupled, performance improvements on existing aircraft are difficult to realize. Secondly, the use of empirical knowledge does not allow for innovative designs as high-fidelity coupled analysis is required. Finally, it is hard to properly explore the design space when all disciplines are uncoupled, as design exploration methods are not very effective.

The workflow of an MDAO analysis can be divided into two different phases, the formulation phase and the execution phase, see Fig. 2.7. The formulation phase of an MDAO framework typically requires the most amount of work, as it takes up about 60 to 80% of the project time [1]. During this phase, the relationships between the various disciplines are established. This includes the integration of the various simulation tools for each discipline, like flow solvers or finite element method software. As the disciplines exchange their inputs and outputs, keeping track of all these variables can get quite complex. An eXtended Design Structure Matrix (XDSM) diagram is created to give a proper overview of the entire optimization process, as shown in Fig. 2.8. The XDSM describes in great detail how the optimization algorithm interacts with the analysis methods of the disciplines. Once the entire optimization process is set up the execution phase can begin. During this phase, the framework starts its optimization process and very little interaction is required from the engineers. Once completed, the design solutions can be evaluated.

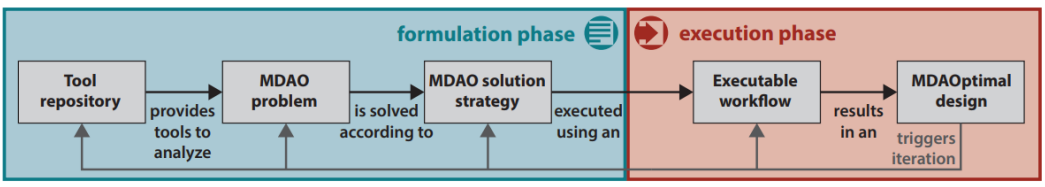

Figure 2.7: General MDAO workflow diagram showing the formulation of all subsequent steps of an optimization problem [1].

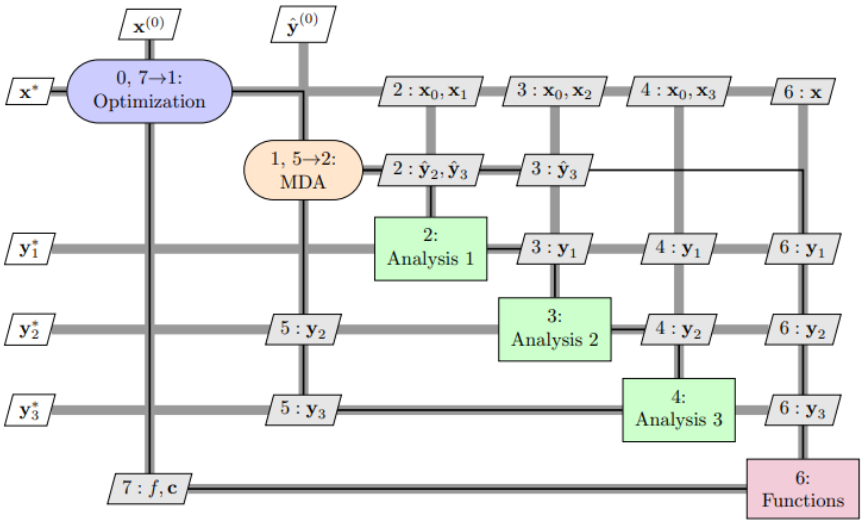

Figure 2.8: XDSM diagram showing all interactions between the various disciplines and the optimization algorithm during the optimization process [24].

MDAO frameworks show very promising results, as they allow engineers to generate a high number of possible solutions once the framework is set up. An example of the timeline for an MDAO application by Flager and Haymaker [25] can be found in Fig. 2.9 and concerns the design of a hypersonic aircraft. Where traditional techniques could take 6 weeks to set up and 4 weeks per iteration, MDAO is able to complete its set-up in 14 weeks but generate new solutions within 1.5 hours. This shows the power of MDAO for the conceptual design phase as its highly advantageous to create a large number of designs. This allows the engineer to explore the design space for unconventional designs.

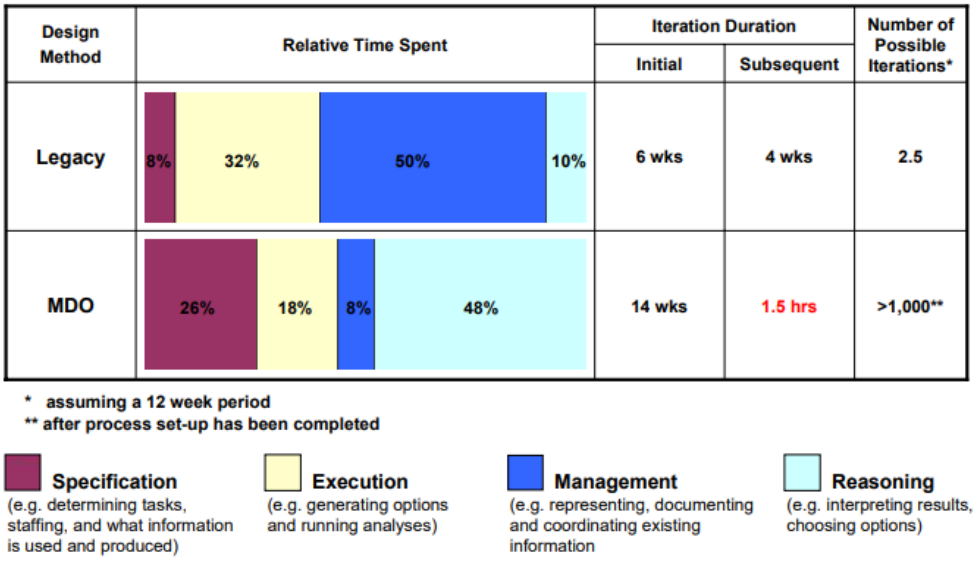

Figure 2.9: Overview of a timeline for an optimization problem concerning a hypersonic aircraft [25].

Sadly, the application of MDAO techniques is not fully realized in the professional field and is mostly restricted to academic research [2, 1, 3]. The reason for this is that the knowledge and expertise required are too high to implement a complete MDAO framework for a design problem. Currently, an expert in MDAO is required to translate a design problem into a proper MDAO framework [2, 1]. Even for MDAO experts, it is proven difficult to integrate high-fidelity software into this framework. Additionally, the

optimization process combined with high-fidelity analysis requires a large amount of computational resources. Finally, as shown in the previous section, the time to set up an MDAO framework is substantially greater than the traditional approach. This results in many projects using traditional design approaches, as the time required to generate the first design is too great. This means a lot of time is spent on interdepartmental communication and repetitive work that could be spent on evaluating a large number of creative design solutions [2, 1].

2.2.2. KBE

KBE is a knowledge-based subset of artificial intelligence that does not utilize a ML algorithm. Its main purpose is to efficiently structure the knowledge contained within a project [26]. KBE techniques are often integrated into MDAO frameworks as they complement each other very well. A popular method of structuring knowledge is semantic web technologies [27, 2]. This method allows for additional information to be stored with data and is fully recognizable by computer programs. These methods are subsequently used to generate ontologies. An ontology is a graph-like structure of data that includes the classification and relationship between various data points. An example of an ontology of the airline KLM can be found in Fig. 2.10, which shows how these relationships are constructed. Ontologies are powerful tools that allow for efficient knowledge processing and transfer between projects.

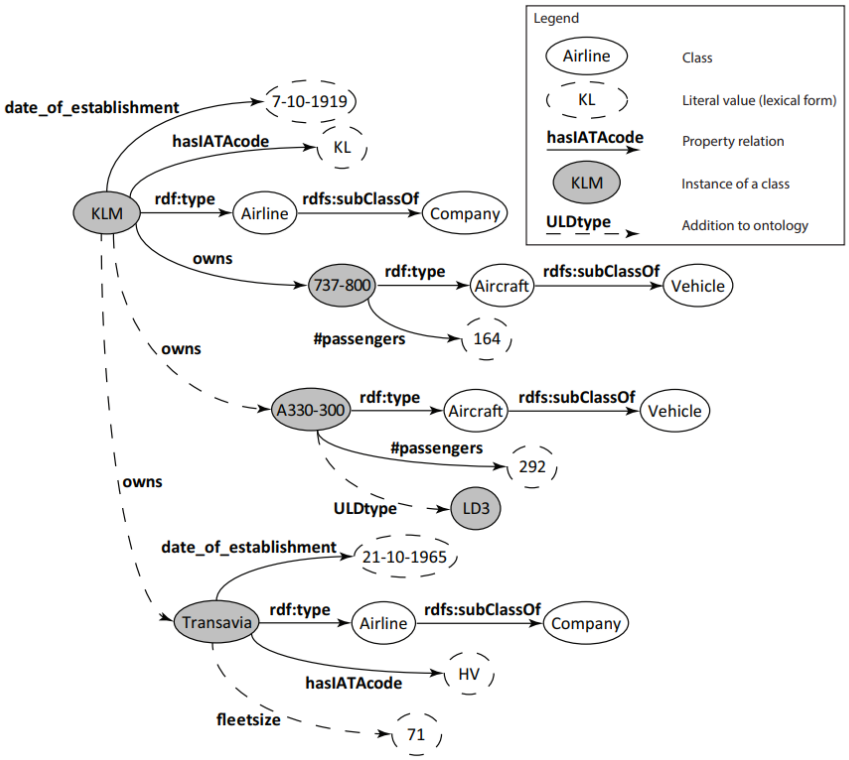

Figure 2.10: Example ontology of the airline KLM, showing the relationship between all data points [2].

Additionally, KBE strategies advertise the use of Object-Oriented Programming (OOP) practices [27]. OOP is typically used for the construction of complex software programs that include a large number of interactions between data. OOP organizes various components and tasks of a project as objects for which a class is constructed. This allows each object to inherit information from other classes. This makes the code more modular and reusable which allows for easy maintenance and modification during the development of the program.

Another useful technique that KBE utilizes is called lazy evaluation. This technique tracks parameters in an ontology together with their relationship with the other parameters. These parameters will only be recalculated if the values of any of its inputs are changed. This decreases the computational effort wasted on needless recalculations.

The techniques discussed so far are not just relevant for MDAO frameworks but could also aid in the construction of a general multi-disciplinary ML optimization problem. These techniques will therefore be taken into account when the optimization frameworks are constructed for this thesis.

2.2.3. Traditional Optimization Algorithms

It is quite difficult to gauge the computational costs required for any given optimization problem. These computational costs can also be greatly impacted by factors like the selected optimization algorithm. Several algorithms are researched in case one algorithm requires too much of the available computational resources.

Simple gradient-based optimization algorithms are quite common in MDAO frameworks, as they are accurate and reliable while being computationally inexpensive [3, 2]. However, gradient-based algorithms struggle in solving optimization problems that contain categorical variables. This makes them unable to be properly compared with ML optimization algorithms. For this reason, three optimization algorithms are explored that are compatible with both multi-objective optimization and categorical variables, namely: Bayesian Optimization (BO), Genetic Algorithm (GA), and Particle Swarm Optimization (PSO).

BO is an optimization algorithm typically used for computationally expensive black-box functions [28]. To use BO, an approximation of the underlying problem is first created in the form of a surrogate model, as can be seen in Fig. 2.11. Random samples of candidate solutions are fed through the objective function to generate data throughout the entire design space. These samples are then used to construct the surrogate model, also known as the acquisition function. Additional data is generated with the objective function and its results are compared with the acquisition function until it sufficiently represents the original function. Additionally, BO has use-cases outside of being directly used as an optimization algorithm. Wu and colleagues [28] have found success in utilizing a BO algorithm to optimize the hyperparameters of various ML algorithms, including random forest and neural networks.

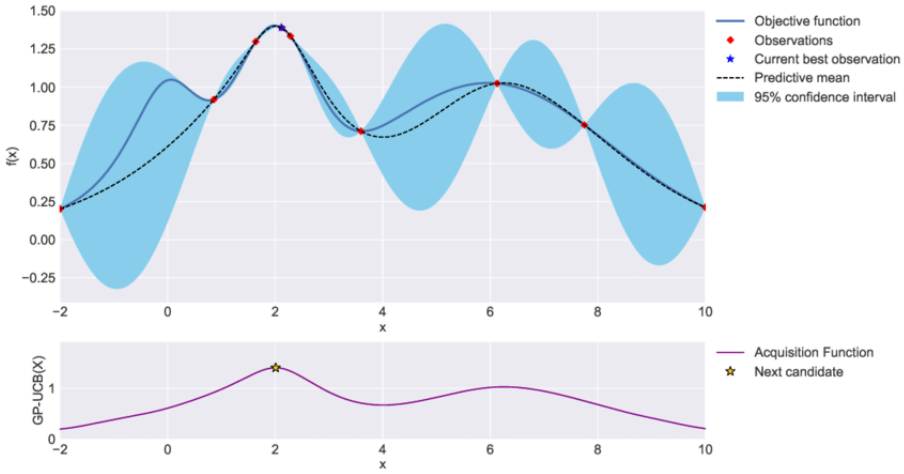

Figure 2.11: Bayesian optimization example showing the process of generating an acquisition function from an objective function [29].

As its name implies, GA is an optimization algorithm that is based on the concept of natural selection [30]. Like with BO, the optimization initializes by generating a collection of samples known as the population. The 'fitness' of each sample is scored using an objective function or underlying simulation. This fitness is then combined with the design vector of the sample and is typically stored in an array, called 'chromosome'. The best chromosomes are then selected from this population. These are referred to as 'elites'. From these 'elites', a number of chromosomes, called 'parents' are selected that will be used to generate new chromosomes. The GA then uses methods called 'crossover' and 'mutation' to alter these 'parents' and generates a new population of chromosomes called 'children'. These children are evaluated and new elites are selected to continue the optimization process, as seen in Fig. 2.12. This process is then repeated until a satisfactory solution is found. GA are typically used for optimization

problems with large design spaces.

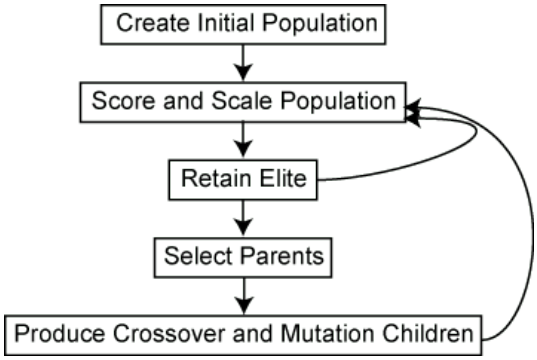

Figure 2.12: Genetic algorithm diagram showing the various steps of a GA optimization.[3]

PSO takes a different approach by generating a number of results using random design variables at various locations in the design space, called particles [30]. The design variables of these particles are then adjusted to look for local optimum solutions within the entire design space as can be seen in the subsequent graphs below, as seen in Fig. 2.13. PSO is known to be a robust method but can be computationally expensive, depending on the number of particles.

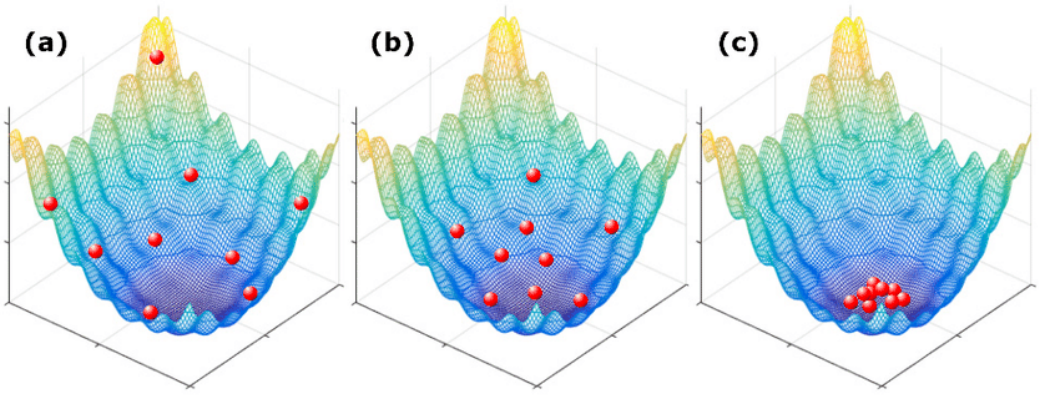

Figure 2.13: Particle swarming algorithm example showing how particles move towards a local minimum [31].

2.3. Software

Much like the optimization algorithms, the selected software can greatly impact the computational resources and must therefore be selected carefully. A similar approach to the previous section is taken, where a selection of possible software is discussed. This gives a good overview and allows for flexibility during the thesis.

2.3.1. Programming Software

During the thesis, any open-source software available will be utilized if applicable, as this greatly decreases the required development time. In recent years, many different ML packages have been developed that aid in the programming of various ML algorithms. Currently, Python is the preferred programming language for data scientists working on machine learning problems, as it is a very popular language with many packages and libraries available[4]. Therefore, all programming for the thesis will be done using Python. Other languages like Rust, Ruby, C++, Julia, or Java have been known to work

[3] Obtained from: `https://nl.mathworks.com/help/gads/what-is-the-genetic-algorithm.html`, accessed on 23-04-2023

[4] `https://www.aiplusinfo.com/blog/7-best-programming-languages-for-machine-learning/`, accessed on 12-04-2023

very well with Machine Learning and can even have a faster execution time, but currently lack the support that Python has[5]. Ultimately, it's up to the user to select a language that they are comfortable with and is appropriate for the task at hand.

The Pytorch package[6] will be used to aid in the programming of the neural networks of machine learning models. It has many features and is currently being used by over 50% of scientists, as listed on paperswithcode.com, a popular repository for scientific papers for various fields[7]. The key strengths of Pytorch are its support for constructing neural networks and the ability to efficiently utilize the physical hardware of a computer to speed up the execution time. Other great options are Tensorflow, Scikit-learn, and Keras [5].

2.3.2. Aerodynamic Analysis

The selection of the flow solver is very important, as it directly impacts the optimization process. While there are a great number of open-source flow solvers, the selection is severely limited by the available computational resources. Computational Fluid Dynamic (CFD) software (Pyaero [8], Flightstream[9], and Openfoam[10]) is most likely computationally too expensive when running a large number of optimizations and would be more suitable for the optimization of a singular design.

Athena Vortex Lattice (AVL) is a potential flow solver that is widely used in the aerospace field as a relatively low-fidelity solver that produces reliable and accurate results [32]. Additional benefits of AVL are its ease of use and integration into a Python program. As AVL is based on the Vortex Lattice Method (VLM), it can only be used for thin lifting surfaces at small angles and is unable to simulate viscous effects. Additionally, XFOIL[11], XFLR5 [12],and MSES [33] could also be great candidates to perform 2D flow analysis. This could be relevant if a hybrid analysis of 2D and 3D flow analysis is required.

In order to properly capture the complex response of High-Lift Devices (HLD), a high-fidelity flow solver is typically used. This is however not feasible as it would far exceed the capabilities of the available computational resources. A solution for this is to use empirical methods to manually calculate the response of the HLD in conjunction with flow solvers to calculate the aerodynamic response of the wing. A concern with using empirical methods is that data needs to be interpolated from a database. This could generate less accurate results than high-fidelity physics-based analysis methods. Olson [34] describes a comprehensive methodology for performing proper semi-empirical analysis of various types of control surfaces at low speeds using AVL. The main advantage of using semi-empirical over fully empirical methods is that this method generates more accurate results for more unconventional aircraft. This allows the optimizer to explore greater design spaces.

The empirical data can be obtained from ESDU[13]. ESDU is a database that contains various methods and data to aid in aerospace design. Included in this database are various empirical methods of determining the aerodynamic response of a great variety of HLD at low speeds [35]. ESDU data items 94028 [36], 94029 [37], and 94030 [38] include methods and data for determining the aerodynamic response of airfoils with plain trailing-edge, split, and single slotted flaps respectively. These methods also include the aerodynamic response of leading-edge devices. ESDU data item 91014 describes a method for determining the increments in maximum lift coefficient for HLD on an entire wing [39]. ESDU data item 92031 describes additional effects due to the deployment of both leading-edge devices with or without trailing-edge flaps. Finally, ESDU data items 93019 [40], 97009 [41], and 97011 [42] describe methods for determining the lift coefficient increment of a wing at zero angle of attack for single-slotted, split, and plain flaps respectively.

[5] https://careerfoundry.com/en/blog/data-analytics/best-machine-learning-languages/, accessed on 12-04-2023
[6] https://pytorch.org/docs/stable/index.html, accessed on 18-03-2023
[7] https://paperswithcode.com/trends, accessed on 11-04-2023
[8] https://pyaero.readthedocs.io/en/stable/, accessed on 14-05-2023
[9] https://www.darcorp.com/flightstream-aerodynamic-modeling-software-aiaa-11/, accessed on 14-05-2023
[10] https://openfoam.org/, accessed on 14-05-2023
[11] https://web.mit.edu/drela/Public/web/xfoil/, accessed on 12-3-2023
[12] http://www.xflr5.tech/xflr5.htm, accessed on 15-3-2023
[13] https://esdu.com/, accessed on 22-04-2023

2.4. State-of-the-Art

The current state of both aerospace optimization and ML application to automated design is now explored. First, aerospace design optimizations relating to control surface design are discussed to get an overview of the methods and techniques used. After this, various ML applications relating to design are explored to get an understanding of how ML methods are currently utilized.

2.4.1. Aerospace Design Projects

The goal of the thesis is to determine the practicality of machine learning applied to aerospace design. As one of the design problems concerns control surfaces, it is important to get an understanding of the design techniques and tools currently used. Two projects regarding the control surfaces design and sizing for a Prandtl plane are discussed. The Prandtl plane has a box wing configuration and has recently regained some attention in academia as the configuration theoretically achieves very low induced drag, see Fig. 2.14. The Prandt plane concept is interesting to investigate as it is a unique design that allows for control surface integration on the top and bottom wings. This could result in innovative methods being developed to generate unconventional control configurations.

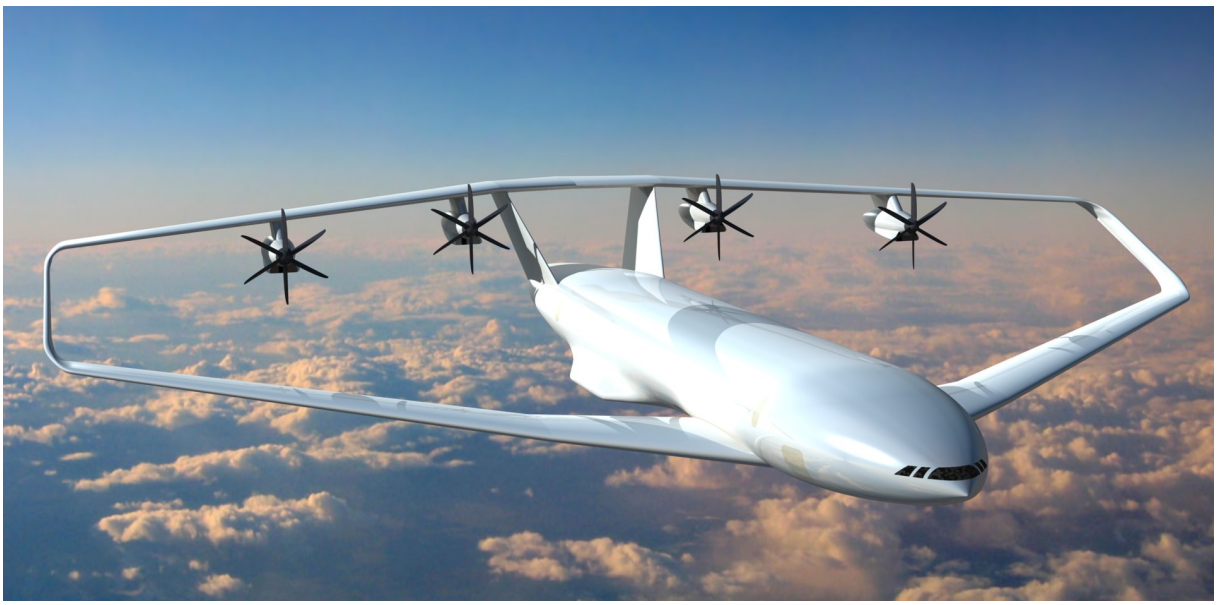

Figure 2.14: Artistic rendering of the Prandtl plane concept[14].

The first project to discuss contains a methodology for the automated control surface design and sizing for any fixed wing configuration [43]. The methodology consists of physics-based aerodynamic analysis, a control allocation algorithm, and the analysis of flight mechanics. The objective of the optimization is to minimize the total control surface area while adhering to some handling quality requirements. The initial design process concerns the selection and positioning of the control surfaces and is developed manually. A decision tree is used to position the control surfaces based on their ability to provide pitch, roll, and yaw control, see Fig. 2.15. This decision tree was then used to perform trade-off studies to determine the optimal configuration. The aerodynamic analysis was performed using a first-order panel method called VSAERO to evaluate the aerodynamic performances. An exhaustive search was then performed over the entire design space to evaluate the resulting designs. This optimization resulted in a configuration with outboard ailerons, inboard elevators, and a conventional rudder in the vertical tails.

[14] Obtained from: `https://phys.org/news/2018-05-radical-closed-wing-aircraft-greener-flight.html`, accessed on 2023-04-24

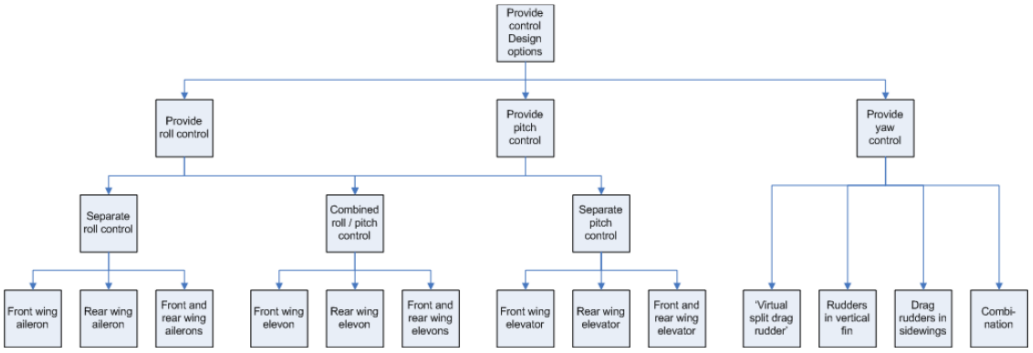

Figure 2.15: Decision tree used for the allocation of control surfaces for a Prandtl Plane [43].

The second project concerns the investigation of the impact of three control allocation algorithms on the sizing of the control surfaces [44]. The sizing optimization is performed with constraints based on the handling- and flying quality criteria, while the spanwise positions of the control surfaces remain constant, similar to the previously discussed project. The aerodynamic analysis is performed using AVL in order to create a database of static and dynamic coefficients. The conclusion mentions that the control surface areas could be further reduced when placed at the mid-wing positions. This shows that including the positioning within the optimization process could be quite beneficial.

2.4.2. Machine learning applications

In this section, various ML applications relating to design problems are discussed. The first ML application concerns the structural optimization of a frame using a RL algorithm [45]. The goal of the optimization is to minimize the total volume of the frame by reducing the cross-sectional area of the members. For the optimization, a DQN algorithm was used in conjunction with a PSO algorithm. The purpose of the DQN was to look for the general optimum solution in the design space while the PSO would perform the local optimization. The action list of the DQN consisted of increasing or decreasing the cross-sectional area of an entire row or column. This was done to simplify the design problem and decrease the computational requirements. Changing the cross-sectional area of each individual member would result in a significantly larger ANN that is expensive to evaluate. The result of the DQN optimization can be found in Fig. 2.16. After this, a PSO algorithm was used to minimize the total volume of the frame. The results show that the combination of RL and PSO was computationally more efficient than exclusively using PSO. Furthermore, this combination made the optimization more robust, as the RL agent prevented the PSO optimizer from starting at bad initial conditions. The RL algorithm is able to optimize a range of frames with varying numbers of rows and columns. Finally, the transfer of knowledge was successfully performed as the trained RL model was able to be re-used to optimize new frames of varying sizes.

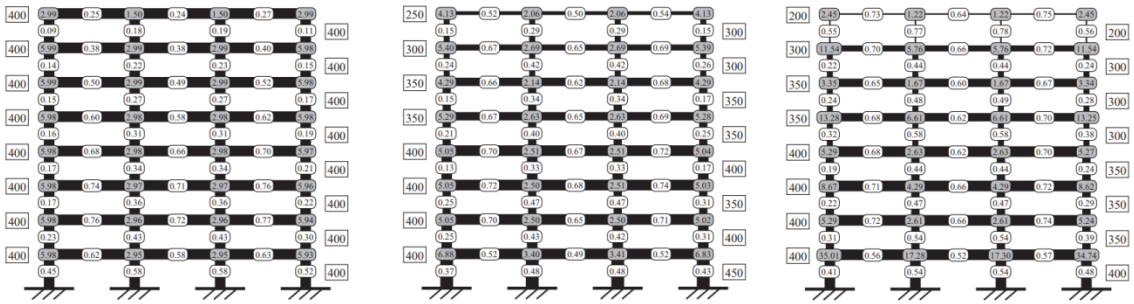

Figure 2.16: RL optimization of a frame after 10, 100, and 1000 iteration. The numbers on the members indicate stress ratios [45].

Within the field of aerospace, ML techniques are frequently applied to aerodynamic shape optimizations

(ASO) or to create surrogate models. An example of such a surrogate model uses an ANN to substitute CFD analysis to improve local aerodynamic performance [46]. This is achieved by utilizing a database of various airfoils that contain both their geometry and aerodynamic performance. The database is then used to train an ANN. The ANN essentially interpolates between the airfoils in the database to generate new airfoils with better aerodynamic performance. This is a great alternative to CFD analysis as the generation of new airfoils can be done much quicker. This could significantly decrease the computational efforts required to obtain objective function values for each iteration of an optimization process. However, this method is less effective for unconventional airfoils, as the ANN is only trained on conventional airfoils.

Hui and colleagues [47] describe an alternative method for ASO by using a deep RL algorithm. The algorithm is called Proximal Policy Optimization (PPO) and is used for multi-objective optimization problems. The main goal of this project is to optimize an airfoil by maximizing the lift-to-drag ratio in two different states while maintaining a constant thickness. The airfoil is parameterized using Free-Form Deformation (FFD) which is a method to describe the geometry of a 2D or 3D object. FFD works differently than other parameterization methods, as it alters a mapping frame instead of the object. The mapping frame consists of multiple control points that can be translated, shown in Fig. 2.17. The RL algorithm was able to reach its optimized design within 15% of the time it took a GA.

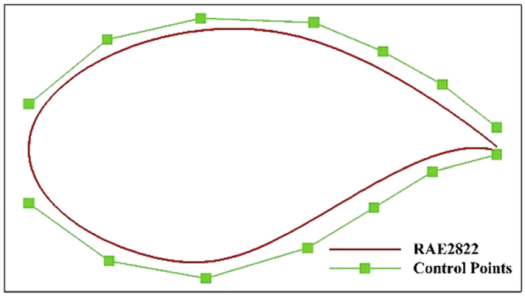

Figure 2.17: FFD parameterization of a RAE2822 airfoil [47].

In the article written by Li et al. [4], an overview is given of various ML algorithms that have successfully been used for ASO, together with a list of challenges that ASO currently faces. A big challenge is the efficient parameterization for compact geometric design. In the article, a modal parameterization method is discussed that is able to describe an airfoil with fewer parameters than conventional methods like FFD. This is quite advantageous as the number of variables greatly impacts the computational costs of the optimization. However, modal parameterization has a tendency to generate unrealistic airfoils with wavy surfaces. This leads to a very inefficient optimization, as all these failed airfoils have to be evaluated. A method has been developed to circumvent this issue. A deep learning model was developed to validate any geometric abnormality in the generated airfoils, see Fig. 2.18. Li et al. also highlight the need for an interactive design optimization tool. This has previously been infeasible, as high-fidelity solvers take too long to evaluate each iteration. However, with the advent of ML-based surrogate models, the feasibility of an interactive design tool increases. An example of this is Webfoil, an interactive airfoil analysis tool powered by surrogate models that can be accessed via a web browser [48].

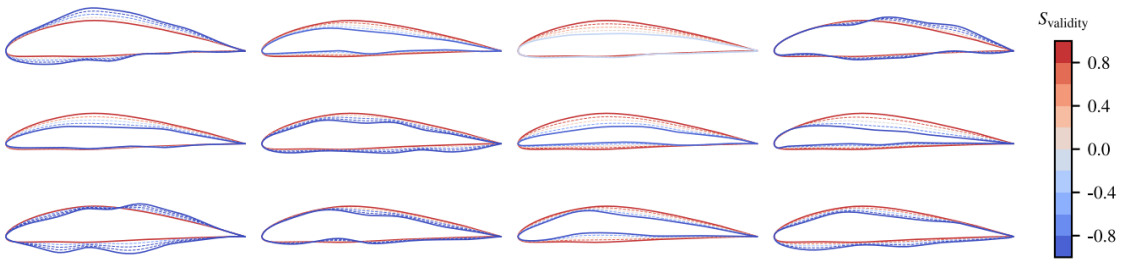

Figure 2.18: Geometric validity study of generated airfoils by an ANN model where infeasible surfaces are evaluated and rejected [4].

Li and Zhang [49] describe an interesting data-based approach to make wing shape optimization more interactive. This is achieved by substituting computationally expensive fluid dynamic solvers with fast and accurate data-based models. To train the data, an extensive database with over 135000 data points of various wing shapes and flight conditions was used. The error of the resulting model was within 0.4% compared to the CFD results. The FFD method is used to construct the geometry of the wing for CFD analysis, see Fig. 2.19. However, this method creates too much geometric freedom. The wing design is therefore translated using a modal parameterization to construct the design variables. The combination of a multi-objective design optimization with a surrogate found a huge decrease in computational requirements. While the optimization took 600 CPU hours to complete with CFD, the data-based optimization only took several minutes with a difference of one to two drag counts.

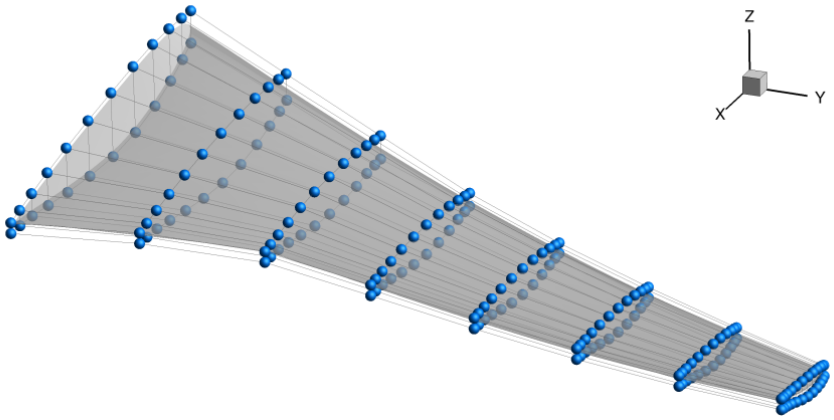

Figure 2.19: FFD representation of a wing, showing all control points [49].

2.5. Discussion

This section serves as a reflection on all information gathered so far. Now the utility of various ML techniques for automated design processes will be discussed. In order to do this, any shortcomings of the current automated design methods have to be established. As stated in section 2.2.1, the main drawback of MDAO is the fact that it is hard to adapt to a professional setting. While the ease of this adaption is far too complex to address here, it does give an indication that there is room for improvement. Additionally, KBE shows the need for a competent knowledge base for automated design problems.

While ML algorithms are being utilized for various engineering applications, the adaption of many ML techniques to automated aerospace design applications remains quite unexplored. ML models offer an alternative method of storing knowledge to KBE. During the training phase, the ML model accumulates all its knowledge. This is by far the most computationally expensive process. However, once a model

is trained, it is able to rapidly make new predictions. This is a great advantage over MDAO optimizers, as the MDAO framework has to run the entire optimization process for a single optimal solution.

Techniques like transfer learning, progressive learning, advisor agents, and multi-objective optimization are able to use the knowledge acquired through training and efficiently adapt a ML model to generate new designs. The use of these techniques could aid in the creation of an interactive design process, where the engineer is able to generate innovative design solutions.

Currently, complex decision-making is done using trade-off tables. Trade-off tables are a powerful tool utilized to make fair decisions on complex problems and can be found at the start of many design projects. This process is not automated, so a new trade-off table has to be created every time the conditions of the design problem change. As noted by Wahler [44], further research revealed that the conclusion of the trade-off table does not lead to the optimal solution. ML algorithms are able to optimize categorical variables quite well and are able to transfer their knowledge to adapt the ML model if the conditions change. This allows for the possibility to integrate the decision process of the trade-off table in the optimization loop. This could result in a more complete design space being explored, leading to more innovative designs. However, this should be done with caution as this could significantly increase the required computational effort.

ML techniques are quite versatile and have the potential to be applied to various different aerospace problems, both as a substitution for older methods and completely new techniques. In order to properly evaluate the adaption of ML techniques to traditional optimization problems, the performance of both optimization algorithms will have to be compared. How this comparison will be performed during the thesis is discussed in Chapter 4.

3

Research Questions

To reach the research objective, two main research questions are constructed. These questions are then further divided into sub-questions that will be answered during the thesis project.

Research Question 1: *How do the computational effort and objective function values of ML and traditional optimization algorithms compare in automated design problems?*

- Sub-question 1.1: *How does the amount of design variables affect the computational effort and objective function values of ML and traditional optimization algorithms?*
- Sub-question 1.2: How are the computational effort and objective function values of ML and traditional optimization algorithms related to the relative complexity of design problems?

Research Question 2: *How are the knowledge retention capabilities of ML algorithms related to the computational effort and objective function values in an automated design problem?*

- Sub-question 2.1: *How is the number of training iterations of a ML model related to the objective function values of the generation of new designs?*
- Sub-question 2.2: *To what extent can a multi-objective function of a trained ML model be altered with the goal of adjusting the resulting performance parameters of the design?*
- Sub-question 2.3: *How are the number of advisor agents and their RL model's similarity to the design objective related to the objective function value of new designs?*

4

Methodology

This chapter will describe the various actions that will be taken during the thesis to answer the research questions. To properly evaluate the performance of both the ML algorithms and traditional optimizers, it is vital that the design problems and analyses are designed simultaneously. This ensures that test results can be reused and computational effort is not wasted. This allows all research questions to be answered in an efficient manner.

4.1. Project Set-up

The set-up of the thesis has to be carefully defined at the start as it determines the scope of the project. Both the computational and software resources available are limited and will greatly determine what is possible to achieve within the given time frame. The main computational resource available is a Windows PC which means the selected software should not exceed the capabilities of this computer. In order to increase computational resources, Google Colab can be used[1]. This is a programming platform developed by Google that lets users utilize computational resources from Google.

4.2. Design problems

In order to answer the aforementioned research questions, two design optimization problems are created of different complexity. Both design problems will be solved using a traditional optimization algorithm and a ML algorithm. As mentioned in section 2.3, the traditional optimization algorithm can only be selected after the required computational effort is established. This means that the BO, GA, and PSO algorithms will have to be tested and the best-performing algorithm will be used for the remainder of the thesis.

As the goal of this thesis is to investigate machine learning performance for the conceptual design phase, the design problems are created to be solved using low-fidelity flow solvers. Because of this, the possible design problems are restricted to the analysis of rigid, subsonic, and small aircraft. In order to manage the available computational resources, the design problems are made as simple as possible, while just capable of answering the research questions.

4.2.1. Design problem 1

The first design problem is a simple aerodynamic optimization of a trapezoidal wing. The objective of this optimization is to maximize the CL/CD for the cruise condition. The design vector consists of the following parameters: wing span, root chord, taper ratio, sweep angle, dihedral, thickness root, thickness tip, and angle of attack. The optimization of this problem is quite straightforward as the design vector contains all parameters required to generate the wing model in AVL. Within AVL, the CL/CD can be calculated by setting the cruise conditions. The design variables are then iteratively adjusted by the optimization algorithms until the optimum wing design is found.

[1] `https://colab.research.google.com/`, accessed 08-05-2023

4.2.2. Design problem 2

The second design problem concerns the selection and sizing of the control surfaces of an aircraft with specified parameters. The goal of this design problem is to optimize three different objectives: the take-off and landing performance and the trimmed CL/CD at cruise. The optimizer is able to add the following HLD to the wing: plain, split, and slotted trailing-edge flap and a leading-edge slat. The empirical methods from ESDU are based on data from conventional wings. This means that unconventional configurations like the Prandtl plane could generate inaccurate results. For this reason, the initial optimizations will model the Cirrus SR20 aircraft. The design vector consists of the following parameters for each control surface: control surface type, chord %, width, and span-wise position. The take-off and landing performances are calculated using AVL.

4.2.3. The Machine Learning Model

For both design problems, the RL method is selected as the ML algorithm. RL seems like the most suitable algorithm to answer the research questions as it has a lot of advantages over other ML algorithms. A RL algorithm does not require any data to train, this is advantageous as the thesis will not be reliant on the availability and completeness of any database. This allows the set-up of the model to be quite flexible as all data is generated during the optimization process. Furthermore, techniques exclusive to RL algorithms could be vital for exploring the research objective.

A diagram is created to illustrate how the RL model is adapted to design problem 2, as can be seen in Fig. 4.1. The agent is a representation of the ML model and uses the policy to make decisions on how to optimize the aircraft design. The action list consists of the adjustments the agent can make to the values of the design variables. Once the agent has altered the control surfaces, various calculations are performed. The aerodynamic response of the selected control surface type is determined using the empirical methods from ESDU. AVL is able to directly integrate the results from the empirical methods into its calculations. Three different aerodynamic evaluations are performed using AVL for both the cruise and the take-off and landing conditions. The trimmed condition can be directly determined using AVL. Finally, the take-off and landing performances are calculated. The results of these calculations then determine the current state which is used to calculate the reward of that iteration. Based on this reward, both the policy and the RL model variables are updated.

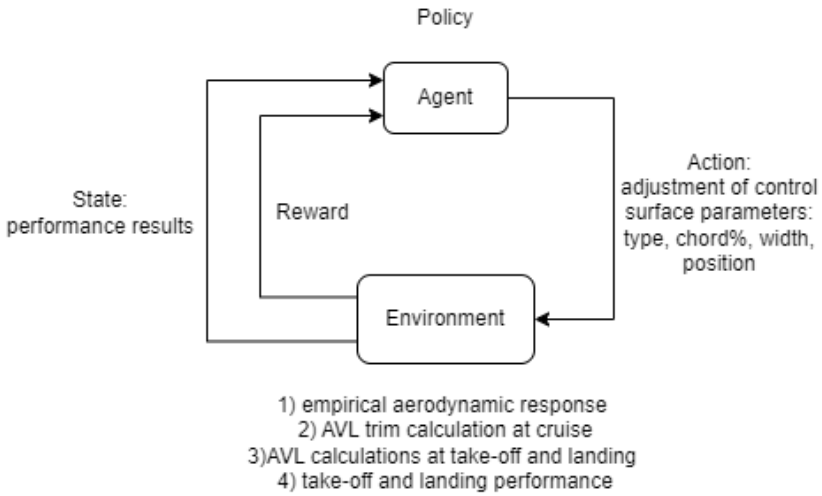

Figure 4.1: Diagram of the reinforcement learning model for design problem 2.

4.3. Analyses

Five analyses are executed to determine various performance metrics in order to answer each of the research sub-questions. These are now discussed, together with their respective expected outcomes.

4.3.1. Cardinality

The first metric will look at the effects of cardinality on the optimization results. This is done by increasing the number of design variables in the design vector and setting the remaining design variables to a constant value. This results in a number of optimizations equal to the length of the design vector. The process of adding variables to the design vector has to be done carefully, as both the order and the selection of design variables could significantly impact the results. In order to properly analyze this impact, a sensitivity study is performed. The sensitivity study ranks each design variable based on a sensitivity index [50]. The first optimization is then performed with the design variable that has the highest sensitivity index. Each subsequent optimization will then add an additional variable to the design vector until all optimizations are performed. A Python package called Salib is used to perform the sensitivity study [51].

Once the optimizations are completed, a comparison is made between the computational effort and objective function values achieved by both optimizers for different numbers of input variables. This performance metric is used to help answer sub-question 1.1. The performance of the cardinality analysis for both design problems is used to determine if the ML optimizer is able to outperform the traditional optimizer at any cardinality. Furthermore, the trends of both optimizer types can be compared for the entire range of design vector lengths.

During this analysis, a separate optimization will be performed using progressive learning. Here the original model will be retrained every time a variable is added to the design vector. The results from this can then be compared with the previous results to determine if progressive learning is a viable method for decreasing the computational effort.

4.3.2. Complexity

The second metric is the difference in both effort and objective function value between the traditional and ML optimizer. Here, the trends in the results of design problem 1 are compared with the results of the more complex design problem 2. Of interest is to see if these trends significantly differ between design problems if the complexity is increased in the form of a larger design vector. This could give context to when it would be applicable to use an ML algorithm over a traditional algorithm based on the complexity of an optimization problem. The results of this analysis are used to answer sub-question 1.2..

It is difficult to estimate if the ML model will outperform the traditional optimizer at this point. ML optimizers typically perform better when more data is used for its training. This should be the case for both design problem 2 and larger design vectors. It is therefore expected that the discrepancy between the objective function value of the traditional and ML optimizer shrinks as the design vector increases in length. Furthermore, it is expected that this discrepancy for the entire range of design vectors will be smaller for design problem 2, compared to design problem 1.

4.3.3. Knowledge Retention

The third metric concerns the knowledge retention of trained machine learning models. The main aim of this analysis is to determine a model's re-usability, as this would greatly decrease computational requirements. For this, transfer learning is applied to the models trained during the previous analyses to generate new aircraft designs. This is done by gradually adjusting a constant parameter the original model is trained on. For example, the second design problem has a fixed wing span as only the control surface parameters are contained in the design vector. The wing span parameter is then slightly increased and results are generated using a model trained on the original wing span. The goal of this analysis is to determine how the results degrade as the constant parameters move further away from the original values the model was trained on. Additionally, the amount of retraining iterations vs. the accuracy of results will be of interest. In order to check the accuracy of the retrained model, a new RL model is trained for each additional wing span value. The results of this analysis consist of the value of the objective functions of both the retrained model (at various amounts of retraining) and the newly trained model, for each design vector length.

This analysis is created to answer sub-question 2.1. The hypothesis for this is that the more computational effort the original ML model has spent, the better it will perform with little to no retraining. It is therefore expected that the model trained on design problem 2 with the complete design vector will

perform the best, as this model will have performed the most amount of calculations during its training phase. This results in the model having trained on more data, which should make it more effective in determining trends.

4.3.4. Design Flexibility

This metric will only be determined for design problem 2, as this is a multi-objective optimization problem. The goal of this analysis is to evaluate the ability to alter a trained model if the resulting performance characteristics are insufficient. For instance, decreasing the importance of the take-off and landing performances to increase the CL/CD at cruise. This can be done by simply adjusting the multi-objective function of a trained model and retraining it for a certain amount of iterations. The main advantage of this is that it requires fewer computational resources, compared to training a completely new model. This would make the optimizer more interactive, which could be advantageous in the conceptual design, as the engineer is able to accurately tweak a design to meet requirements. For this analysis, it is of interest to compare how the amount of retraining relates to the quality of the results.

The results of this analysis will be used to answer sub-question 2.2. It is expected that very little retraining is required to generate proper results. The results can be evaluated by training a new model on the new objective function.

4.3.5. Advisor Agents

Once the ML optimizer for design problem 2 has been successfully set up, it can be used to generate models trained on different aircraft configurations. Once a sufficient number of models are trained, they can be used as advisor agents to help with the training of a new model on a unique configuration. The goal of using advisor agents is quite similar to transfer learning as they are both methods for decreasing the required computational effort. However, the key difference is that with advisor agents a completely new model is being generated, whereas transfer learning adds layers to a neural network. Of interest for this analysis is the response to the number of advisor models and the similarity between the aircraft configuration of the advisor and the new model.

This analysis will be used to answer sub-question 2.3. It is expected that the more similar advisors are, the better they will perform. However, increasing the number of advisors should also increase the quality of results. Therefore the comparison between similarity and the number of advisors will be of interest.

5

Conclusion

5.1. Conclusions

In this literature review, the current state of automated design and related ML techniques are explored. This led to the discovery of several opportunities for ML to aid automated design. Predominantly, the ability of a ML model to adapt its knowledge to a new model could prove quite beneficial, as this could significantly reduce the required computational effort. Techniques that can achieve this are transfer learning, progressive learning, advisor agents, and multi-objective optimization. Transfer learning can be used to adapt the knowledge of one ML model to solve new tasks. Progressive learning can be used to solve increasingly complex problems by retraining a model used for a simpler problem. Advisor agents are trained RL models that can be used to efficiently solve new problems. Finally, ML could be used to adjust the multi-objective function to alter designs based on required performance parameters.

Additionally, the use of trade-off tables to make decisions could be eliminated as these can be prone to error and non-autonomous. Trade-off tables typically contain categorical variables which traditional optimization algorithms have a hard time properly optimizing. RL algorithms make decisions based on an environment. This allows them to adopt the trade-off table within the optimization loop.

The aim of the thesis is to explore the utility of various ML techniques to automated design processes. The results of the thesis could form the basis for the development of new automated design practices that are aided by ML techniques. The addition of ML to automated design could result in the generation of innovative designs. Additionally, ML could aid in the discovery of more efficient methods of optimization which could lead to a more interactive designing process.

References

[1] Imco van Gent. “Agile MDAO Systems - A Graph-based Methodology to Enhance Collaborative Multidisciplinary Design”. PhD thesis. Delft, 2019. DOI: https://doi.org/10.4233/uuid:c42b30ba-2ba7-4fff-bf1c-f81f85e890af.

[2] Maurice Frederik Maria Hoogreef. “Advise, Formalize AND Integrate MDO Architectures”. PhD thesis. Delft, 2017. DOI: https://doi.org/10.4233/uuid:cc2af611-6d78-4439-9b10-7e62ae579029.

[3] Joaquim R. R. A. Martins and Andrew Ning. *Engineering Design Optimization*. Cambridge University Press, 2021. ISBN: 781108833417.

[4] Jichao Li, Xiaosong Du, and Joaquim R.R.A. Martins. “Machine learning in aerodynamic shape optimization”. In: *Elsevier, Progress in Aerospace Science* 134 (2022). DOI: https://doi.org/10.1016/j.paerosci.2022.100849.

[5] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras TensorFlow*. O’Reilly, 2019. ISBN: 978-1-492-03264-9.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2017. ISBN: 9780262035613.

[7] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. second. MIT Press, 2020. ISBN: 9780262039246.

[8] Aristotelis Lazaridis, Anestis Fachantidis, and Ioannis Vlahavas. “Deep Reinforcement Learning: A State-of-the-Art Walkthrough”. In: *Journal of Artificial Intelligence Research* 69 (2020). DOI: https://doi.org/10.1613/jair.1.12412.

[9] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-Learning”. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI’16. Phoenix, Arizona: AAAI Press, 2016, pp. 2094–2100. DOI: https://doi.org/10.48550/arXiv.1509.06461.

[10] Thanh Thi Nguyen et al. “A Multi-Objective Deep Reinforcement Learning Framework”. In: *Engineering Applications of Artificial Intelligence* 96 (2020). DOI: https://doi.org/10.48550/arXiv.1803.02965.

[11] Conor F. Hayes and Roxana Rădulescu. “A practical guide to multi-objective reinforcement learning and planning”. In: *Autonomous Agents and Multi-Agent Systems* 36:26 (2022). DOI: https://doi.org/10.1007/s10458-022-09552-y.

[12] Jindong Wang and Yiqiang Chen. *Introduction to Transfer Learning*. second. Springer, 2023. ISBN: 978-981-19-7583-7.

[13] Marco Wiering and Martijn van Otterlo. *Reinforcement Learning : State-of-the-Art*. Springer, 2012. ISBN: 978-3-642-27644-6.

[14] Haytham M. Fayek, Lawrence Cavedon, and Hong Ren Wu. “Progressive learning: A deep learning framework for continual learning”. In: *Neural Networks* 128 (2020). DOI: https://doi.org/10.1016/j.neunet.2020.05.011.

[15] Andrei A. Rusu et al. *Progressive Neural Networks*. 2016. DOI: https://doi.org/10.48550/arXiv.1606.04671.

[16] Michael G. Madden and Tom Howle. “Transfer of Experience between Reinforcement Learning Environments with Progressive Difficulty”. In: *Artificial Intelligence Review* 21 (2004). DOI: http://dx.doi.org/10.1023/B:AIRE.0000036264.95672.64.

[17] Jaehong Yoon et al. *Lifelong Learning with Dynamically Expandable Networks*. 2018. DOI: https://doi.org/10.1016/j.neunet.2020.05.011.

[18] Kaiqing Zhang, Zhuoran Yang, and Tamer Basar. *Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms*. 2019. DOI: https://doi.org/10.48550/arXiv.1911.10635.

[19] Annie Wong et al. “Deep multiagent reinforcement learning: challenges and directions”. In: *Artificial Intelligence Review* 56 (2022). DOI: https://doi.org/10.1007/s10462-022-10299-x.

[20] Felipe Leno da Silva, Ruben Glatt, and Anna Helena Reali Costa. “Simultaneously Learning and Advising in Multiagent Reinforcement Learning”. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. AAMAS ’17. Sao Paulo, Brazil: International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 1100–1108.

[21] Sriram Ganapathi Subramanian et al. “Multi-Agent Advisor Q-Learning”. In: *Journal of Artificial Intelligence Research* 74 (2022). DOI: https://doi.org/10.48550/arXiv.2111.00345.

[22] Zhongkai Hao et al. “Physics-Informed Machine Learning: A Survey on Problems, Methods and Applications”. In: (2022). DOI: https://doi.org/10.48550/arXiv.2211.08064.

[23] Ehsan Haghighata et al. “A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics”. In: *Computer Methods in Applied Mechanics and Engineering* 379 (2021). DOI: https://doi.org/10.1016/j.cma.2021.113741.

[24] Joaquim R. R. A. Martins and Andrew B. Lambe. “Multidisciplinary Design Optimization: A Survey of Architectures”. In: *AIAA Journal* 51 (2013). DOI: https://doi.org/10.2514/1.J051895.

[25] Forest Lee Flager and John Haymaker. *A Comparison of Multidisciplinary Design, Analysis and Optimization Processes in the Building Construction and Aerospace Industries*. 2007.

[26] G La Rocca and M J L van Tooren. “Knowledge-based engineering to support aircraft multidisciplinary design and optimization”. In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering.* International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 1041–1055. DOI: {http://dx.doi.org/10.1243/09544100JAERO592}.

[27] I.O. Sanyaa and E.M. Shehab. “A framework for developing engineering design ontologies within the aerospace industry”. In: *International Journal of Production Research* 53 (2015). DOI: http://dx.doi.org/10.1080/00207543.2014.965352.

[28] Jia Wu et al. “Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization”. In: *JOURNAL OF ELECTRONIC SCIENCE AND TECHNOLOGY* 17 (2019). DOI: https://doi.org/10.11989/JEST.1674-862X.80904120.

[29] Tinu Theckel Joy et al. “A Flexible Transfer Learning Framework for Bayesian optimization with Convergence Guarantee”. In: *Expert Systems with Applications* 115 (2020). DOI: https://doi.org/10.1016/j.eswa.2018.08.023.

[30] Dan Simon. *Evolutionary Optimization Algorithms*. John Wiley Sons, Inc., 2013. ISBN: 978-0-470-93741-9.

[31] Kayvan F. Tehrani et al. “Adaptive optics stochastic optical reconstruction microscopy (AO-STORM) by particle swarm optimization”. In: *Biomedical Optics Express* 8 (2017). DOI: https://doi.org/10.1364/BOE.8.005087.

[32] Mark Drela and Harold Youngren. *MIT AVL User Primer*. 2022. URL: https://web.mit.edu/drela/Public/web/avl/AVL_User_Primer.pdf (visited on 04/22/2023).

[33] Mark Drela. *A User’s Guide to MSES 3.05*. 2007. URL: https://web.mit.edu/drela/Public/web/mses/mses.pdf (visited on 05/22/2023).

[34] Erik D. Olson. “Semi-Empirical Prediction of Aircraft Low-Speed Aerodynamic Characteristics”. In: *American Institue of Aeronautics and Astronautics* (2015). DOI: http://dx.doi.org/10.2514/6.2015-1679.

[35] *ESDU 97002: Information on the use of Data Items on high-lift devices*. IHS-Groups UK- based subsidiary, Technical Indexes Ltd., 1992.

[36] *ESDU 94028: Increments in aerofoil lift coefficient at zero angle of attack and in maximum lift coefficient due to deployment of a plain trailing-edge flap, with or without a leading-edge high-lift device, at low speeds.* IHS-Groups UK- based subsidiary, Technical Indexes Ltd., 1994.

[37] *ESDU 94029: Increments in aerofoil lift coefficient at zero angle of attack and in maximum lift coefficient due to deployment of a trailing-edge split flap, with or without a leading-edge high-lift device, at low speeds.* IHS-Groups UK- based subsidiary, Technical Indexes Ltd., 1994.

[38] *ESDU 94030: Increments in aerofoil lift coefficient at zero angle of attack and in maximum lift coefficient due to deployment of a single-slotted trailing-edge flap, with or without a leading-edge high-lift device, at low speeds.* IHS-Groups UK- based subsidiary, Technical Indexes Ltd., 1994.

[39] *ESDU 91014: Maximum lift of wings with trailing-edge flaps at low speeds.* IHS-Groups UK- based subsidiary, Technical Indexes Ltd., 2000.

[40] *ESDU 93019: Wing lift increment at zero angle of attack due to deployment of single-slotted flaps at low speeds.* IHS-Groups UK- based subsidiary, Technical Indexes Ltd., 2000.

[41] *ESDU 97009: Wing lift coefficient increment at zero angle of attack due to deployment of trailing-edge split flaps at low speeds.* IHS-Groups UK- based subsidiary, Technical Indexes Ltd., 2003.

[42] *ESDU 97011: Wing lift coefficient increment at zero angle of attack due to deployment of plain trailing-edge flaps at low speeds.* IHS-Groups UK- based subsidiary, Technical Indexes Ltd., 2003.

[43] D. A. J. van Ginneken, Mark Voskuijl, and Michel J. L. van Tooren. “Automated Control Surface Design and Sizing for the Prandtl Plane”. In: *American Institue of Aeronautics and Astronautics* (2010). DOI: http://dx.doi.org/10.2514/6.2010-3060.

[44] N.F.M. Wahler. *The Impact of Control Allocation on Optimal Control Surface Positioning and Sizing*. Delft, 2022.

[45] Kazuki Hayashi and Makoto Ohsaki. “Reinforcement learning for optimum design of a plane frame under static loads”. In: *Engineering with Computers* 37 (2021). DOI: https://doi.org/10.1007/s00366-019-00926-7.

[46] Shuyue Wang et al. “Database self-expansion based on artificial neural network: An approach in aircraft design”. In: *Elsevier, Aerospace Science and Technology* 72 (2017). DOI: https://doi.org/10.1016/j.ast.2017.10.037.

[47] Xinyu Hui et al. “Multi-object aerodynamic design optimization using deep reinforcement learning”. In: *AIP Advances* 11 (2021). DOI: https://doi.org/10.1063/5.0058088.

[48] Univesity of Michigan. *Webfoil homepage*. 2023. URL: http://webfoil.engin.umich.edu/ (visited on 04/26/2023).

[49] Jichao Li and Mengqi Zhang. “Data-based approach for wing shape design optimization”. In: *Aerospace Science and Technology* 112 (2021). DOI: http://dx.doi.org/10.1016/j.ast.2021.106639.

[50] Max Rein. *Adjustment of the MDAO Problem Formulation using Sensitivity Analysis to Reduce the Computational Cost within Aircraft Design*. Delft, 2022. URL: %7Bhttp://resolver.tudelft.nl/uuid:aadb7e34-62af-47bb-86bc-8ccb6dd44cd8%7D.

[51] Jon Herman and Will Usher. “SALib: An open-source Python library for Sensitivity Analysis”. In: *The journal of open source software* 2 (2017). DOI: https://joss.theoj.org/papers/10.21105/joss.00097.

[52] Jan Mariens. *Wing Shape Multidisciplinary Design Optimization*. Delft, 2012. URL: %7Bhttp://resolver.tudelft.nl/uuid:3b1c6432-cfbf-4fec-894b-9f6b870015f5%7D.

[53] Ebrahim Sumiea et al. *Deep Deterministic Policy Gradient Algorithm: A Systematic Review*. Nov. 2023. DOI: 10.21203/rs.3.rs-3544387/v1.

[54] Shagun Sodhani, Amy Zhang, and Joelle Pineau. *Multi-Task Reinforcement Learning with Context-based Representations*. 2021. DOI: https://doi.org/10.48550/arXiv.2102.06177.

[55] Jacob Beck et al. *A Survey of Meta-Reinforcement Learning*. 2023. DOI: https://doi.org/10.48550/arXiv.2301.08028.

III

Appendices

A

Additional Transfer Learning Results

Fig. A.1 shows the generated ASK21 wing planforms with the highest objective value. Here we see that the original SR22 and TL models were both unable to optimize the planform. The CL model was only able to slightly improve the design. All models shrink the wing span, even the SR22 model. This is interesting, as the results from the F50 wing optimization would imply that the model would increase this variable until it reaches the upper bound. In Fig. A.2, the average changes each framework has made to the design variables of the ASK21 wing relative to the starting wing are plotted. The results are taken from 500 optimizations per model. There is quite a bit of variance between the wing span, kink location and wing twist. Here we see that the SR22 model tries to increase the wing span, despite the fact that this will not result in a more efficient wing. This is consistent with the results from the F50 optimizations. The ASK21 and CL model were able to adapt and learn to decrease the wing span. However, the TL model was not able to do so as it still increased the wing span.

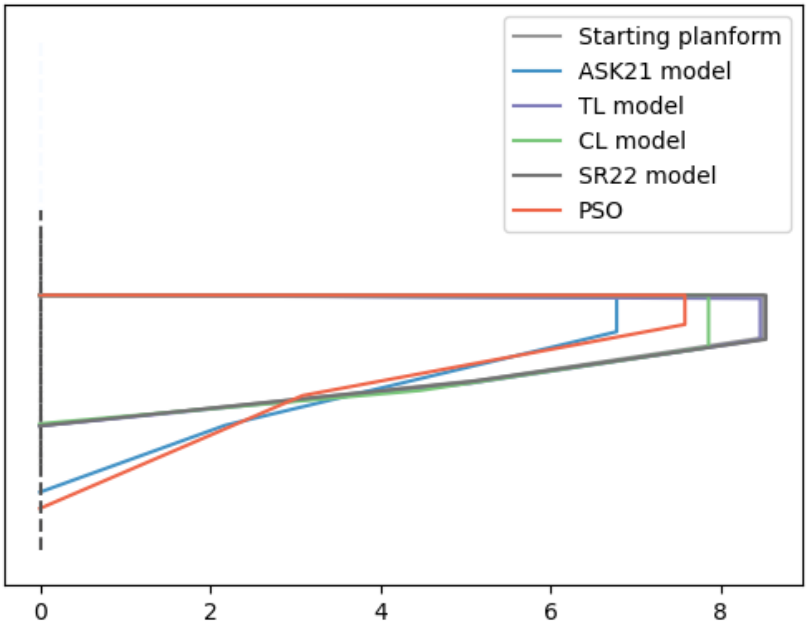

Figure A.1: Optimized ASK21 wing planforms using the ASK21 models and PSO.

Similarly, the design variable changes from the F50 models are illustrated in Fig. A.3. These results are consistent with the conclusions drawn in the paper. We see that the CL model increases the root chord, thus introducing a kink to the planform. This is interesting as the original SR22 model actually decreases the root chord. We do see a lot of variance between the variables again. This variance is unexpected for the TL and F50 model as they were able to reach similar average objective values.

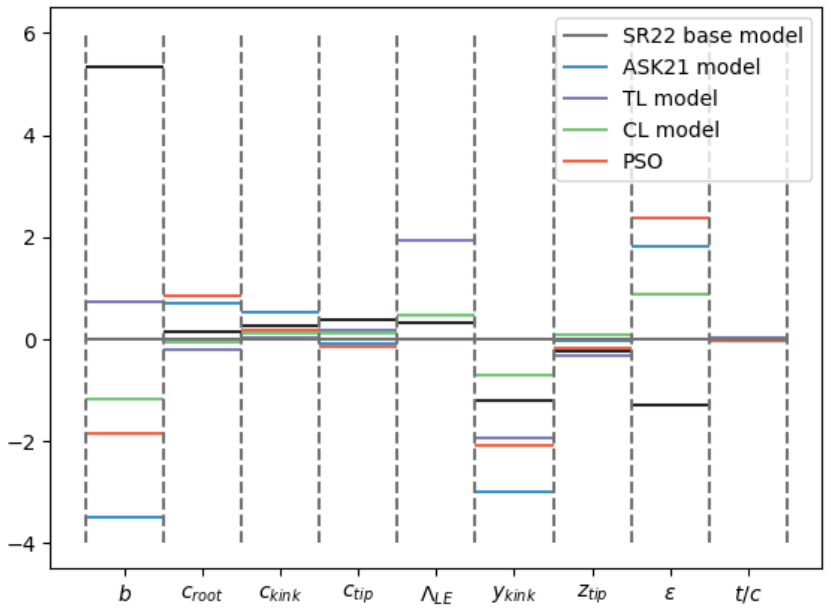

Figure A.2: Average design variable changes by the ASK21 models.

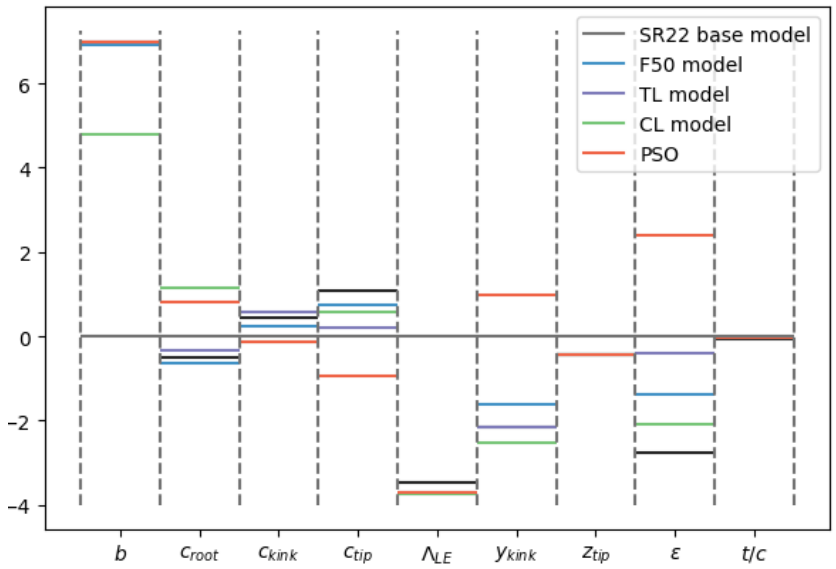

Figure A.3: Average design variable changes by the F50 models.

B

Additional Environment Evaluation

In Fig. B.1, the rewards collected by an older version of the framework are plotted. The gain R_1 in this version of the framework scaled on the basis of the increase of the new maximum objective value at the current step relative to the previous maximum objective value it has reached during the optimization. This was done to encourage the agent to improve the design as quickly as possible and not continuously collect small rewards. However, due to the limited changes the agent can make to design vector at each step, the model learned to exploit the environment. The agent would optimize the wing while simultaneously violating the constraints, as can be seen by the increasing cumulative penalties before step 20. This would delay the collection of the gain to increase its value. Just before termination at step 20, the agent complies with all constraints. This caused the relative increase in objective value to be rather large. The total rewards collected are now dominated by that first gain its has collected. Interestingly, the model did try to further optimize the wing after collecting this gain. However, the improvements it makes are rather small and it fails to optimize the wing on many of the steps. This behaviour should be discouraged, as the agent is intentionally violating constraints and not efficiently trying to optimize the wing.

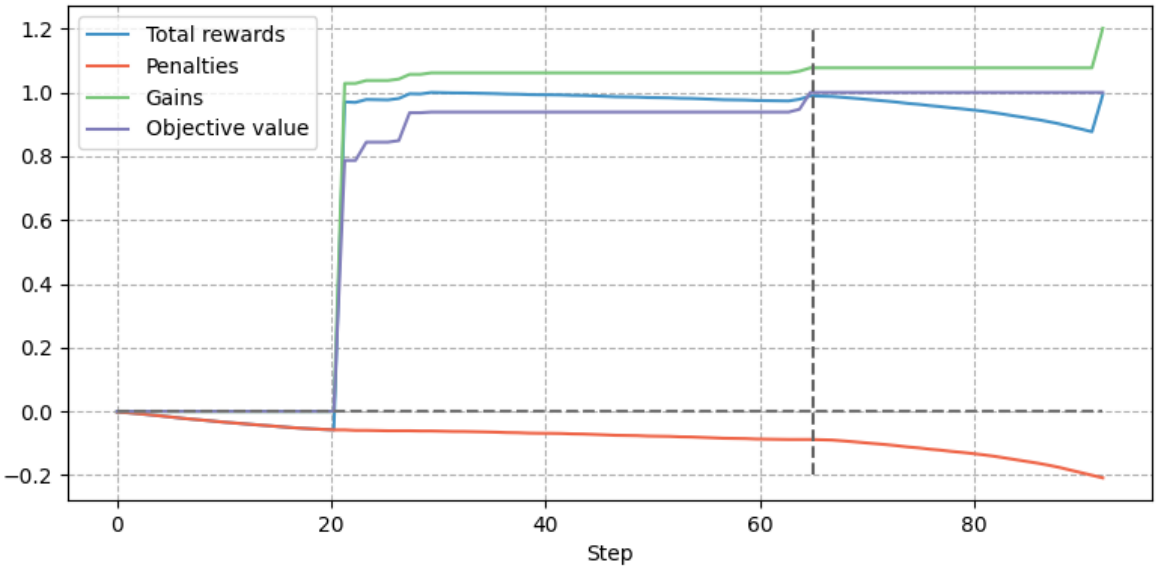

Figure B.1: Normalized rewards obtained from the environment during a single optimization by an old version of the framework, objective value reached: 30.80. The step at which the optimum wing was found is indicated by the grey line.

The reward collection of 500 DA50 wing optimizations are plotted using the DA50 model, see Fig. B.2. The collected rewards scale exponentially with the reached objective values. Here we see that the

environment calibrated to optimize the SR22 wing is well suited to optimize the DA50 wing. This is to be expected, as both reference aircraft are quite similar.

Figure B.2: Rewards obtained from the environment vs objective value reached by the DA50 model.

Much like the DA50 model, the F50 model shows a great response to the environment, see Fig. B.3. Once again the rewards scale exponentially with the reached objective values. The penalties do seem to increase the higher the objective value becomes. This means that in order to reach a high objective value, the model makes more mistakes by violating the constraints or not optimizing the wing at every step.

Figure B.3: Rewards obtained from the environment vs objective value reached by the F50 model.

As expected, the ASK21 wing could not be optimized without re-calibration of the environment, see Fig. B.4. While the collected gains did scale with the reached objective value, they were not enough to guide the agent through the design space. This further illustrates the importance of scaling the gains exponentially with the reached objective value. The agent collected quite a high number of penalties and the results are scattered, explaining the low average objective values reached by the ASK21 model.

Figure B.4: Rewards obtained from the environment vs objective value reached by the ASK21 model.

C

Additional Insights on the Framework Setup

C.1. Optimization Algorithm Selection

The Genetic and Bayesian optimization algorithms [30] were also used to solve the design problem. Both algorithms had comparable results to the PSO algorithm at the start of the thesis when the design problem had a smaller design vector and fewer constraints. However, when increasing the size of the design vector and the number of constraints, their performance degraded rather quickly. The Baysian optimization algorithm was able to optimize the wing about 70% of what PSO was able to but this took over 24 hours to achieve. The GA was unable to find a higher objective value than the starting condition in 2 hours.

C.2. Flow Solvers

XFOIL[1] was also used to determine the profile drag coefficient of the designed wings [52]. However, this resulted in unrealistic shapes like heavily tapered wings and quadrupled the computational efforts. This illustrates that the selected solvers will heavily impact the generated wings shapes. This could be particularly interesting for this project, as the design problem mostly generated rectangular wings. The F50 is a much larger, heavier and faster aircraft than the SR22, so it is expected that the planform shapes would be different between the two aircraft. The use of CFD solvers could increase the diversity of wing shapes the DRL framework can learn to generate. The optimization of transonic aircraft could be especially interesting. However, the CFD analysis of an entire wing can be rather costly. In order to incorporate CFD solvers into the DRL framework, it is suggested a RL algorithm is seleted that is more efficient in its training process than the PPO algorithm. This would decrease the number of evaluations required to train the agent, thus decreasing the computational efforts.

C.3. Alternative RL algorithms

An alternative RL algorithm called DDPG was also tested [53]. This algorithm is quite popular and should theoretically be a great candidate for a DRL framework. However, its performance was significantly worse than the PPO algorithm for this particular design problem. This performance discrepancy could be due to incompatibilities between the algorithm and the design problem or the calibration of the environment. This illustrates the different response each particular RL algorithm can have to the complete setup of an optimization framework, and its selection should be carefully considered.

There are alternative RL methods that incorporate TL capabilities into their framework during the training phase. Two examples are Multi-Task [54] and Meta Learning [55]. The Multi-Task Learning technique trains a RL model to solve multiple problems during the initial training phase by having a dynamic environment. The environment is adjusted during the training phase to force the agent to solve new

[1] `https://web.mit.edu/drela/Public/web/xfoil/`, accessed on 12-3-2023

tasks. Once trained, the model should then be able to solve tasks that are similar to the range of tasks its trained on. The Meta Learning technique trains a model to generate new results using data of a new design space. It then re-calibrates its results for alternative design spaces.

These techniques could be used to create a single framework that is able to optimize the wings for a large range of aircraft. These frameworks could be used for multiple different projects without the need for a costly retraining phase. However, these techniques require the construction of a more complex framework. The calibration of the environment to optimize the wings of multiple aircraft could be challenging. These techniques are quite new, so their performance and consistency are less proven.