EPFL - MSc in Computational Science and Engineering

Master Thesis

Aerodynamic shape optimization via surrogate modelling with convolutional neural networks

Supervisors:
Prof. Pascal Fua
Dr. Pierre Bruno Baqué

Student:
Francesco Bardi
orcid.org/0000-0001-7761-8640
Student n°: 269236

Computer Vision Laboratory - CVLAB

Academic year 2018-2019
Submitted on June 21th, 2019

Abstract

Aerodynamic shape optimization has become of primary importance for the aerospace industry over the last years. Most of the method developed so far have been shown to be either computationally very expensive, or to have low dimensional search space. In this work we present how Geodesic Convolutional Neural Networks [1] can be used as a surrogate model for a computational fluid dynamics solver. The trained model is shown to be capable of efficiently exploring a high dimensional shape space. We apply this method to the optimization of a fixed wing drone, the *eBee Classic*.

Chi ha provato il volo
camminerà guardando il cielo
perché là è stato
e là vuole tornare.

Leonardo da Vinci

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Aerodynamic shape optimization has been of interested since the very early stages of aeronautical engineering. With recent consideration about energy efficiency and emission control this field has flourished to become one one the biggest challenges of the aerospace industry. Several techniques have been developed to address the optimization problem, but most of the methods developed so far are computationally very demanding. The recent advancements in geometric deep learning [2] have opened a new perspective on how to approach the problem. Moreover, open source libraries, such as TensorFlow and PyTorch, have given the opportunity to access state of the the art deep learning tools.
We rely on previous work by Baqué et al. [1] and follow-up developments in the context of EPFL's DeepShape project. In this project we build on top of the developed code base for geometric Deep Learning and optimization.

1.1 Aerodynamic shape optimization methods

In its most generic form an aerodynamic shape optimization problem can be formulated as follows:

$$\begin{aligned} &\underset{\boldsymbol{X}\in\Omega}{minimize} \quad \mathcal{F}(\boldsymbol{X}) \\ &subject\ to \quad \begin{cases} \mathcal{G}_i(\boldsymbol{X}) \leq 0 \\ \mathcal{H}_i(\boldsymbol{X}) = 0 \end{cases} \end{aligned}$$

where is Ω denotes the sets of feasible shapes. The objective function $\mathcal{F}$, and the constraints $\mathcal{G}$ and $\mathcal{H}$, represent quantities of interest that are in general the results of a computationally expensive numerical simulation, in this case the solution of the Navier Stokes equations. The problem can be addressed mainly via two approaches:

- Direct optimization
- Surrogate modelling

The first approach is the most expensive in terms of computations as it requires to probe the numerical simulator at each optimization step. There are two possible strategies to find a new query point $\boldsymbol{X}$; gradient based methods, such as the adjoint method, rely on sensitivity analysis while gradient-free methods include Genetic Algorithms and Particle Swarm Optimization techniques. Surrogate modelling consists in building a model F that can be used as an approximation of the numerical simulator and optimized with respect to the design variables; two different strategies are available:

- Parametric models
- Non parametric models

Parametric methods build a regressor which interpolates $\mathcal{F}$ on a low dimensional parametrization of the geometry $\boldsymbol{X}$; the most popular one in the field is the Kriging method. Our approach belongs to the non parametric models since we use a Geodesic Convolutional Neural Network which takes as input the mesh representing the geometry, and not some predefined interpolation parameters. It is important to stress out that all the mentioned methods (including our method) do not offer the guarantee to converge to the global optimum. However the interest is to find a design with better performances than the baseline configuration. We refer the interested reader to [3] for a more comprehensive and detailed analysis.

1.2 Deep learning shape optimization

The outline of the present work is as follows. In Chapter 2 we review the theory of neural networks and deep learning, describing the operations used in our model and its architecture. In Chapter 3 we describe the automatic framework based on OpenFOAM that we have used to obtain the ground truth values. In Chapter 4 we show how the initial geometry can be parametrized, introducing a new set of design variables, and we illustrate the optimization process. Chapter 5 is dedicated to the analysis of the obtained results, including a comparison of the aerodynamic performances of the *eBee Classic* with of one optimal shapes. Finally, in Chapter 6 we summarize the achievements of the work and we give a directions for possible future developments.

Chapter 2

Deep learning

Our intention is to use a deep neural network that is capable of predicting the aerodynamic performances of a given shape (in the form of a triangulated surface) that can be used as a surrogate model of a computational fluid dynamics solver. After recalling the basic structure of a neural network we show how the standard operations can be extended to perform regression on non euclidean data. Finally we describe the structure of our model.

2.1 Feedforward neural networks

The basic structure of a feedforward neural network with one input layer, L hidden layers and one output layer is shown in Figure 2.1. Denoting by $\boldsymbol{x}^{(0)}$ the input vector $\boldsymbol{x}^{(k+1)}$ is obtained via:

$$\begin{aligned}
&\boldsymbol{x}^{(k+1)} = \phi((\boldsymbol{W}^{(k+1)})^T \boldsymbol{x}^{(k)} + \boldsymbol{b}^{(k+1)}) \\
&\boldsymbol{W}^{(1)} \in \mathbb{R}^{D\times K}, \boldsymbol{b} \in \mathbb{R}^K \\
&\boldsymbol{W}^{(k)} \in \mathbb{R}^{K\times K}, \boldsymbol{b} \in \mathbb{R}^K \quad for \quad k = 2, ..., L \\
&\boldsymbol{W}^{(L+1)} \in \mathbb{R}^{K\times 1}, \boldsymbol{b} \in \mathbb{R}^1
\end{aligned}$$

$W^k_{i,j}$ is he weight of the connection between $x_i^{(k-1)}$ and $x_j^{(k)}$, $\boldsymbol{b}$ is called the bias term and the activation function $\phi(x) \in \mathbb{C}^0 \colon \mathbb{R} \to \mathbb{R}$ is applied point wise. Common activation functions are:

- Sigmoid: $\frac{1}{1+\exp^{-x}}$
- Tanh: $\frac{\exp^x - \exp^{-x}}{\exp^x + \exp^{-x}}$
- Rectified linear Unit (ReLU): $\max\{0, x\}$
- Leaky ReLU: $\max\{\alpha x, x\}$

It can be proven that this architecture can approximate any continuous function on a bounded domain [4].

2.1.1 Training

Assuming that we use the neural network $f(\boldsymbol{x}_n)$ to regress a value y_n, being $\ell(x)$ a chosen loss function, the total loss is:

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}_n = \frac{1}{N}\sum_{n=1}^{N} \ell(f(\boldsymbol{x}_n), y)$$

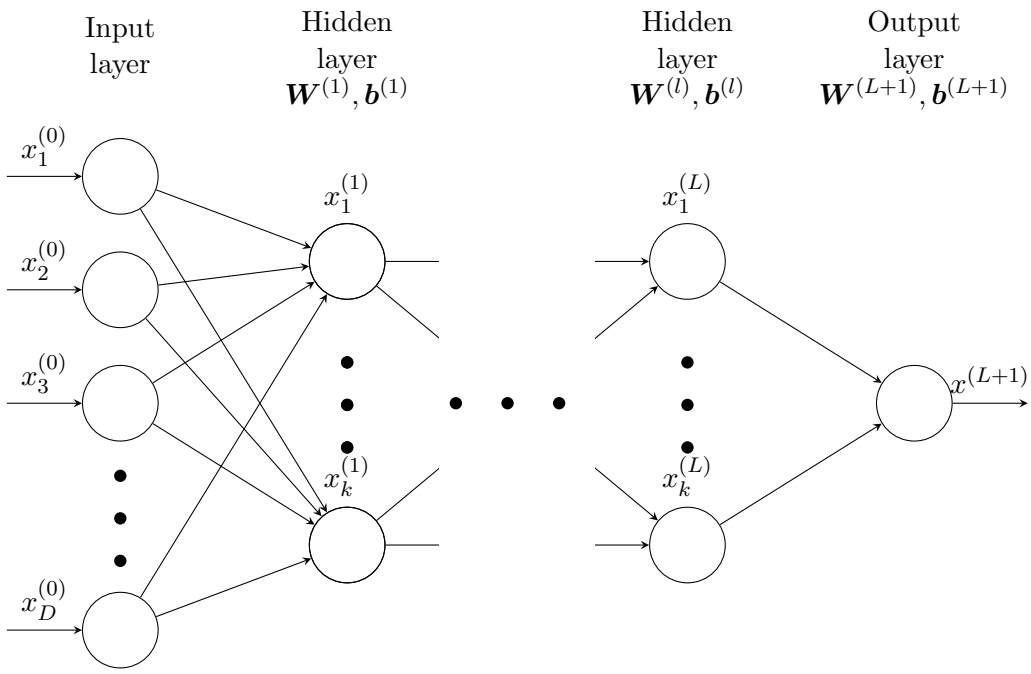


Figure 2.1: Basic structure of a feed forward neural network

The gradient of $\mathcal{L}$ with respect to the weights and bias terms can be efficiently obtained via the Backpropagation Algorithm 1. $\mathcal{L}$ can then be minimized using an appropriate optimization algorithm e.g. Stochastic Gradient Descent (SGD) or Adam [5].

Parameter: $\boldsymbol{W}^{(k)}, \boldsymbol{b}^{(k)} \quad k = 1, \ldots, L+1$
Data : $y_n, \boldsymbol{x}_n$
Result : $\frac{\partial \mathcal{L}_n}{\partial W_{i,j}^{(l)}}$

for $k = 0, ..., L$ **do**
  $\boldsymbol{z}^{(k+1)} = \boldsymbol{W}^{(k+1)})^T \boldsymbol{x}^{(k)} + \boldsymbol{b}^{(k+1)}$
  $\boldsymbol{x}^{(k+1)} = \phi(\boldsymbol{z}^{(k+1)})$
end
$\delta^{(L+1)} = \ell' \left(y_n - \mathbf{x}^{(L+1)}\right) \phi' \left(z^{(L+1)}\right)$
for $k = L, .., 1$ **do**
  $\boldsymbol{\delta}^{(l)} = \left(\boldsymbol{W}^{(l+1)} \boldsymbol{\delta}^{(l+1)}\right) \odot \phi' \left(\boldsymbol{z}^{(l)}\right)$
end
$\frac{\partial \mathcal{L}_n}{\partial W_{i,j}^{(l)}} = \delta_j^{(l)} x_i^{(l-1)}$
$\frac{\partial \mathcal{L}_n}{\partial b_j^{(l)}} = \delta_j^{(l)}$

Algorithm 1: Backpropagation algorithm

2.2 Convolutional neural networks

Convolutional neural networks (CNNs) have been successfully applied in a wide range of computer vision tasks, ranging from image segmentation to pose estimation. CNNs are typically built by the composition of convolutional and pooling layers. A convolutional layer transforms an input $f_l(x) \quad l = 1, \ldots, p$ in a q dimensional feature map $g_m(x) \quad m = 1, \ldots, q$:

$$g_m(x) = \phi\left(\sum_{l=1}^{p} \left(f_l \star \gamma_{m,l}\right)(x)\right)$$

where $\phi(x)$ is the previously introduced activation function, and the convolution operation between a function $f(x)$ and a compact support kernel $\gamma(x)$ over a domain Ω is defined as:

$$(f \star \gamma)(x) = \int_{\Omega} f(x - x')\gamma(x')dx'$$

Denoting by $\mathcal{N}(x)$ the neighborhood of x and by $P(x)$ a permutation invariant function the pooling operation reads:

$$g_l(x) = P\left(\left\{f_l\left(x'\right) : x' \in \mathcal{N}(x)\right\}\right), \quad l = 1, \ldots, q$$

When $P\left(f(x)\right) = \|f(x)\|_{L^p_{\mathcal{N}(x)}} \quad p = 1, 2, \infty$ we call it average-, energy-, or max-pooling respectively. If Ω represents a 2D image then the result of the convolution between an image I and a $M \times N$ kernel K is:

$$S(i,j) = (I * K)(i,j) = \sum_{m}^{M}\sum_{n}^{N} I(i+m, j+n)K(m,n)$$

Differently from a dense layer a convolutional layer is characterized by a sparse weight matrix [6](see Figure 2.2) and weight sharing (different edges share the same weights). Moreover it exhibits shift invariance; this property makes CNNs very suitable for treating images, or more in general, data on regular grids.

2.2.1 Geodesic convolution

The standard convolution operation cannot be used for non Euclidean domain, due to lack of shift invariance, and the generalization is not trivial. Geometric deep learning is an active field of research that devolopes convolutional kernels for non euclidean domain such as graphs and manifolds [2]. A possible approach [7] is to build a set of weighting functions $v_k(x) \quad k = 1, \ldots, K$ to construct a patch operator of the form:

$$D_k(x)f = \int_{\mathcal{N}(x)} f\left(x'\right) v_k\left(x, x'\right) dx', \quad k = 1, \ldots, K$$

being $\mathcal{N}(x)$ neighbourhood of x. At this point it is possible to compute the convolution between f and a kernel g of size K as:

$$f \star g = \sum_{k \in 1,\ldots,K} g_k D_k f$$

For a surface defined by a set of vertices $\boldsymbol{V} \in \mathbb{R}^{N_V \times 3}$ and an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N_V \times N_V}$, if we denote by $\mathcal{N}^i = \{j : A_{i,j} = 1\}$ the set of neighbours of the vertex $\boldsymbol{V}^i$ the interpolation operator can be written:

$$(D_k f)^i = \textstyle\sum_{j \in \mathcal{N}^i} f^j \exp\left(\frac{-\left(\rho\left(\boldsymbol{V}^i, \boldsymbol{V}^j\right) - \alpha_{k\rho}\right)^2}{\sigma_{k\rho}}\right) \exp\left(\frac{-\left(\theta\left(\boldsymbol{V}^i, \boldsymbol{V}^i\right) - \alpha_{k\theta}\right)^2}{\sigma_{k\theta}}\right)$$

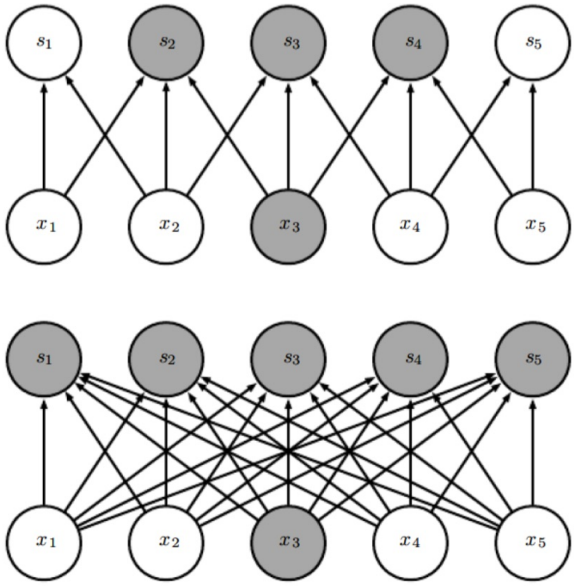


Figure 2.2: Sparse weight connections [6].

where ρ and θ constitutes a local polar system of coordinates [8]. The parameters $\alpha_{k\rho}, \alpha_{k\theta}, \sigma_{k\rho}, \sigma_{k\theta}$ are manually initialized but are treated as learnable parameters during the training process. In Figure 2.3 we show the 0.8 level sets for an example of the interpolation operator with $K = 24$.

2.3 Our model

Relying on the introduced operations it is possible to build a Geodesic Convolutional Neural Network (GCNN) that given a surface $\left\{\boldsymbol{V}^{(m)}, \boldsymbol{A}^{(m)}\right\}$, can predict its aerodynamic performances, replacing an expensive call to a numerical simulator. The outputs of the network are N_Y scalar fields $\boldsymbol{Y} \in \mathbb{R}^{N_V \times N_Y}$ and N_z global scalars $\boldsymbol{z} \in \mathbb{R}^{N_z}$. More specifically, given a data set of M shapes, we train the network $F_{\boldsymbol{\omega}}(\boldsymbol{V}, \boldsymbol{A})$ to optimize the following function:

$$\mathcal{L}(\boldsymbol{\omega}) = \sum_{m}^{M} \left\| F_{\boldsymbol{\omega}}^{\boldsymbol{Y}}(\boldsymbol{V}^{(m)}, \boldsymbol{A}^{(m)}) - \boldsymbol{Y}^{(m)} \right\|_{Frob}^{2} + \lambda_{\boldsymbol{z}} \left\| F_{\boldsymbol{\omega}}^{\boldsymbol{z}}(\boldsymbol{V}^{(m)}, \boldsymbol{A}^{(m)}) - \boldsymbol{z}^{(m)} \right\|_{2}^{2}$$

where $\boldsymbol{\omega}$ is the vector of trainable parameters. In our case we aim to regress 3 scalar global quantities $\boldsymbol{z} = \left[L, D, M_y\right]$ the lift, the drag and the pitching moment respectively and 2 scalar fields, the pressure and the x component of the wall shear stress $\boldsymbol{Y} = \left[\boldsymbol{p}, \boldsymbol{\tau}_x\right]$. Even though our final scope is to build a surrogate model for the global scalars only, it is useful to include the fields as it was shown to prevent overfitting. Training the network to predict more fields could have increased the accuracy of the model, however it would have also increased the memory requirements, we hence have chosen the two fields that are more correlated with $\boldsymbol{z}$. The set of training shapes is obtained by introducing a parametrization of a reference geometry and by applying random deformations (Chapter 4), whereas the ground truth values are obtained using a computational fluid dynamics solver (Chapter 3). The loss function is optimized using the Adam optimizer.

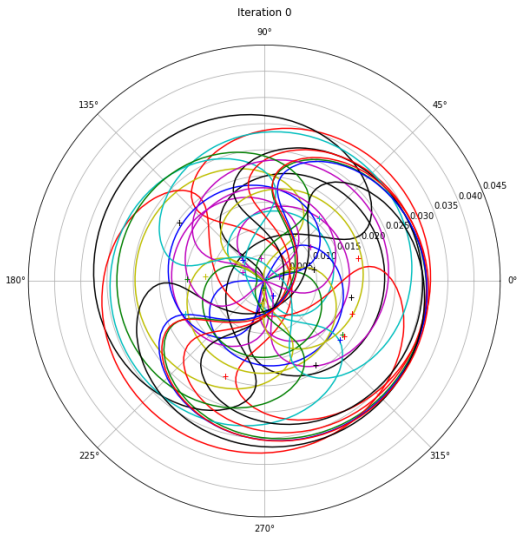


Figure 2.3: 0.8 level set for $K = 24$ randomly initialized
Image credit: Luca Zampieri, orcid.org/0000-0002-5369-0514

2.3.1 GCNN architecture

The architecture of our network resembles the one shown in Figure 2.4 [1]. The first part of the model $F_0^{\boldsymbol{\omega}}$ preprocess the input $\{\boldsymbol{V}, \boldsymbol{A}\}$ and constructs a set of features by means of the previously introduced geodesic convolution operations. $F_{\boldsymbol{z}}^{\boldsymbol{\omega}}$ is then used to predict the global scalars $\boldsymbol{z}$ via average pooling and two dense layers. $F_{\boldsymbol{Y}}^{\boldsymbol{\omega}}$ generates instead the scalars fields $\boldsymbol{Y}$ relying on an additional set of geodesic convolutions and pointwise operations. Differently from [1] the new network takes advantage of a GPU efficient implementation of geodesic convolutions, removing the need to use a Cube-Mesh mapping.

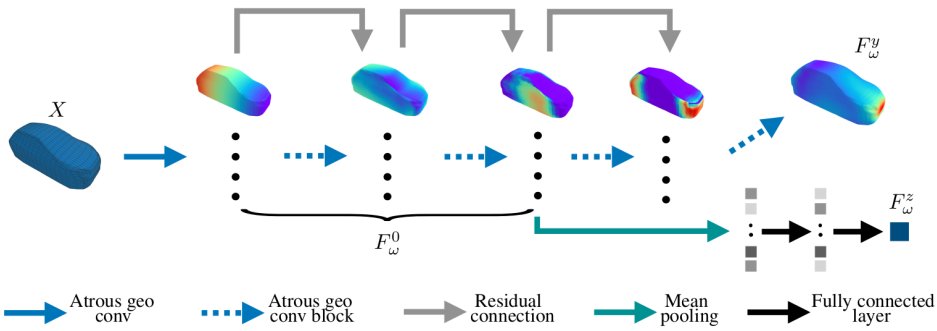


Figure 2.4: Network architecture [1]

Chapter 3

Data set generation

The ground truth values $\boldsymbol{Y}$, $\boldsymbol{z}$ for each geometry are obtained using OpenFOAM, a free software mainly dedicated to Computational Fluid Dynamics (CFD). After giving an introduction to the equations and the numerical methods used by the framework, we describe the characteristics of the automatic pipeline that we have built.

3.1 Navier Stokes equations

The Navier Stokes equations is a system of non linear partial differential equation describing the motion of a fluid. Indicating by u_x, u_y, u_z, ρ, μ the three components of the velocity, the density and the dynamic viscosity respectively, for a Newtonian incompressible fluid it yields:

$$\rho\left(\frac{\partial u_x}{\partial t}+u_x\frac{\partial u_x}{\partial x}+u_y\frac{\partial u_x}{\partial y}+u_z\frac{\partial u_x}{\partial z}\right)=-\frac{\partial p}{\partial x}+\mu\left(\frac{\partial^2 u_x}{\partial x^2}+\frac{\partial^2 u_x}{\partial y^2}+\frac{\partial^2 u_x}{\partial z^2}\right)+\rho g_x \tag{3.1a}$$

$$\rho\left(\frac{\partial u_y}{\partial t}+u_x\frac{\partial u_y}{\partial x}+u_y\frac{\partial u_y}{\partial y}+u_z\frac{\partial u_y}{\partial z}\right)=-\frac{\partial p}{\partial y}+\mu\left(\frac{\partial^2 u_y}{\partial x^2}+\frac{\partial^2 u_y}{\partial y^2}+\frac{\partial^2 u_y}{\partial z^2}\right)+\rho g_y \tag{3.1b}$$

$$\rho\left(\frac{\partial u_z}{\partial t}+u_x\frac{\partial u_z}{\partial x}+u_y\frac{\partial u_z}{\partial y}+u_z\frac{\partial u_z}{\partial z}\right)=-\frac{\partial p}{\partial z}+\mu\left(\frac{\partial^2 u_z}{\partial x^2}+\frac{\partial^2 u_z}{\partial y^2}+\frac{\partial^2 u_z}{\partial z^2}\right)+\rho g_z \tag{3.1c}$$

$$\frac{\partial u_x}{\partial x}+\frac{\partial u_y}{\partial y}+\frac{\partial u_z}{\partial z}=0 \tag{3.1d}$$

The first three equations represents the conservation of the momentum whereas the fourth, called the continuity equation, represents the conservation of mass. For a derivation we refer the interested reader to [9]. OpenFOAM allows to find a numerical solution to the Navier Stokes Equations by means of the finite volume discretization method. However when the flow is turbulent Direct numerical simulation (DNS) of Equations 3.1 is computational expensive since the grid needs to resolve the Kolmogorov length scale $L/\eta \sim Re^{3/4}$ in each direction, meaning that we would need $\approx Re^{9/4}$ nodes, where the Reynolds number is defined as:

$$Re=\frac{\rho U L}{\mu} \tag{3.2}$$

In our case $L \approx 0.5$ [m], $U = 12$ $[\frac{\mathrm{m}}{\mathrm{s}}]$ so $Re \approx 4\times 10^5$. This method, which is not usually used for industrial applications, would be unfeasible for our case. A trade off between computational cost and solution accuracy has to be found. A common approach, when the interest is in the average properties of the flow and there is no need to capture the turbulent fluctuations, is to

use Reynolds-averaged Navier Stokes equations (RANS). There are several RANS models, and choosing the appropriate models highly depends on the application. We selected the SST $k-\omega$ that according to [10] suits best for external aerodynamics. This method introduces two additional transport equations to model the Reynold stress tensor, one for the turbulent kinetic energy k and another one for the turbulence frequency ω. It is a combination of the $k-\epsilon$ and the $k-\omega$ models, and is designed to avoid the extra sensitivity of the $k-\omega$ to the inlet free-stream turbulence properties, while modelling very well the near wall region. OpenFOAM implements a near wall treatment that automatically switch between the low-Re and the wall function approach according to the position of the near wall grid centroid as described in [11]. This feature is particularly well suited to build an automatic pipeline.

3.2 Simulation setup

The simulation constitutes of the following steps:

- Creation of a volumetric mesh, representing the flow domain around the drone.
- Impose boundary conditions.
- Iteratively solve the flow equations.
- Post process the result, obtaining the quantities of interest.

3.2.1 Mesh

The `snappyHexMesh` utility is used to automatically build a mesh from an object surface. Indicating by $\boldsymbol{l} = [L_x, L_y, L_z]$ the dimensions of the geometry bounding box and by $\boldsymbol{x}_c$ the bounding box center, the flow domain is defined as the box with extrema $\boldsymbol{x}_{min}, \boldsymbol{x}_{max}$:

$$\boldsymbol{x}_{min} = \begin{bmatrix} x_c - 2.5L_x \\ y_c - 3.5L_y \\ z_c - 3.5L_z \end{bmatrix} \boldsymbol{x}_{max} = \begin{bmatrix} x_c + 10.5L_x \\ y_c + 3.5L_y \\ z_c + 3.5L_z \end{bmatrix} \tag{3.3}$$

We consider the object aligned with the positive x axis. Moreover we add refinement zone with extrema $\boldsymbol{y}_{min}, \boldsymbol{y}_{max}$:

$$\boldsymbol{y}_{min} = \begin{bmatrix} x_c - 1.25L_x \\ y_c - 0.75L_y \\ z_c - 1.25L_z \end{bmatrix} \boldsymbol{y}_{max} = \begin{bmatrix} x_c + 4.5L_x \\ y_c + 0.75L_y \\ z_c + 1.25L_z \end{bmatrix}$$

The dimension h of the first cell is set in order to achieve a desired dimensionless wall distance y^+:

$$\begin{aligned} Re &= \frac{\rho U L_x}{\mu} \\ C_f &= (2\log_{10}(Re) - 0.65)^{-2.3} \\ u &= \sqrt{0.5 C_f U^2} \\ h &= y^+ \nu / u \end{aligned}$$

where the Schlichting correlation [9] for a turbulent boundary layer on a smooth flat plate, is used to estimate the local skin-friction coefficient C_f. The biggest cell of the domain has dimension

$H = 16h$. We have chosen a target $y^+ = 300$, expecting the first cell centroid to be in the logarithmic sublayer. However we found that the maximum y^+ on the object surface was no bigger than 150 for the simulated geometries. Seeking $y^+ < 1$, in order to better capture flow separation for high angle of attack, would have increased excessively the number of grid elements. For each geometry a quality check for the mesh ensures that the following have acceptable values:

- Mean and max mesh non-orthogonality.
- Max skewness.
- Max aspect ratio.

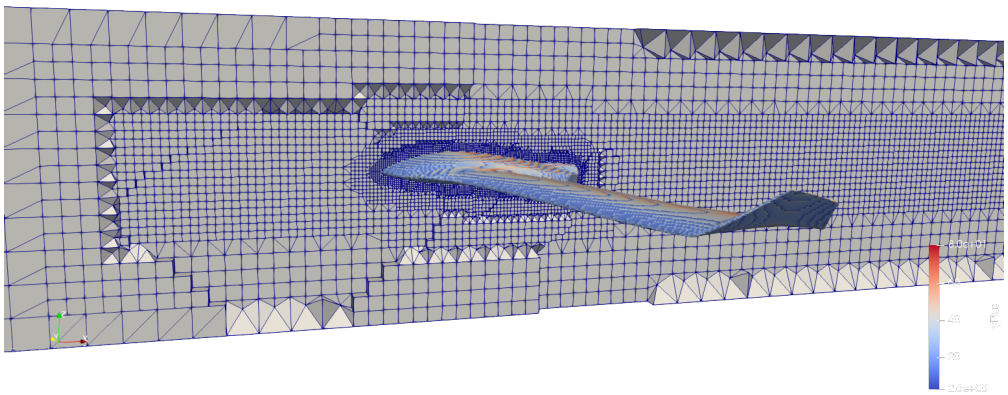


Figure 3.1: Crinkle clip on the y-normal plane, showing the mesh and the y^+ value on the drone surface.

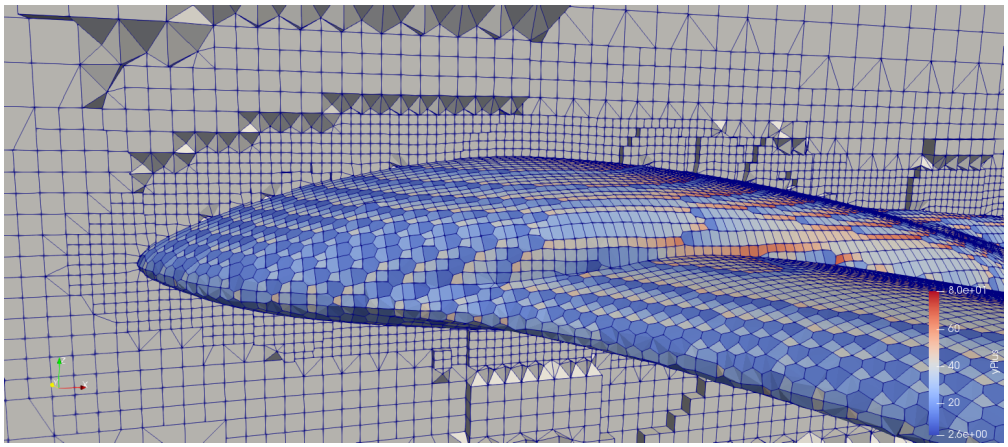


Figure 3.2: Close up of Figure 3.1

3.2.2 Boundary condition

In Table 3.1 we report the boundary conditions for each field and boundary patch. The `freestream` boundary condition implements a free-stream condition which switches between a Dirichlet (`fixedValue`)

Table 3.1: Boundary condition for each field and boudary patch.
`domainFrontiers` denotes all the bounding planes of the domain a part from the inlet

Boundary patch	U	p	k	omega	nut
inlet	fixedValue (12,0,0)	freeStreamPressure	fixedValue	fixedValue	fixedValue
domainFrontiers	freestream	freeStreamPressure	zeroGradient	zeroGradient	zeroGradient
objectSurface	fixedValue (0,0,0)	zeroGradient	kqRWallFunction	omegaWallFunction	nutkWallFunction

and an homogeneous Neumann (`zeroGradient`) condition depending on the sign of the flux. The value of turbulent kinetic energy k and the turbulence frequency ω at the inlet patch are set according to:

$$
\begin{aligned}
I &= 0.16 Re^{-\frac{1}{8}} \\
l &= 0.07 L \\
k &= \frac{3}{2}(UI)^2 \\
\omega &= k/l
\end{aligned}
$$

where I is the turbulence intensity and l the turbulence length scale.

3.2.3 Solution

The simulation is performed with the incompressible flow solver `simpleFoam` by initializing the flow using the potential flow solver `potentialFoam`. `simpleFoam` implements the Semi-Implicit Method for Pressure Linked Equations (SIMPLE) [12] which is a segregated method to iteratively solve the Naviers Stokes equations (in our case the RANS equations).

3.3 Quantities of interest

We describe here how the ground truth values $\boldsymbol{Y} = \left[\boldsymbol{p}, \boldsymbol{\tau}_x\right]$ and $\boldsymbol{z} = \left[L, D, M_y\right]$, used to train the neural network, are obtain from the results of the simulations. The scalar values are computed using the post processing utility which integrates the pressure p and the wall shear stress $\boldsymbol{\tau}$ over the object surface.

$$
\begin{aligned}
\mathbf{F}_p &= \sum_i \rho_i \mathbf{s}_{f,i} \left(p_i - p_{ref}\right) \\
\mathbf{F}_v &= \sum_i s_{f,i} \cdot \left(\mu \mathbf{R}_{\text{dev}}\right) = \sum_i \left\|s_{f,i}\right\|_2 \boldsymbol{\tau}_{f,i}
\end{aligned}
$$

Where $\boldsymbol{s}_{f,i}$ is the face area vector of the i-th boundary cells of the volumetric mesh and $\boldsymbol{R}_{\text{dev}}$ is the deviatoric stress tensor:

$$
\sigma = -p\boldsymbol{I} + \mu \boldsymbol{R}_{\text{dev}} = -\begin{pmatrix} p & 0 & 0 \\ 0 & p & 0 \\ 0 & 0 & p \end{pmatrix} + \mu \begin{pmatrix} 2\frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} & \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} \\ \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} & 2\frac{\partial v}{\partial y} & \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \\ \frac{\partial w}{\partial x} + \frac{\partial u}{\partial z} & \frac{\partial w}{\partial y} + \frac{\partial v}{\partial z} & 2\frac{\partial w}{\partial z} \end{pmatrix}
$$

The lift and drag values are then: $L = F_{p,z} + F_{v,z}$ and $D = F_{p,x} + F_{v,x}$ respectively. The two scalar fields p and $\boldsymbol{\tau}_x$ are interpolated from the boundary of the volumetric mesh to the geometry vertices $\boldsymbol{V}$.

Chapter 4

Optimization

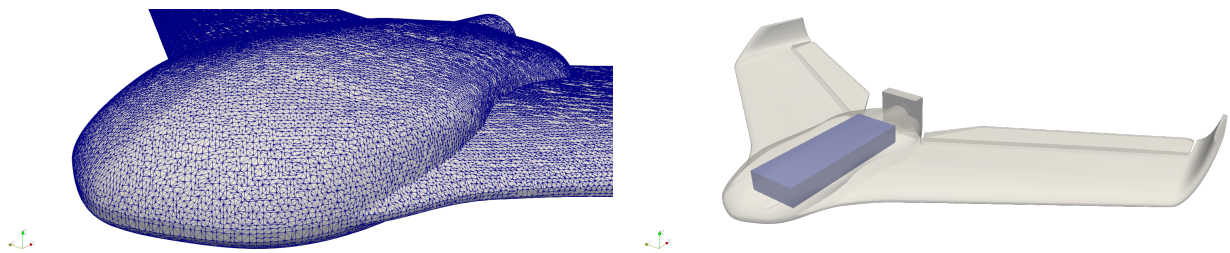
The aim of the project is to optimize the the aerodynamic performances of a fixed drone while satisfying aerodynamics and geometric constraints. Denoting by Ω the set of feasible geometries the problem reads:

$$\begin{aligned} \underset{\boldsymbol{V}\in\Omega\subset\mathbb{R}^{N\times 3}}{\textit{minimize}} \quad & -L(\boldsymbol{V})/D(\boldsymbol{V}) \\ \textit{subject to} \quad & \begin{cases} L_{obj} - L(\boldsymbol{V}) \leq 0 \\ M_y(\boldsymbol{V}) = 0 \end{cases} \end{aligned}$$

In order to reduce the number of optimization variables, to ensure a smooth deformation of the surface, and to handle the geometric constraints, a parametrization of the vertices $\boldsymbol{V} = \boldsymbol{P}(\boldsymbol{V}^{(0)}, \boldsymbol{z})$ is introduced. $\boldsymbol{V}^{(0)}$ denotes the original vertices and $\boldsymbol{z}$ is a vector of the new design variables.

4.1 Pre processing

The neural network architecture is designed to take as input a triangulate mesh. As a first step we converted the 3D CAD format file to a triangulated surface. Since the original drone contains inner parts which should not be fed to the network we have used OpenFOAM to create volumetric mesh around it, and we have then retrieved the external surface only with the `surfaceMeshTriangulate` utility. Finally MeshLab was used to decimate the geometry, obtaining a final file with 108974 vertices which is shown in Figure 4.1(a).



(a) A close up on the reference surface $\{\boldsymbol{V}^{(0)}, \boldsymbol{A}^{(0)}\}$

(b) The blue box is the fuselage. The box in the tail represents the zone where the vertices cannot be deformed.

Figure 4.1: *eBee Classic*

4.2 Parametrization

We define $P^{(i)}(\boldsymbol{z}^{(i)}, \boldsymbol{V})$ as a continuous and differentiable transformation of the input vertices $\boldsymbol{V}$ according to a design variables vector $\boldsymbol{z}^{(i)}$. Moreover we indicate as $P^{(i)} \circ P^{(j)}$ the composition of two parametrization. We now report the three types of parametrization that we used:

- Radial Basis Function interpolation
- Rotation
- Projection

4.2.1 Radial Basis Functions

In order to smoothly deform the surface we rely on the Radial Basis Functions interpolation tecnique (RBFs) [13]. Indicating by $\boldsymbol{c}_i$ the i-th row of the $N_C \times 3$ matrix $\boldsymbol{C}$ representing a set of the control points, we want to find a function $\boldsymbol{d}\colon \mathbb{R}^3 \to \mathbb{R}^3$ that interpolates $\boldsymbol{D} \in \mathbb{R}^{N_C \times 3}$ at the control points locations. The RBF map is then defined as follows:

$$d_j(\boldsymbol{x}) = \sum_{i=1}^{N_C} W_{i,j} \phi(\|\boldsymbol{x} - \boldsymbol{c}_i\|_2) + p_j(\boldsymbol{x}) \quad j = 1, 2, 3 \tag{4.1}$$

where $\phi\colon \mathbb{R}^3 \to \mathbb{R}$ is a radially symmetric kernel function and $\boldsymbol{p}\colon \mathbb{R}^3 \to \mathbb{R}^3$ is a polynomial term, usually of first order, used to recover translation and rotations:

$$\begin{aligned} &\boldsymbol{p}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \\ &\boldsymbol{A} \in \mathbb{R}^{3\times 3}, \quad \boldsymbol{b} \in \mathbb{R}^3 \end{aligned}$$

Common radial basis functions are:

- Gaussian splines: $\phi(\|\boldsymbol{x}\|_2) = e^{-\|\boldsymbol{x}\|_2^2/r^2}$
- Multiquadratic biharmonic splines: $\phi(\|\boldsymbol{x}\|_2) = \sqrt{\|\boldsymbol{x}\|_2^2 + r^2}$
- Inverse multiquadratic biharmonic splines: $\phi(\|\boldsymbol{x}\|_2) = \frac{1}{\sqrt{\|\boldsymbol{x}\|_2^2 + r^2}}$
- Thin-plate splines: $\|\boldsymbol{x}/r\|_2^2 \ln(\|\boldsymbol{x}/r\|_2)$

In this project we have used the multiquadratic biharmonic splines family setting with $r = 0.4$ [m]. If we aim to interpolate a deformation of the control points $\boldsymbol{D}$, it must hold that:

$$d_j(\boldsymbol{c}_i) = D_{i,j} \quad i = 1, \ldots, N_C \quad j = 1, 2, 3$$

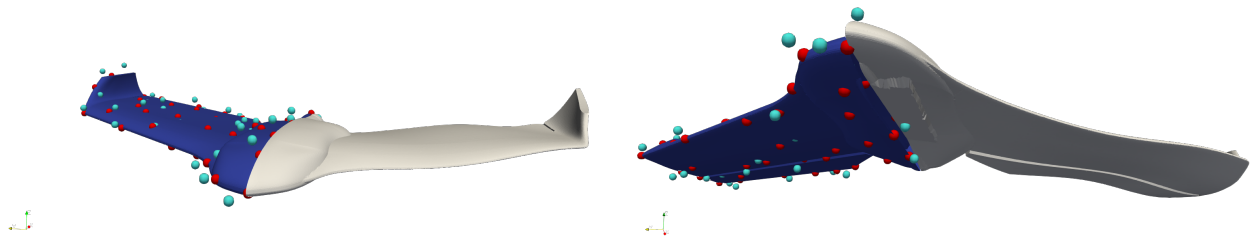
Moreover, by imposing:

$$\begin{aligned} \sum_{i=1}^{N_C} W_{i,j} &= 0 && j = 1, 2, 3 \\ \sum_{i=1}^{N_C} C_{i,j} W_{i,j} &= 0 && j = 1, 2, 3 \end{aligned}$$

the $N_C \times 3 + 12$ unknown parameters can be found, solving the following symmetric linear system:

$$\underbrace{\begin{bmatrix} \boldsymbol{M} & \mathbf{1} & \boldsymbol{C} \\ \mathbf{1}^T & \mathbf{0} & \mathbf{0} \\ \boldsymbol{C}^T & \mathbf{0} & \mathbf{0} \end{bmatrix}}_{\boldsymbol{H}} \begin{bmatrix} W_{1,j} \\ \vdots \\ W_{N_C,j} \\ b_j \\ A_{j,1} \\ A_{j,2} \\ A_{j,3} \end{bmatrix} = \begin{bmatrix} D_{1,j} \\ \vdots \\ D_{N_C,j} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad j = 1, 2, 3$$

where $\boldsymbol{M} \in \mathbb{R}^{N_C \times N_C} : M_{i,j} = \phi\left(\|\boldsymbol{c}_i - \boldsymbol{c}_j\|_2\right)$ and $\mathbf{1} \in \mathbb{R}^{N_C \times 1}$. The positions of the deformed vertices $P_{i,j}^{(RBF)}(V_{i,j}, \boldsymbol{D})$ is then found via Equation 4.1. It is not necessary to solve the system for every new deformation $\boldsymbol{D}$, since the matrix $\boldsymbol{H}^{-1}$ can be precomputed and used to find the new weights in linear time. In Figure 4.2 we show the reference geometry (blue) and a random configuration where 50 control points $\boldsymbol{C}$ (red) are randomly deformed with $|D_{i,j}| \leq 0.02$.



(a) Front view.

(b) From this view point it is possible to see the projection done by the parametrization $P^{(proj)}$ on the fuselage box.

Figure 4.2: Reference geometry (blue) and a random configuration. Control points $\boldsymbol{C}$ (red) and deformed control points $\boldsymbol{C} + \boldsymbol{D}$ (light blue).

4.2.2 Projection

This parametrization is designed to ensure that the geometric constraints are satisfied at every iteration of the optimization process. The requirements impose to have a fuselage which is a rectangular cuboid of dimensions $\boldsymbol{l} = \left[L_x, L_y, L_Z\right] = \left[220, 75, 30\right]$ $[mm]$, see Figure 4.1(b). When a vertex enters the fuselage volume it needs to be projected back on its surface. The projection of the deformed vertices on the rectangular cuboid, which without loss of generality we will assume centred in $(0, 0, 0)$, is done trough a linear map. Defining as $\boldsymbol{y}$ the deformed position of a vertex and as $\boldsymbol{x}$ its original position the following must holds:

$$y_i + (x_i - y_i)t_i = \text{sign}(x_i)l_i \quad i = 1, 2, 3$$

Among the three candidates the one satisfying:

$$-\boldsymbol{l} \leq \boldsymbol{y} + (\boldsymbol{x} - \boldsymbol{y})t_i \leq \boldsymbol{l}$$

is selected. We will denote this projection operation as $P^{(proj)}(\boldsymbol{V})$. An example is shown in Figure 4.2(b).

4.2.3 Rotation

Given an axis s, specified by two points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, and a rotation angle θ, the rotation of $\boldsymbol{x}_0$ around s is defined as:

$$\boldsymbol{x}_1 = \boldsymbol{R}(\boldsymbol{x}_0 - \boldsymbol{p}_1) + \boldsymbol{p}_1$$

where:

$$\boldsymbol{R} = \begin{bmatrix} \cos\theta + u_x^2\,(1-\cos\theta) & u_x u_y\,(1-\cos\theta) - u_z \sin\theta & u_x u_z\,(1-\cos\theta) + u_y \sin\theta \\ u_y u_x\,(1-\cos\theta) + u_z \sin\theta & \cos\theta + u_y^2\,(1-\cos\theta) & u_y u_z\,(1-\cos\theta) - u_x \sin\theta \\ u_z u_x\,(1-\cos\theta) - u_y \sin\theta & u_z u_y\,(1-\cos\theta) + u_x \sin\theta & \cos\theta + u_z^2\,(1-\cos\theta) \end{bmatrix}$$

$$\boldsymbol{u} = \frac{\boldsymbol{p}_2 - \boldsymbol{p}_1}{\|\boldsymbol{p}_2 - \boldsymbol{p}_1\|_2}$$

This parametrization, from now on called $P^{(\boldsymbol{R})}(\boldsymbol{V}, \theta)$, is used to vary the angle of attack θ of the drone (a rotation around the y axis); however it could also be used to rotate some specific parts of the geometry, such as the elevators.

4.2.4 Parametrization gradient

Our aim is to optimize an objective function $F(\boldsymbol{V})$. By using a composition of the described parametrizations we can write the set of vertices $\boldsymbol{V}$ as $P(\boldsymbol{V}^{(0)}, \boldsymbol{z})\colon \mathbb{R}^{3\times N} \times \mathbb{R}^{Nz} \to \mathbb{R}^{3\times N}$. Hence the gradient of the objective function $F(\boldsymbol{V})$ with respect to the design variable $\boldsymbol{z}$ can be computed as:

$$\frac{\partial F(\boldsymbol{V})}{\partial z_k} = \sum_{i=1}^{N}\sum_{j=1}^{3} \frac{\partial F(\boldsymbol{V})}{\partial V_{ij}} \frac{\partial V_{ij}}{\partial z_k}$$

In our case $\boldsymbol{z} = [\theta, \boldsymbol{D}]$ and $\boldsymbol{V} = \boldsymbol{P}(\boldsymbol{V}_0) = \boldsymbol{P}^{(\boldsymbol{R})}(\boldsymbol{P}^{(proj)}(\boldsymbol{P}^{(RBF)}(\boldsymbol{V}_0, \boldsymbol{D})), \theta)$ thus:

$$\frac{\partial V_{ij}}{\partial \theta} = \frac{\partial P^R_{i,j}}{\partial \theta}$$

$$\frac{\partial V_{ij}}{\partial D_{k,l}} = \sum_{m=1}^{N}\sum_{n=1}^{3} \frac{\partial P^R_{i,j}}{\partial P^{proj}_{m,n}} \frac{\partial P^{proj}_{m,n}}{\partial D_{k,l}} \qquad \frac{\partial P^{proj}_{ij}}{\partial D_{k,l}} = \sum_{i=1}^{N}\sum_{j=1}^{3} \frac{\partial P^{proj}_{m,n}}{\partial P^{RBF}_{i,j}} \frac{\partial P^{RBF}_{i,j}}{\partial D_{k,l}}$$

We remind that we compute the gradient with respect to the deformation of the control points $\boldsymbol{D}$ and not their initial position $\boldsymbol{C}$.

4.3 Optimization

Our aim is to find a solution to Problem [ref], by relying on a Geodesic Convolutional Neural Network that we use as a surrogate model of a computational fluid dynamics solver. We recall that the network (Chapter 2) returns a prediction of the lift $L^{\boldsymbol{\omega}}$, drag $D^{\boldsymbol{\omega}}$ and pitching moment $M_y^{\boldsymbol{\omega}}$. With the parametrization introduce in the previous section, the optimization problem can be reformulated as:

$$\underset{\boldsymbol{z}=[\theta, \boldsymbol{D}\in\mathbb{R}^{N_C\times 3}]}{minimize} \quad -L^{\boldsymbol{\omega}}(P(\boldsymbol{V}^{(0)}, \boldsymbol{z}))/D^{\boldsymbol{\omega}}(P(\boldsymbol{V}^{(0)}, \boldsymbol{z}))$$

$$subject\ to \quad \begin{cases} L_{obj} - L^{\boldsymbol{\omega}}(P(\boldsymbol{V}^{(0)}, \boldsymbol{z})) \leq 0 \\ M_y^{\boldsymbol{\omega}}(P(\boldsymbol{V}^{(0)}, \boldsymbol{z})) = 0 \end{cases}$$

where in our case $N_C = 50$, so that there are in total 151 optimization variables. It is important to remark that the parametrization can be modified (e.g. by changing the control points of the RBF parametrization) at any time during the optimization process, since the network takes as input a triangulated surface $\{\boldsymbol{V}, \boldsymbol{A}\}$ and not the design variables.

The gradient of the objective function with respect to the design variables $\boldsymbol{z}$ is obtained as described in Section 4.2.4, using automatic differentiation. The objective function can then be optimized using either a gradient based method or an evolutionary algorithm. We have decided to ignore the constraint on the pitching moment, being aware that the found design may be statically unstable, nonetheless in order to find a design for which $M_y = 0$ the parametrization should have included the deflection of the elevators; we reserve for the future this additional requirement. However the problem still present the inequality constraint for the lift value.

We have decided to solve the optimization via the SciPy implementation of [14] which was designed for large scale optimization problems and also allows equality constraints.

4.3.1 Optimization process

The introduced parametrizations allows to generate an initial data set Ω of shapes simply by a random sampling of the design variables $\boldsymbol{z}$ with $|D_{i,j}| \leq 0.02$ $[m]$ and $-20° \leq \theta \leq 20°$, for which the aerodynamic quantities are computed as described in Chapter 3. The network $F^{\boldsymbol{\omega}}$ trained on Ω is then used to compute a surrogate objective function. Since the initial dataset is not representative of the whole shape space ($|\Omega| = 184$), in order to increase the performances of the network we alternate the optimization phase to a retraining phase, where we train the network for $E = 10$ epochs. At every iteration i we run the optimization algorithm K times for N steps, starting from a different geometry (randomly sampled with $3° \leq \theta \leq 6°$ and $|D_{i,j}| \leq 0.02$ $[m]$) in order to have a better exploration. This part could be made more efficient by using an evolutionary algorithm. The best of the K geometries is sent to the numerical simulator and is used to augment the dataset Ω. We summarize the procedure in Algorithm 2.

```
Parameter: ω
Data      : An initial data set of simulated geometries Ω
Result    : A set of optimal shapes B

for i = 1, ..., 285 do
    Augment Ω with simulated geometries
    for e = 1, ..., E = 10 do
        Train the network F^Ω
    end
    for k = 1, ..., K = 5 do
        Sample a new random shape V^(k), 3° ≤ θ ≤ 6°, |D_{i,j}| ≤ 0.02 [m]
        for j = 1, ..., N = 50 do
            Perform one optimization step.
        end
    end
    Send the best geometry V^(k*) to simulator.
end
```

Algorithm 2: Optimization routine

Chapter 5

Results

The aim of the project was to optimize the aerodynamic performances of the *eBee Classic*, a fixed wing drone from senseFly, taking into consideration geometric and aerodynamic constraints. To achieve the result we have trained a Geodesic Convolutional Neural Network to predict global aerodynamic quantities (Chapter 2) on a data set of random shapes (Chapter 3). The network was then used as a surrogate model and optimized with respect to the design variables that parametrize the initial geometry: the control points deformation $\boldsymbol{D}$ and the angle of attack θ. The optimized shape are sent to the CFD simulator and added to training set (Chapter 4). In Figure 5.1 the outline of the whole optimization process is shown.

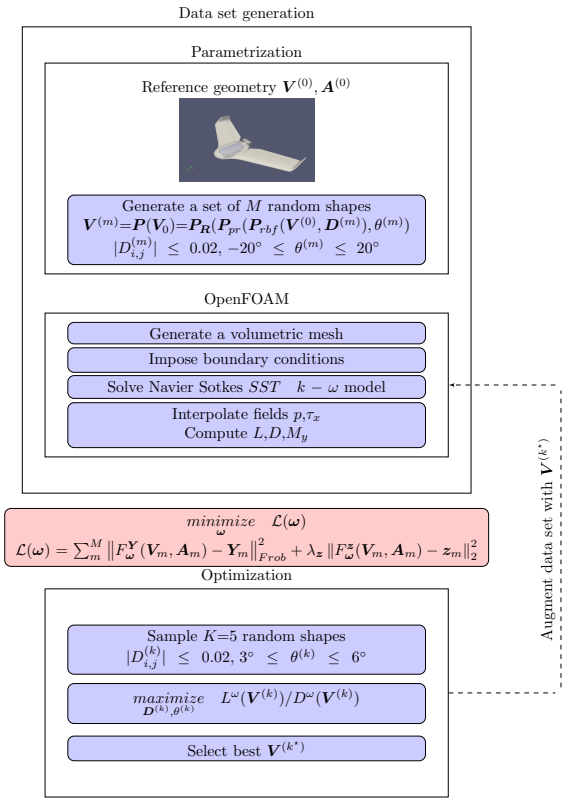


Figure 5.1: Schema of the whole work.

5.1 Optimization process

In Figure 5.2 we show how the lift-to-drag ratio L/D (that we will call the objective) evolves over the iterations of the optimization process. Both the ground truth values and the predictions of the network are shown with their centred moving average (with a window size $n = 20$). Moreover the relative error (and its moving average) is also shown.

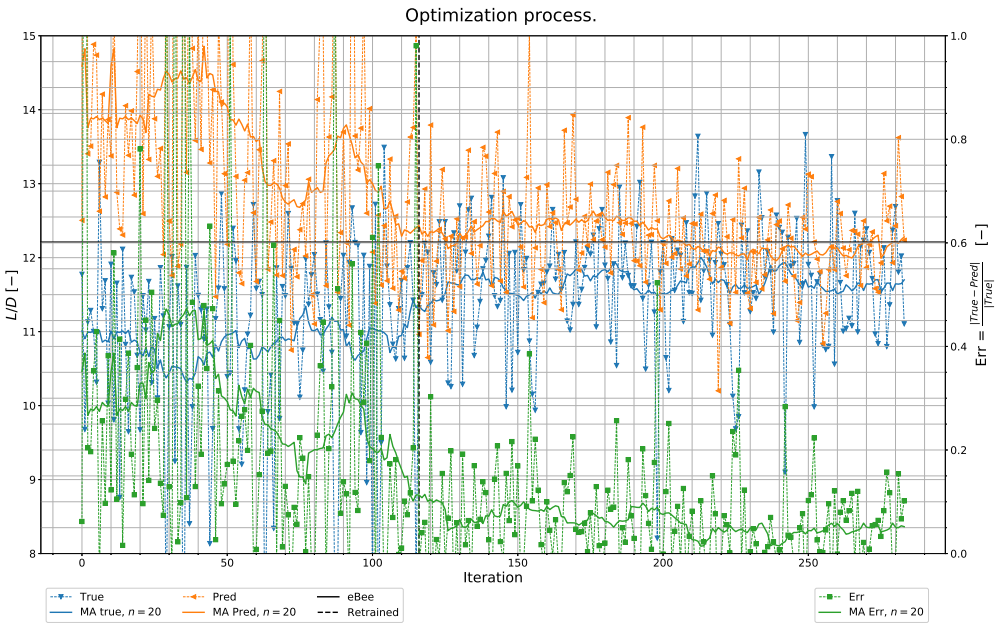


Figure 5.2: Evolution of the objective over the optimization process.

5.1.1 Objective evolution

Figure 5.2 clearly shows that the mean objective value increases over time. The reference objective represents the maximum value of lift-to-drag ratio obtained by simulating the reference geometry for $N_\theta = 20$ equispaced angles of attack $-2° \leq \theta \leq 7°$. Even though the moving average is smaller than the reference objective, our interest is to find an increasing number of geometries better than the *eBee*. In fact a better result in average could have been obtained by exploiting a neighbourhood of the reference geometry, just by applying very small deformations $\boldsymbol{D}$ on the reference shape. In Figure 5.3 we show the cumulative number of geometries, for which the objective is better than a threshold value, over the optimization process. At the end of the 284 iterations we have obtained 49 shapes that are better than the original geometry and 5 for which the improvement is better than 8%. We have decided to increase the exploration of the shape space by applying a random deformation $D_{i,j}$ for each 50 control points before performing the optimization steps as described in Section 4.3.1. Nonetheless the model could be prone to optimize the initial geometry by always converging to a previously visited optimal shape. This does not happen, from a visual inspection of the found geometries, see Appendix. In order to have a confirmation and to quantify the differences in between the optimal geometries $\mathcal{B}$, we have computed the following distance indicator $d_{i,j}$ between two shapes:

$$d_{i,j} = \frac{\|\boldsymbol{D}_i - \boldsymbol{D}_j\|_{Frob}}{\max \|\boldsymbol{D}_k - \boldsymbol{D}_l\|_{Frob}} \quad i = 1, \dots N \quad j = 1, \dots i \tag{5.1}$$

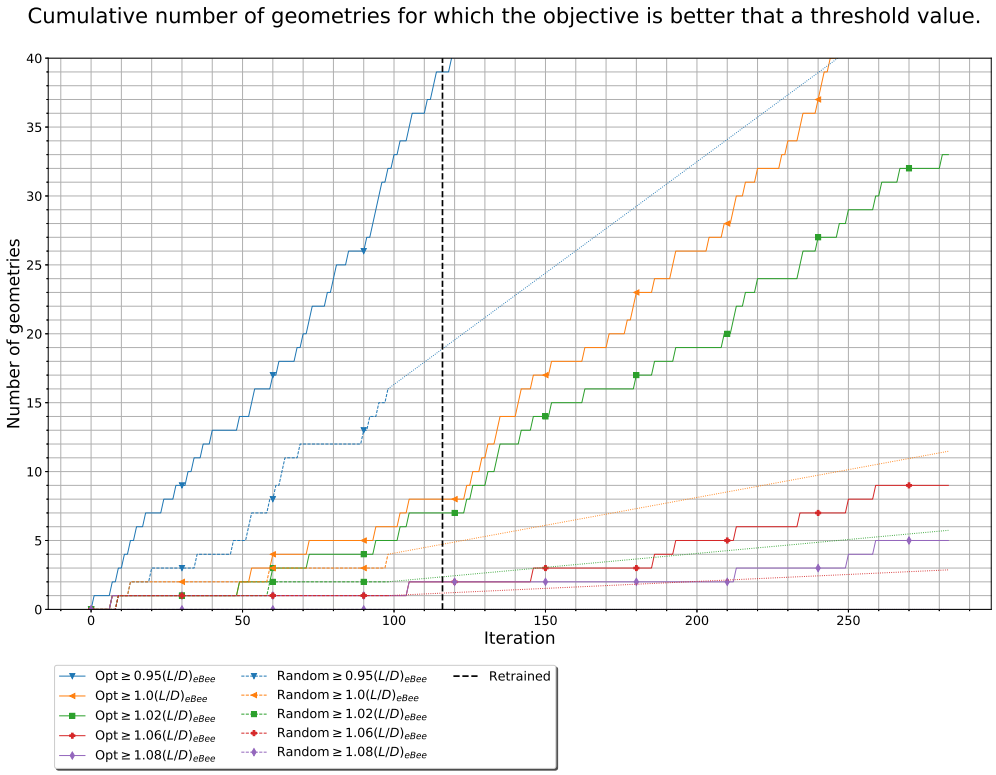


Figure 5.3: Cumulative number of geometries for which the objective is better than a threshold value, specified with respect to the best lift-to-drag ratio of the *eBee Classic*

where $\{V^i, V^j\} \in \mathcal{B}$ and $V^l, V^k \in \Omega$. The distance indicator is shown in Figure 5.4 where the shapes are ordered according to their objective values. Interestingly there are no evident patterns, suggesting that the optimization process is not stuck in a local minimum.

5.1.2 Comparison with random sampling

The test dataset has been designed so that it can also be used to make a fair comparison between our optimization process and a smart random sampling. 99 test geometries are generated by sampling uniformly the angle of attack within the range $3 \leq \alpha \leq 6$ and $|D_{i,j}| \leq 0.02$ $[m]$. In Figure 5.3 we compare the efficiency of the random sampling with the optimization process. For the random sampling, since only 99 geometries are available for $Iter > 99$, we show a linear extrapolation. Surprisingly the random sampling is capable of obtaining 4 shapes better than the reference. However it is clearly outperformed by the optimization. For the optimization the slope of the cumulative curve increases over the iterations, especially after iteration 116 where the network was retrained up to convergence on the test data set. In Figure 5.5 we show the distribution of the objective for the random sampling and for the optimization process over different iterations ranges.

5.1.3 Analysis of the network

From Figure 5.2 we see that the relative error of the prediction and the ground truth values decrease over time. However, since the averages do not coincide, the R^2 score is negative. In order to understand better the behaviour of the network we have analysed the following models:

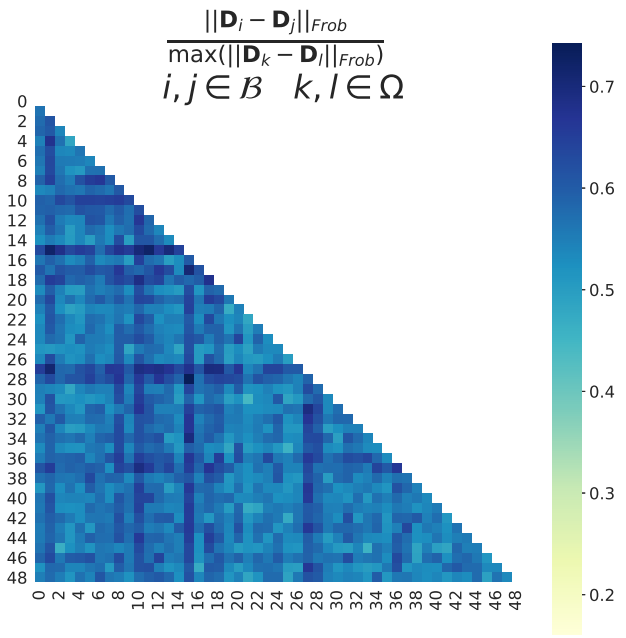


Figure 5.4: Distance indicator $d_{i,j}$.

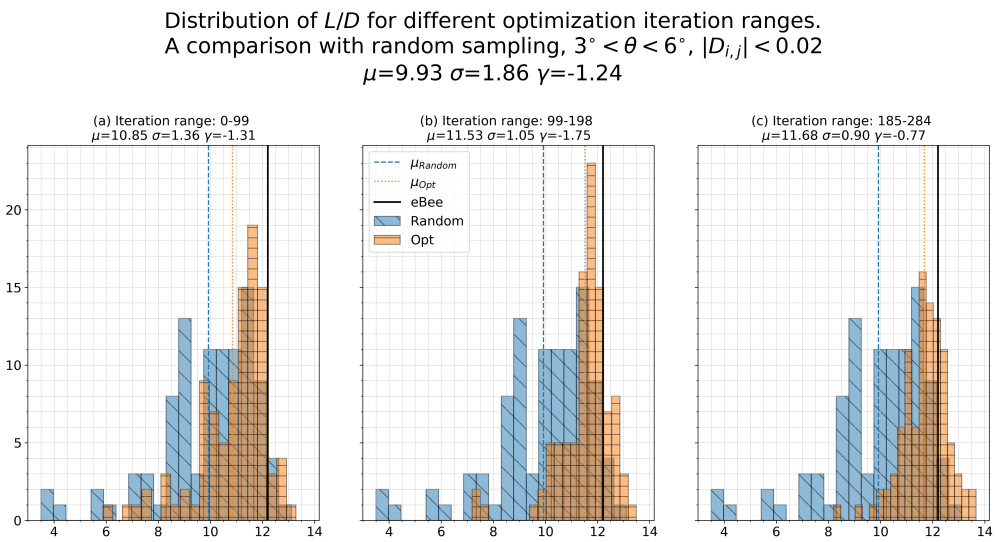


Figure 5.5: Distribution of the objective values.

- We denote by F the architecture described in Section 2.3.1, using $K = 7$ kernels for the geodesic convolutions:

  $F^{(0)}$: a model trained up to convergence on the test data set with Ω as a training set.

  $F^{(1)}$: the model $F^{(0)}$ retrained over the optimization process up to iteration 116.

  $F^{(2)}$: the model $F^{(1)}$ retrained on the augmented dataset up to convergence on the test data set.

  $F^{(3)}$: the model $F^{(2)}$ retrained over the optimization process from 116 to last iteration.

- F_D a model with the same architecture as F with an additional layer that downsample the geometry before the convolutional layers (with a downsampling factor of 3), with $K = 32$ and an additional upsampling layer for the fields stream. The model was trained up to convergence on the whole augmented dataset.

For each of them we have computed (for all the predicted global quantities) the mean error $\boldsymbol{e}$, its standard deviation and the Explained Variance EV (Table 5.1) :

$$e_i = y_i - f_i \quad \bar{\boldsymbol{e}} = \frac{1}{N}\sum_i^N e_i \quad \bar{\boldsymbol{y}} = \frac{1}{N}\sum_i^N y_i$$
$$EV = 1 - \frac{\sum_i^N (e_i - \bar{\boldsymbol{e}})^2}{\sum_i^N (y_i - \bar{\boldsymbol{y}})^2}$$

where $\boldsymbol{y}$ and $\boldsymbol{f}$ are the true values, and the predictions respectively. From the shown metrics it is clear that the network performance improves over the optimization process. The online learning process resembles an adversarial training; at each iteration we seek $\boldsymbol{V}$ that maximizes $L^{\boldsymbol{\omega}}(\boldsymbol{V})/D^{\boldsymbol{\omega}}(\boldsymbol{V})$. Nonetheless, during the first iterations of the process there are shapes for which the relative error is very high. These geometries significantly affects the retraining phase, since gradient of the training loss $\mathcal{L}_{\boldsymbol{z}}$ depends on the errors of the lift and drag:

$$\begin{aligned}\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\boldsymbol{z}}(\boldsymbol{\omega}, \boldsymbol{V}) = {} & 2\nabla_{\boldsymbol{\omega}} L^{\boldsymbol{\omega}}(\boldsymbol{V})(L^{\boldsymbol{\omega}}(\boldsymbol{V}) - L) + 2\nabla_{\boldsymbol{\omega}} D^{\boldsymbol{\omega}}(\boldsymbol{V})(D^{\boldsymbol{\omega}}(\boldsymbol{V}) - D) \\ & + 2\nabla_{\boldsymbol{\omega}} M_y^{\boldsymbol{\omega}}(\boldsymbol{V})(M_y^{\boldsymbol{\omega}}(\boldsymbol{V}) - M_y)\end{aligned}$$

The optimization process hence allows to explore regions of the shape space where the model poorly performs, increasing its accuracy. However since the error is due to an overprediction of L/D, the high gradients of the adversarial geometries drives the network to underestimate L/D on the test set. Moreover, we see that this problem is not present for model DF which is trained on the whole data set. In order to mitigate this tendency it may be useful, during the retraining phase, to clip the gradient by its global norm $\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\boldsymbol{z}}(\boldsymbol{\omega}, \boldsymbol{V})$ with a clipping value c estimated during the initial training:

$$\omega_i = \omega_i \frac{c}{\max\{\|\nabla_{\boldsymbol{\omega}}\mathcal{L}_{\boldsymbol{z}}(\boldsymbol{\omega}, \boldsymbol{V})\|_2, c\}} \tag{5.2}$$

Finding the most effective strategy for the retraining phase is of fundamental importance and deserves to be investigated in the future.

5.2 Optimal shape

We have found 49 shapes with a better lift-to-drag ratio than the reference geometry. However, for a proper comparison we need to check the aerodynamic performances for different angles of attack. We have done this analysis only for one shape. In the future all the best geometries should be checked. This analysis can indeed be done relying on already the built models. In Figure 5.6 we compare the two shapes for $N_\theta = 20$ equispaced angles of attack $-2° \leq \theta \leq 7°$. The optimal shape presents a better lift-to-drag ratio for all θ due to a significant increase in the lift. On the other side the pitching moment M_y, calculated with respect to the center of the fuselage box, is also significantly smaller. This last consideration shows the need to include the deflection of the elevators in the parametrization and to add the equality constraint for M_y in the optimization

		$F^{(0)}$	$F^{(1)}$	$F^{(2)}$	$F^{(3)}$	DF
D 1.04 (0.30)	$\bar{\boldsymbol{e}}$	0.03 (0.29)	-0.14 (0.23)	-0.09 (0.20)	-0.08 (0.19)	-0.04 (0.18)
	EV	0.06	0.43	0.55	0.61	0.64
L 10.39 (3.45)	$\bar{\boldsymbol{e}}$	0.24 (2.16)	0.17 (1.44)	0.38 (1.32)	0.36 (1.28)	0.09 (0.75)
	EV	0.61	0.83	0.85	0.86	0.95
M_y -1.11 (0.66)	$\bar{\boldsymbol{e}}$	0.00 (0.39)	-0.02 (0.30)	-0.07 (0.26)	-0.07 (0.25)	0.04 (0.29)
	EV	0.66	0.80	0.84	0.86	0.80
L/D 9.93 (1.85)	$\bar{\boldsymbol{e}}$	-0.07 (1.85)	1.50 (1.29)	1.28 (1.23)	1.24 (1.19)	0.50 (1.33)
	EV	0.01	0.52	0.56	0.59	0.49

Table 5.1: Metrics for different models, for all the global quantities. The mean error $\bar{\boldsymbol{e}}$ is shown with its standard deviation.

problem. Finally in Figure 5.7 we show the surface pressure field for both the geometries. The optimal shape presents a smaller pressure in the proximity of the leading edge, which is the reason for the increase in the lift. We attribute this to the fact that this geometry has a more cambered shape. Moreover, the pressure is smaller also in the region close to the the winglets, which may indicate a smaller downwash force.

Comparison of the eBee Classic with one optimal shape with 8% increase in the lift-to-drag-ratio.

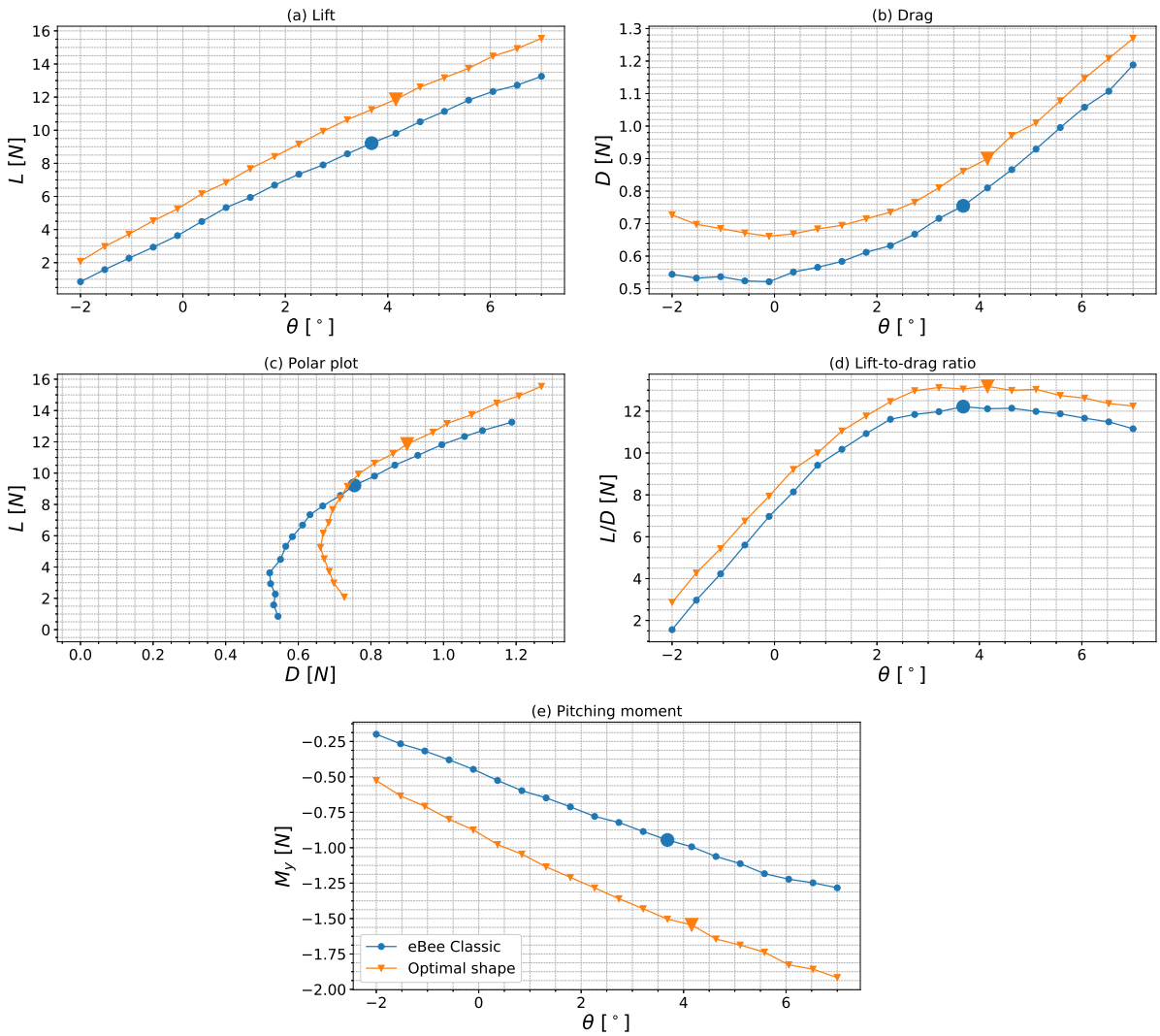


Figure 5.6: Comparison of the aerodynamic performance of the *eBee Classic* with one optimal shape with an increase in lift-to-drag ratio of 8%. The aerodynamic quantities are shown for an angle of attack $2° \leq \theta \leq 7°$

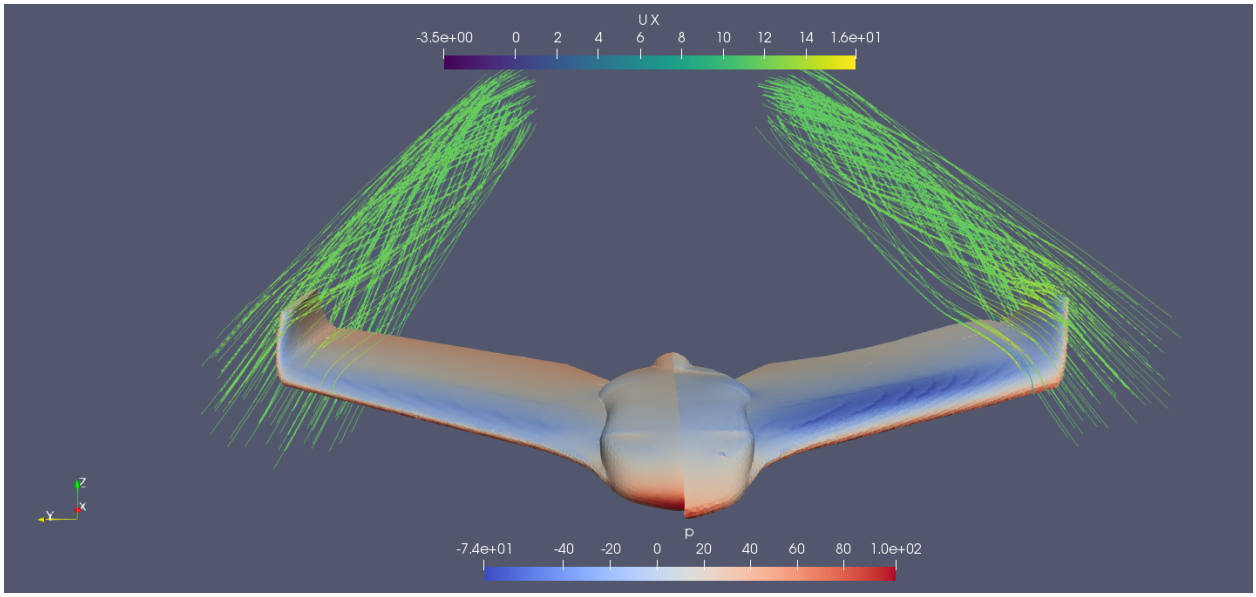


Figure 5.7: Pressure field distribution for the *eBee Classic* (left) and for the optimal shape (right). Streamlines visualization, obtained via integration from a sphere source point positioned in front of the winglets.

Chapter 6

Conclusions and future work

We have used a Geodesic Convolutional Neural Network that has been used as surrogate model to predict the aerodynamic performances of fixed wing drone. Via an optimization process we have obtained 49 geometries with a lift-to-drag ratio better than the reference. We have presented an analysis of the aerodynamic performances for one of the optimal shape. It was shown how the model evolves over the optimization process, demonstrating that it increases its accuracy and that is capable of exploring a high dimensional space, clearly outperforming a random search strategy. The future work should include the following point:

- Compare the optimization process with the Kriging method.
- Analyse all the best found geometries.
- Add the deflection of the elevators to the parametrization of the geometry and include the constraint on the pitching moment in the optimization problem.
- Research different network architectures.
- Further investigate how the retraining strategy affects the optimization process.

Bibliography

[1] P. Baque, E. Remelli, and P. Fua, "Geodesic Convolutional Shape Optimization," 2018.

[2] M. M. Bronstein, J. Bruna, Y. Lecun, A. Szlam, P. Vandergheynst, and C. V. May, "Geometric deep learning : going beyond Euclidean data," pp. 1–22.

[3] S. N. Skinner and H. Zare-Behtash, "State-of-the-art in aerodynamic shape optimisation methods," *Applied Soft Computing*, vol. 62, pp. 933–962, 2018.

[4] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, 1989.

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

[6] Yoshua Bengio, "Deep Learning Book," *MIT Press*, 2015.

[7] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 5425–5434, 2017.

[8] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on Riemannian manifolds,"

[9] H. Schlichting, *Boundary- Layer Theory.*

[10] H. K. Versteeg and W. Malaskekera, *An Introduction to Computational Fluid Dynamics: The Finite Volume Method.* 1995.

[11] F. R. Menter, M. Kuntz, and R. Langtry, "Ten Years of Industrial Experience with the SST Turbulence Model," 2003.

[12] L. S. Caretto, A. D. Gosman, S. V. Patankar, and D. B. Spalding, "Two calculation procedures for steady, three-dimensional flows with recirculation," in *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics* (H. Cabannes and R. Temam, eds.), (Berlin, Heidelberg), pp. 60–68, Springer Berlin Heidelberg, 1973.

[13] M. D. Buhmann, *Radial Basis Functions: Theory and Implementations.* Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 2003.

[14] R. H. Byrd, M. E. Hribar, and J. Nocedal, "An Interior Point Algorithm for Large-Scale Nonlinear Programming," *SIAM Journal on Optimization*, vol. 9, no. 4, pp. 877–900, 2003.

Appendix