Semester Project

Laboratory of Intelligent Systems EPFL-LIS

Automated Propeller Geometry Measurement from 3D Scanning Data

January 10, 2019

Author:
Pauline Maury Laribière

Supervisors:
Prof. Dario Floreano
Sebastian John Steffen
Anand Bhaskaran

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Summary:

In this project, an algorithm to enable drone performance analysis based on 3D scanning data has been made.

By taking as input an STL file and positions provided by a user (in terms of percentage along the blade), the algorithm computes the following parameters: the shape of the aerofoil, blade twist and chord length at each given position as well as the tip radius and hub radius of the propeller. It then computes and outputs the aforementioned parameters, points along the contour of each aerofoil and the associated aerodynamic parameters. For instance, figure 1 illustrates an aerofoil at the middle of the blade with its associated chord length and blade twist.

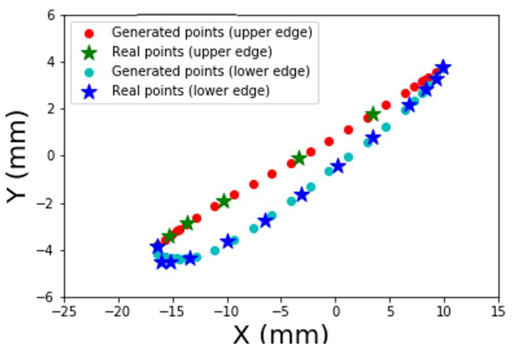


FIGURE 1: Aerofoil shape

Satisfying results were obtained regarding the aerofoil shapes and parameters, proving the possibility of a reverse engineering approach based on 3D scanning. Those can be output to be used in a software for aerodynamic performance analysis.

Contents

1 Introduction

In recent years, drones have become increasingly popular for both personal and professional applications. Due to their worldwide adoption, their quality as well as reliability and performance are continuously being improved resulting in the rise of new models and designs. The analysis of the performance of drones is thus a highly active field of research, especially when it comes to the design of the propellers used for the actuation of these drones.

However, there are currently no automated methods for the estimation of the aerodynamic properties of the propellers based solely upon their geometry. Indeed, the most common means by which their performance is evaluated is through wind tunnel tests which require a lot of infrastructure and are impractical, especially when it comes to large propellers [1]. The Blade Element Momentum Theory [2] is therefore often used to model their aerodynamic properties but this requires information about the geometry of the propellers which are not typically provided by the manufacturers. Moreover, there exist no common database for professional to a analyze a wide range of aerofoil.

A more simple and practical alternative to determine the geometry of the propellers would be to scan them and reverse engineer their 3-dimensional (3D) shape. 3D scanning technologies have been greatly developed in the past few years and processing techniques are under constant improvement, thus enabling the extrapolation of the shape of a 3D object based on the geometric samples of its surface. To do this, the scanner is first used to compute the distance of each point of the surface and to take multiple measurements at different positions. The multiple scans are then put on a reference system through process of alignment and are eventually merged to create the final model. The result is therefore a 3D model that has been reverse-engineered from a 3d scanning of a given object.

Multiple software applications exist for such a 3D modelling task, however the choice of method relies heavily on the specifications of the project such as the size of the object, the accuracy required, the budget, the amount of time available, etc. An optimal approach would be to combine multiple technologies by taking each of their benefits.

The parameters of the aerofoil could then be found through analytic computations which would enable the computation of the lift and drag of the propellers which would provide information about the performance of the drones to which they are attached.

1.1 Goal of the project

The goal of this project is to hence to create a software tool that would enable an accurate estimation of the thrust and power consumption of propellers based on 3D coordinate points obtained through 3D scanning.

The inputs consist of STL files and the output is the shape of the aerofoil at different locations along the radius of the blade as well as the the hub radius, tip radius, blade twist and chord length. Those parameters can then be used as input in the Blade Element Momentum Theory to determine the lift and drag characteristics at each radial position of the blade. To apply this theory and find the lift and drag coefficients, the aerodynamic softwares can be used.

Additionally, further research on potential processes to further improve the automation of the 3D scanning of propellers that could facilitate the study of aerodynamics is done.

1.2 Initial concepts

An aerofoil is a cross section of a blade. According to the Oxford Dictionnary, it is 'a structure with curved surfaces designed to give the most favourable ratio of lift to drag in flight, used as the basic form of the wings, fins, and tailplanes of most aircraft'. Hence the shape of aerofoil are studied to create propellers with optimal aerodynamics properties[3].

The Blade Element Momentum Theory takes as input geometric parameters: the aerofoil shape with its chord length and twist angle and aerodynamic parameters: Reynolds and Mach numbers. It first cuts the propeller down into discrete segments along the radius. And then uses the law of conservation of momentum to estimate the relative flow angleΦ between the air velocityα and the blade segment β. The lift C_L are drag C_D coefficient are determined as a function of the Reynolds number Re and this difference of angle α. Those function f_1 and f_2 depends on the aerofoil shape. Figure 1.1 illustrates this explanation.

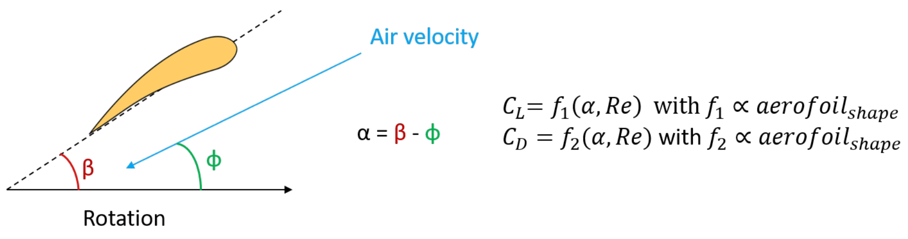

Air velocity
β
ϕ
Rotation
α = β - ϕ
$C_L = f_1(\alpha, Re)$ with $f_1 \propto aerofoil_{shape}$
$C_D = f_2(\alpha, Re)$ with $f_2 \propto aerofoil_{shape}$

FIGURE 1.1: Blade Element Momentum Theory

The thrust and torque generated by the propeller can then be predicted with the lift and drag coefficients.

1.3 State of the art

1.3.1 3D scanning techniques

The main 3D scanning techniques are classified into three groups: contact, non-contact active and non contact-passive. They are described in the following paragraphs.

Contact 3D scanning

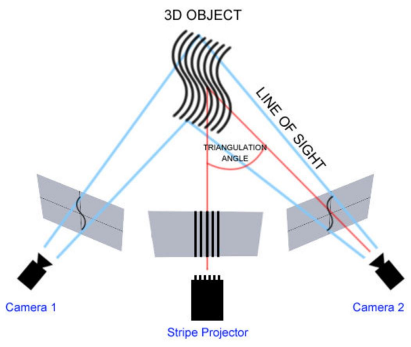
In contact 3D scanning, a probe physically touches a firmly fixed object at several points and samples the different positions of the surface. The most commonly used example of this technology is Coordinate Measuring Machine [4] (CMM) where point clouds are generated by a regression algorithm.

Non-contact active 3D scanning

Non contact active 3D scanning technologies [5] and [6] rely on the emission of radiation or light. These methods are currently the most commonly used techniques for 3D modelling as they directly give the point clouds and colour information of the 3D data. They are easy to use and have the ability to capture details until 10μmeter of accuracy.

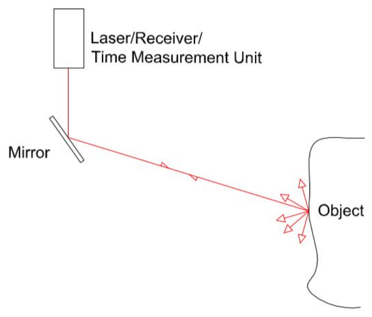
There are three main types of methodologies:

- Structured light: the structured light method consists of projecting a pattern of light upon the given object. By measuring the deformation of this pattern on the object, its shape can then be deduced as illustrated in figure 1.2a.
- Time of flight (laser pulse): in the case of time of flight 3D scanning technology, more commonly known as Lidar, a pulsed laser beam is projected onto a surface and its reflection is then collected by a sensor,as illustrated in figure 1.2b. The measurement of the time of flight of the laser beam between its emission and reception then provides geometrical information of the object's surface.
- Laser triangulation: in laser triangulation, as illustrated in figure 1.2c, a sensor projects a laser spot onto the object. Depending on the distance of the object, the reflected light falls at a certain angle on a receiving element. By determining the position of the spot on the receiver, the distance from the object to the receiver is computed.



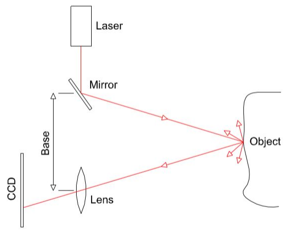
(A) Structured Light method

Source[1]



(B) Time of Flight Method

Source [6]



(C) Laser Triangulation method
Source [6]

FIGURE 1.2: Non contact active

Non-contact passive 3D scanning

Non-contact passive 3D scanning techniques enable the modelling of a 3D object through the detection of ambient radiation, i.e. visible or infrared light. In this method, a user first identifies features or points through a software application which are then measured by the sensor.

An example of this process is photogrammetry [7] which reconstructs a 3D model of an object from 2D digital pictures taken at different angles as in figure 1.3 with the use of computer vision and computational geometry algorithms. Objects of multiple sizes can the be scanned by simply changing the size of the lens of the sensor.

[1] $https://cdn.instructables.com/F8B/05AI/I7CCLCOX/F8B05AII7CCLCOX.MEDIUM.jpg$

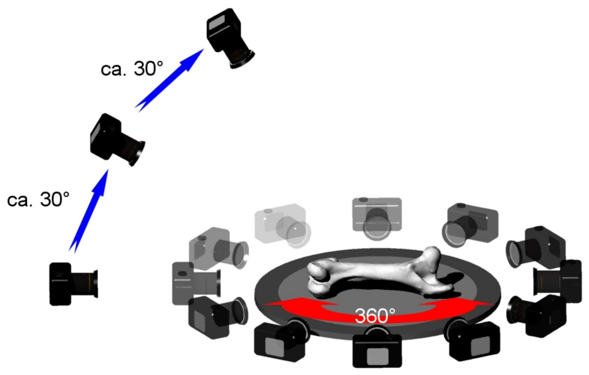


FIGURE 1.3: Photogrammetry

Source[2]

Different softwares are available for this process, such as:

- PhotoModeler[3] Scanner by EOS systems Inc which performs semi-automatic modelling. Its advantage is that all the point clouds are in a unique reference frame and alignment or similarity transformations are not required.
- 3DF Zephr[4] which performs an automatic and a very user-friendly interface.
- and many others.

Comparison of different methods

Table 1.1 on page 5 allows to compare the various methods presented above in terms of price, resolution, advantages, disadvantages, etc.

1.3.2 Parametrisation

Many parametrisation techniques have been developed for aerofoils as they are fundamental for aerodynamic optimization. In order to choose the most suitable technique for this project, the following objectives are assessed:

- The number of degrees of freedom and hence the number of parameters should be minimized to ensure computation efficiency.
- The parameters should be simple to formulate.
- The technique should be able to represent a wide range of existing airfoils
- The continuity of the design space should be ensured. Indeed, no degenerate state of a polynomial curve should appear at any parameter combination.

The various parametrisation techniques, including their benefits and defaults are described below.

[2] $https://dinosaurpalaeo.files.wordpress.com/2013/12/photogr\_tut\_01.jpg$
[3] $https://www.photomodeler.com/index.html$
[4] $https://www.3dflow.net/3df-zephyr-pro-3d-models-from-photos/$

	Contact [4]	**Non-contact active [6]**			**Non-contact passive [7]**
	CMM	**Structured Light**	**Time of flight**	**Laser triangulation**	**Photogrammetry**
Typical accuracy	1 μm	5-100 μm sub-μm with blue LED	-2mm @ 30m -4mm @ 50m -7mm @ 100m	40-100 μm	Depends on picture pixel/4
Price @50μm[a]	150'000CHF	15'000CHF	50 micro not possible	50'000CHF	500CHF (camera price)[b]
Typical range[c]	In contact with probe	2cm-150m	1m-km	10cm-km	10cm-km
Pros [5]	-No problem with reflective or transparent surface -Not sensitive to external light -Least prone to error	-Cheap -Fast (ms)	-Can capture textures and colors -Can scan translucent surface (phase comparison)	-Can scan moving scene -Fast (ms)	-Fast (no set-up, only take few photo) -Cheap -Very flexible (change lens to change size)
Cons [5]	-Very slow (hours) -No complex shape (no assembled parts) -Only stationary	-Not for reflective or transparent surface -Sensitive to external light -Only stationary	-Only stationary	-Not for reflective or transparent surface -Expensive	-Not for reflective or transparent surface -Need processing (software) -Only stationary
Domain[3]	Small, simple object with high quality in lab	Terrestrial	Terrestrial	-Mobile -Terrestrial -Airborne	-Close-range -Aerial (UAV) -Terrestrial
Application examples[5]	-High quality scanning -Quality checking -Reference Scanning	-Tool free calibration -Scanning of architecture elements	-3D shape scanning [138 du [3]] -Documentation of Historic Buildings -Kinect	-Extracting Road Information -Street object recognition -Urban Environment Modelling	-Topography of Natural Areas -Automatic Camera Calibration -Line detection and matching

TABLE 1.1: Comparative table of the 3D scanning methods

[a]The prices were determined from many examples on scanner selling webstites such as https://www.aniwaa.fr/comparatif/scanners-3d/ and https://www.3dnatives.com/3D-compare/scanner

[b]The price for the photogrammetry depends on the quality of the pictures and hence, on the camera. As $0.05mm$ is the needed resolution and the propeller size is in the order of $20x20cm$, then $Nb_{Pixels} = \frac{200}{0.05} * \frac{200}{0.05} = 16MPixels$ at least. Moreover, an effective sensor is required to have efficient focus. Cameras from 500CHF would be accurate enough for the task.

[c]Same as for the prices

Bezier Curves method

Aerofoils consist of two main parameters: the camber line and the thickness distribution. When added or removed from the camber line, the thickness distribution gives the coordinates of the upper and lower surfaces. Hence, to describe the profile of the aerofoil, the typical practice is to use a set of Bezier curves [8] which can describe n-degree curves through (n+1) control points.

PARSEC method

PARSEC is also a very common method to parametrise aerofoils [10]. This method uses eleven parameters to define the shape: the leading-edge radius r_{LE}, the upper X_{UP}, Z_{UP} and lower crest locations X_{LO}, Z_{LO}, the upper and lower curvature Z_{xxUP}, Z_{XXLO}, the trailing edge coordinate Z_{TE} and direction α_{TE}, the trailing edge wedge angle β_{TE} and the thickness ΔZ_{TE}. The following combination of shape function describes the aerofoil shape:

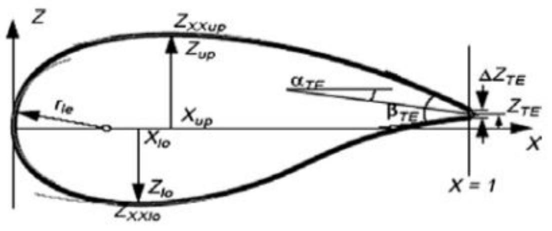


FIGURE 1.4: Parsec taken from [9]

$$Z_k = \sum_{n=1}^{6} a_{n,k} X_k^{\frac{n-1}{2}} \tag{1.1}$$

The main issue of PARSEC is that it does not provide sufficient control over the shape of the trailing edge.

Sobieczky, Modified Sobieczky and 'Navier Stokes' methods

To overcome the lack of control of the trailing edge, the PARSEC method is used in combination with the Sobieczky method as in [10], which give the trailing edge a concave shape.

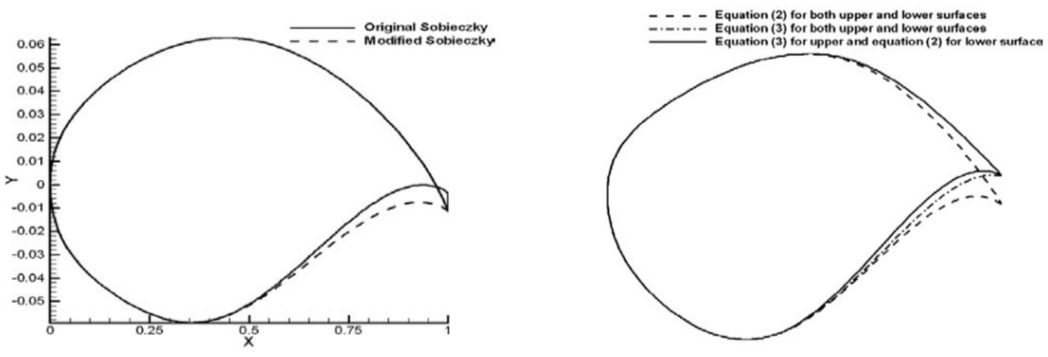
However, the feasibility of this new trailing edge is not ensured. Hence, the modified Sobieczky method [10] was created to overcome this issue. First, the upper surface is created. Then, a constraint is imposed on the lower surface to finish at the trailing edge of the upper surface.

However, this modified method does not entirely solve the issues with the trailing edge as it has a tendency to pull the trailing edge downwards, which can induce a pressure gradient on the upper surface. Hence, the 'Navier-Stokes' method from [10] overcomes this by flattening the upper surface of the aerofoil. This method is advantageous for the modelling of the upper surface but not of the lower surface, where it is less flexible.

A better solution is thus to use the modified Sobieczky method to model the lower surface and use the 'Navier-Stokes' method to model the upper surface.

BEZIER/PARSEC method

The Bezier-Parsec method [9] was developed to extend and improve the Bezier parametrization by taking advantage of both the Bezier and the Parsec methods. This method uses the PARSEC variables as parameters to define the four Bezier curves (leading edge and trailing edge of the camber line and thickness distribution). In addition to these parameters, there are two main parametrizations BP 3333 and BP3434, which have different polynomial curve



(A) Sobieczky modified method (B) 'Navier Stokes' method

FIGURE 1.5: Sobieczky and improved method picture from [9]

orders. For instance, BP3434 is especially used to control the trailing edge. This method has a lot of benefits and can cover a wide range of aerofoils.

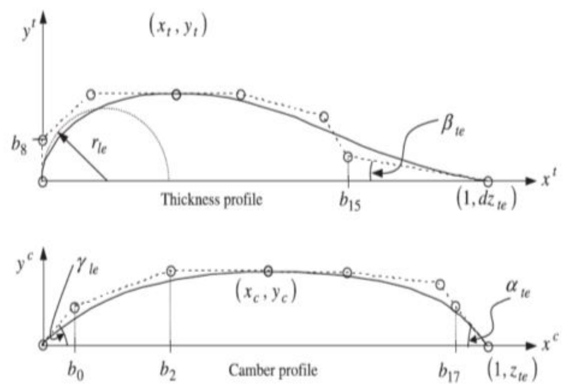


FIGURE 1.6: BP3434 from [9]

Improved Geometric Parameter method

The Improved Geometric Parameter (IGP) aerofoil parametrization method [11] is a recent method that decouples the camber and the thickness to construct them separately with fewer parameters and a clear physical representation (the same as PARSEC). The camber is expressed based on Bezier polynomials and the thickness is expressed by the polynomial basis function.

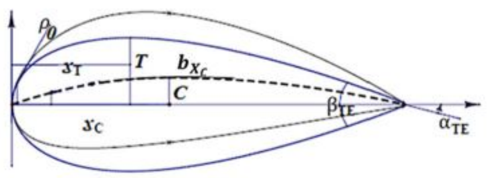


FIGURE 1.7: Improved Geometric Parameter from [9]

The main advantage of this method is that the control parameters can also be related to the aerofoil shape parameters making an easy link with the general aerodynamic theory. The 8 parameters are the leading edge radius ρ_0. the camber line abscissa x_C, the lower surface abscissa x_l, the maximum thickness and camber T and C, the trailing edge boat-tail angle β_{TE} and the angle between camber line and chord line on trailing edge α_{TE}.

2 Methodology

The input of this project is an STL file, a format native to the stereolithography CAD software created by 3D Systems. It describes the geometry of a three-dimensional object surface with triangles giving the unit normal and vertices ordered by the right hand rule using a three-dimensional Cartesian coordinate system. The file scale is by default in millimeters, however this can vary depending on the software used to read the file or set its parameters.

The steps used to process the data contained in these STL files are then presented in the following sections. The notations used in this report are explained in appendix A. The method presented here until the end of the point selection process could be generalized to any other files types containing 3D point cloud.

All the code and figures can be found on the following github repository:

$$https://github.com/PaulineMauryL/PropellerProject$$

A ReadMe file explaining its use can be found in the repository and is attached in appendix B.

2.1 File pre-processing

2.1.1 STL file to python

As reference for the shape and appearance, the STL file of the propeller was first openened by means of a 3D software called 3D Viewer by Microsoft[1]. The propeller is then shown in figures 2.1a and 2.1b.

The points of the surface of the propeller were then extracted from the STL file by means of the numpy-stl library[2] and stored in a format that could be used efficiently with python later on.

As the point cloud contains multiple duplicate points, it is important to remove them for computational and simplicity purposes. Indeed, in the case of the propeller shown above, certain points are present up to more than 20 times.

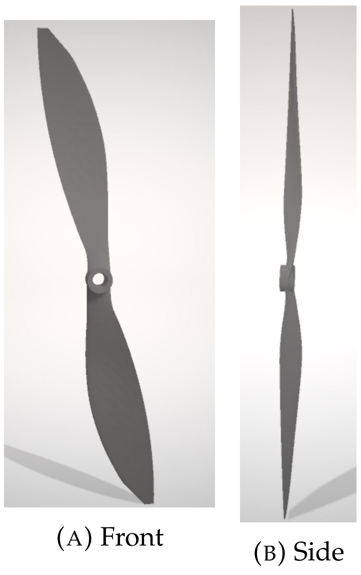

(A) Front (B) Side

FIGURE 2.1: Propeller in 3D Viewer

[1]https://www.microsoft.com/en-us/p/3d-viewer/9nblggh42ths?activetab=pivot:overviewtab
[2]https://pypi.org/project/numpy-stl/

2.1.2 Principal directions of the propeller

An important step of the pre-processing is to find the three principal directions of the propeller to obtain a reference of direction for future computations.

As the density of points in the different regions of the propeller is uneven, the center position of the propeller cannot be found by simply computing the average of the given points. Thus, to compute the position of the center of the propeller P_{mid}, the extreme values in each of the three dimensions of the propeller are first computed and the intersection point of the vectors connecting the extreme values in each direction is taken as center point.

As the tip of the blade has been aligned on the z-axis, the steering vector in the length of the propeller $\vec{n}$ is first computed by subtracting the coordinates of the center point from the coordinates of the furthermost point at the top of the blade. $\vec{n}$ is the plotted on the point cloud as illustrated in figure 2.2.

To find the direction that goes through the hub of the propeller, the three closest points to the middle of the propeller $A,\ B,\ and\ C$ are computed. With those three points, two vectors $\vec{AB}$ and $\vec{AC}$ can be traced and their cross product is computed. The resulting vector $\vec{v} = \vec{AB} \wedge \vec{AC}$ is then the direction that goes through the hub of the propeller.

Note:
- To compute the direction that goes through the hub of the propeller, the assumption has been made that there is a circular hole in the center of the propeller. This assumption should be true for most aerofoil as the center of blade always rotates around the middle point.
- This computation is only possible in the case where the two vectors are not co-linear. If it is the case, another point D is assigned and used for the computation.

Finally, the last direction $\vec{w}$ goes along the side of the propeller. It is found through the following cross product $\vec{w} = \vec{v} \wedge \vec{n}$.

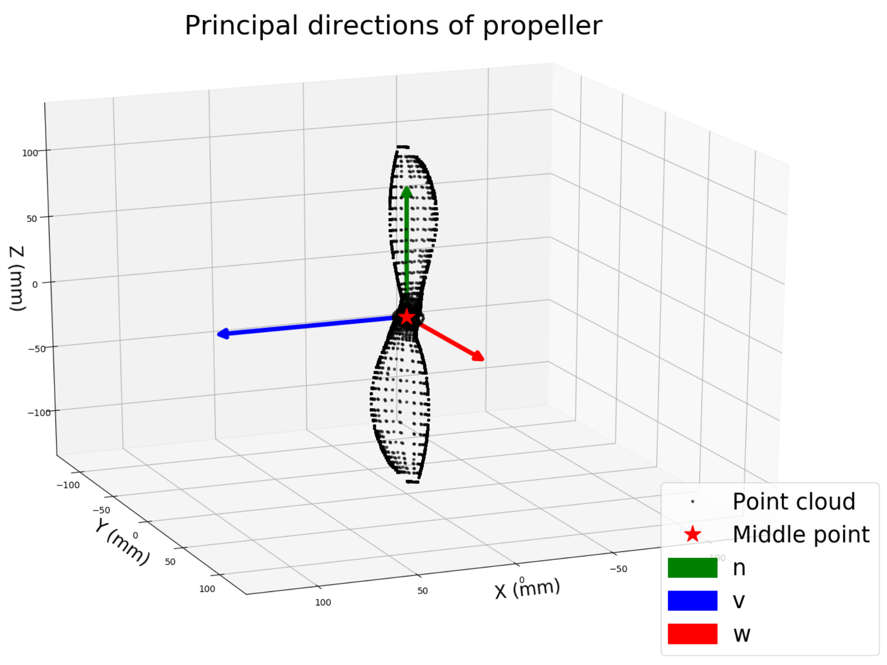


FIGURE 2.2: Principal directions of propeller

2.1.3 Propeller alignment

To facilitate further computation, the point cloud is translated such that the point P_{mid} computed above in 2.1.2 is centered on the origin. It is then rotated to have the direction of the length of the blade $\vec{n}$ aligned with the z-axis. Therefore the point that is the farthest away from the middle of the propeller is considered as the tip of a blade and is considered as a reference for the 3 dimensional rotation computation. Finally, it is rotated to align the vector going to the side of the propeller $\vec{w}$ along the x axis.

2.1.4 Blades selection

Making the assumption that propellers have a rotational symmetry about the axis of rotation, it is possible to select only one of them to avoid any redundant computations. The main concept is therefore to compute the equation of the plane at the center of the propeller and classify all the points above that plane as part of the upper blade and all the points under the plane as part of the lower blade. In this case:

$$d_{n-mid} = -\vec{n} * P_{mid} = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \wedge \begin{pmatrix} x_{mid} \\ y_{mid} \\ z_{mid} \end{pmatrix} \tag{2.1}$$

where $\vec{n}$ is the vector perpendicular to the plane.

Each point of the propeller is then tested in against the following equation $ax + by + cz + d_{n-mid} > 0$. If the condition holds true, the given point is categorized as part of the upper blade, otherwise it is categorized as part of the lower blade. The points of one of the blades are then used for the following computations.

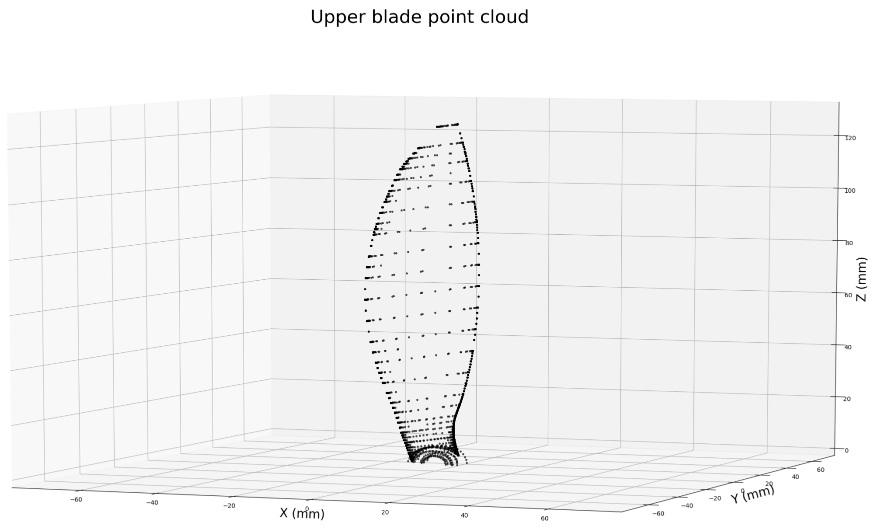


FIGURE 2.3: Upper blade

<u>Note:</u> The method presented in this project works in the case of a propeller with two blades (one on each side of the plane). In the case of propellers with N_{blade} blades where $N_{blade} > 2$, there should be N_{blade} equations of planes going through the middle points with a normal vector along the direction of each blade. The points would then be assigned to their respective blades following an equation similar to the one presented above.

2.2 Propeller section computation

To be able to compute the aerodynamic characteristics of the aerofoils, a major required step is to compute its shape at specific positions.

The following subsections present the algorithm.

2.2.1 Plane equation

In order to obtain the sections of the propeller required by the Blade Element Momentum Theory, a first step is to compute the equation of the plane of each blade element that is to be studied. The equation of the plane is computed thanks to the middle point P_{n-mid} described above and the length steering vector which is the vector normal to the plane $\vec{n}$.

Similarly, the equation of a plane with a normal vector $\vec{n} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$ and point $P = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ is given by $ax + by + cz + d = 0$. Hence, d of each plane is computed as $d_n = -\vec{n} * P$. This is the general case of the blade differentiation computation.

In the algorithm, the number of aerofoil segments N_{seg} to be considered along the blade must be given as input by the user. Then the d parameter at the tip of the blade (d_{n-max}) and at the middle point (d_{n-mid}) of the propeller are computed. Thus, the distance of translation between each plane is $\Delta = \frac{d_{n-max} - d_{n-min}}{N_{seg}}$. The equations of each planes are then taken computed one after the other by adding Δ to each previous d:

$$Plan_0 : ax + by + cz + d_{n-mid} = 0$$

$$Plan_1 : ax + by + cz + (d_{n-mid} + \Delta) = 0$$
$$Plan_2 : ax + by + cz + (d_{n-mid} + 2 * \Delta) = 0$$
$$\vdots$$
$$Plan_N : ax + by + cz + (d_{n-mid} + N * \Delta + \epsilon) = ax + by + cz + d_{max} + \epsilon = 0$$

For the last plane, a small ϵ is added to ensure that all points will be considered to be below it (even close neighbour that could have been omitted) in future stages.

As illustration in figure 2.4, the points have different colours in between different planes.

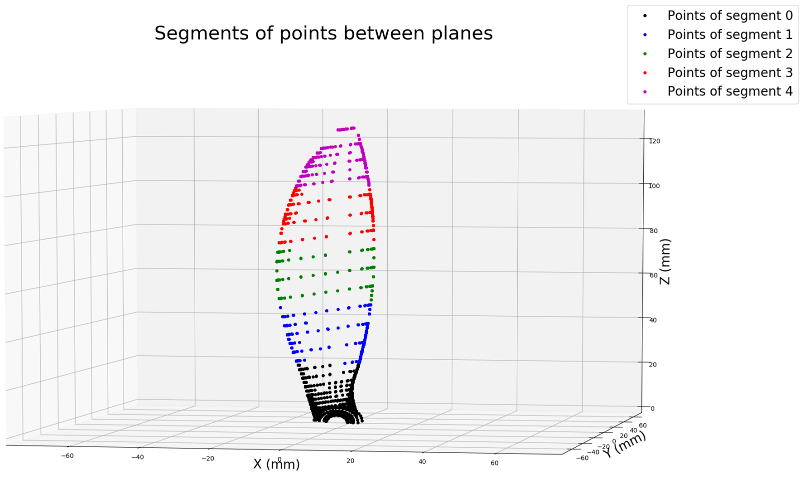


FIGURE 2.4: Points between planes

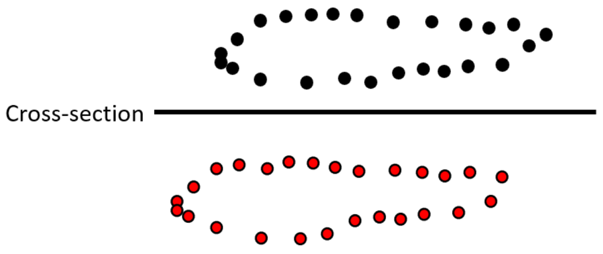
2.2.2 Aerofoil shape

Points selection for projection

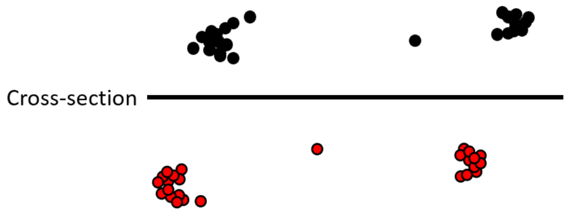
The selection of points to consider for projection is a very important step for the whole algorithm as all following stages of the process rely heavily on the points selected for the computations. Indeed, when points are not properly selected, interpolations are made on point distribution that are sparse in most of the contour and very dense in specific location and hence, the interpolated curve do not represent the aerofoil shape at all.

The simplest idea is to compute the distance of each point to the plane of projection and keep only a selected number of closest points. However, a drawback of this method is that the points are very unevenly selected as the STL file of the propellers do not have an homogeneous repartition of points. Indeed, due to the shape of the blade, there is a very high amount of points to describe the contours of the blades and much less points towards the interior. In figure 2.3, an illustration of this problem can be seen where contrarily to the points describing the contours of the blades, those towards the center are arranged in a sequence of rows followed by empty spaces. The selection of points based on this method therefore only leads to the selection of the points along the edges of the blade.

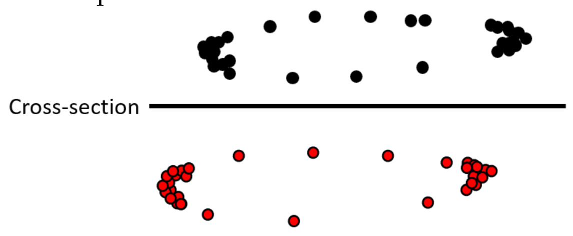
Figure 2.5b illustrates the ideal distribution of points versus the real one. This uneven distribution of points can be observed in ll the point cloud figure: there are much more points on the contour of the propeller than elsewhere.



(A) Ideal distribution of points



(B) Real distribution of points (first method)



(C) Real distribution of points (second method)

FIGURE 2.5: Point distribution

The second method that was considered is based on the same computation used for the blade segmentation which forces all the points within 2 planes parallel to the projection plane to be selected. A length δ is chosen with respect to the overall density of points and all points within this distance, above and below the plane are taken. For instance, if the plane of projection is

$$P : ax + by + cz + d_{n-mid} = 0 \tag{2.2}$$

then all points respecting

$$ax + by + cz + (d_{n-mid}+\delta) > 0 \quad and \quad ax + by + cz + (d_{n-mid}-\delta) < 0 \tag{2.3}$$

are kept.

Despite the points being more evenly selected with respect to the previous method, this method also presented the issue of keeping much more points on the outer edges than towards the center of the propeller as in figure 2.5c. Hence, interpolation was performed on very unevenly distributed points and the points on the contour did not have enough weights (weren't numerous enough) compared to the one on the outer corner and hence, wasn't successful.

Another strategy was then considered consisting of taking at least a row of points from both sides of the propeller while keeping track of the number of added points each time the distance from the original plane was increased by δ. The addition of a large number of points added in one step would then act as an indication that a row had been detected. Whereas this method did improve the selection process, it still led to an uneven distribution with too many points along the edges of the blade reducing thus the accuracy of the interpolation stage following this one.

Implemented: The method that was kept for this part of the process was therefore the following method which, similar to the ones presented above, first consisted of computing the equation of the plane. This plane was then shifted by a distance δ which enabled the determination and selection of the "new points" above and below any given plane by determining the points between itself and the planes above and below it. In the case of this project, a $\delta = 0.05mm$ yielded the best visual results without greatly reducing the computational speed. In theory, by decreasing this value, the accuracy of the results could be improved, however this has no significant effect on the selection of the points and only presented a disadvantage of increasing the computational time.

Each "new point" is then only added to the previously selected points it if has a greater distance than the chord of the section's aerofoil divided by 40[3] from all from these points. This restriction is crucial, as mentioned previously, since the points are very unequally distributed in space. Without this restriction, almost all selected points lie along the edges of the blade and the interpolation stage explained in the later sections 2.2.2 yields poor results.

The algorithm is repeated until one of the two following condition is met:

1. A threshold number τ of points have been selected (τ = 30)
2. the planes used for considering new points yield a greater than 0.5mm (there have been 100 iterations with a $\delta = 0.05mm$ added each time) to the aerofoil plane.

The pseudo-code of the algorithm is presented on appendix C.

With this selection method, some aerofoil have very few points (especially at the tip of the propeller as there chord length is smaller) and thus either show less accurate interpolation or can not compute their interpolation at all (5 points needed for fourth order curve). Hence, the planes that do not have enough points are not taken into account in later computations.

[3]Once again, many values were tested and the plots visually inspected afterwards. Dividing by 40 seemed to be the optimal choice

Edge distinction

To determine the interpolation of the aerofoil shape, the upper and lower edges of the aerofoil must be distinguished with one curve fitting the upper edge and another curve fitting the lower one. The points must therefore first be divided into two categories. The method employed here consists of computing the equation of the least squares line between all given points and then use it to divide them into two categories: upper edge and lower edge. The points were first kept in 3-dimension to perform this computation. However, the least squares led to a separation of the points along the length axis, not along the aerofoil section as desired. The least squares line is thus only computed in 2 dimensions, approximating the y value based on the x value.

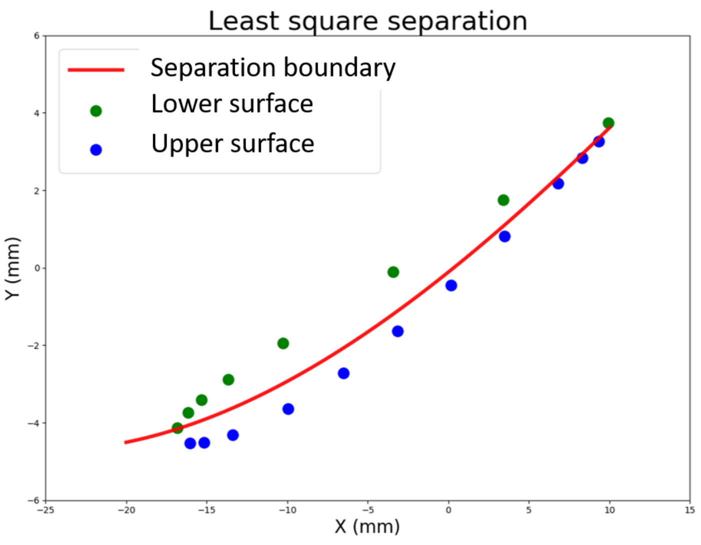


FIGURE 2.6: Least squares separation

Note: To enable this, the blade is rotated in the pre-processing stage, where the blade is aligned with the z-axis.

Interpolation

Finally, the next stage consists of interpolating the points along both sides of the blade. Initially, as mentioned in the previous subsection, the 3 dimensional coordinates of the points were considered. Hence, for each plane, four interpolations were performed: one along the lower edge above the plane, one along the upper edge above the plane, one along the lower edge under the plane and one along the upper edge under the plane as in figure 2.7.

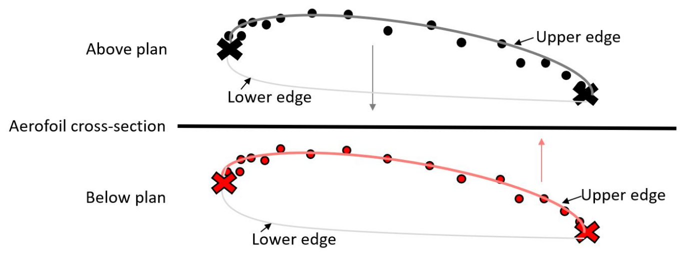


FIGURE 2.7: Another strategy

The goal was then to assign a couple of points to each other with one of those points on one of the edges (upper or lower) above the plane with another point along the same edge below the plane. However, in 3 dimensional coordinates, it was not possible to determine properly which pair of points should be assigned together as the number of points on each side were not necessarily equal such that strong assumptions would need to be provided and implemented in the algorithm. For instance, assigning pairs of points that were closest to each other when projected on the plane between two aerofoils did not lead to conclusive results. Furthermore, selecting points at uniform intervals on the curves to make projections was not efficient as the shape of the curve on the z-axis was not representative of the real shape of the propeller.

Implemented: Thus, similar to the previous subsection, the points were only considered in two dimensions for this stage, i.e. all selected points above and under the plane were interpolated by considering only the x and y coordinates. The assumption is therefore that, in this case, the loss of information along the z axis does not degrade the performance of the algorithm as the deviation along the z axis for the selected points is small. To select the best curve approximation for this interpolation, the points were exported into Matlab and were fitted by means of the curve fitting tool[4]. By visualizing the curve fitting of multiple functions, it could be determine that the most appropriate fit could be found with polynomial curves or smoothing splines. However, as each aerofoil section may have a different shape, it cannot be said that any specific interpolation curve is appropriate for all curves. As such, the fourth order polynomial was kept as it enables an appropriate fitting of the real shape of the blade and fits most curves with relatively low error.

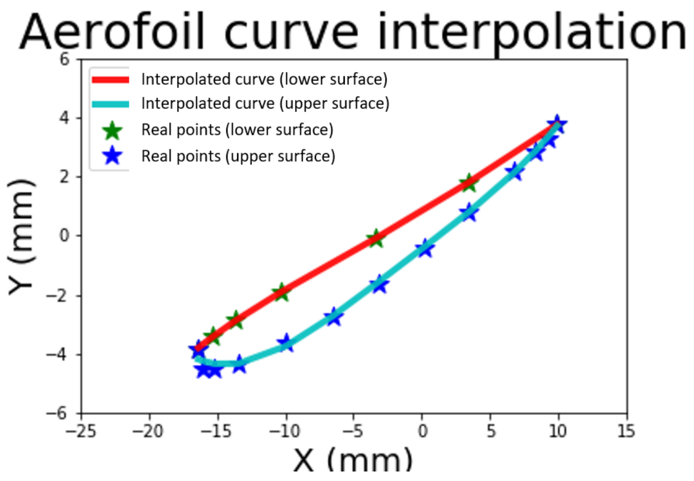


FIGURE 2.8: Interpolation

Moreover, the other aerofoil parametrisation techniques mentioned in the State of the Art were not used. First, it made no sense to use a parametrisation describing the shape with geometric parameters as to compute those parameters, the points would have first needed to be interpolated leading to even less accuracy. Second, no function nor library were found to compute the control points of an interpolation with Bezier curves.

Generation of points

The outputs of the preceding interpolation stage are the selected points on both sides of each plane (between aerofoils) with their associated optimal parameters for the 4th order polynomial curve.

However, these outputs are not appropriate inputs for the X-foil software application for two major reasons:

1. When combined and plotted, the two interpolation lines do not describe a continuous shape of the aerofoil as the last points at each end of the curves do not share a common point.

2. The distribution of these points is still uneven as some parts of the aerofoil do not have any points and the number of points is also different for each aerofoil section.

The first of these issues can be solved by first determining the point with the smallest (respectively largest) value along the x axis on whole the points of the aerofoil. Then 100

[4]https://ch.mathworks.com/help/curvefit/curvefitting-app.html

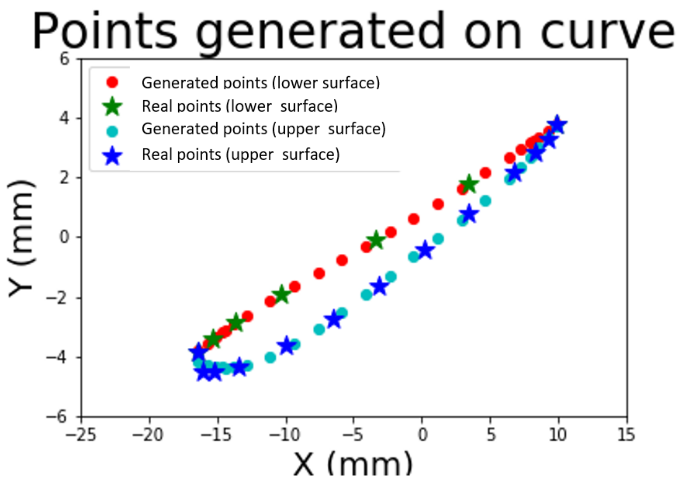


FIGURE 2.9: Aerofoil points reconstruction

points are linearly taken from the smallest to the highest x values. Both sides computes their y coordinate based on this same range along the x axis. By computation of the y coordinate of each given x coordinate with the optimal 4th order polynomial, it can then be verified that the two curves now share a common point on both sides of the aerofoil.

The second issue is solved by simply computing 100 equally spaced points between the smallest and largest values along the x axis of the aerofoil. Their corresponding y coordinates can the be found on both sides through computation with the optimal parameters of the 4th order polynomial found previously. The resulting plots are then presented in chapter 3.

2.3 Propeller parameters

In this section, the parameters of the propeller's shape are determined and can be seen in figure 2.10. These parameters consist of:

- The tip radius
- The hub radius
- The chord length
- The blade twist

2.3.1 Tip radius

The tip radius is the distance between the point at the centre of the propeller and a tip of one of the blades. It is shown on figure 2.10. In this case, this consists of finding the maximum distance between the point at the centre of the propeller and all other points.

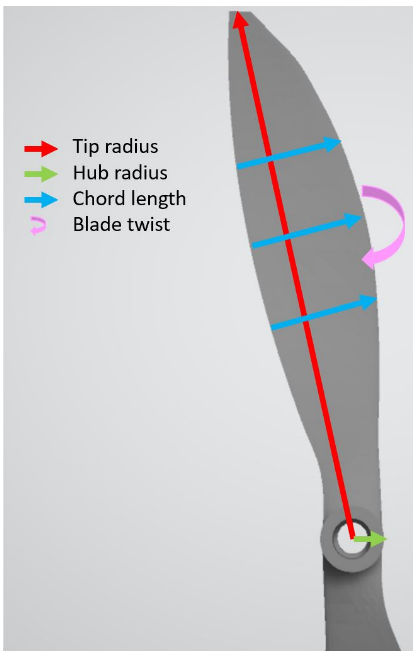


FIGURE 2.10: Parameters

Note: Ideally, the two blades should be equidistant from the centre of the propeller such that two maximal distances should be found. However, this is not necessarily the case.

2.3.2 Hub radius

The hub is the central portion of the propeller through which the axle passes. The hub radius is the distance between the center of the propeller and the outer edge of this cylindrical region. The following paragraphs present the explanation for the computation of the hub inner and outer radius.

Hub inner radius The hub inner radius length $r_{hub-inner}$ is first determined by computing the mean distance between the middle point P_{mid} and its three closest points to the middle point. The 3 closest points are taken instead of just a single point to ensure a more robust value in the case where the middle point is not perfectly centered.

This value then enables us to find a point in the inner radius $P_{hub-inner}$ in the direction of the side of the propeller $\vec{w}$ through the following equation:

$$P_{hub-inner} = P_{mid} + r_{hub-inner} * \vec{w} \tag{2.4}$$

Hub outer radius To compute the point on the outer radius $P_{hub-outer}$, the equation of the plane going through P_{mid} and with normal vector $\vec{n}$ is computed as in section 2.2.1:

$$a_w * x + b_w * y + c_w * z + d_{w-mid} = 0 \tag{2.5}$$

The algorithm then tests each point with this equation after which the one giving the highest result is characterized as a point on the outer side of the hub, $P_{hub-outer}$. $P_{hub-inner}$ and $P_{hub-outer}$ are then represented in black on figure 2.11.

To ensure that no outlier point would come from an upper part of a twisted blade as well as to reduce the computation time, only the points within a small distance to the plane with normal vector $\vec{n}$ and reference point P_{mid} are selected. Therefore, two planes parallel to it and shifted of $\pm\Delta d$ are computed and every point in between those two planes (same methodology as in section 2.4) are kept for the computation. These points are represented in cyan in figure 2.11.

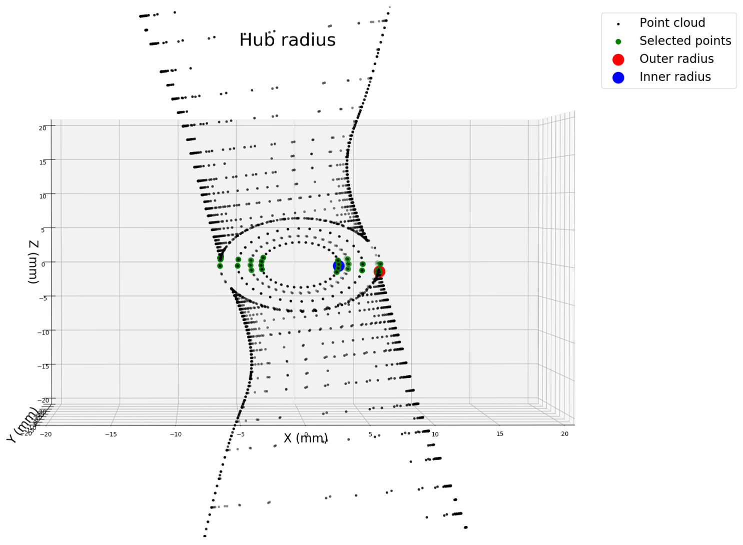


FIGURE 2.11: Hub radius points

Then the hub radius is then simply:

$$r_{hub-outer} = norm(P_{hub-outer} - P_{middle}) \tag{2.6}$$

2.3.3 Chord length

The chord length is the distance between the trailing edge and the point on the leading edge where the chord intersects the leading edge, it is represented on figure 2.10. To obtain the chord length of the blade elements, the algorithm to compute each aerofoil shape as in section 2.2.2 must first be done. The chord length can then be computed with the extreme points along the x axis determined previously with both y approximations (upper and lower curve). By computing the average y coordinate for each of these extremities and computing the norm distance from one to the other, the chord length is determined.

$$ChordLength = mean(norm(P_{mid}, closest_{points})) \tag{2.7}$$

2.3.4 Blade twist

The blade twist is the angle of rotation of the aerofoil with respect to the plane of the hub, it is represented on figure 2.10. As for the chord length, two points are obtained: x minimum (respectively maximum) with the mean of its two y coordinates. The vector between those points is then computed. The angle between this vector and the x-axis represents the blade twist and is computed for each aerofoil section.

$$BladeTwist = angle(ChordLength, x - axis) \tag{2.8}$$

Note: The values of the chord length and blade twist, as well as the graphs on which the values were computed are in appendice B.

2.4 Inputs to aerodynamic software

The aerodynamic parameters as well as the aerofoil shape are needed as input to aerodynamic software. It should then be able to output the lift and drag coefficient of the blade at different angle of attack to further enable the thrust and torque of the propeller.

2.4.1 Aerodynamic parameters output

First, the radius r, distance from the center of the propeller to the aerofoil cross-section is computed. As a reminder, the user must give as input of the file the position in percentage $pos(\%)$ along the radius of the blade. Hence, the radius is

$$r = \frac{pos(\%)}{100} * R \quad [mm] \tag{2.9}$$

(R is the tip radius).

The velocity of the fluid with respect to the object, u, is then

$$u = r * rpm \quad [mm/min] \tag{2.10}$$

The Reynold number is computed with

$$Re = \frac{u * ChordLength}{\nu} \tag{2.11}$$

with $\nu = 14.88[mm^2/min]$, the kinematic velocity.

The Mach number is computed with

$$M = \frac{u}{c} \tag{2.12}$$

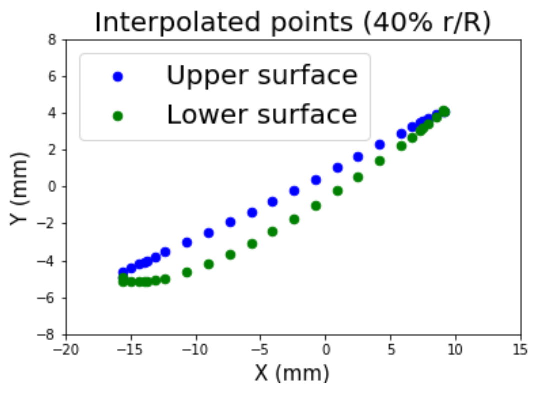
with c the speed of sound $c = 343[m/s] = 20580000[mm/min]$.

Note: The table for the radius, Reynold and Mach numbers along every 10% of the radius are in appendix C.

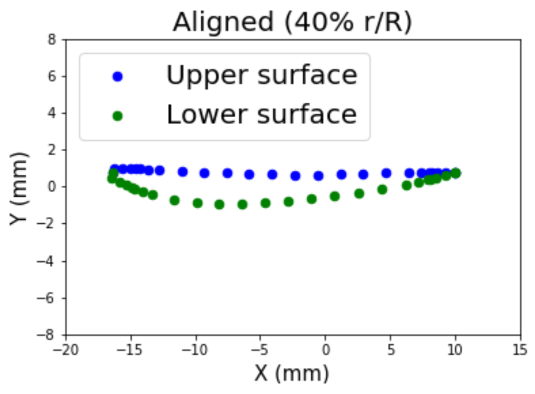
Those parameters are also output in a txt file if the rotation speed rpm of the propeller is given as input to the algorithm.

2.4.2 Aerofoil points

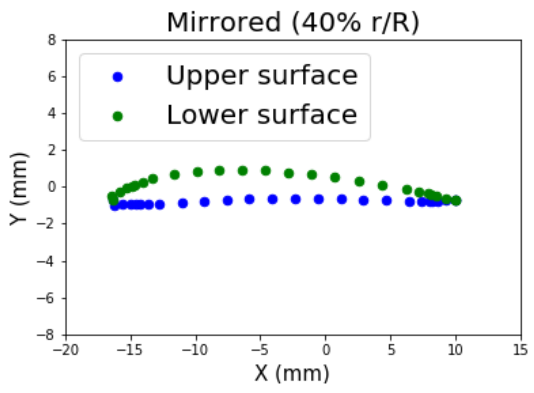
In the case of the X-foil software [12], which was tested during the project, the points must be formatted specifically. The process is explained in the section.



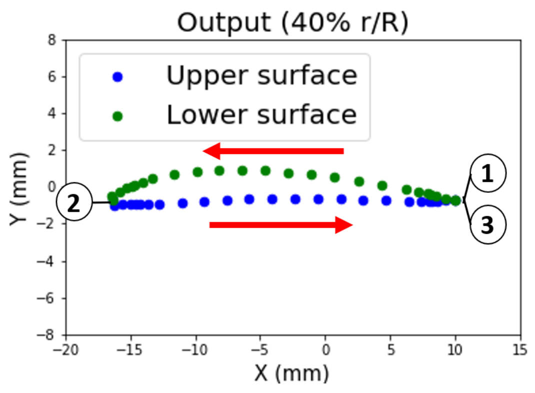
(A) Points selection



(B) Alignment of chord length with x-axis



(C) Mirrored



(D) Order of points in file

FIGURE 2.12: Preparation of X-foil input

First, as observed in the Airfoil Tools database.[13] the points are not linearly computed but more points are given near the edges of the blade. Hence, the same procedure was followed here as can be seen in figure 2.12a. The points are taken in the range of the x-coordinates and the y is computed with the optimal parameters of a fourth order curve.
Afterwards, the aerofoil must be rotated to have its chord length parameter parallel to the x-coordinates as in figure 2.12b.

Afterwards, the aerofoil was mirrored to have its more 'curved' side above as it is the general way of representing the aerofoils in X-foil [12] and in the database AirfoilTools [13]. Previous version of Xfoil required the input to be normalized with a range between 0 and 1 on the x coordinate ($ChordLength = 1$). However, there is also a command in Xfoil which does it automatically. The command to enter is 'NORM' before loading the aerofoil.

Finally, the values printed in the file that will be given as input of X-foil must be of a specific shape and order. Each point is given on a line with its x-coordinate followed by a space and then y-coordinate (and optionnaly a space and a z-coordinate). This step can be observed in figure 2.12c.[5].
The first datapoint must be the trailing edge. Afterwards, the data points are the next point to the leading edge and then to the trailing edge as in figure as explained in the xfoil documentation. The file can be either a .txt or a .dat file. In this project .txt files were made.

In X-Foil, many strategies were tested to find the lift and drag coefficients of the aerofoil:

- Normalizing the aerofoil before creating the txt file and normalizing it directly in X-Foil
- Allowing a higher number of possible iterations to find the optimal parameters
- Testing the output with different aerofoils (at different positions such as 30%, 40%, 50%, 60%, 75%)
- Using the recommended 'PANE' command of the XFoil documentation to have a better point distribution
- Testing different Reynolds and Mach numbers

However, no strategy gave satisfying results as will be discussed in the results.

[5]Previous version of X-foil had more restrictive needs but with version 6.99 this method works.

3 Results and discussion

To evaluate the reliability of the algorithms presented previously, the following sections present a comparison between the real measurement values and the computed values of certain parameters of the propellers.

3.1 Results

3.1.1 Aerofoil parameters

Table 3.1 enables the comparison between the real values of the hub tip and hub radius versus the values computed according to the methodology presented in the previous chapter.

	Hub radius	Tip radius (R)
Computed	6.541	126.8067
Real	7.15	127.00[1]
Relative error	0.080	0.0015

TABLE 3.1: Computed versus real values

The resulting Chord length and Blade Twist for different aerofoil positions were also obtained and are shown in appendix D. Their root mean squared error (RMSE) was also computed for each of the 16 linearly spaced aerofoils from $20\% r/R$ to $95\% r/R$. The results are presented in table 3.2.

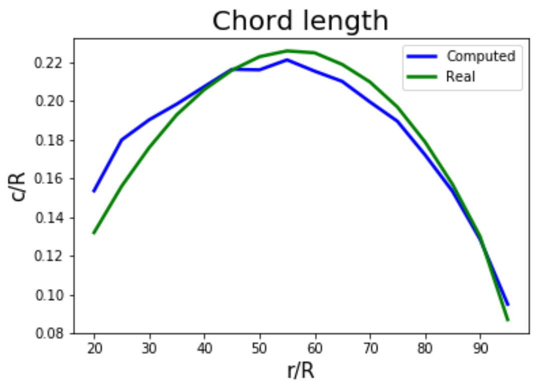
	Chord length	Blade twist
RMSE	0.05959	8.75817

TABLE 3.2: Difference between real and computed model

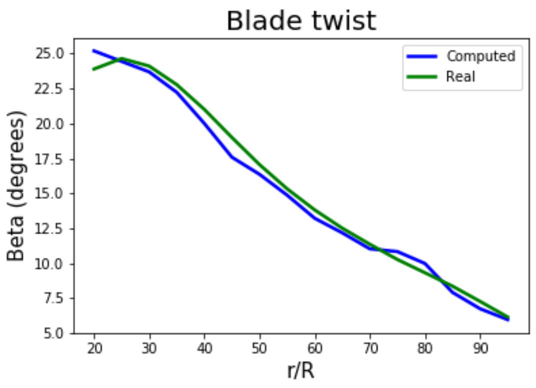
By taking chord length and blade twist on equally spaced aerofoils, figure 3.1a and 3.1b can be obtained where the values along the axes are normalized by the tip radius R. r/R is thus the normalized position along the blade and c/R is the normalized chord length[2]. The real values can then be compared to the computed values with accurate results except when close to the hub of the propeller.

[1]Manufacturer: 9 inches diameter propeller. Hence radius = 4.5 inches = 127cm.

[2]As in section 2.2.2, the aerofoils that are close to the tip and the hub of the propeller are not taken into account as the number of points is too low



(A) Chord length



(B) Blade twist

FIGURE 3.1: Plots of real vs computed parameters

3.1.2 Aerodynamic software

The algorithm is able to output:

- The blade parameters: hub radius, tip radius
- The interpolated aerofoil shape with 2 dimensional points
- The associated blade twist and chord length of each aerofoil
- The associated aerodynamic parameters: Reynolds and Mach numbers for each aerofoil

Those output can be used as input in different aerodynamic softwares.

In this project, the Xfoil software was chosen for testing. However, when entering the files in X-Foil and computing the lift and drag coefficient, no results could be found in viscid mode despite a convergence in non viscid mode. Indeed, When a Reynolds number was given as input, the results did not converged.

One reason that could explain this is that the points are taken from two different interpolated curves and the transition between them may not be sufficiently smooth. Even though the algorithm ensures that the transition from one curve to the other is continuous (their first and last points are the same such that both curves should join), the transition on the edges might be too sharp and hence unrealistic, making it impossible for X-Foil to converge in viscid mode.

3.2 Discussion

3.2.1 Project methodology and results

The algorithms implemented require at least one input: the STL file of the propeller. However the following inputs can optionally be given to tune the desired performance.

- $N_{aerofoil}$: Number of aerofoils desired (default = 50).
- δ: Minimum distance for point selection iteration (default = 0.05 mm). Its minimum value should be the resolution of the 3D scan, otherwise, computational speed would be lost without any reason.
- max_{it}: Maximum number of iterations with δ for point selection (default = 100, hence the maximum distance between a selected point and the aerofoil section is 5 mm). This parameter has to be chosen carefully. Indeed, if it is too small, then there is often an insufficient amount of selected points to compute the interpolation. However, the higher it is, the less the accuracy can be ensured.
- The rpm of the propeller (default is 6500) to compute the Reynolds and Mach numbers for each aerofoil.

As the assumption is made that STL files are by default in mm, no additional information about the metric needs to be provided.

The outputs are then:

- The x and y coordinates of the points of each aerofoil; 100 points for the lower side and 100 for the upper side (stored in a txt file).
- The tip radius, hub radius, chord length and blade twist parameters for each section of the aerofoil (in a csv file).
- Reynolds and Mach number for each aerofoil (in a csv file).

The following 4 python libraries were used in this project: numpy, pandas, numpy-stl and scipy which need to be installed to run the algorithms presented in the previous chapter. Numpy and pandas enable the processing of the data, scipy enables the determination of the optimal parameters for the curve interpolation and numpy-stl enables the conversion of the stl propeller file into a format that can be processed in python.

The 4th order polynomial curves were chosen to find the best fit of the interpolation of the edges of the aerofoils instead of the aerofoil parametrization explained in the state of the art in section 1.2 for multiple reasons. First, using methods with parameters as Parsec, Sobieczky or IGP did not make sense because an interpolation would initially have been required to compute them. Furthermore, Bezier curves could not be computed efficiently based on the low number of points that were selected. Finally, post visualization of the shape of aerofoil with 2 to 3 inflection points, taking the fourth order polynomial made sense because this order of polynomial can have 3 inflection points without overfitting. Also when observing the points with Matlab's curve fitting tool (cftool), the best fits were observed with polynomials with orders above 3 and smoothing splines.

A major issue that had to be adressed to obtain the shape of the aerofoil was the uneven distribution of the points contained within the STL file. Indeed, the most crucial step of the whole algorithm is the selection of the points with which to compute the aerofoil's shape (explained in the pseudo-code in appendix C). The selection of the points is also subject to a trade off between the computation time and the accuracy: the smaller the increase in δ at each iterative step, the better the accuracy. However, this results in a longer computational time required to select the points. For the best accuracy the δ should therefore be equal to the accuracy of the 3D scanning tool.

The difference between the computed and real parameters may be due to one or more of the following reasons:

1. The points are not well distributed along the blade and the STL file does not contain sufficient information for each plane such that in extreme cases (in the middle between two line of points on smooth surface) the algorithm can not work properly as points are too far away.
2. A fourth order curve is used to interpolate the selected points. However, usual optimal parametrisation methods do not use such polynomials but rely on Bezier curves or the other parametrisation techniques. Hence, the propeller was not 'created' based on the parametrisation and a polynomial curve will never be a perfect fit.
3. Two curves are interpolated separately based on different points along the upper and lower sides of the aerofoils and then put back together to 'close' the shape with a point that is the mean of the interpolated point. Hence, the position of the points along the edges may not be sufficiently reliable as their final position may be different than the interpolated one.

All useful parameters are output from the algorithm and can be taken as input for further analyze on aerodynamic softwares. Hence future work for this project would be to test and compare the outputs of aerodynamic softwares. Either by improving the aerofoil shape for the Xfoil software to converge or by testing the algorithm on other aerodynamic softwares such as Pablo[3] and JavaFoil[4]. Currently, no results were obtained with the X-foil software. As explained in the results, this is probably due to the sharp transition between the two interpolated curves. Indeed, forcing them to begin and end at the same point may not be a sufficiently strong enough continuity condition to ensure good results. Moreover, X-Foil does not take as usual input points distributed on fourth order polynomial and might not be robust to very small variation of points distributions.

3.2.2 3D scanning choice for reverse engineering

While researching and coding segments of this project, many unexpected issues occurred, some of which are useful to develop a better understanding of the optimal 3D scanning technology to use to facilitate a reverse-engineering process.

The major point to take into consideration is the distribution of the points. In this project, STL file was taken as input and converted into 3 dimensional points in space. The problem is that STL file contains a lot of points on edges and non-smooth surfaces. However, on smooth surfaces, points are very sparse. Hence, when computing results for an aerofoil, points on the sides must be filtered as there are many and points along the edges tend to be further away from the cross-section reducing the precision of the algorithm. To ensure a more accurate and faster algorithm, a denser and more uniformly distributed point cloud should ideally be obtained from the 3D scanning. Hence, an STL file might not be the most appropriate solution to store the data after the 3D scanning. Other file formats than STL

[3]http://www.pdas.com/pablo.html
[4]http://www.mh-aerotools.de/airfoils/javafoil.htm

were studied but the same issue arises, i.e. uneven distributions of points on smooth surface is an intrinsic part of all representations.

As STL is the leading format in 3D scanning and printing, a compromise could be to still use it but with very high precision to obtain a denser point cloud on smooth surfaces. However, in such a case, the filtering stage of the points along the edges would still be required as there would be even more points in such regions.

Another solution would consist of generating points by interpolation on smoother surfaces to obtain a high density of points before applying the algorithms described previously.

Moreover, the 3D scanning technique used to represent the aerofoil does not need to be neither mobile, nor very fast, nor capture texture or colors. Hence, all presented 3D scanning technologies would be able to perform the task.

If price is the main consideration, it would make sense to simply use photogrammetry as it is the cheapest method available and can give very accurate results with an accurate and precise camera. Moreover, the camera is not task specific and could be used for other purposes. There are only two drawbacks of photogrammetry which can be solved relatively easily. First, the need to use software afterwards to reconstruct the image (but there are many effective and free software). Second, the sensibility to the possibly reflective surface of propeller such that the surface of the propeller should be prepared with a matting powder or a coating spray.

If the accuracy is the most important parameter and needs to be very high, then a CMM is the most reliable technique and no surface preparation is required. However, the price is much higher than for photogrammetry and the device is task specific.

4 Conclusion

In this project, the implementation of an algorithm to reverse engineer the shape of propellers from 3D scanning data is described. The first parts of the algorithm could be extended to any 3 dimensional point coordinate data. More specifically, the algorithm can output aerofoil shape with points at any given position along the blade, blade parameters (hub radius, tip radius, chord length and blade twist) with close accuracy, plots to visualize those data and associated aerodynamic parameters (Reynolds and Mach) for each aerofoil. Extracting the lift and drag characteristics of these shapes remains would be a new possibilities with those output taken in aerodynamic softwares as input. Additionally, some insights were given to better choose the 3D scanning technique for further improvements and research of objects shape reverse engineering.

No previous papers were found concerning the reverse engineering of propeller from 3 dimensional scanning data for propellers and no complete method for general object shape study either. The algorithm presented here has no inspiration from any other study.

This algorithm demonstrates the possibility of extracting the aerofoil shapes of propellers based solely on an STL file. This shows the possibility for drone scientists as well as other researchers to reverse engineer 3 dimensional shape of object based on 3D scanning solely. Hence, studies could be conducted more easily as the issue that manufacturers do not typically publish propellers shape and parameters would not be a difficult problem to solve anymore.

This project proves that it is possible to output the important aerodynamic parameters of a propeller based only on its 3 dimensional points. This opens new possibilities for professional drones engineer to create common databases with a wide number of propeller and their associated aerodynamic characteristics enabling to gain time and improve the design phase.

To conclude, I would like to thank my supervisors, Professor Dario Floreano, Sebastian Steffen and Anand Bhaskaran of the Laboratory of Intelligent Systems (LIS) for their advice and support.

A Notations

Symbol	**Signification**
P_{middle}	Point in the middle of the propeller
$P_{hub-inner}$	Point on the inner side of the hub
$P_{hub-outer}$	Point on the outer side of the hub
$r_{hub-inner}$	Radius from center of propeller to $P_{hub-inner}$
$r_{hub-outer}$	Radius from center of propeller to $P_{hub-outer}$
d	Bias in equation of plan ax + by + cz + d = 0
d_{n-mid}	Bias of plan with normal vector $\vec{n}$ and at point P_{mid}
d_{w-mid}	Bias of plan with normal vector $\vec{w}$ and at point P_{mid}
N_{blade}	Number of blade of propeller
$N_{aerofoil}$	Number of aerofoil to compute
Δ	Distance between two aerofoil planes
δ	Distance between two planes during point selection
τ	Maximum number of points to select for an aerofoil section
κ	Minimum distance between two points selected for an aerofoil section
max_{it}	Maximum number of iteration of δ to select the points for each section
u	Velocity of the fluid
ν	Kinematic viscosity
Re	Reynolds number
M	Mach number

B Github ReadMe File

The github read me file provides an explanation for the user.[1]

Propeller-Project

The user only has to interact with the file propeller_information.py.

The file propeller_information.py imports its functions from:

- preprocessing.py to preprocess the aerofoil,
- aerofoil_shape.py to compute the points on each aerofoil,
- parameters.py to compute the propeller parameters,
- final_aerofoil_plot.py to plot the aerofoil with their blade twist and chord length,
- aerodynamic_parameters.py to compute the Reynolds and Mach numbers,
- xfoil_output.py to output the .txt file in the right shape for the X-foil software

INPUTS – when launched, the algorithm asks the following inputs

1. 'Enter your stl file name (without .stl)': The user should give as input the name of the stl file (ex: propeller) This file should be in the same folder as propeller_information.py.
2. 'Enter your positions (in percentage from hub to tip)': The user should give as input the positions of the aerofoil in percentage from hub to tip (ex: 40 50 for 40 r/R and 50 r/R)
3. 'Aerodynamic parameters ? (1: yes, 0: No)': The user should enter 1 if he/she wants to output the Reynolds and Mach numbers in a .txt file. And enter 0 otherwise.
4. 'Enter rpm of propeller (integer)': If the user wants to output the Reynolds and Mach numbers, then he/she should also enter the rpm considered.
5. 'Plots ? (1: yes, 0: No)': The user should enter 1 if he/she wants to see the plots with the aerofoils and parameters. And enter 0 otherwise.

OUTPUTS – depending on the inputs, the algorithm returns in the 'output' folder

1. aerofoil_XX_.txt (with XX being the position r/R) – For each position of aerofoil given, the algorithm outputs the aerofoil points in the right order for xfoil.
2. blade_parameters.csv – The parameters of the blade: hub radius, tip radius, chord length and blade twist for each positions
3. aerodynamic_parameters.csv – Reynolds and Mach numbers for each position depending on the given rpm speed
4. plot_XX_.png (with XX being the position r/R) – The plots with the aerofoil points, blade twist and chord length for each positions

[1]It can also be found online on $https://github.com/PaulineMauryL/PropellerProject$

C Point selection algorithm

Algorithm 1 Select point around plane

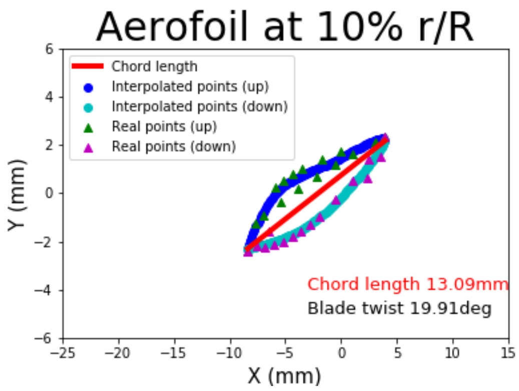
```
count = 0
$\delta$ = 0.05 mm
$\tau$ = 30
$\kappa$ = $\frac{width}{40}$
$aplane_{prev} \leftarrow aplane$
$bplane_{prev} \leftarrow bplane$
$aplane \leftarrow aplane + \delta$
$bplane \leftarrow bplane - \delta$
while number points taken $< \tau$ do
  count = count + 1
  if count > 100 then
    return
  end if
  for all points $i$ do
    ADD = true
    if between ($aplane_{prev}$ and $aplane$) OR between ($bplane_{prev}$ and $bplane$) then
      for all points already taken do
        if distance $< \kappa$ then
          ADD = false
        end if
      end for
      if ADD == true then
        add point to point taken
      end if
    end if
  end for
  $aplane_{prev} \leftarrow aplane$
  $bplane_{prev} \leftarrow bplane$
  $aplane \leftarrow aplane_{prev} + \delta$
  $bplane \leftarrow bplane_{prev} - \delta$
end while
```

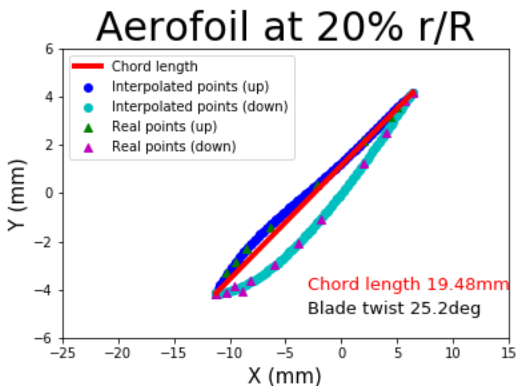
with $aplane$ standing for "above the plane" and $bplane$ standing for "below the plane".

D Chord length and Blade twist

pos% (r/R * 100)	**Blade Twist computed** $[deg]$	**Blade Twist real** $[deg]$	**c/R computed**	**c/R real**
20	25.19	23.90	19.483	0.132
25	24.44	24.65	22.818	0.156
30	23.69	24.11	24.142	0.176
35	22.23	22.78	25.172	0.193
40	20.00	21.01	26.319	0.206
45	17.58	19.00	27.448	0.216
50	16.37	17.06	27.416	0.223
55	28.07	15.33	28.070	0.226
60	27.33	13.82	27.330	0.225
65	26.66	12.51	26.658	0.219
70	25.32	11.36	25.316	0.210
75	24.05	10.27	24.047	0.197
80	21.85	9.32	21.850	0.179
85	19.45	8.36	19.453	0.157
90	16.30	6.15	12.043	0.130

TABLE D.1: Blade twist and Chord length





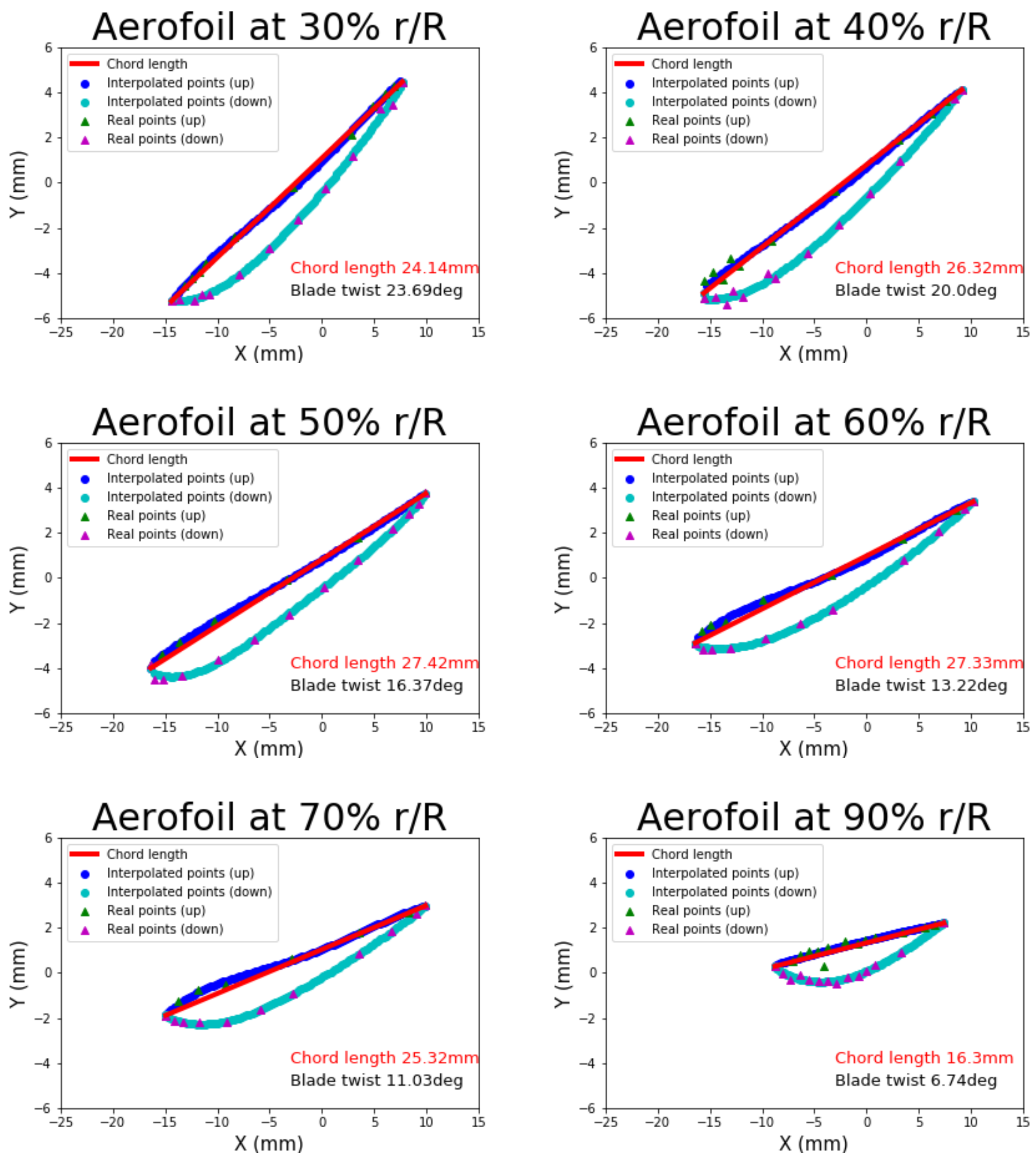


FIGURE D.2: Aerofoil for parameters computation

E Aerodynamic parameters

Here are the Reynolds and Mach number to enter in the X-foil software depending on the aerofoil position.

$pos\%$ **(r/R * 100)**	**r** $[mm]$	**Re**	**M**
10	12.68	1208	0.004
20	25.36	3597	0.008
30	38.042	6686	0.012
40	50.72	9719	0.016
50	63.40	12′655	0.020
60	76.08	15′138	0.024
70	88.76	16′360	0.028
80	101.44	16′137	0.032
90	114.13	13′544	0.036

TABLE E.1: Reynolds and Mach numbers

F Gantt Chart

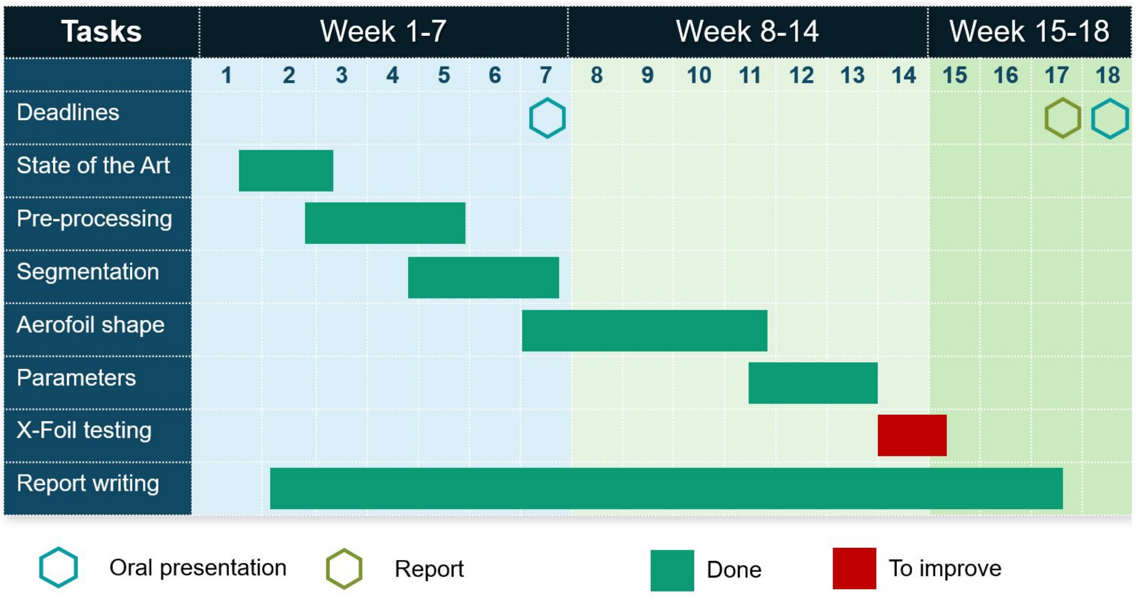


FIGURE F.1: Gantt Chart

Bibliography

[1] M. S. Kuester et al. “Wind Tunnel Testing of Airfoils for Wind Turbine Applications”. en. In: *Wind Engineering* 39.6 (Dec. 2015), pp. 651–660. ISSN: 0309-524X. DOI: 10.1260/0309-524X.39.6.651. URL: https://doi.org/10.1260/0309-524X.39.6.651 (visited on 01/01/2019).

[2] M. K. Rwigema. “Propeller Blade Element Momentum Theory With Vortex Wake Deflection”. en. In: (), p. 9.

[3] J. Cho and S.-c. Lee. “Propeller blade shape optimization for efficiency improvement”. In: *Computers & Fluids* 27.3 (Mar. 1998), pp. 407–419. ISSN: 0045-7930. DOI: 10.1016/S0045-7930(97)00035-2. URL: http://www.sciencedirect.com/science/article/pii/S0045793097000352 (visited on 01/01/2019).

[4] V. Carbone et al. “Combination of a Vision System and a Coordinate Measuring Machine for the Reverse Engineering of Freeform Surfaces”. en. In: *The International Journal of Advanced Manufacturing Technology* 17.4 (Jan. 2001), pp. 263–271. ISSN: 0268-3768, 1433-3015. DOI: 10.1007/s001700170179. URL: http://link.springer.com/10.1007/s001700170179 (visited on 12/19/2018).

[5] M. Daneshmand et al. “3D Scanning: A Comprehensive Survey”. In: *arXiv:1801.08863 [cs]* (Jan. 2018). arXiv: 1801.08863. URL: http://arxiv.org/abs/1801.08863 (visited on 09/21/2018).

[6] W. Boehler and A. Marbs. “3D SCANNING INSTRUMENTS”. en. In: (), p. 4.

[7] T. Luhmann. “Close range photogrammetry for industrial applications”. In: *ISPRS Journal of Photogrammetry and Remote Sensing*. ISPRS Centenary Celebration Issue 65.6 (Nov. 2010), pp. 558–569. ISSN: 0924-2716. DOI: 10.1016/j.isprsjprs.2010.06.003. URL: http://www.sciencedirect.com/science/article/pii/S0924271610000584 (visited on 12/19/2018).

[8] R. W. Derksen and T. Rogalsky. “Bezier-PARSEC: An optimized aerofoil parameterization for design”. In: *Advances in Engineering Software*. Advances in Structural Optimization 41.7 (July 2010), pp. 923–930. ISSN: 0965-9978. DOI: 10.1016/j.advengsoft.2010.05.002. URL: http://www.sciencedirect.com/science/article/pii/S0965997810000529 (visited on 10/03/2018).

[9] N. P. Salunke, J. A. R. A, and S. A. Channiwala. “Airfoil Parameterization Techniques: A Review”. en. In: *American Journal of Mechanical Engineering, American Journal of Mechanical Engineering* 2.4 (Jan. 2014), pp. 99–102. DOI: 10.12691/ajme-2-4-1. URL: http://pubs.sciepub.com/ajme/2/4/1/abstract.html (visited on 09/21/2018).

[10] A. Shahrokhi and A. Jahangirian. “Airfoil shape parameterization for optimum Navier–Stokes design with genetic algorithm”. en. In: *Aerospace Science and Technology* 11.6 (Sept. 2007), pp. 443–450. ISSN: 12709638. DOI: 10.1016/j.ast.2007.04.004. URL: http://linkinghub.elsevier.com/retrieve/pii/S1270963807000557 (visited on 09/23/2018).

[11] X. Lu et al. “An improved geometric parameter airfoil parameterization method”. In: *Aerospace Science and Technology* 78 (July 2018), pp. 241–247. ISSN: 1270-9638. DOI: 10.1016/j.ast.2018.04.025. URL: http://www.sciencedirect.com/science/article/pii/S1270963817304686 (visited on 09/21/2018).

[12] M. Drela. *XFoil Software*. URL: http://web.mit.edu/drela/Public/web/xfoil/ (visited on 12/19/2018).

[13] *E63 (4.25%) (e63-il)*. URL: http://airfoiltools.com/airfoil/details?airfoil=e63-il (visited on 12/20/2018).

[14] F. M. A and S. T. A. *Low Cost Reverse Engineering Techniques for 3d Modelling of Propellers*.

[15] M. Gupta et al. “A Practical Approach to 3D Scanning in the Presence of Interreflections, Subsurface Scattering and Defocus”. en. In: *International Journal of Computer Vision* 102.1-3 (Mar. 2013), pp. 33–55. ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-012-0554-3. URL: http://link.springer.com/10.1007/s11263-012-0554-3 (visited on 09/21/2018).

[16] E. P. Baltsavias. “A comparison between photogrammetry and laser scanning”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 54.2 (July 1999), pp. 83–94. ISSN: 0924-2716. DOI: 10.1016/S0924-2716(99)00014-3. URL: http://www.sciencedirect.com/science/article/pii/S0924271699000143 (visited on 12/19/2018).

[17] *CLARK Y AIRFOIL (clarky-il)*. URL: http://airfoiltools.com/airfoil/details?airfoil=clarky-il (visited on 12/20/2018).

[18] *Engineering*. en-US. URL: https://www.apcprop.com/technical-information/engineering/ (visited on 12/20/2018).