TORUS Geometry

Varun Unnithan

October 11, 2024

This document describes the mathematical approach to our generated parametric model for a toroidal propeller. A fully parametric propeller model will allow us to mesh the geometry at any resolution without having to interpolate, as well as defines a more convenient translation between input parameter spaces and geometric changes in a propeller. This document will outline how the cross-sectional 2D shape is defined, before explaining the remaining math that allows for the airfoil to be remapped and cast along a 3D curve to create the toroidal propeller blade.

1 Defining the 2D Airfoil Shape

The airfoil shape is defined using the NACA 4-digit series equations, which provide a standardized method for generating airfoil geometries based on parameters controlling camber and thickness.

1.1 NACA 4-Digit Airfoil Equations

The mean camber line $y_c(t)$ is defined piecewise:

$$y_c(t) = \begin{cases} \frac{m}{p^2}(2pt - t^2), & 0 \leq t \leq p \\ \frac{m}{(1-p)^2}[(1-2p) + 2pt - t^2], & p < t \leq 1 \end{cases}$$

The thickness distribution $y_t(t)$ is given by:

$$y_t(t) = 5 \times \text{thickness} \times \left[0.2969\sqrt{t} - 0.1260t - 0.3516t^2 + 0.2843t^3 - 0.1036t^4\right]$$

where:

- t is the normalized chord length parameter ($0 \leq t \leq 1$),
- m is the maximum camber,
- p is the location of maximum camber,
- thickness is the maximum thickness as a fraction of the chord.

1.2 Constructing Upper and Lower Surfaces

The coordinates of the upper and lower surfaces are calculated as:

$$\begin{aligned} x_u(t) &= t - y_t(t)\sin(\theta_c(t)) \\ y_u(t) &= y_c(t) + y_t(t)\cos(\theta_c(t)) \\ x_l(t) &= t + y_t(t)\sin(\theta_c(t)) \\ y_l(t) &= y_c(t) - y_t(t)\cos(\theta_c(t)) \end{aligned}$$

where $\theta_c(t)$ is the angle of the camber line slope:

$$\theta_c(t) = \arctan\left(\frac{dy_c}{dt}\right)$$

The NACA airfoil definition requires the thickness to be applied perpendicular to the camber line, requiring the above consideration. However, for the sake of simplicity and computational efficiency (particularly for the normalization of mesh sizes), $\theta_c(t)$ can be set to zero, assuming negligible camber line slope or that the model already represents a sufficient portion of the airfoil's geometry space without needing this consideration. When $\theta_c(t) = 0$, the thickness is just applied in the y-direction.

1.3 Parametrization Over t

The parameter t is divided into two intervals:

- $[0, 1)$ for the upper surface,
- $(1, 2]$ for the lower surface (shifted to maintain continuity).

The complete airfoil coordinates are defined using Heaviside functions:

$$x_{\text{airfoil}}(t) = x_u(t) \cdot H(1-t) + x_l(2-t) \cdot H(t-1)$$
$$y_{\text{airfoil}}(t) = y_u(t) \cdot H(1-t) + y_l(2-t) \cdot H(t-1)$$

where H is the Heaviside step function. This ensures that the airfoil function, $\mathbf{f}(t) = (x(t), y(t))$, is an injective function with respect to parameter t.

2 Transformations Along the Blade Centerline

To account for changes in blade orientation and cross-sectional size along the centerline, rotation and scaling transformations are applied as functions of the parameter s, which parametrizes the centerline curve.

2.1 Rotation Transformation

The rotation matrix $R(s)$ for an angle of attack $\alpha(s)$ is:

$$R(s) = \begin{bmatrix} \cos\alpha(s) & -\sin\alpha(s) \\ \sin\alpha(s) & \cos\alpha(s) \end{bmatrix}$$

The angle of attack $\alpha(s)$ can be defined as a polynomial function of s:

$$\alpha(s) = a_\alpha s^4 + b_\alpha s^3 + c_\alpha s^2 + d_\alpha s + e_\alpha$$

2.2 Scaling Transformation

Scaling factors $S_x(s)$ and $S_y(s)$ modify the airfoil dimensions:

$$S_x(s) = a_{sx} s^4 + b_{sx} s^3 + c_{sx} s^2 + d_{sx} s + e_{sx}$$
$$S_y(s) = a_{sy} s^4 + b_{sy} s^3 + c_{sy} s^2 + d_{sy} s + e_{sy}$$

These functions allow for parametric scaling along the blade's path, as they are parameterized by s, the parameter for the centerline of the toroidal blade. The two scaling functions scale the airfoil's x and y dimensions. Traditional propellers and other definitions of toroidal propellers may represent the airfoil transformations along the blade by performing rotations in all three axis. Instead, for the sake of simplicity, as well as being able to place every 2D airfoil perfectly perpendicular to the

centerline, the rotation about z (the yaw) is represented with the angle of attack function, and for roll and pitch rotations, the scaling functions are used as a proxy. This is because for any roll or pitch, you can represent it by its projection on the plane perpendicular to the centerline by simply scaling either the x or y dimension, respectively, by $cos(\phi)$, where ϕ is the angle to have been rotated by in that axis.

2.3 Applying Transformations

The transformed airfoil coordinates are:

$$\begin{bmatrix} X_{\text{rotated}}(s,t) \\ Y_{\text{rotated}}(s,t) \end{bmatrix} = R(s) \cdot \begin{bmatrix} x_{\text{airfoil}}(t) \\ y_{\text{airfoil}}(t) \end{bmatrix}$$

Scaling is applied as:

$$X_{\text{scaled}}(s,t) = S_x(s) \cdot X_{\text{rotated}}(s,t)$$
$$Y_{\text{scaled}}(s,t) = S_y(s) \cdot Y_{\text{rotated}}(s,t)$$

2.4 Function Space Analysis for Transformation Parameters

In designing the transformations for angle of attack, x-scale, and y-scale along the blade centerline, it is essential to select function forms that can model a wide variety of possible transformation behaviors while minimizing the number of parameters required. This section details the analysis performed to determine suitable function forms that can approximate a large function space relevant to airfoil transformations in toroidal propellers.

2.4.1 Objective

The goal is to identify a function form $f(s)$ that:

- Can approximate a wide range of possible transformation behaviors along the blade centerline.
- Uses a minimal number of parameters to reduce computational complexity and overfitting risks.

To achieve this, a Monte Carlo simulation was performed where randomly generated functions, representing possible transformation behaviors, were fitted using candidate function forms. The performance of each candidate function form was evaluated based on its ability to fit the randomly generated functions.

2.4.2 Random Function Generators

A set of random function generators that produce diverse function behaviors was defined to model a sample space that a function may try to fit. Here, $X \sim \mathcal{U}(a,b)$ represents a parameter X randomly sampled from a uniform distribution with lower and upper bounds of a and b, respectively.

1. **Random Sinusoidal Function**: $f_{\sin}(s) = A\sin(2\pi Fs + \phi)$ where $A \sim \mathcal{U}(0.5, 2)$ is the amplitude, $F \sim \mathcal{U}(1,3)$ is the frequency, and $\phi \sim \mathcal{U}(0,\pi)$ is the phase shift.
2. **Random Polynomial Function**:

$$f_{\text{poly}}(s) = \sum_{i=0}^{d} c_i s^i$$

where $d \sim \mathcal{U}\{1,8\}$ is the degree, and $c_i \sim \mathcal{U}(-2,2)$ are the coefficients.

3. **Random Logistic Function**:

$$f_{\text{logistic}}(s) = \frac{L}{1 + e^{-k(s-s_0)}}$$

where $L \sim \mathcal{U}(1, 50)$ is the carrying capacity, $k \sim \mathcal{U}(8, 100)$ is the growth rate, and $s_0 \sim \mathcal{U}(0.1, 0.9)$ is the midpoint.

4. **Random Piecewise Linear Function**:

$$f_{\text{interp}}(s) = \text{Interpolate}\,\{(s_i, y_i)\}$$

where s_i are n' points uniformly sampled in $[0, 1]$ and $y_i \sim \mathcal{U}(-1, 1)$. To avoid excessively chaotic behavior, the points taken from $y_i \sim \mathcal{U}(-1, 1)$ is downsampled in size so as to have some continuity across s. What this means is that regardless of how many sample data points are being generated, $n' \sim \mathcal{U}(3, 8)$, with the rest of the data points being the result of linear interpolations between those sampled data points.

These function generators were chosen to represent behaviors such as cyclic patterns, sharp transitions, and random fluctuations, which are characteristic of possible airfoil transformations along a toroidal blade.

2.4.3 Candidate Fitting Functions

The following function forms were considered to fit the randomly generated data:

1. **4th-Degree Polynomial**:

$$f_{\text{poly4}}(s) = a_4 s^4 + a_3 s^3 + a_2 s^2 + a_1 s + a_0$$

2. **6th-Degree Polynomial**:

$$f_{\text{poly6}}(s) = a_6 s^6 + a_5 s^5 + a_4 s^4 + a_3 s^3 + a_2 s^2 + a_1 s + a_0$$

3. **Fourier Series (Order 2)**:

$$f_{\text{Fourier2}}(s) = A_0 + \sum_{k=1}^{2} [A_k \cos(2\pi k s) + B_k \sin(2\pi k s)]$$

4. **Fourier Series (Order 3)**:

$$f_{\text{Fourier3}}(s) = A_0 + \sum_{k=1}^{3} [A_k \cos(2\pi k s) + B_k \sin(2\pi k s)]$$

5. **Sinusoidal Function**:

$$f_{\sin}(s) = A \sin(Bs + C) + D$$

These functions were chosen to represent different levels of complexity and flexibility, with varying numbers of degrees of freedom. Thus, only fitting functions with similar degrees of freedom will be compared, with dimensionality being a external factor to consider (4th-Degree Polynomial vs Order 2 Fourier Series and 6th-Degree Polynomial vs Order 3 Fourier Series). The sinusoidal function is fitted as a reference.

2.4.4 Fitting Procedure

For a randomly generated function $f_{\text{random}}(s)$:

1. Generate n data points $\{(s_i, y_i)\}$ where s_i are uniformly sampled in $[0, 1]$ and $y_i = f_{\text{random}}(s_i)$.
2. Fit each candidate function $f_{\text{candidate}}(s)$ to the data using least squares optimization to minimize the mean squared error (MSE):

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} [y_i - f_{\text{candidate}}(s_i)]^2$$

3. Record the MSE and fitting parameters for each candidate function.

This process was repeated for a large number of iterations ($N = 10000$) to sample a wide variety of random functions.

2.5 Results and Analysis

The average MSE for each candidate function over all iterations is summarized below:

Candidate Function	**Mean Error (MSE)**	**Number of Parameters**
4th-Degree Polynomial	2.064	5
6th-Degree Polynomial	0.845	7
Fourier Series (Order 2)	2.441	5
Fourier Series (Order 3)	1.193	7
Sinusoidal Function	5.748	4

Table 1: Mean squared error (MSE) for candidate fitting functions over 10,000 iterations.

2.5.1 Interpretation

- The **6th-degree polynomial** achieved the lowest average MSE, indicating a superior ability to fit the random functions, at the cost of increased complexity (7 parameters).
- The **4th-degree polynomial** had a moderate MSE while using fewer parameters (5 parameters), striking a balance between flexibility and simplicity.
- The **Fourier series** functions, while theoretically capable of modeling periodic behaviors, performed worse than the polynomials in this context, possibly due to the non-periodic nature of some random functions generated.
- The **sinusoidal function** had the highest MSE, suggesting it is insufficiently flexible to model the diverse function space considered.

Considering the trade-off between model complexity and fitting accuracy, the **4th-degree polynomial** was somewhat holistically selected for modeling the transformation functions. It provides adequate flexibility to approximate a wide range of behaviors with a manageable number of parameters. Higher-degree polynomials or functions with more parameters could lead to overfitting and increased computational burden without substantial gains in modeling capability for the intended application.

3 Generating the Centerline Curve with Splines

A Non-Uniform Rational B-Spline (NURBS) curve is used to define the blade's centerline due to its flexibility and precision in modeling complex natural shapes.

3.1 Coordinate System and Control Point Definition

The generation of the blade centerline begins by dividing the lateral face of the cylindrical hub into n equal sections, where n corresponds to the number of blades. For a section, a local 2D coordinate system is established:

- The positive x-axis ($+x$) extends horizontally across the section.
- The positive y-axis ($+y$) extends vertically downward towards the bottom of the cylinder.

This coordinate system facilitates the placement and manipulation of control points within each blade's sector. The origin, $(0, 0)$ is placed 1 unit down from the top left of the section. This leaves some padding so that the blade isn't intersecting the top of the cylindrical hub. This 2D coordinate system can be extended to 3D by adding a z-axis, creating a coordinate system denoted L.

- The positive z-axis ($+z$) extends radially out from the cylinder. For any point (x, y), the scaled z-vector will always normal to the lateral cylinder face.

While extending the z-axis radially creates a non-Euclidean coordinate frame, it creates a natural relation between the z-coordinate and the radial distance of control points, which can help with optimization.

3.2 Centerline Inset and Hub Intersection

To facilitate the boolean union operation between the blade and the hub, the centerline is inset slightly into the hub's volume. This is achieved by setting:

$$R_{\text{inset}} = R_{\text{hub}} - \frac{R_{\text{hub}}}{8}$$

This inset ensures that the generated blade intersects with the hub, allowing for a seamless geometric union that results in a single, manifold mesh suitable for simulation and analysis.

3.3 Control Points Placement

A cubic NURBS spline requires control points that define the curve's shape. For each blade, four control points are defined:

- P_1: $(0, 0, 0)_L$
- P_2: $(x_2, y_2, z_2)_L$
- P_3: $(x_3, y_3, z_3)_L$
- P_4: $(x_4, y_4, 0)_L$

1. **First Control Point** P_1: The coordinates of P_1 in the global coordinate system are:

$$P_1 = \left(R_{\text{inset}},\, 0,\, \frac{L_{\text{hub}}}{2} - 1 \right)$$

   This is taken such that the local origin for a blade's lateral face section lies on the global x-axis. Thus the coordinate for P_1 in the global cylindrical and Cartesian frame are the same numerically.

2. **Local to Global Coordinate Transformation**: Let C be the global cylindrical frame, represented by coordinates $(r, \theta, z)_C$. Coordinate frame L is simply a remapping of this cylindrical frame, where change in x_L is the same as a change in θ_C, a change in y_L is the same as a change in $-z_C$, and a change in z_L is the same as a change in r_C. Thus, the change of basis matrix becomes:

$$\mathbf{M} = \begin{bmatrix} \mathbf{e}_{x_L} & \mathbf{e}_{y_L} & \mathbf{e}_{z_L} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

Therefore, for local control point P_i, its relative global cylindrical coordinate is found with:

$$\mathbf{P_i}_{C'} = \mathbf{M} \cdot \mathbf{P_i}_{L}$$

To get the global coordinates for control point P_i, the offset of origins needs to be considered, so adding the origin vector, which is just $\mathbf{P_1}$, will give the absolute coordinates.

$$\mathbf{P_i}_{C} = \mathbf{P_i}_{C'} + \mathbf{P_1}_{C}$$

3.4 NURBS Curve Construction

With the control points defined, the cubic NURBS spline $C(s)$ is constructed as:

$$C(s) = \frac{\sum_{i=0}^{3} N_{i,3}(s) w_i P_i}{\sum_{i=0}^{3} N_{i,3}(s) w_i}$$

where:

- $N_{i,3}(s)$ are the cubic B-spline basis functions,
- w_i are the weights associated with each control point. For this case, so as to reduce the parameters needed for optimization, the weights are fixed with $w_i = 1$
- s is the normalized parameter along the spline ($0 \leq s \leq 1$).

The knot vector $\{u_i\}$ for a cubic NURBS with clamped ends is:

$$\{u_i\} = \{0, 0, 0, 0, 1, 1, 1, 1\}$$

3.5 Spline Fitting and Symmetry Considerations

The cubic NURBS spline provides the necessary flexibility to define smooth and continuous centerline curves for each blade while maintaining rotational symmetry across the hub. By defining control points in a non-Euclidean coordinate space, where the z-axis is radial with respect to the hub, the resulting centerline ensures that each airfoil section remains perpendicular to the centerline.

The spline fitting process involves adjusting the control points P_1 and P_2 to achieve the desired curvature and alignment of the blade. By manipulating these control points, complex blade geometries can be accurately modeled with a minimal number of parameters.

4 Extruding the Airfoil Using Frenet-Serret Frames

The Frenet-Serret frame provides an orthonormal basis at each point along the curve, allowing the airfoil to be correctly oriented perpendicular to the centerline. This is a relative path coordinate system that moves along the centerline to orient and define the airfoil through.

4.1 Calculating the Frenet-Serret Frame

At each point s along the curve $C(s)$:

1. **Tangent Vector** $T(s)$:

$$T(s) = \frac{C'(s)}{\|C'(s)\|}$$

2. **Normal Vector** $N(s)$:

$$N(s) = \frac{T'(s)}{\|T'(s)\|}$$

3. **Binormal Vector** $B(s)$:

$$B(s) = T(s) \times N(s)$$

4.2 Positioning the Airfoil

The final 3D coordinates of the airfoil are:

$$\begin{aligned} X_{\text{final}}(s,t) &= X_c(s) + X_{\text{scaled}}(s,t)N_x(s) + Y_{\text{scaled}}(s,t)B_x(s) \\ Y_{\text{final}}(s,t) &= Y_c(s) + X_{\text{scaled}}(s,t)N_y(s) + Y_{\text{scaled}}(s,t)B_y(s) \\ Z_{\text{final}}(s,t) &= Z_c(s) + X_{\text{scaled}}(s,t)N_z(s) + Y_{\text{scaled}}(s,t)B_z(s) \end{aligned}$$

where (X_c, Y_c, Z_c) are the coordinates of the centerline $C(s)$, and (N_x, N_y, N_z), (B_x, B_y, B_z) are components of $N(s)$ and $B(s)$.

5 Generating Evenly Spaced Mesh Points

Using a linear spacing for the airfoil parameter, t, yields a mesh with uneven cell sizes. Since

$$\frac{\mathrm{d}^2\mathbf{f}}{\mathrm{d}t^2} \neq 0 \text{ where } \mathbf{f}(t) = (x(t), y(t))$$

and $\mathbf{f}$ is the vector airfoil function, the derivative of $\mathbf{f}$ changes along the airfoil's surface. This means that an even linearly sampled set for the input will have the output's spacing be inconsistent. This would cause the outputted mesh to have differential cell sizes, with the leading edge, or regions where $|\frac{\mathrm{d}\mathbf{f}}{\mathrm{d}t}|$ is the largest, having larger cells compared to the trailing edge.

Uniformly distributed mesh points along the airfoil perimeter ensure consistent cell sizes in the final mesh, improving CFD and meshing accuracy. The goal is to find some set or vector of parameter values, $\mathbf{t}$, that when plugged into the symbolic equations will create even cell sizes in the mesh. Even sizes of meshes are determined by covering the same arc distance along the airfoil.

5.1 Computing Arc Length Parameterization

The arc length $s(t)$ along the airfoil is:

$$s(t) = \int_{t_0}^{t} \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}\, dt$$

Due to the complexity of the NACA equations, numerical integration is employed.

5.2 Generating t Values Corresponding to Equal Arc Length Increments

1. **Total Arc Length** S_{total}:

$$S_{\text{total}} = s(t_{\text{end}})$$

2. **Arc Length Increment** Δs:

$$\Delta s = \frac{S_{\text{total}}}{n}$$

where n is the desired number of intervals (mesh resolution).

3. **Iterative Calculation of t Values**:

   Starting from t_0, solve for t_i such that:

   $$s(t_i) - s(t_{i-1}) = \Delta s$$

   This is not analytically solvable, so numerical methods are used to approximate the solution. This is done using a resolution of n_{fine} iterations over the entire airfoil edge, however this can be refined further to be more accurate, albeit with non-negligible computational costs. This integration at each iteration is done using the technique from the QUADPACK Fortran library, called by a SciPy proxy function. It uses adaptive Gaussian quadrature methods to approximate the integral.

4. **Generate Mapping Function** $t(s)$:

   A linear interpolation between the n_{fine} (by default 1000) integrated data points yields a function $t(s)$, which when inputted the current cumulative arc distance, outputs the t value that would produce the current coordinate point. Evaluating

   $$t(\bar{\mathbf{s}}) = \mathbf{t}$$

   where

   $$\bar{\mathbf{s}} = \{s_i = i \times \Delta s | i \in [0, n]\}$$

   yields the vector $\mathbf{t}$, whose elements are t-values that will produce even mesh spacing.

6 Assembling the Complete Propeller

6.1 Duplicating Blades

The blade geometry is duplicated n times (as specified by the number of blades parameter), rotated evenly around the hub's axis:

$$\theta_i = \theta_0 + \frac{2\pi i}{n}$$

6.2 Generating the Hub

The hub is modeled as a cylinder with radius R_{hub} and length L_{hub}. A parametric representation in cylindrical coordinates is used:

$$\begin{aligned} X_{\text{hub}}(\theta, z) &= R_{\text{hub}} \cos\theta \\ Y_{\text{hub}}(\theta, z) &= R_{\text{hub}} \sin\theta \\ Z_{\text{hub}}(\theta, z) &= z \end{aligned}$$

7 Mesh Generation and Boolean Operations

7.1 Creating Mesh Faces

Using the computed coordinates, mesh faces are generated by connecting adjacent points in the parameter grids. This is connected in an order such that the cross product for the normals of the face will point outwards.

7.2 Triangulation

At this point, there are $n + 1$ bodies, where n corresponds to the number of blades and the extra body is the hub. In order to merge them into a contiguous mesh, PyVista is used. This library requires bodies to first be triangulated, which is done by simply cutting each rectangular mesh face along the diagonal.

7.3 Geometric Union

The blades and hub are combined into a single mesh using boolean union operations in PyVista, resulting in a manifold mesh suitable for simulations.

8 Appendix: Mathematical Expressions and Parameters

Default parameters for now

8.1 Airfoil Parameters

- Maximum camber $m = 0.02$
- Location of maximum camber $p = 0.4$
- Maximum thickness thickness $= 0.5$

8.2 Transformation Coefficients

- Angle of attack coefficients $a_\alpha, b_\alpha, c_\alpha, d_\alpha, e_\alpha$
- Scaling coefficients $a_{sx}, b_{sx}, c_{sx}, d_{sx}, e_{sx}$ and $a_{sy}, b_{sy}, c_{sy}, d_{sy}, e_{sy}$

8.3 NURBS Parameters

- Degree $p = 3$
- Knot vector $\{0, 0, 0, 0, 1, 1, 1, 1\}$
- Control points P_i
- Weights $w_i = 1$ (uniform)