AIAA AVIATION Forum
5-9 June 2017, Denver, Colorado
18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference

10.2514/6.2017-3327

Component-based Geometry Manipulation for Aerodynamic Shape Optimization with Overset Meshes

Ney R. Secco *
John P. Jasa *
Gaetan K. W. Kenway †
Joaquim R. R. A. Martins ‡
University of Michigan, Ann Arbor, Michigan, United States

Mesh generation for high-fidelity CFD simulation and aerodynamic shape optimization is a time-consuming task. We can model complex geometries accurately using overset meshes where multiple high-quality structured meshes corresponding to different aircraft components overlap to model the full aircraft configuration. Nevertheless, from the geometry manipulation standpoint, most methods operate on the entire geometry rather than on each component, which diminishes the advantages of overset meshes. To address this issue, we introduce a geometry module that operates on individual components and automatically computes their intersections to automate the overset mesh updates during optimization. This method is also differentiated to compute derivatives with respect to component-based design variables and is integrated within an optimization framework. Using these automatically updated meshes and the corresponding derivatives, we perform aerodynamic shape optimization including the wing-body intersection for the DLR-F6 geometry and achieve a reduction of 15 drag counts (5%) compared to the baseline design.

I. Introduction

With stricter environmental regulations, increased air traffic, and thinning profit margins, airlines and aircraft manufacturers are looking to make the next generation of aircraft as efficient as possible. Several novel aircraft concepts such as the blended wing body (BWB) [1], truss-braced wing (TBW) [2], double-bubble layout [3], and joined wings [4] are under investigation. An airplane configuration can be seen as a group of individual components, such as lifting surfaces, bodies, fairings, and engines; this reflects on how we parametrize the shape of the airplane. Furthermore, from the aerodynamic perspective, junctions between these components cause interference drag due to the appearance of different flow features such as separation bubbles. The interference drag depends not only on the overall flow condition, but also on the detailed shape of the junction.

Low-fidelity tools such as lifting line theory and panel codes can capture general trends associated with individual aircraft components, but they do not model enough of the necessary physics to accurately resolve the junction flow. Surrogate models for interference drag [5, 6] can be coupled to low-fidelity tools as an approach for preliminary aircraft design [7]. However, they are limited in the number of design parameters that can be used for the training, and the predictions are only valid within the training domain of the surrogate model. Therefore high-fidelity computational fluid dynamics (CFD) is essential for detailed analysis and optimization of junction geometries.

Another motivation to use high-fidelity models is that we want to design these aircraft quickly without extensive wind tunnel testing or other costly evaluation methods, while maintaining an acceptable level of accuracy. Thus, computational simulations have taken hold as the main design and analysis tools used in aircraft design. It is challenging to obtain the correct level of detail in the simulations and to model the physics well within a reasonable time frame. There is a large initial effort to produce realistic models and analyze them accurately. However, once these computational models and analyses are established, several design can be evaluated at comparatively low cost. The design space for detailed aerodynamic shape description is broad since it requires a high number of variables to parametrize the aircraft

*Ph.D. Candidate, Department of Aerospace Engineering, AIAA Student Member
†Research Investigator, Department of Aerospace Engineering
‡Professor, Department of Aerospace Engineering, AIAA Associate Fellow

geometry. Thus, gradient-based optimization techniques are necessary to intelligently search the design space for the optimum configuration, and gradients can be efficiently computed via the adjoint method [8].

The use of high-fidelity tools for aerodynamic analysis and optimization requires the discretization of the domain using meshes. Rapidly creating meshes for complex geometries for computational fluid dynamics (CFD) simulations has historically been challenging.

Multi-block meshes work well for simple geometries but are cumbersome for models with multiple complex features, such as the truss-braced wing. Previously, we used a CFD solver based on structured multiblock meshes to optimize this type of configuration, but this limited possible changes in the wing-truss junction geometry due to difficulties in generating and warping the cells in the concave regions around these junctions [2]. A recurring issue is that surface deformations frequently generated negative volume cells after the mesh warping procedures.

The use of overset meshes [9] is a promising approach to overcome this issue. This methodology mitigates these mesh-related problems because we can independently generate high-quality meshes for each component. These dedicated meshes can also be automatically generated using, for instance, hyperbolic mesh generation algorithms [10, 11]. It is also necessary to create meshes specifically for each junction, called *collar meshes*, that allow information to be passed between the primary component meshes [12]. Besides easier mesh generation, another advantage of the overset approach is that we can still make use of high-quality and memory-efficient structured meshes, since overset meshes are composed of multiple independent structured blocks. Because of this, overset CFD may excel in optimizations where the geometry components shift a large amount, such as optimizing the location of the wing-body intersection for a conventional airplane, or when independently modifying components during an optimization.

On the other hand, these advantages may be hampered if the geometry manipulation method used in the optimization is limited. Many geometry creation tools exist [13, 14, 15, 16], but most do not have a seamless integration between geometry description and mesh generation. Furthermore, these tools might not provide derivatives of the geometry description with respect to the design variables, making them unfeasible for gradient-based optimization. An alternative type of geometry manipulation method used in aircraft design optimization parametrizes shape perturbation rather than the shape itself, such as the free-form deformation (FFD) approach [17, 18]. This approach not only reduces the number of design variables [19] but also allows the use of externally provided geometries in native formats. Nevertheless, these modules operate on the entire geometry at once, which ignores the valuable component subdivision information. For instance, in a wing-body configuration, it is not possible to translate the wing without deforming the fuselage shape.

The goal of this work is to develop a geometry manipulation approach that allows independent parametrization of shape deformations at both the individual component and entire aircraft levels while taking into account intersections among these components. The approach is implemented in a tool called *pySurf* that uses unstructured discrete surfaces to describe each primary component as inputs. It then uses these surfaces to compute intersections between components and regenerate collar meshes automatically at each optimization iteration. Derivatives of mesh points with respect to shape deformation design variables are also computed using this tool, enabling gradient-based aerodynamic shape optimization.

To demonstrate the developed tool we perform high-fidelity aerodynamic shape optimization of a wing-body intersection for a conventional aircraft. An overset solver that solves the RANS equations is used for aerodynamic analysis. We demonstrate the feasibility of this geometry manipulation methodology and also give new insights regarding wing-body junction design.

II. Methodology

As mentioned in the previous section, we aim to extend a component-based approach used in CFD solvers to geometry manipulation within the optimization framework to fully take advantage of the overset approach. Consequently, we also need to dynamically handle junctions among these primary components during optimization. This is because we need collar meshes to adequately model the junction area between two components.

Automatically producing a collar mesh between two components is a multifaceted challenge: we must find the intersection curve, march a surface mesh outward on the surfaces of both components, and project this mesh back onto the surfaces during the marching process.

To enable gradient-based aerodynamic shape optimization, we need the derivatives of the functions of interest with respect to the design variables. For this purpose, the entire process of finding the intersection curve and producing

a collar mesh must be differentiated. We use automatic differentiation (AD) to efficiently compute the derivatives of the generated mesh points with respect to the design variables. Further details regarding the derivative computation process are included in Sec. II-E.

A. Overall architecture

To address the difficulty of automatically creating collar meshes, we develop a new geometry module called pySurf which performs the following sequence of operations:

1. import primary components individually to represent the full vehicle geometry.
2. apply geometric changes on each primary component separately, based on their own design variables.
3. compute intersections between these multiple primary components.
4. automatically generate collar meshes for these intersections. These meshes should be structured to be compatible with the CFD solver used in this work.
5. compute derivatives of the functions of interest with respect to the primary components' design variables.

These operations are represented in Fig. 1. First we focus on the forward execution of the code (top to bottom). The user should provide discrete surface representations (composed of triangles and quads), structured surface meshes, and FFD boxes for each primary component (such as a wing or fuselage).

B. Surface deformation: FFD

For each optimization iteration, pySurf receives the design variables from the optimizer. Then it uses the individual FFD boxes [18] to update the triangulated surface nodes and the structured surface meshes of each component based on these high-level design variables.

Each FFD box can be seen as a structured volume mesh that encompasses the surface that we want to modify. The nodes of the FFD blocks act as control points: moving a node from the FFD block warps the embedded surface. It is important to embed both surface descriptions of a primary component in the same FFD so that they have consistent displacements, otherwise the collar mesh and the component volume mesh are modeling different geometries.

The FFD deformation process for each component is independent of any other component, since we also have separate overset meshes for each component. This gives the designer and optimizer freedom to control components' variations individually.

C. Intersection computation

Once the triangulated surface is updated, we use highly efficient intersection algorithms with alternating digital tree (ADT) searches [20] to compute intersection curves between primary components. The ADT groups discrete elements based on their spatial location so we can efficiently find which pairs of elements likely intersect. We then use fast pairwise triangle-triangle intersection algorithms [21] on these filtered elements to identify intersections.

Specifically, we create rectangular bounding boxes for each primary component based on the minimum and maximum coordinates in the x, y, and z directions. We then find the union of these bounding boxes, which is a smaller rectangular volume. Next, we determine which triangle elements are within this smaller box and construct an ADT based on only these triangles. We use the ADT to find triangles that possibly intersect. Finally, we use a pairwise triangle-triangle intersection algorithm [21] on the candidates given by the ADT search to compute the intersection curve(s) of the primary components. These intersection curves can then be used as starting curves for the hyperbolic surface mesh generation detailed in the next section.

An arbitrary number of primary components can be intersected using this method. Each pairwise component intersection may have multiple intersection curves and each is generated and stored for the user to select as possible surface mesh starting curves.

We also added features dedicated to the manipulation of these intersection curves, such as merging, splitting, and remeshing, so that the user can accurately control the topology and the number of nodes used to describe the intersection, since this curve is the starting point for the hyperbolic surface marching.

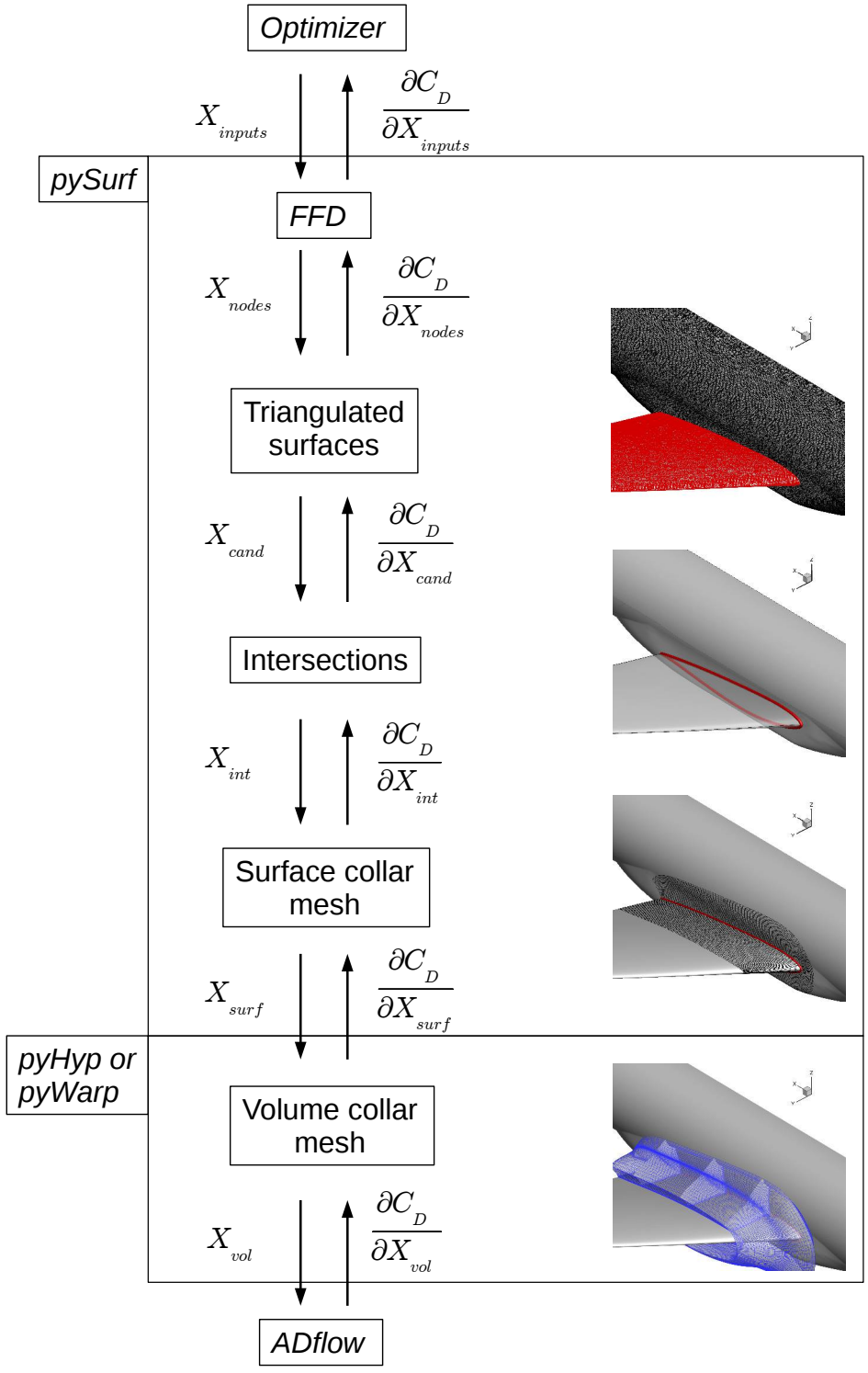


Figure 1. Data passing scheme for pySurf. The top-to-bottom arrows illustrate the forward mode execution of the optimization framework, whereas the bottom-to-top arrows indicate the reverse mode automatic differentiation used to backpropagate derivatives. Here we are interested in C_D, but any output could be substituted in its place.

D. Hyperbolic mesh generation

Having identified the intersection curves, we can now use established hyperbolic mesh-marching algorithms [10] to produce the surface collar mesh. This type of mesh generation algorithm starts from a baseline curve and then uses a marching scheme to generate the next layer of the surface mesh. This process is repeated until the desired number of cells and mesh extension is reached. One advantage of hyperbolic marching schemes over other mesh generation methods (such as transfinite interpolation [22]) is that they do not require the outer boundary of the mesh domain to be defined, making the process easier to automate, especially for collar meshes.

We added features to the standard scheme to improve control over the mesh generation. For instance, the user can choose to preserve special surface features, such as trailing edge corners, as the surface mesh is marched. In this case, we identify which node from the baseline curve is closest to the guide curve and then we project this node onto the guide curve. In addition, we locally modify the marching equations for this node so that it marches in a direction tangent to the guide curve.

We also added a feature to preserve the relative node spacings from the baseline curve throughout the entire mesh. This feature works as follows: after every new layer is generated, we redistribute the nodes of this new layer using the same relative arc-lengths of the nodes in the baseline curve. This is important when generating meshes near trailing edges, since the artificial dissipation associated with the marching scheme tries to smooth the nodal intervals near the trailing edge despite the local high refinement.

A comparison of meshes generated without and with these geometry-preserving options is shown in Fig 2. In the left image, we simply extrude the mesh without considering any geometric features. In the middle image, we set the upper and lower edge of the blunt trailing edge as guide curves. This forces the nearest point of the extruded mesh layer to coincide with the guide curve, which preserves the edge geometry. The bunched spacing at the leading and trailing edges dissipates as we march away from the intersection curve. In the right image, we require the relative spacing between the mesh points in the extruded layer to be the same as in the previous layer along with the previous guide curves. This produces a high-quality surface mesh appropriate for CFD.

During mesh marching, we must be able to project the mesh points back onto the components' surfaces so the mesh obeys the prescribed geometry. For discrete surfaces, finding the nearest surface triangle by brute force would not be tractable, especially when we must regenerate the mesh for each optimization iteration. Thus, we use the ADT algorithm once again to efficiently compute the projection of the mesh point onto the nearest surface element.

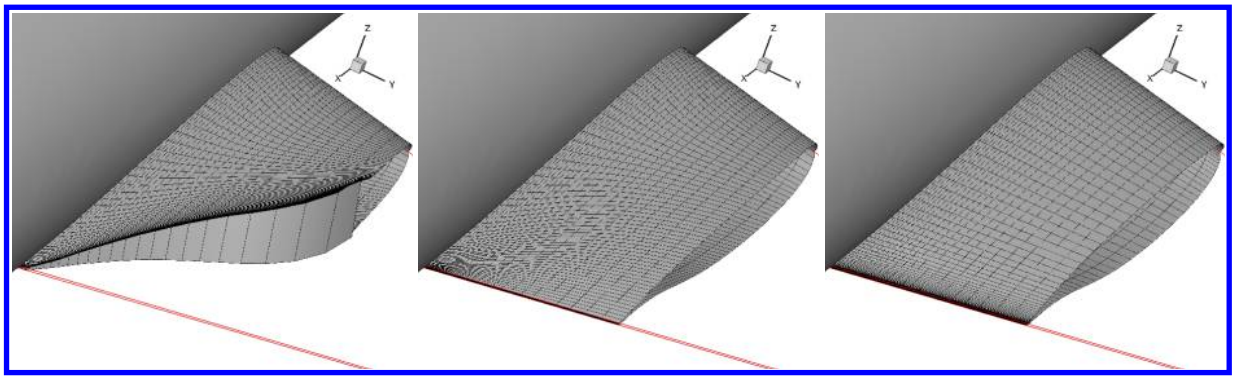


Figure 2. From left to right: the portion of the collar mesh on the wing extruded with no geometry feature information, the wing mesh extruded with trailing edges (red) set as guide curves, and the wing mesh extruded with trailing edge guide curves and relative spacing retention for each marched layer. Only the rightmost mesh has acceptable quality.

E. Automatic Differentiation

We need to compute accurate derivatives of all processes within pySurf efficiently to enable effective gradient-based aerodynamic shape optimization. The overall framework uses reverse mode AD for this purpose, which we discuss in detail in Sec. IV-A. Therefore, pySurf also uses the same AD method to internally backpropagate derivatives. The backpropagation is illustrated by the bottom-to-top arrows of Fig. 1.

The automatically differentiated code for pySurf was generated using Tapenade [23], but we had to selectively alter the differentiation process in each pySurf module to get an efficient final code. For instance, we store the results of the ADT searches used by the intersection computation outside of the differentiated code since we already know which elements intersect from the forward execution of the program. The same applies to the projections done during the hyperbolic mesh generation because we already know the location of the projections from the forward execution. Therefore, there is no need to differentiate the ADT search procedure itself. A similar approach can be applied to the linear system solutions inside the hyperbolic marching codes, since we already know the solutions for these problems from the forward execution.

In Sec. II-C we mentioned that pySurf has several tools to control the intersection curves. During forward execution, pySurf stores all the intermediary steps required by the user to generate the appropriate curve for hyperbolic marching, then it uses the same steps, but in reverse order, to perform the reverse propagation of derivatives.

Additionally, each layer of the hyperbolic marched mesh depends on the previous layer, so we use the chain rule to propagate the reverse mode derivatives, which requires storing information from all intermediate layers during the initial forward march.

III. Computational tools

pySurf on its own is just a geometry module, so it needs to be coupled with other analysis tools and optimizers to provide a fully capable aircraft design optimization framework. High-fidelity aircraft design optimization requires a multidisciplinary framework that should be computationally efficient and massively parallelized due to the expensive cost of each analysis. Furthermore, gradient-based optimization is necessary due to the high number of design variables [8]. Thus, this framework should also be capable of computing first order derivatives of both objective and constraint functions with respect to each design variable.

The MACH framework (MDO of Aircraft Configurations with High fidelity) [24] is developed with these needs in mind. This framework uses Fortran and C++ routines wrapped in a Python interface to perform efficient high-fidelity aerostructural optimization. For this work specifically, we only use modules relevant to aerodynamic shape optimization, which include a geometry modeler (pyGeo), volume mesh generator (pyHyp), mesh warper (pyWarp), aerodynamic solver (ADflow), Optimizer (SNOPT), and the newly added pySurf module as the collar mesh generator. These modules are described in detail in the next sections.

The framework is shown in Fig. 3. To use this framework, the user should provide for each primary component:

structured surface mesh with the desired resolution for the CFD analysis. This surface mesh is used to generate the volume mesh with hyperbolic extrusion methods.

unstructured discrete (triangulated) surface mesh that is used as reference for intersection computation and collar mesh generation.

FFD block that contains the surfaces described above and whose control points give the necessary shape control resolution for the optimization problem.

Here we should clarify why we need two descriptions—one structured and the other unstructured—for each primary component. The structured mesh of the primary components has the necessary surface resolution required by the CFD solver, which is already an approximated description of the underlying continuous surface representation. pySurf cannot use the same resolution to march new hyperbolic surface meshes for the collars because this would cause an additional loss of information compared to the original continuous representation. Therefore, the user also needs to provide another surface representation that is finer than the CFD solver requirements so that the hyperbolic marching algorithm has a more accurate description of the surface geometry compared to the primary structured mesh. Since this finer surface mesh is not actually used by the CFD solver, it can be unstructured. The use of triangulated surfaces for automatic mesh generation is recognized as a possible practice for automatic mesh generation [25], and it can be faster than healing and using analytic representations, such as the ones used by CAD tools [12].

Now we discuss aspects of each module in the optimization framework. Further details regarding the framework are described by Kenway et al. [24].

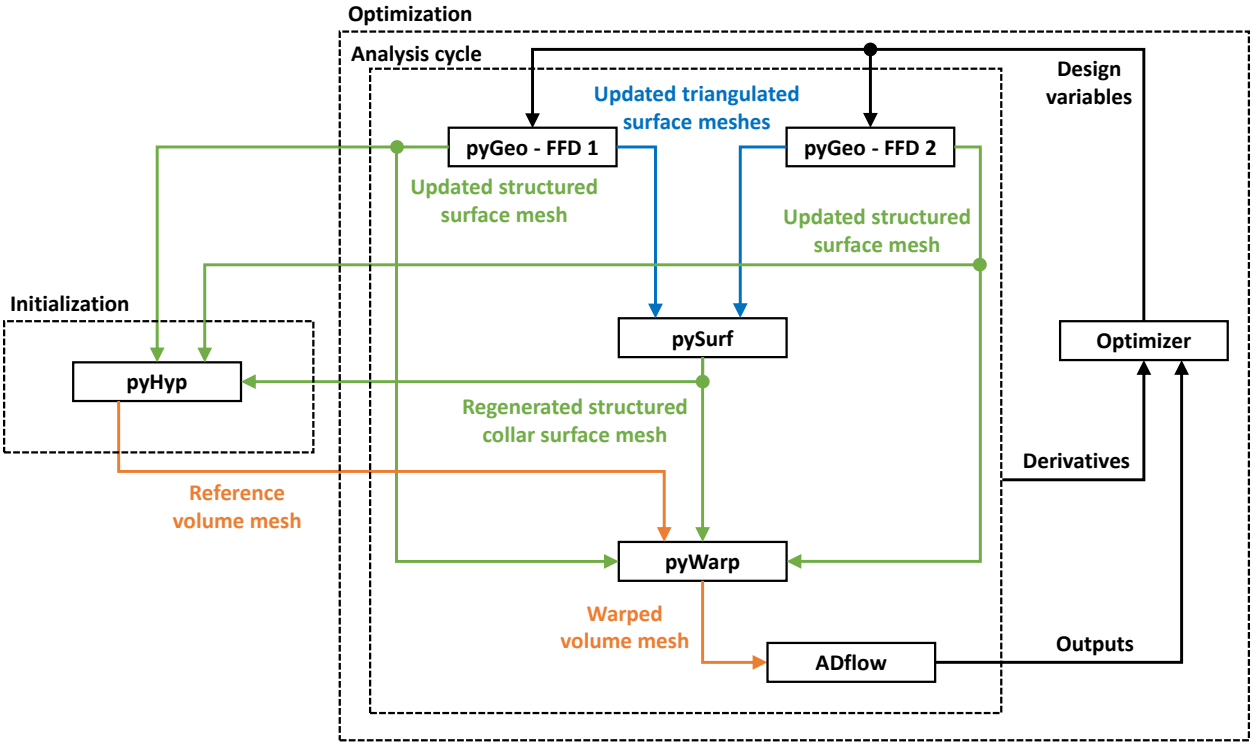


Figure 3. Aerodynamic shape optimization process. Green arrows represent structured surface meshes, blue arrows indicate triangulated surfaces, orange arrows represent structured volume meshes, and black arrows indicate design variables and functions of interest. The initialization cycle is executed only at the beginning of the optimization to generate the reference volume meshes for mesh warping. The derivatives are computed using reverse automatic differentiation, so they depend on all components of the analysis cycle.

A. Geometry modeler—pyGeo

pyGeo is a Python module that generates and manipulates geometries and is able to translate high-level design variables (twist, chord, shape) to mesh-level changes. It complements pySurf's capabilities by handling the primary components' geometries, whereas pySurf focuses on intersections and collar meshes. pyGeo uses an FFD approach to parametrize changes in geometry [18]. First, we generate a box of control points surrounding each primary component to define a trivariate B-spline mapping within this volume. Then we use a Newton search to determine the parametric coordinates of the structured surface mesh nodes within their corresponding B-spline boxes. We repeat the same process for the surface nodes given in the unstructured surface meshes so that both surface representations are parametrized in the same B-spline mapping. Any changes done to the control points can be transfered to the embedded surface nodes using this B-spline mapping. Therefore, the positions of these control points are the primary aerodynamic shape variables of the problem.

This method parametrizes changes in geometry rather than the parameters defining it, so there is no need to use CAD-based tools in the optimization process. Furthermore, the B-spline mapping is easily differentiable [19], which allows the derivatives of the surface nodes' position with respect to the control points' position to be computed efficiently.

B. Collar mesh generator—pySurf

pySurf is the module presented in this work that produces collar meshes between intersected geometry components as described in Section II. It operates using the unstructured surface meshes that are updated by pyGeo's FFDs throughout the optimization to compute intersections among primary components.

The surface collar meshes are regenerated by pySurf during each optimization iteration. Conversely, the surface meshes of the primary component meshes are not recomputed, but are updated via FFD deformations.

C. Volume mesh generator—pyHyp

The previous modules operate on surface meshes. However, we need volume meshes for CFD analysis. We use hyperbolic volume mesh marching schemes [11] to extrude the surface meshes to obtain the volumetric collar meshes. This methodology is applied to the structured surface meshes of each primary component and collars. This step is only done in the initialization phase to get the baseline volume meshes. These meshes are then warped by the mesh warping module (pyWarp) throughout the optimization.

The same advantages of hyperbolic mesh generation over other methods previously discussed for surface meshes apply here, especially those regarding the degree of automation and robustness. The main user-defined parameters are initial cell size and the marching distance. The selected marching distance should allow a reasonable amount of overlap among the collar mesh and the primary components' meshes. Additionally, cells within the collar mesh should be smaller than the cells in the primary component meshes, so that they have priority during the overset hole cutting process.

The surface meshes are extruded to a small extension (such as 3 mean aerodynamic chords). Then pyHyp uses the bounding box of these near-field volume meshes to create a background mesh that reaches an appropriate distance for the farfield boundary conditions. The background mesh generation process is discussed in more detail in Sec. V-B.

D. Volume mesh warping—pyWarp

Within the optimization loop, we warp the initial volume meshes generated by pyHyp based on the updated component surfaces instead of regenerating new ones for each design point. The warping is done using pyWarp. This tool propagates the surface nodes deformations to the volume nodes using an inexact explicit interpolation algorithm to reduce computational cost [26]. This is computationally less expensive than regenerating new volume meshes at each design point.

pyGeo provides the surface updates for the primary components, while pySurf gives the updated surfaces for the collar meshes. The background mesh has no associated surface, and thus remains fixed throughout the optimization.

Previously, pyWarp simply warped all mesh points based on all surfaces in the flow domain. We have extended pyWarp's capabilities so individual volume meshes can be warped using only the surfaces present in each individual component. For instance, the wing surface nodes only affect the warping of the wing mesh. This takes advantage of the independent nature of each component by allowing the warped volume meshes to not be influenced by surfaces that are not present in individual components. Specifically, this means that volume mesh spacing is better preserved, especially at near-wall regions where fine spacing must be achieved to accurately resolve viscous flows.

pyWarp is also automatically differentiated using Tapenade so that it can compute the derivatives of volume node positions with respect to surface node movements.

E. CFD solver—ADflow

Once volume meshes are generated and warped to reflect the changes in geometry, we use the CFD solver, ADflow, to perform aerodynamic analysis on the new configuration and to obtain lift and drag coefficients. These coefficients are used by the objective and constraint functions in the optimization problem.

The aerodynamic solver used in this work is a newly-implemented version of SUmb called ADflow [27, 28]. It is capable of solving Euler, laminar Navier–Stokes, and RANS equations in multiblock structured overset meshes in a thoroughly parallelized fashion with a cell-centered finite volume formulation. The inviscid fluxes are discretized with artificially dissipated central-differencing while the viscous fluxes use standard central-differencing.

ADflow initially solves steady problems using time-marching schemes (DDADI or RK4) accelerated with geometric multigrid to approach the basin-of-attraction. It then switches to a Newton–Krylov formulation to converge to the steady solution. The switching criterion is determined when the time residuals drop below an user-specified threshold.

This CFD solver uses implicit hole cutting [29, 30] to determine which cells should be blanked, interpolated, or actually computed. The interpolated cell values are computed using a first order interpolation method. Finally, zipper meshes are used to fill the surface gaps, giving a watertight surface for the integration of total aerodynamic forces [31].

In this work, we update the overset connectivities and zipper meshes at every optimization iteration. Further details regarding the overset implementation in ADflow are presented by Kenway et al. [32].

A discrete adjoint approach gives the derivatives of the aerodynamic forces and moments with respect to the nodal coordinates [28, 33]. We use Tapenade's forward mode algorithmic differentiation [23] to compute the partial derivatives of the CFD problem. Then we use PETSc [34, 35, 36] to solve the adjoint system with a preconditioned GMRES [37]. The derivatives currently computed by ADflow assume frozen overset interpolation weights. In other words, the linearized code does not consider the effect of the mesh coordinates on the overset interpolation weights. Nevertheless, the weights are updated in the next forward execution of the code.

F. Optimizer—SNOPT

We use SNOPT (Sparse Nonlinear Optimizer) [38], which is a gradient-based optimizer that implements the sequential quadratic programming method. The user must provide functions of interest and their gradients. SNOPT handles large-scale nonlinear optimization problems with thousands of constraints and design variables, making it suitable for aerodynamic shape and aerostructural optimizations [24, 32, 39, 40].

IV. Derivative computation

In this section we discuss the derivative computation procedure applied by the optimization framework. We also present test cases used to verify the general behavior of the aerodynamic functions of interest and the accuracy of the gradients.

A. Overview of the derivative computation process

Aerodynamic shape optimization problems usually have a high number of design variables compared to the number of functions of interest. The adjoint method [8] is an efficient way of computing derivatives for this scenario. However, the implementation of this method requires a significant amount of effort as it requires partial derivatives of the residual equations used by the CFD code.

ADflow uses a hybrid-adjoint approach [28], in which the partial derivatives of the discrete adjoint system are computed with the reverse mode AD of Tapenade. Lyu et al. [33] describes specific details regarding the differentiation of the RANS equations.

The computation of partial derivatives with reverse mode AD starts at the last code of the analysis chain, which is the CFD Solver (see Fig. 3). The volume mesh derivatives are transformed into surface node derivatives by pyWarp. Then pySurf receives the derivatives of the surface collar mesh points and executes its own reverse mode AD to backpropagate derivative information to the triangulated surface nodes. This process is illustrated by Fig. 1.

Finally, pyGeo accumulates the derivatives coming from the triangulated surfaces (given by pySurf) and from the structured surface meshes (given by pyWarp) of each primary components and then uses the FFD mapping to backpropagate derivatives to the FFD control points, which can then be translated in derivatives with respect to design variables, closing the reverse AD propagation.

This procedure needs to be applied for every function of interest in order to assemble the corresponding adjoint systems.

B. Gradient verification: cylinder case

The derivative computation is verified in a test case in which we intersect a flat plate and a cylinder, as shown in Fig. 4. Here we use the vertical position of the plate as the design variable and the lateral position of the intersection as the output variable. This allows us to compute derivatives in three distinct ways: analytically, by automatic differentiation, and using finite differences. We compare the average of the derivatives at each point in the intersection curve in Table 1. The finite differences results agree with automatic differentiation up to the eleventh digit, since the problem is locally linear (the triangles are planar elements). The error with respect to the analytical result is due to the surface discretization, and the relative error ranges between 0.27% and 1.34%. We will investigate how this gradient accuracy affects the aerodynamic optimization process in Sec. IV-E.

Table 1. Derivative comparison for the intersection points of the plate with the cylinder for different refinements levels of the triangulation. FD denotes finite differencing with the optimal step size and AD denotes automatic differentiation. FD and AD results match exactly because the derivatives on the local level are linear due to the discrete nature of the planar triangles composing each surface.

Number of triangles	20,288 triangles		112,208 triangles		1,019,852 triangles	
Method	Value	Rel. error	Value	Rel. error	Value	Rel. error
Analytic	0.57735026919		0.57735026919		0.57735026919	
FD	0.57283881276	-0.78%	0.57580421562	-0.27%	0.56963993328	-1.34%
AD	0.57283881278	-0.78%	0.57580421563	-0.27%	0.56963993329	-1.34%

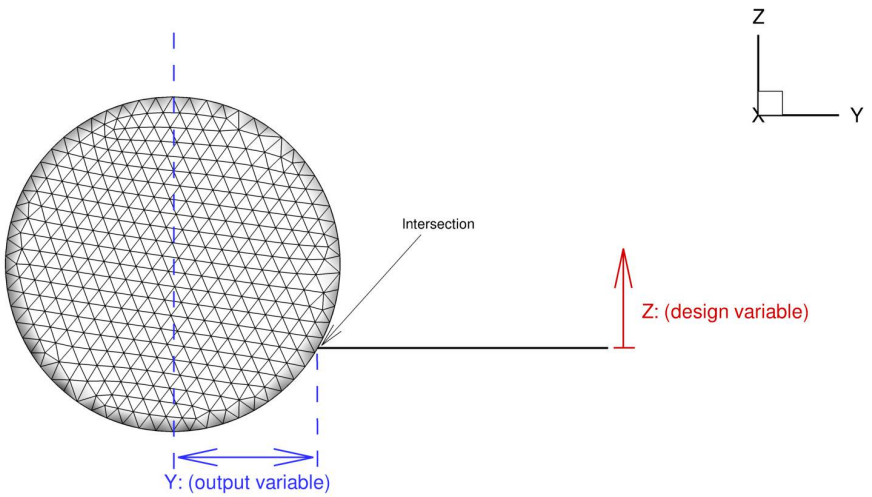


Figure 4. Cylinder-plate intersection problem variables used to verify derivatives. pySurf is capable of computing the derivatives of the intersection point coordinate with respect to the vertical position of the plate, $\partial Y/\partial Z$. The results are shown in Table 1.

C. Gradient verification: CRM case

We used pySurf to examine the wing-body intersection curve of the Common Research Model (CRM) [41] as we move the wing from a low-wing to a high-wing position, as shown in Fig. 5. The wing and fuselage are both triangulated surfaces and we recompute the intersection curve for each configuration. The intersection curve is used as a seed curve to march the collar mesh.

Because the process is fully differentiated, we can calculate the gradients of the intersection curve points with respect to the design variables. In this example, we computed the derivative of the trailing edge point's Y-coordinate with respect to the design variable, the vertical position of the wing. The derivatives are plotted in Fig. 6 for the CRM fuselage model. The wing and fuselage surfaces are inherently discontinuous, so it was not clear if the gradients would be smooth or accurate enough. We found that the computed gradients are indeed suitable for gradient-based optimization, since they locally follow the shape of the fuselage well.

D. Noise issues in the functions of interest

We integrated pySurf into the framework and now seek to verify the overall derivatives. The entire chain of code used in the optimization is differentiated in both forward and reverse mode, allowing us to use the dot product test to verify the consistency of the linearized code [42].

We propose a simple case to verify the accuracy of the computed derivatives in a transonic flow application. We generate a simple geometry consisting of a tapered, swept wing based on a Boeing 717 and a cylindrical fuselage, which are then simulated in transonic flow conditions (Mach 0.8). We define a single design variable to the geometry: the wing vertical position with respect to the fuselage (Fig. 7). We generate the surface meshes of the primary

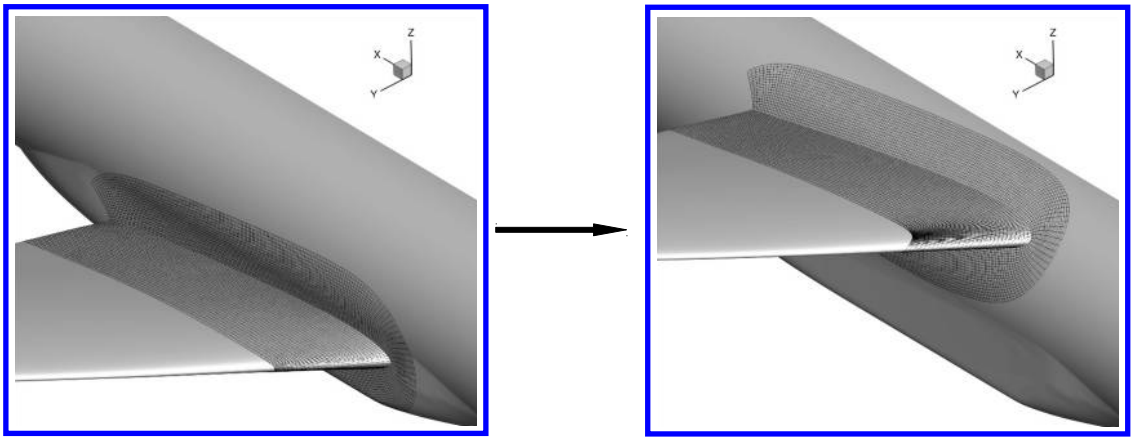


Figure 5. Low-wing configuration shifted to high-wing configuration with automatically generated collar meshes.

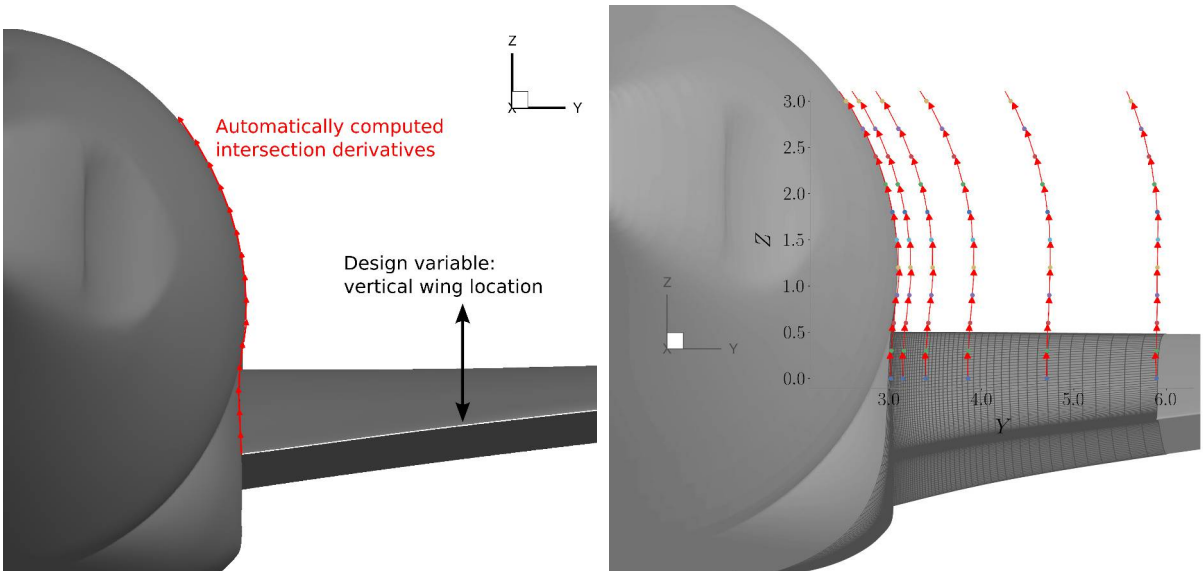


Figure 6. Aft view of the CRM wing-body junction. The left figure shows the design variable definition and the tracking of trailing edge intersection point. The figure on the right shows how surface mesh points move as the wing is translated. The red arrows indicate the AD-computed derivatives for the trailing edge intersection position (left) and surface mesh points (right) as we vertically translate the wing. The computed gradients for triangulated surface intersections are smooth and accurate enough for gradient-based optimization.

components in ICEMCFD. The fuselage mesh remains fixed while the wing mesh is vertically translated according to the design variable value. The fuselage-wing intersection curve and the collar mesh are automatically generated by pySurf for each design point.

Then we run flow simulations and derivative computations for a range of wing positions; the results are shown in Fig. 8. The derivatives are consistent with the overall trend of the function. However, when we consider small displacements (right side of Fig. 8), we see that the function shows noisy patterns with amplitudes of hundredths of drag counts. The noise is mainly caused by topological changes in the zipper mesh used for force integration (Fig. 9).

The drag coefficient curve shown in Fig. 8 suggests that there is a minimum around $y_{\text{wing}} = 0.5$. Next, we use this same test case to verify the effects of the noise in optimization problems.

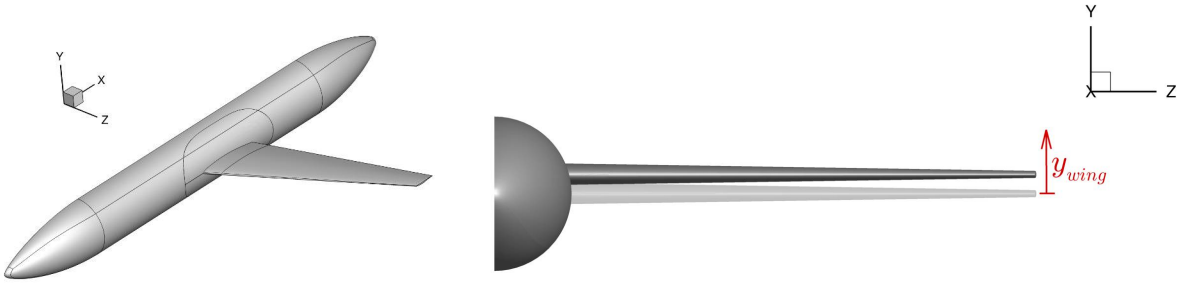


Figure 7. Transonic configuration used for wing translation study. The y_{wing} design variable controls the vertical displacement of the wing.

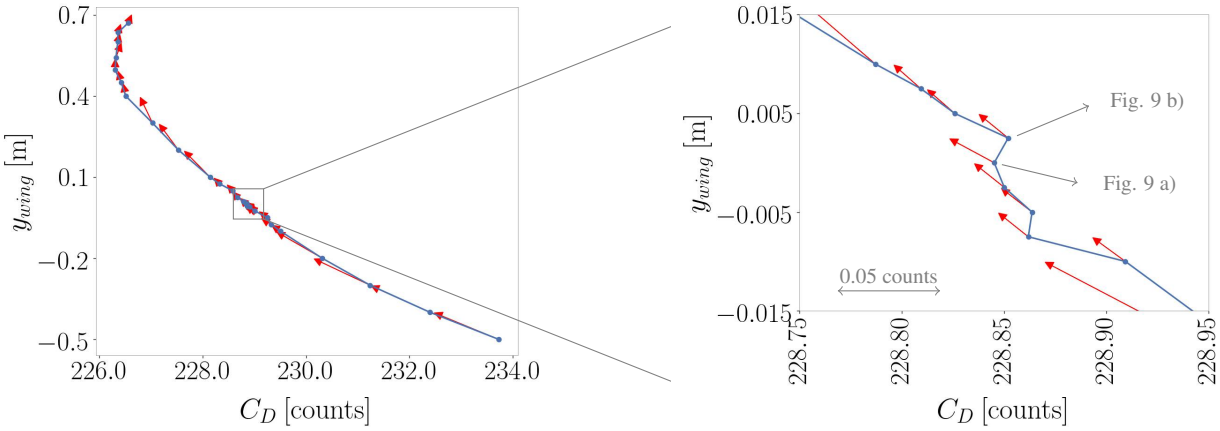
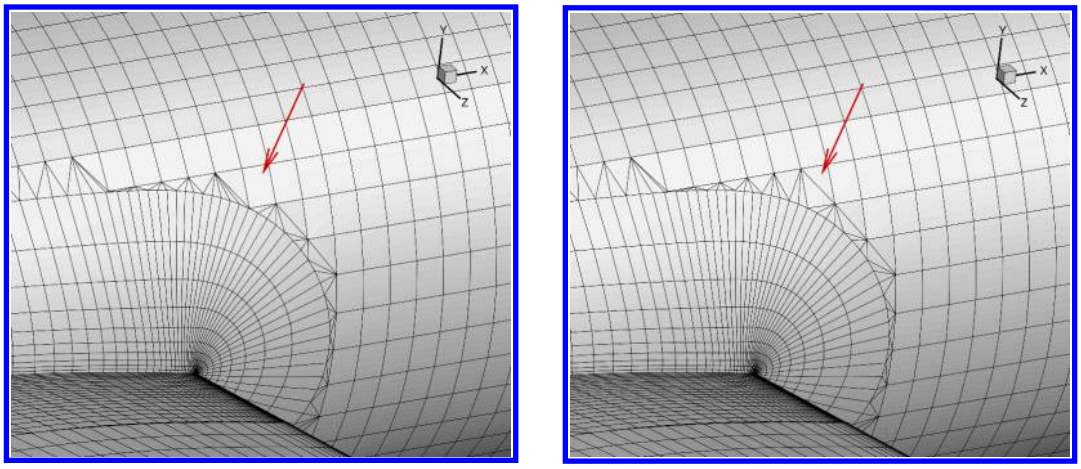


Figure 8. Drag variations due to the vertical translation of the wing on a transonic airplane configuration. The red arrows represent gradients. We observe noise at small steps, but the gradients are still consistent with the overall trend. The noise is mainly associated with changes in the zipper mesh. The zipper mesh change illustrated in Fig. 9 caused the gradient inconsistency between the two indicated points.



a) $y_{wing} = 0.0000$ m b) $y_{wing} = 0.0025$ m

Figure 9. Moving the collar mesh by a small amount blanks another cell of the fuselage mesh during the hole cutting process. This changes the topology of the zipper mesh used for the integration of surface forces, causing the noise in the drag curve shown in Fig. 8.

E. Effects of numerical noise in an univariate optimization problem

We study the effects of the noise in a relatively simple case before moving to a more complex optimization problem. Here we use the same transonic configuration introduced in the previous section. We control the vertical position of the wing relative to the fuselage to minimize drag, which is a univariate optimization problem.

Figure 10 shows the optimization progress. The optimizer gets trapped in a noisy valley when it reaches the flat design space surrounding a potential optimum and then halts due to numerical difficulties. Noise becomes an issue in regions of the design space where the gradients are small, such as near the optimum, since the relative error between the gradient and the actual trend of the noisy function is significant. Nevertheless, the optimization still improves the baseline design and converges towards the global optimum. Therefore, we proceed to more complex optimization cases.

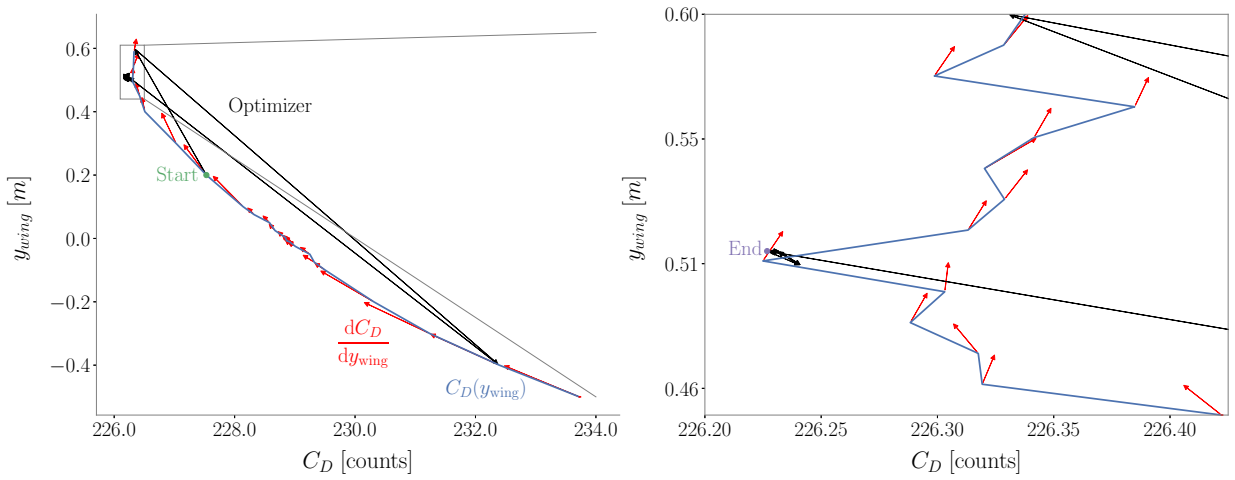


Figure 10. Optimization path for the wing translation case. The plot on the right is a zoomed-in view showing the last iterations of the optimizer. The optimizer is stuck in the noise, but it managed to improve the design.

V. DLR-F6 optimization

To demonstrate the capabilities of pySurf, we perform aerodynamic shape optimization of the DLR-F6 wing-body configuration [43]. This model was extensively used for drag prediction benchmarks [44, 45] and as a baseline for aerodynamic shape optimization cases [46, 47, 48].

The baseline DLR-F6 configuration shows a separation bubble at the trailing edge of the wing-body junction at $C_L = 0.5$ [44], which makes this case particularly interesting for wing-body junction optimizations. In fact, a fairing was designed for the third drag prediction workshop (DPW3) to remove the recirculation region and improve drag convergence studies [49], and this configuration is named DLR-F6-FX2B.

In one of the previous works regarding the DLR-F6 aerodynamic shape optimization [48], the FFD volume would affect all contained surface mesh points, regardless of whether they are from the wing or fuselage. Now we can define each component and its surfaces separately using pySurf, and thus have individual control of the deformation for each component. For example, we can modify the twist and shape of the wing without having any effect on the fuselage.

This separation bubble served as motivation to use the MACH framework, now with the pySurf module, to perform aerodynamic shape optimization of the DLR-F6 wing-body junction starting from its original geometry. The objective of all optimization cases described here is the minimization of drag coefficient (C_D) at $C_L = 0.5$, $M = 0.75$, and $Re = 5 \cdot 10^6$, following the same standards as DPW3.

Since pySurf now allows independent manipulation of the wing and fuselage, we define four major groups of design variables: angle of attack, wing twist, wing shape, and fuselage shape variables. We conduct a series of

optimizations using multiple combinations of these groups to investigate the effects of each design variable in the overall improvement of the design. Table 2 shows a summary of the overall optimization problem.

Besides the lift constraint, we also consider thickness constraints for the wing and a *bubble-gum* constraint for the fuselage, which means the fuselage shape cannot go inside its original outer-mold line. The same constraint was adopted during the FX2B fairing design.

Table 2. General description of the DLR-F6 aerodynamic shape optimization

	Variable/function	Description	Quantity
Minimize	C_D	Drag coefficient	
with respect to	α	Angle of attack	1
	n_{fuse}	Normal displacement of fuselage FFD control points	16
	τ_{wing}	Wing twist	8
	z_{wing}	Vertical displacement of wing FFD control points	128
		Total design variables	**153**
Subject to	$C_L = 0.5$	Lift constraint	1
	$t/t_{\text{init}} \geq 0.99$	Wing thickness constraint	100
	$\Delta z_{\text{wing,TE,upper}} = -\Delta z_{\text{wing,TE,lower}}$	Fixed trailing edge	8
	$\Delta z_{\text{wing,LE,upper}} = -\Delta z_{\text{wing,LE,lower}}$	Fixed leading edge	8
		Total constraints	**117**

A. Design variables and constraints

The angle of attack is a necessary variable to meet the lift constraint. The geometric design variables affect the position of the FFD control points.

The wing FFD is composed by a volumetric block of 8 spanwise by 8 chordwise by 2 vertical nodes (Fig. 11). The 8 wing twist variables (τ_{wing}) apply uniform rotation around the quarter-chord to each chordwise section of the FFD block. The 128 displacement variables (z_{wing}) move each control point of the FFD in the vertical direction to provide airfoil shape control. To avoid shear twist, the control points on either side of the upper and trailing edges are constrained to move equal amounts but in opposite directions. This ensures that the airfoil chordline is affected only by twist variables, not shape variables [39].

Since this is a purely aerodynamic shape optimization problem, we need to enforce thickness constraints to obtain a realistic design. In this case we define a grid of 10×10 points over the wing planform and then we enforce that the thicknesses at these locations should never go below 99% of the baseline thicknesses.

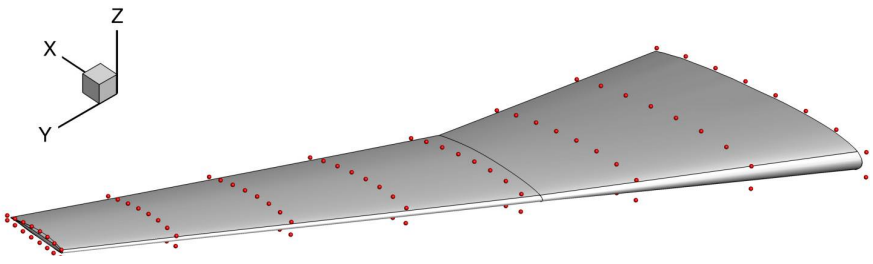


Figure 11. Wing FFD. The control points (red dots) are manipulated by the twist and vertical displacement variables.

The fuselage FFD consists of a block of $8 \times 6 \times 2$ nodes that encompasses the region around the wing intersection (Fig. 12). Nodes in the outer two layers of the FFD are fixed to guarantee C_1 continuity with the undeformed part of

the fuselage [17], so this leaves a total of 16 free control points. These control points are only allowed to move in a direction normal to the fuselage surface. Furthermore, they can only move away from the center of the fuselage, ensuring the bubble-gum constraint.

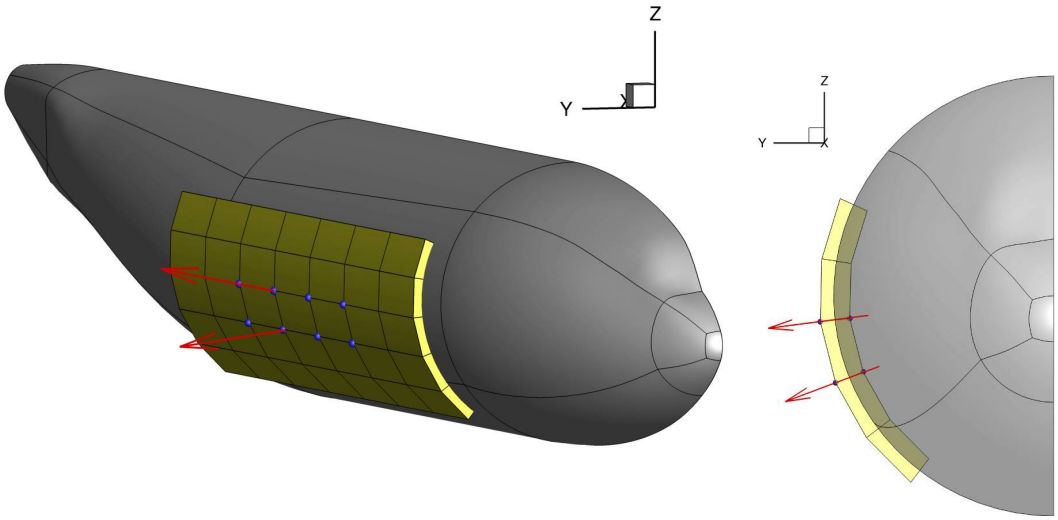


Figure 12. Fuselage FFD. The blue dots indicate free control points. The other control points remain fixed to guarantee C_1 continuity within the undeformed region.

To better understand this design optimization problem we steadily increase the problem complexity until we reach the desired formulation expressed in Table 2. We do this by activating each group of design variables one at a time. Therefore we defined five optimization problems that are identified by the tags shown in Table 3.

Table 3. Identification tags for each optimization problem.

Tag	Active design variables	Case description
B	α	C_L matching for the baseline configuration
F	α, n_{fuse}	Fairing optimization
F+T	α, n_{fuse}, τ_{wing}	Fairing and wing twist optimization
F+T+S	α, n_{fuse}, τ_{wing}, z_{wing}	Fairing, wing twist, and wing shape optimization
T	α, τ_{wing}	Wing twist optimization
T+S	α, τ_{wing}, z_{wing}	Wing twist and shape optimization

B. Case setup

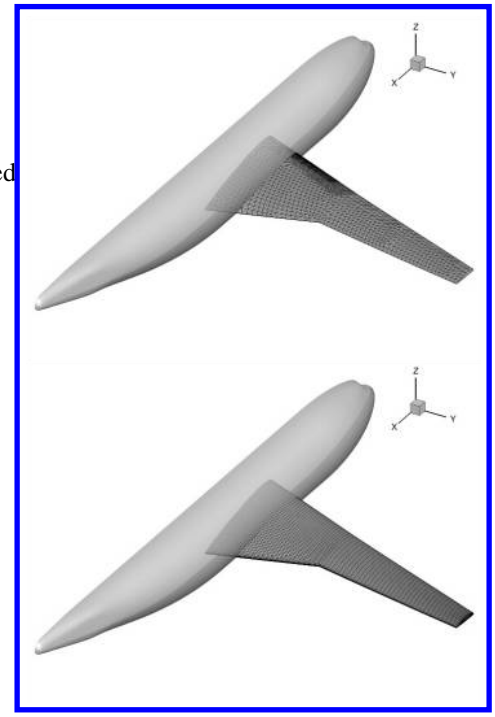
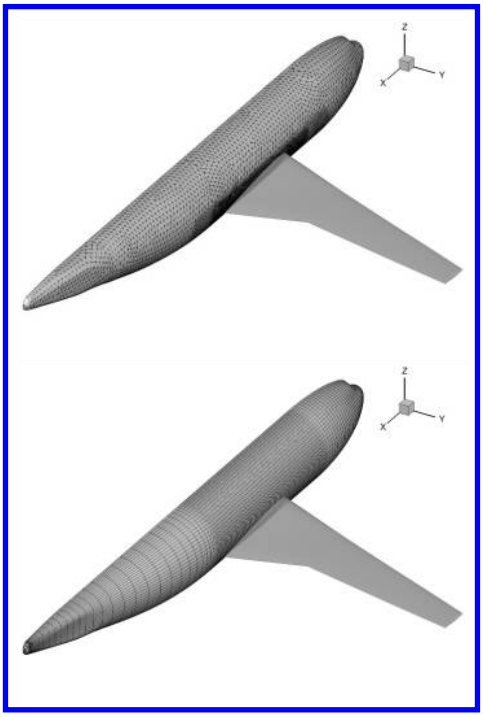
To set up the analysis and optimization cycle in the MACH framework we need to provide triangulated surfaces and structured surface meshes for all primary components. We imported the DLR-F6 IGES files in ICEMCFD to generate both triangulated surfaces and structured surface meshes for the wing and fuselage (Table 4).

The wing and fuselage triangulated surfaces are refined in the high-curvature regions (such as the leading edge) where we expect the collar mesh to grow. The wing triangulated surface has 9,948 elements while the fuselage triangulated mesh has 19,902 elements. The wing structured surface mesh has 127,488 cells and the fuselage structured surface mesh has 105,344 cells. During the triangulated surface generation, we also generated bar elements to outline important features of the geometry, such as the two corners of the blunt trailing edge.

In the next step, we use pySurf to perform the necessary operations for the collar mesh generation, which are to:

1. Compute the intersection between the wing and fuselage triangulated surfaces.

Table 4. Inputs required by pySurf. All triangulated meshes and structured surface meshes are generated with ICEMCFD.

	Wing	Fuselage
Triangulated surface		
Structured surface meshes		

2. Split the intersection curve at the wing leading edge and trailing edge corners. This splits the intersection curve into three segments: upper skin, lower skin, and blunt trailing edge.

3. Create node distributions for each of these segments in an appropriate manner for CFD simulations (for instance, with refined leading and trailing edges.)

4. Merge the three remeshed curve segments back into a single curve.

5. Use the merged curve as a starting feature for the hyperbolic surface mesh marching on the wing and on the fuselage to generate the surface collar mesh.

pySurf saves the order of these processes for the reverse propagation of derivatives. The execution of the outlined steps generates the collar mesh shown in Fig. 13.

Before tackling the optimization problem itself, we use pyHyp in an initialization step to automatically extrude the surface meshes into volume meshes. The extrusion parameters, such as initial cell size and growth ratios, follow the DPW3 guidelines for the coarse mesh level. We enforce slightly smaller values of these parameters (95%) for the collar mesh to ensure that its cells have higher priority during the implicit hole cutting process. The near field meshes are extruded up to 2.5 times the mean aerodynamic chord of the model. The background mesh is generated in two steps. First we generate a block of Cartesian cells to fill the bounding box defined by the near field meshes. Finally, we use hyperbolic extrusion once again to march the surface of this Cartesian mesh to reach 100 mean aerodynamic chords, which is the farfield distance specified by DPW3. Figure 14 displays the symmetry plane of the overset meshes. Now that we have a volumetric mesh, we can proceed to the aerodynamic analysis of the geometry.

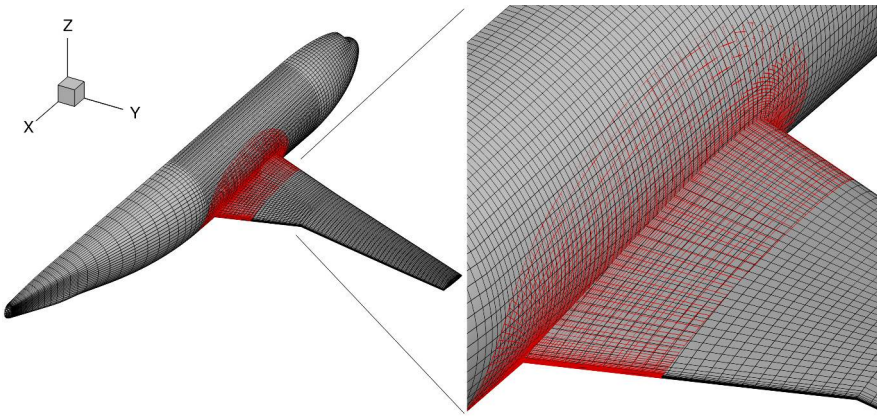


Figure 13. Structured surface meshes used for the volume mesh extrusion. The primary components' meshes (wing and fuselage) are shown in black, and the automatically generated collar mesh is shown in red. A detailed view of the collar mesh is shown on the right. The collar mesh generation parameters should be adjusted so that the cells have similar sizes at the interfaces.

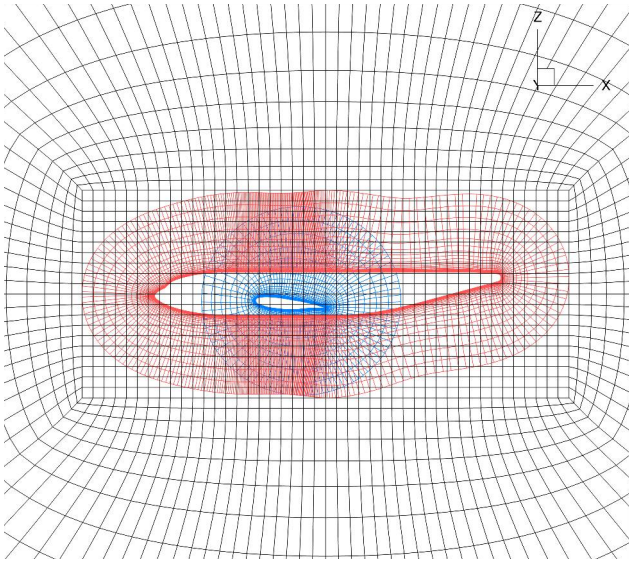


Figure 14. Symmetry plane showing the fuselage near field mesh (red) the wing near field mesh (blue) and the background mesh (black) made by a Cartesian block and an O-mesh.

C. Baseline configuration studies

To make sure we are simulating the right physics, we run preliminary aerodynamic analysis on the baseline configuration to compare with the DPW3 data set.

We used the procedure described in the previous section to generate meshes for both the baseline DLR-F6 configuration and the one with the FX2B fairing. The same mesh topology and number of cells are used for both cases. We generated three mesh levels for a drag convergence study (Table 5). The use of automatic collar mesh generation accelerated the process since the mesh parameters could be easily changed in our script.

The ADflow drag predictions were also evaluated in the sixth Drag Prediction Workshop (DPW6) [50]. One important conclusion obtained during this benchmark study was that the standard Spalart–Allmaras (SA) turbulence model [51] could overestimate the size of separation bubbles in wing-body junctions [52], and that this could be mitigated by applying modifications to the turbulence model, such as the rotation correction [53], and the quadratic

Table 5. Mesh levels used for drag convergence study. The maximum y^+ values are computed based on the converged CFD results.

Level	Number of cells (million)	max y^+
Coarse	1.1	1.4
Medium	3.0	1.1
Fine	8.8	0.6

constitutive relation (QCR) [54]. We perform the drag convergence study for both the standard SA model and the modified version with rotation correction and QCR relation (SA-R-QCR2000).

Figure 15 displays the drag coefficients obtained for different mesh levels and turbulence models. The QCR version of the turbulence model predicts higher drag values and the difference in drag values between the two components is not immediately evident, except for the finer meshes.

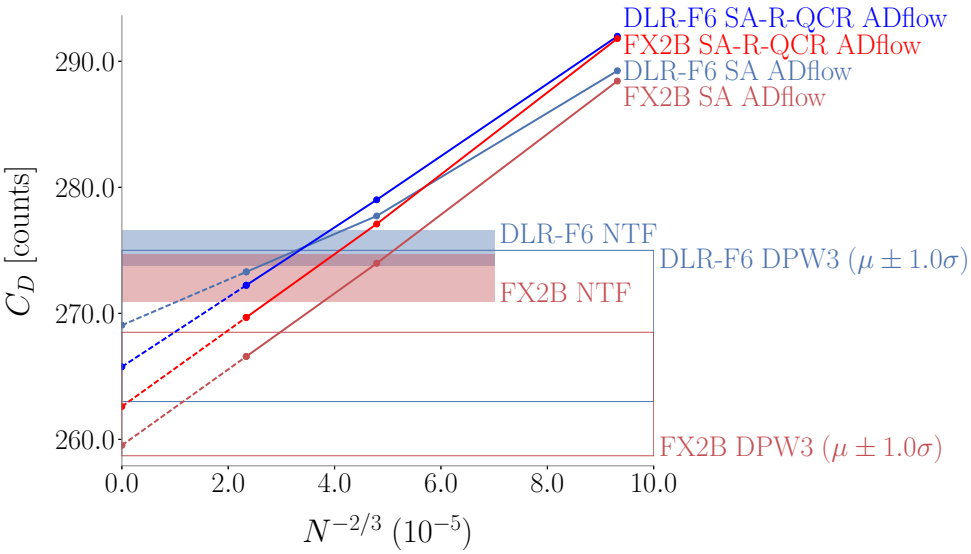


Figure 15. Drag convergence study for the DLR-F6 configurations. ADflow results are compared against the average values from DPW3 [45] and also against the experimental data from National Transonic Facility (NTF) [55]. Blue colors refer to the baseline DLR-F6 configurations and red colors refer to the DLR-F6-FX2B configuration. The shaded area represents the uncertainty in the wind tunnel measurements. ADflow predictions are within 1.0 standard deviation of the average values from DPW3. The kink in the convergence curve for the baseline configuration analyzed with standard SA model is related to the bubble growth shown in Fig. 17.

The ADflow simulations show the same flow features seen in DPW3: the baseline version shows the separation bubble on the wing-body junction trailing edge, while the FX2B fairing removes it (Fig. 16). The separation bubble of the baseline configuration obtained from different mesh levels and turbulence models is shown in Fig. 17. We see that the bubble size predicted by the standard SA model increases as the mesh is refined. This phenomenon was also observed in DPW6, where the overestimated bubble size caused inconsistencies when comparing against wind tunnel data [50]. In addition, this separation bubble growth for the standard SA model is related to the change in slope of the drag convergence curve for the baseline configuration (DLR-F6 curve in Fig. 15). The curves based on the QCR model do not show the same change in slope because the bubble growth is contained by the turbulence model type. The same applies to the DLR-F6-FX2B case since the separation bubble is negligible due to the fairing, and thus it does not affect drag convergence.

We use the SA-R-QCR2000 turbulence model for the optimization because it showed better drag convergence characteristics. In addition, we use the coarse mesh level (1.1 million cells) for the optimization to limit the computational cost of the optimization.

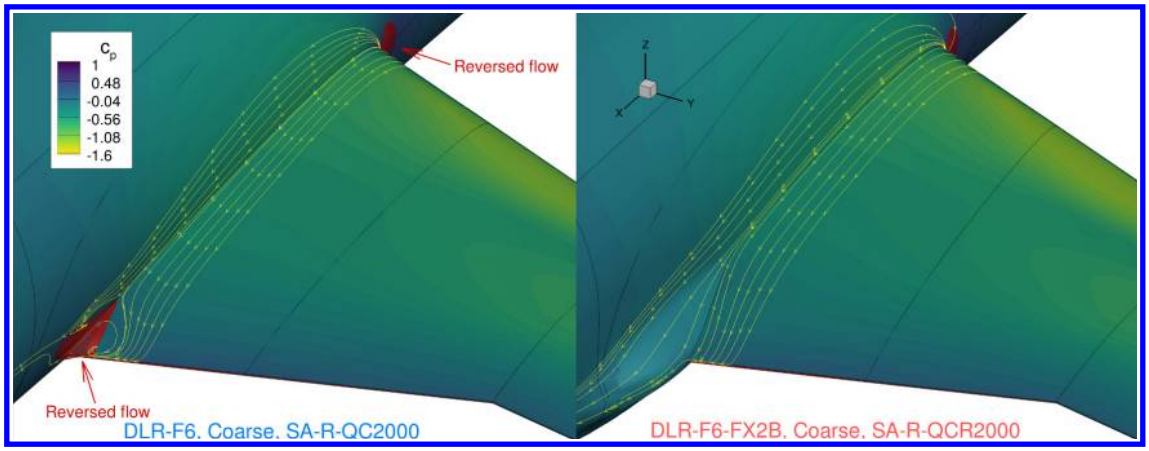


Figure 16. Wing-body junction flow patterns predicted by ADflow for the DLR-F6 configuration. The baseline geometry has a separation bubble on the junction trailing edge (left), while the addition of the FX2B fairing removes it (right). This is the same trend observed in DPW3.

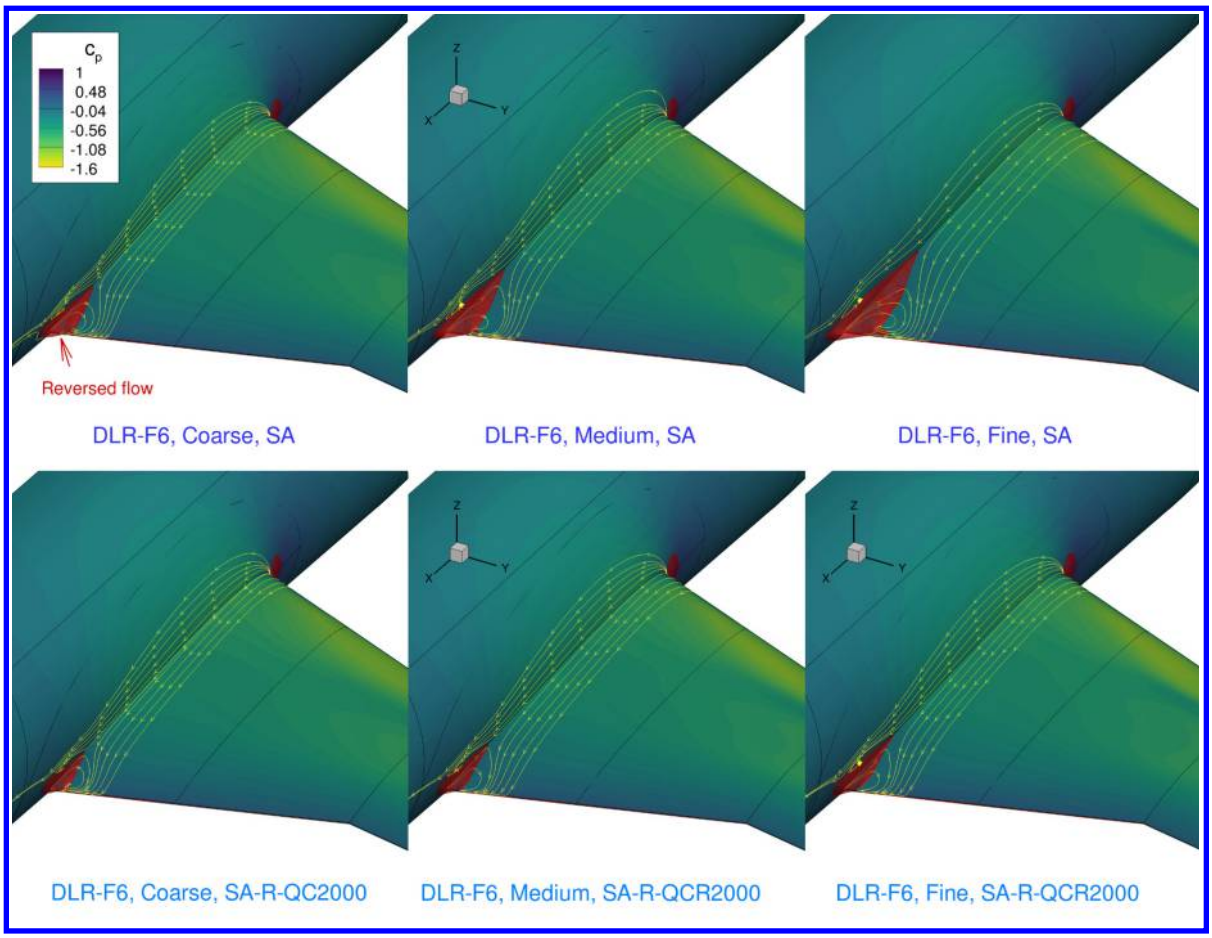


Figure 17. Effect of grid refinement and turbulence model variant in the recirculation bubble size. The standard SA model predicts a larger separation bubble as the grid is refined. On the other hand, the bubble size has small variations due to grid refinement for the SA-R-QCR2000 model, which usually gives better agreement with experimental data and better drag convergence characteristics, as seen in DPW6.

D. Optimization results

At first we try an optimization with just with the fuselage shape variables (Problem F in Table 3). All other design variables presented in Table 2 are fixed in this case. Figure 18 shows the convergence of the optimization problem, and we achieve a reduction of approximately 5 drag counts. Figure 19 shows details of the optimized fairing. The designed fairing completely eliminates the recirculation regions, as shown in Fig. 20.

Figure 18 also shows that some design variables reach the prescribed bounds. The fairing extends even more if we relax these bounds, but excessive stretching leads to mesh warping and overset hole cutting problems. The explanation for this behavior is that since the optimizer does not have freedom to modify the wing geometry, it is (inefficiently) using the fairing to change the lift distribution. The fairing gets bigger to unload the inboard region of the wing and shift the lift distribution towards the tip. The increased outboard load is slightly closer to the desired elliptical distribution, as shown in Fig. 21.

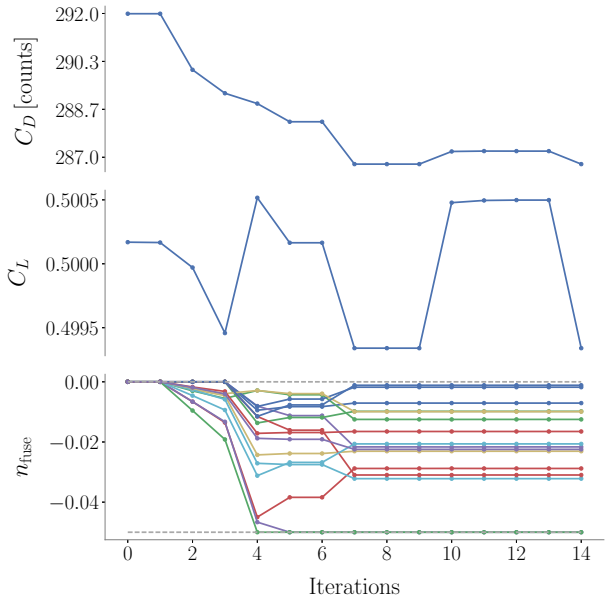


Figure 18. Fairing-only optimization history. Some normal displacement design variables went to the lower boundary of -0.05.

Next, we solve the additional optimization problems described in Table 3 to investigate the effect of the other design variables. Some relevant performance figures regarding these optimization, such as optimality and wall time, are listed in Table 6. No optimization reaches the desired reduction in optimality since they each terminate due to numerical difficulties caused by noise in the functions of interest. This follows the same trend observed in the univariate optimization case discussed in Sec. IV-E. Nevertheless, the optimizer still reduces the drag compared to the baseline configuration for every case. The summary of the drag coefficients obtained for each optimized configuration is shown in Table 7 and Fig. 22.

Table 6. Figures regarding each optimization problem.

Problem	Initial Optimality	Final Optimality	Wall time (hours)	Procs
F	$7.4 \cdot 10^{-1}$	$1.4 \cdot 10^{-1}$	1.2	64
F+T	$1.8 \cdot 10^{-2}$	$7.7 \cdot 10^{-4}$	4.3	128
F+T+S	$1.2 \cdot 10^{-2}$	$4.7 \cdot 10^{-3}$	6.0	128
T	$1.7 \cdot 10^{-2}$	$1.4 \cdot 10^{-4}$	1.9	128
T+S	$1.2 \cdot 10^{-2}$	$7.2 \cdot 10^{-3}$	3.5	128

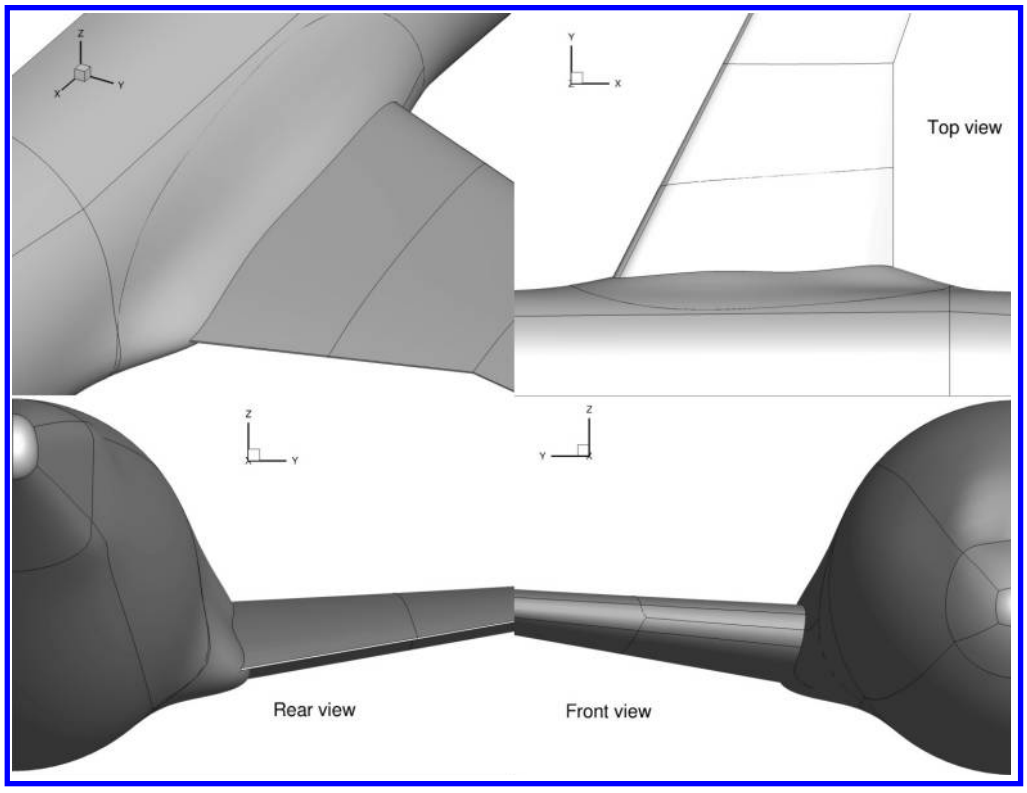


Figure 19. Optimized fairing for the fairing-only optimization (Problem F).

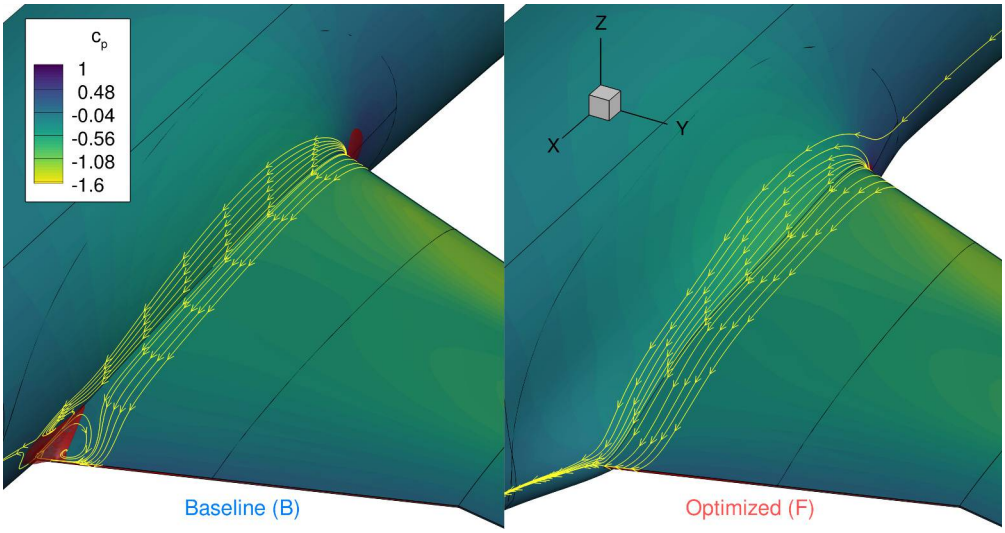


Figure 20. Wing-body junction flow before and after the fairing optimization (Problem F). Red regions indicate reversed flow. The redesigned fairing reduces the recirculation bubble.

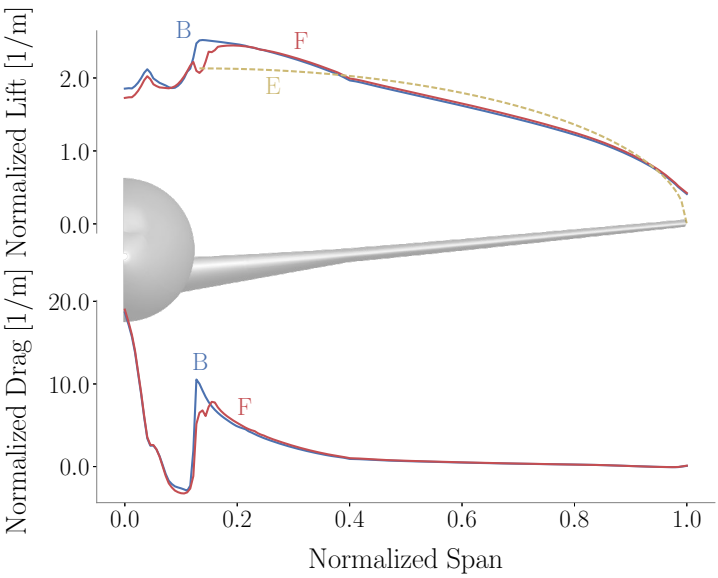


Figure 21. Comparison of lift and drag distributions between the baseline (B) and the fairing-optimized configuration (F). The equivalent elliptical distribution (E) is computed using wing-alone span and lift.

Table 7. Aerodynamic coefficients obtained for each optimization.

Problem	C_L	C_D (counts)
B	0.5002	291.99
F	0.4993	286.76
F+T	0.4995	281.28
F+T+S	0.5000	276.85
T	0.5000	280.47
T+S	0.4995	278.61

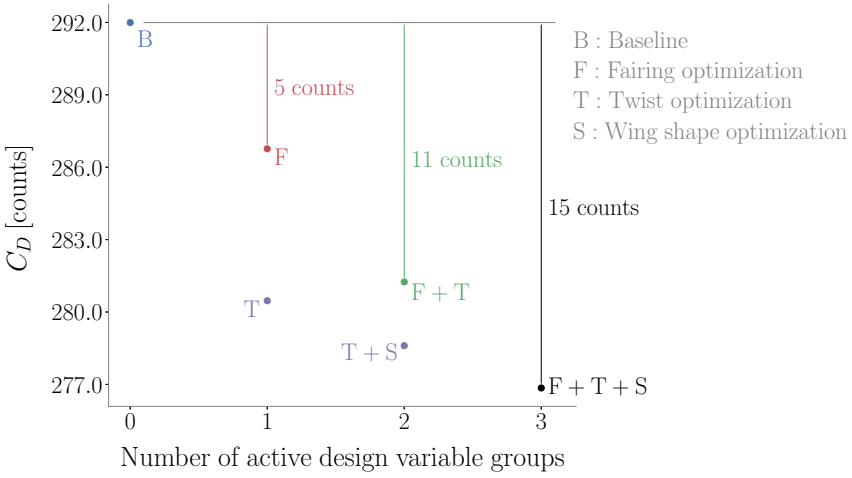


Figure 22. Progression of the optimized drag value due to the additional active design variables. The design generally improves as we add more degrees of freedom to the optimization.

As we can see from Fig. 22, adding design variables tends to reduce the drag. The full optimization (F+T+S) achieves a 15 drag count (5%) improvement compared to the baseline configuration. However, the fairing and twist optimization (F+T) converged to a higher drag value than the twist alone optimization (T), but the difference between the two optimized configurations is smaller than 1.0 drag count. Thus, the noise level might have prevented the F+T optimization from reaching a drag value smaller that the one from Problem T.

Figures 23 and 24 show how the optimized fairing geometry changes due to the addition of twist design variables and wing shape variables, respectively. The fairing size progressively reduces as we include more degrees of freedom in the optimization, and the fairing design variables do not reach the prescribed bounds anymore. Since the optimizer has additional degrees of freedom to tailor the wing, the fairing design variables only make local adjustments to the wing-body junction flow pattern, which require smaller deformations. Therefore, the fairing design variables no longer reach their bounds as shown in Fig. 25.

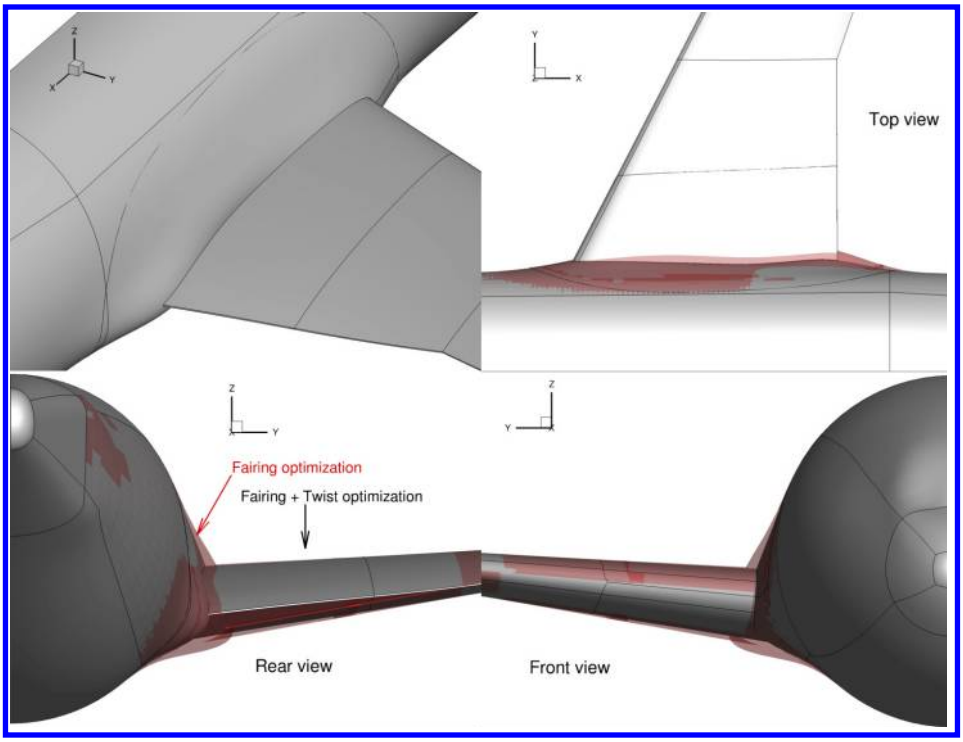


Figure 23. Comparisons of the optimized fairing for Problem F (red) and Problem F+T (grey). The fairing gets smaller for Problem F+T since the optimizer can change twist variables to control the spanwise lift distribution.

We observe similar behavior for twist variables. Figure 25 shows that the magnitude of twist variation also decreases when we activate shape design variables. The shape variables can control the airfoil camber, which allows the optimizer to reach the desired lift distribution more efficiently than using twist alone. This is because shape variables can further improve the airfoil aerodynamic efficiency (L/D) while still attaining an elliptical lift distribution.

Figure 26 shows the normal displacements on the fuselage surface for the F, F+T, and F+T+S optimization cases. The fuselage shape is changed the most when the optimizer can only control the fairing geometry and not the wing. In each case, the leading and trailing edges are modified the most since these regions have recirculated flow.

Figure 27 shows the lift and drag distributions for the different optimized configurations. The inclusion of twist variables allows the optimizer to reach a more elliptical lift distribution that reduces induced drag. In addition, optimizations with twist variables reduced the fuselage contribution to the overall lift and transferred this load to the wing, where lift can be generated more efficiently.

The introduction of wing shape design variables results in designs with smoother lift distributions (see Fig. 28), since the optimizer can fine-tune the thickness and camber along the wing. Smoother pressure distributions have smaller pressure gradients that prevent accelerated boundary layer growth.

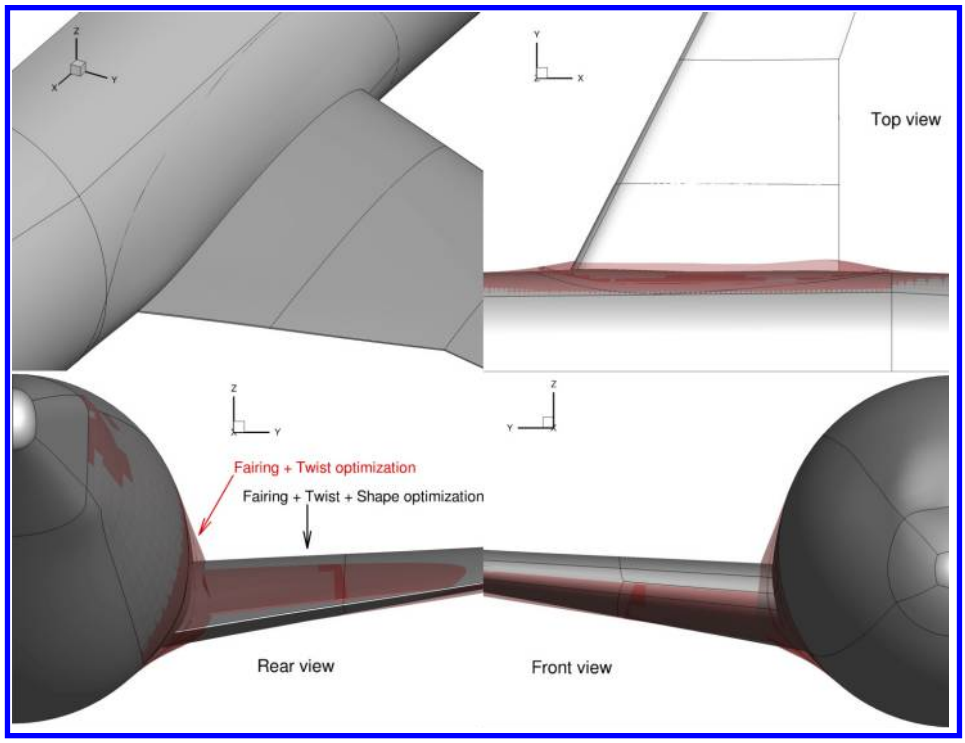


Figure 24. Comparisons of the optimized fairing for Problem F+T (red) and Problem F+T+S (grey). The wing shape variables also contribute to the improvement of the wing-body junction flow, so the optimizer could make the fairing even smaller.

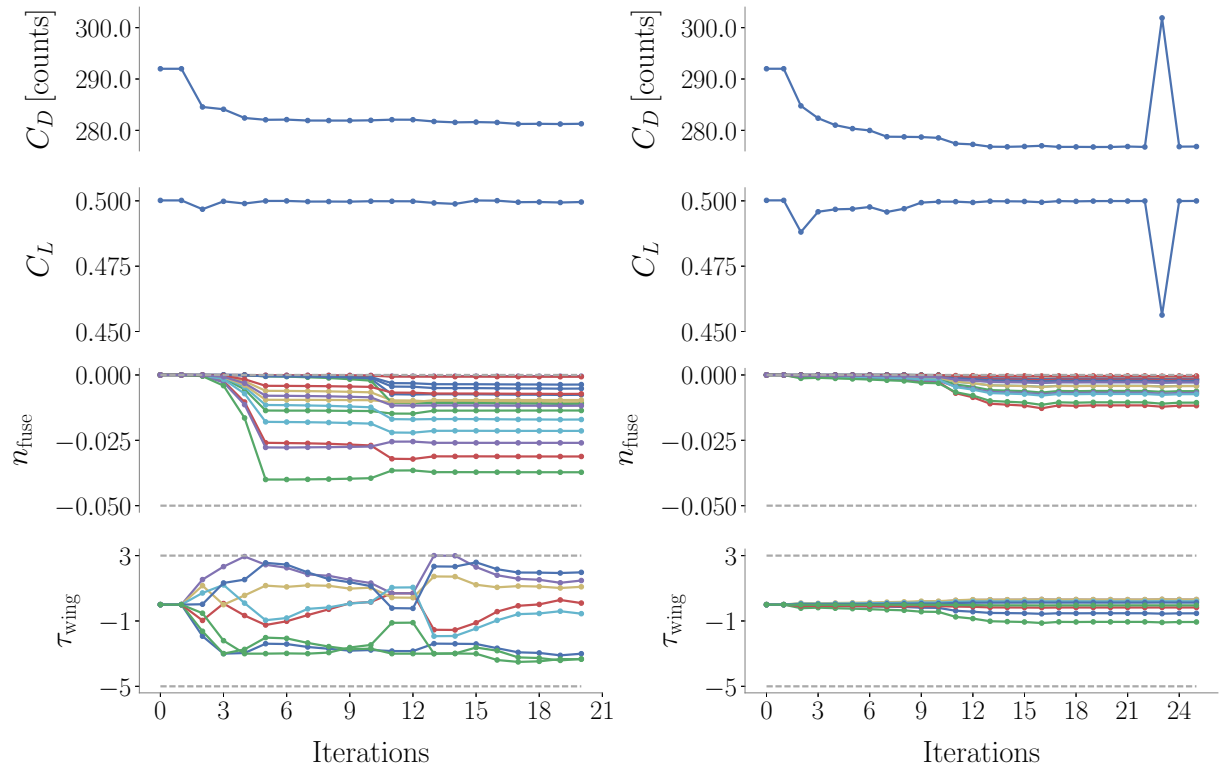


Figure 25. Fairing and twist optimization history (left) and fairing, twist, and shape optimization history (right). The fairing design variables do not reach the lower bound in either case. Due to the large number of shape design variables, they are not shown here.

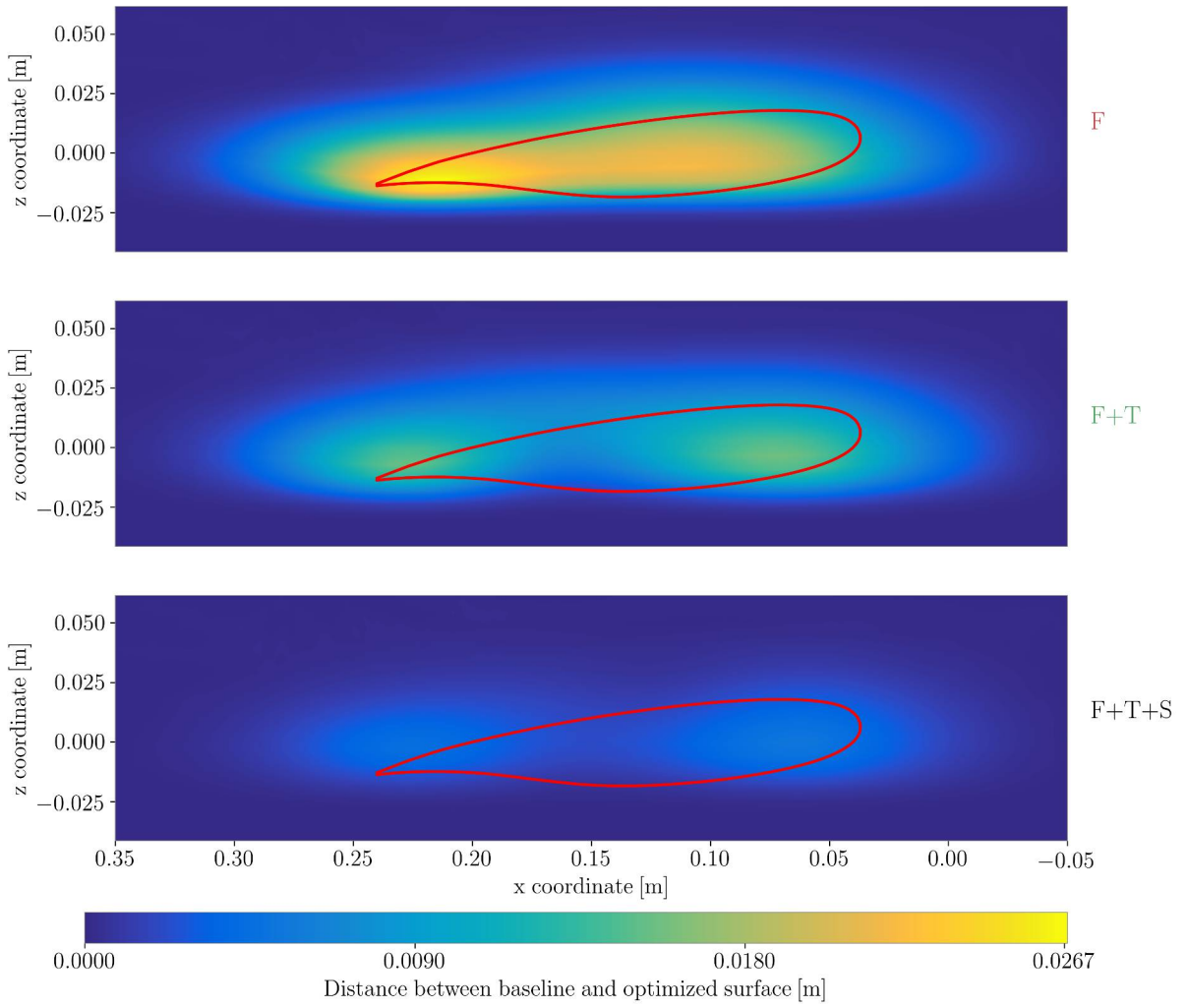


Figure 26. Contour plot showing the distance that the fuselage surface moved during optimization. The optimizer extends the fairing both at the leading and trailing edges, since both regions initially show recirculated flow. The fuselage surface moves a relatively large amount when the optimizer can only control the fairing.

The fairing design variables effectively remove the separated region of the junction trailing edge in all optimization cases in which they are active, as seen in Fig. 29. Twist and wing shape optimization without the fairing design variables might still show minor separated regions (see Fig. 30). In fact, the main differences between drag distributions of the T+S and F+T+S configurations shown in Fig. 31 are located around the junction region (between 10% and 30% of the semi-span). On the other hand, Problems T+S and F+T+S converged to practically the same wing geometry. Figure 31 indicates that the wing twist distribution and the airfoil sections of the optimized configurations are similar, except in the vicinities of the wing-body junction, where the effects of fairing design variables are noticeable.

Figure 32 shows the drag convergence plot for the F+T+S optimization along with the baseline and FX2B configuration. The optimized result shows a drag reduction of 15 counts (5%) at the coarsest mesh level, which was used for all optimization cases. The continuum estimates show the optimized configuration achieves a drag reduction of 10 counts (4%). Furthermore, the relative ranking of the configurations in terms of drag remains the same for all mesh levels.

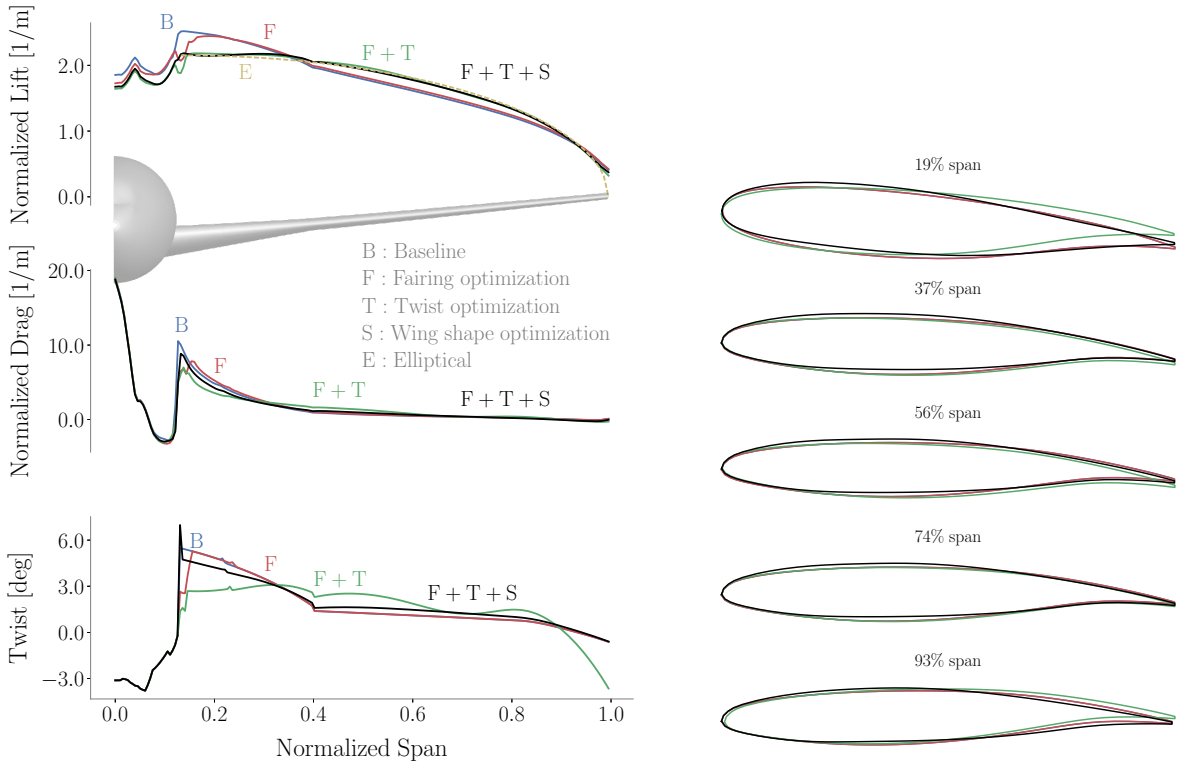


Figure 27. Comparison of lift and drag distributions for all fairing optimizations. The inclusion of twist variables (T) allows the optimizer to achieve more efficient lift distributions, which are closer to an elliptical one. The elliptical wing distribution is computed using the lift and span of the wing alone, not including the fuselage.

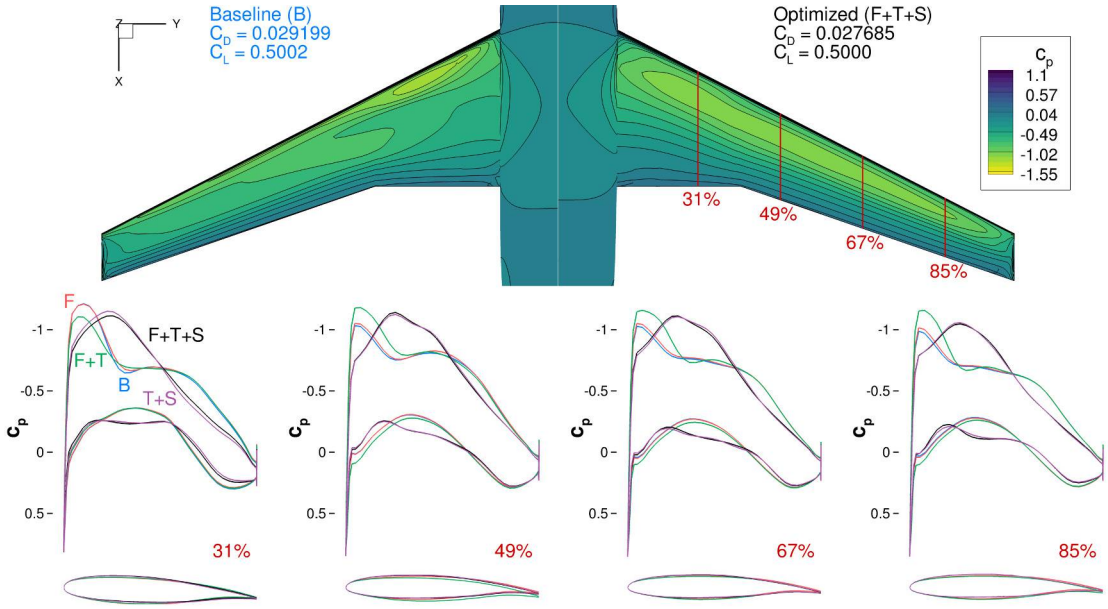


Figure 28. Pressure coefficient slices of all optimized configurations. The optimizer can design wings with smooth pressure distributions as we activate wing shape variables (S). Even though Problems T+S and F+T+S have different design spaces, they show similar airfoils and lift distributions since they converged to practically the same wing design.

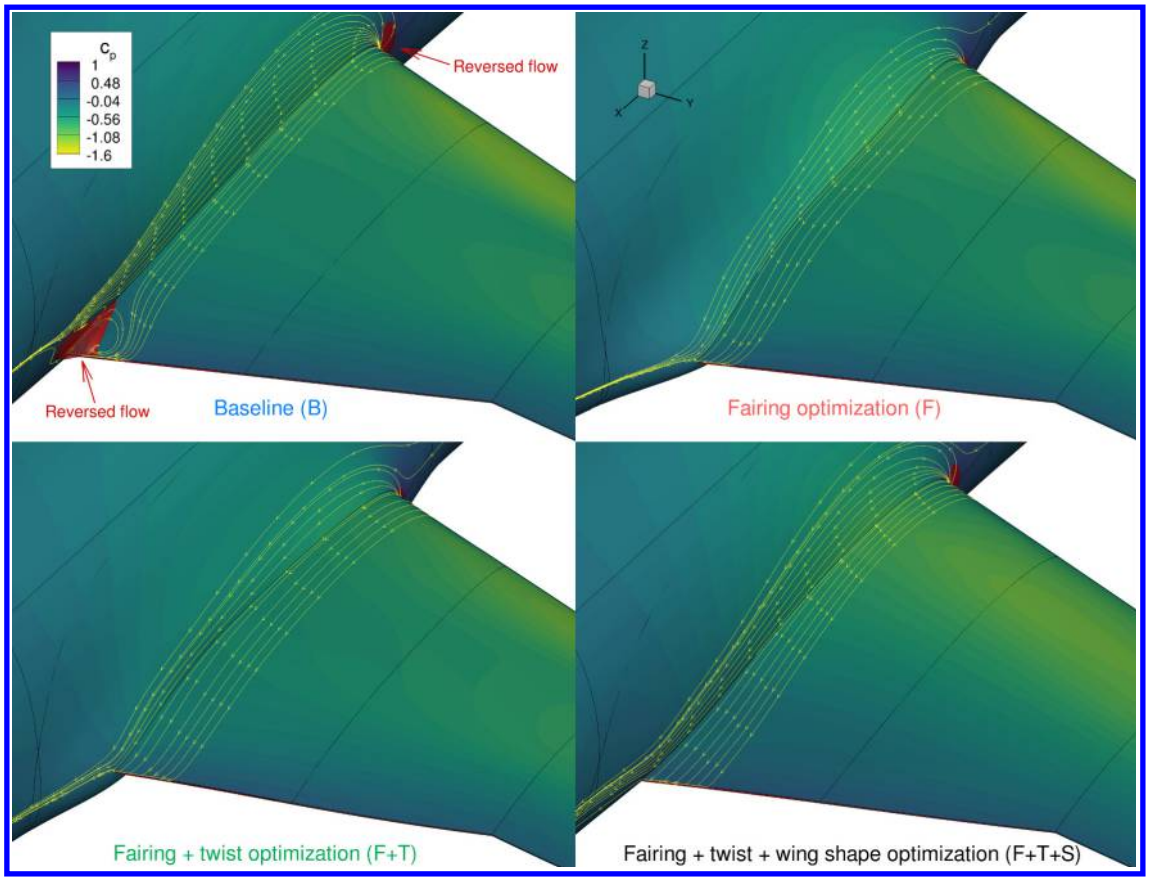


Figure 29. Wing-body junction trailing edge after every optimization problem. Red regions indicate reversed flow. The redesigned fairings eliminated the recirculation bubble in all cases. The wing shape variables achieve smoother chordwise pressure distributions.

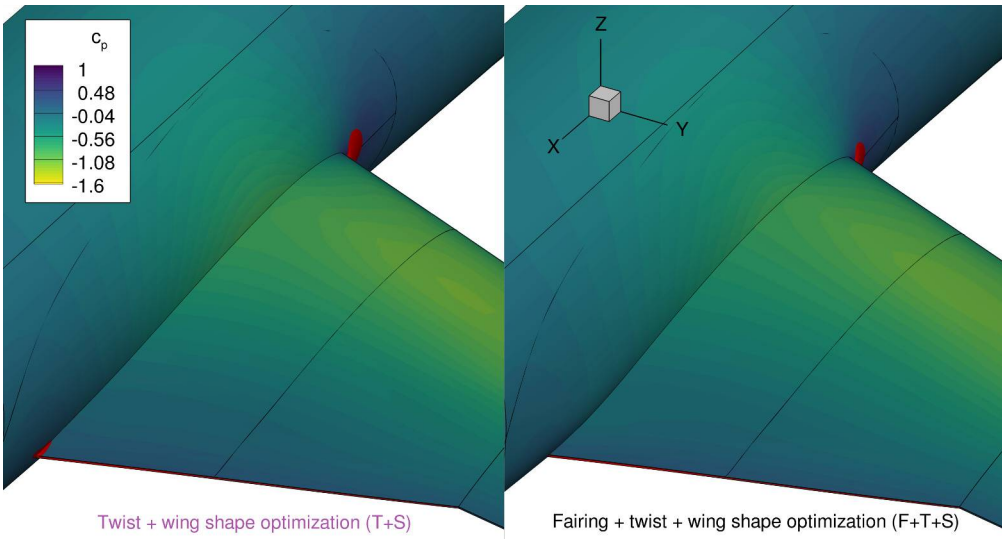


Figure 30. Comparison of the optimized configuration with and without fairing design variables. Optimization without the fairing design variables might still show separated regions (red).

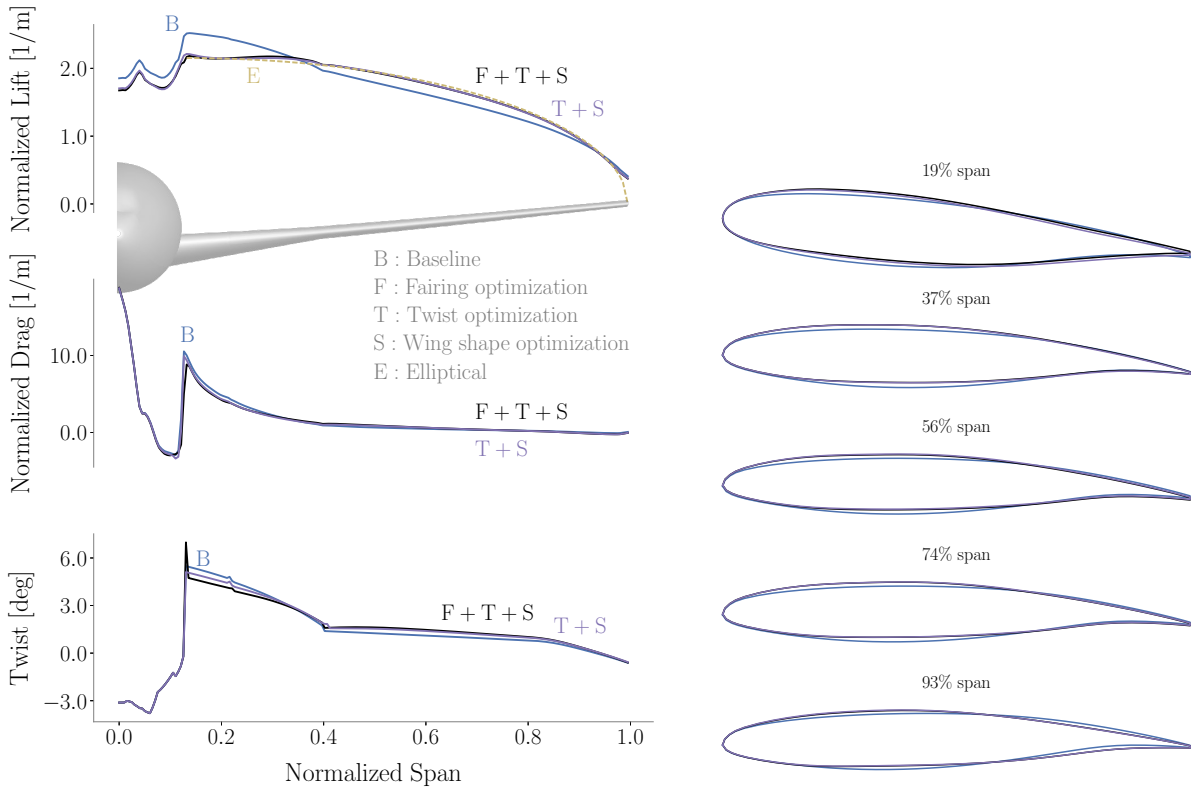


Figure 31. Comparison of lift and drag distributions of the twist and shape optimized configurations with (F+T+S) and without (T+S) fairing variables. The resulting twist and airfoil distributions get progressively more similar as we move towards the tip. The F+T+S configuration has a smaller drag in this region as well, which leads to the improvement seen in Fig. 22.

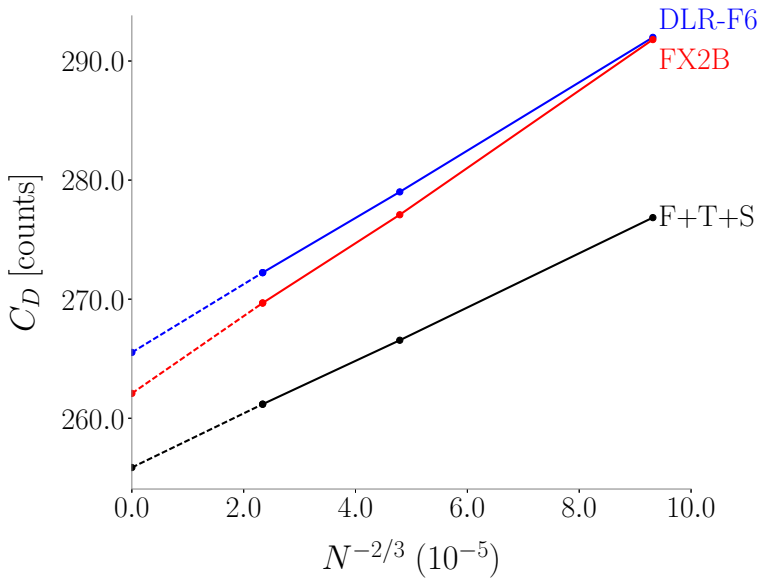


Figure 32. Drag convergence study for the DLR-F6 configurations including the F+T+S optimization case. Blue colors refer to the baseline DLR-F6 configurations, red colors refer to the DLR-F6-FX2B configuration, and black colors refer to the optimized result. All simulations use SA-R-QCR2000 as turbulence model. The improvement provided by optimized design is still present on the finer mesh levels.

VI. Conclusions

We implemented a new geometry module, pySurf, to automate collar mesh generation based on intersections and also to allow independent geometry component manipulation in aerodynamic shape optimization problems. The intersection computation and surface mesh generation are differentiated to allow the use of gradient-based optimization techniques.

We demonstrated that independent component manipulation, mesh warping, and automatic collar mesh generation are feasible for optimization tasks. We noticed the noise introduced to the functions of interest due to the updates in the overset hole cutting and zipper meshes. We verified that the optimizer still manages to improve the design, even though it does not strictly meet the optimality criteria. The noise and discontinuities become an issue in flat regions of the design space, such as near the optimum, as the function variations become inconsistent with the gradients.

We used the new geometry manipulation approach to design the wing-body junction of the DLR-F6. We showed that a fairing-alone optimization might try to solve issues due to the overall configuration, which was unproductive. Adding wing design variables gave more options to the optimizer to solve these issues, thus the fairing design variables can be used to locally adjust the wing-body junction flow. Furthermore, each new design variable group progressively improves the drag of the optimized configuration. The optimizer achieves a reduction of 15 drag counts (5%) compared to the baseline design when all variables are active.

In the future we intend to extend the features in pySurf to automatically handle intersections and collar mesh generation for more complex topologies, such as simultaneous intersections of three components. We also plan to pursue approaches to reduce the noise associated with changes in the overset connectivity, such as increasing the interpolation stencil size, or switching to a frozen connectivity approach by the end of the optimization.

Another possible line of work is to improve the robustness of the hole cutting method to avoid mesh failures during the optimization. Each mesh failure forces the optimizer to reduce its step size, thus increasing the optimization time. A more robust hole cutting also allows the exploration of broader design spaces.

Because the developed framework automatically produces structured overset meshes for complex geometries, we can now study a variety of applications beyond the conventional aircraft configuration. The truss-braced wing (TBW) configuration lends itself well to overset-based studies due to the wing-strut junctions. Optimizing the location of the strut relative to the wing and fuselage using high-fidelity methods has not been done, in part due to the difficulty associated with mesh warping when dealing with multiple components and moving intersections. The use of pySurf may make this application possible now, and a TBW optimization might shed some light on the advantages and drawbacks of this specific configuration.

VII. Acknowledgements

The authors are thankful for support from the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1256260, from NASA through award No. NNX14AC73A–Technical Monitor Tristan Hearn, and also from the Brazilian Air Force.

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562.

References

[1] Lyu, Z. and Martins, J. R. R. A., "Aerodynamic Design Optimization Studies of a Blended-Wing-Body Aircraft," *Journal of Aircraft*, Vol. 51, No. 5, September 2014, pp. 1604–1617. doi:10.2514/1.C032491.

[2] Ivaldi, D., Secco, N. R., Chen, S., Hwang, J. T., and Martins, J. R. R. A., "Aerodynamic Shape Optimization of a Truss-Braced-Wing Aircraft," *Proceedings of the 16th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Dallas, TX, June 2015, AIAA 2015-3436.

[3] Drela, M., "Development of the D8 Transport Configuration," *29th AIAA Applied Aerodynamics Conference*, American Institute of Aeronautics and Astronautics (AIAA), jun 2011. doi:10.2514/6.2011-3970.

[4] Lee, H., Kim, Y., Park, G., Kolonay, R., Blair, M., and Canfield, R., "Structural Optimization of a Joined Wing Using Equivalent Static Loads," *Journal of Aircraft*, Vol. 44, No. 4, 2007, pp. 1302–1308.

[5] Philippe-André, Té, trault, Schetz, J. A., and Grossman, B., "Numerical prediction of interference drag of strut-surface intersection in transonic flow," *AIAA journal*, Vol. 39, No. 5, 2001, pp. 857–864.

[6] Duggirala, R. K., Roy, C. J., and Schetz, J. A., "Analysis of interference drag for strut-strut interaction in transonic flow," *AIAA journal*, Vol. 49, No. 3, 2011, pp. 449–462.

[7] Gur, O., Bhatia, M., Schetz, J. A., Mason, W. H., Kapania, R. K., and Mavris, D. N., "Design Optimization of a Truss-Braced-Wing Transonic Transport Aircraft," *Journal of Aircraft*, Vol. 47, No. 6, 2010. doi:10.2514/1.47546.

[8] Peter, J. E. V. and Dwight, R. P., "Numerical Sensitivity Analysis for Aerodynamic Optimization: A Survey of Approaches," *Computers and Fluids*, Vol. 39, No. 3, March 2010, pp. 373–391. doi:10.1016/j.compfluid.2009.09.013.

[9] Steger, J. L., Dougherty, F. C., and Benek, J. A., "A chimera grid scheme.[multiple overset body-conforming mesh system for finite difference adaptation to complex aircraft configurations]," 1983.

[10] Chan, W. M. and Buning, P. G., "Surface grid generation methods for overset grids," *Computers & fluids*, Vol. 24, No. 5, 1995, pp. 509–522.

[11] Chan, W. M. and Steger, J. L., "Enhancements of a three-dimensional hyperbolic grid generation scheme," *Applied Mathematics and Computation*, Vol. 51, No. 2, 1992, pp. 181–205.

[12] Chan, W. M., Gomez, R. J., Rogers, S. E., and Buning, P. G., "Best practices in overset grid generation," *AIAA paper*, Vol. 3191, 2002, pp. 2002.

[13] Hahn, A. S., "Vehicle sketch pad: a parametric geometry modeler for conceptual aircraft design," *48th AIAA Aerospace Sciences Meeting and Exhibit*, 2010.

[14] Hwang, J. T. and Martins, J. R. R. A., "GeoMACH: Geometry-centric MDAO of Aircraft Configurations with High Fidelity," *Proceedings of the 14th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, Indianapolis, IN, Sept. 2012.

[15] Haimes, R. and Drela, M., "On the construction of aircraft conceptual geometry for high-fidelity analysis and design," *50th AIAA Aerospace sciences meeting including the new horizons forum and aerospace exposition*, 2012, p. 683.

[16] David, L. R. and Sturdza, P., "A rapid geometry engine for preliminary aircraft design," Tech. rep., AIAA-2006-0929, 2006.

[17] Sederberg, T. W. and Parry, S. R., "Free-form Deformation of Solid Geometric Models," *SIGGRAPH Comput. Graph.*, Vol. 20, No. 4, Aug. 1986, pp. 151–160. doi:10.1145/15886.15903.

[18] Kenway, G. K., Kennedy, G. J., and Martins, J. R. R. A., "A CAD-Free Approach to High-Fidelity Aerostructural Optimization," *Proceedings of the 13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, Fort Worth, TX, Sept. 2010, AIAA 2010-9231.

[19] Samareh, J. A., "Survey of Shape Parameterization Techniques for High-Fidelity Multidisciplinary Shape Optimization," *AIAA Journal*, Vol. 39, No. 5, 2001, pp. 877–884.

[20] Bonet, J. and Peraire, J., "An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems," *International Journal for Numerical Methods in Engineering*, Vol. 31, No. 1, 1991, pp. 1–17.

[21] Möller, T., "A fast triangle-triangle intersection test," *Journal of graphics tools*, Vol. 2, No. 2, 1997, pp. 25–30.

[22] Eriksson, L., "Generation of boundary-conforming grids around wing-body configurations using transfinite interpolation," *Aiaa Journal*, Vol. 20, No. 10, 1982, pp. 1313–1320.

[23] Hascoët, L. and Pascual, V., "The Tapenade Automatic Differentiation tool: Principles, Model, and Specification," *ACM Transactions On Mathematical Software*, Vol. 39, No. 3, 2013.

[24] Kenway, G. K. W., Kennedy, G. J., and Martins, J. R. R. A., "Scalable parallel approach for high-fidelity steady-state aeroelastic analysis and adjoint derivative computations," *AIAA journal*, Vol. 52, No. 5, 2014, pp. 935–951.

[25] Chan, W. M., "Best Practices on Overset Structured Mesh Generation for the High-Lift CRM Geometry," *55th AIAA Aerospace Sciences Meeting*, 2017, p. 0362.

[26] Luke, E., Collins, E., and Blades, E., "A fast mesh deformation method using explicit interpolation," *Journal of Computational Physics*, Vol. 231, No. 2, 2012, pp. 586 – 601. doi:http://dx.doi.org/10.1016/j.jcp.2011.09.021.

[27] van der Weide, E., Kalitzin, G., Schluter, J., and Alonso, J., "Unsteady Turbomachinery Computations Using Massively Parallel Platforms," *Proceedings of the 44th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA, Forth Worth, TX, 2006. doi:10.2514/6.2006-421.

[28] Mader, C. A., Martins, J. R. R. A., Alonso, J. J., and van der Weide, E., "ADjoint: An Approach for the Rapid Development of Discrete Adjoint Solvers," *AIAA Journal*, Vol. 46, No. 4, April 2008, pp. 863–873. doi:10.2514/1.29123.

[29] Lee, Y. and Baeder, J. D., "Implicit hole cuttinga new approach to overset grid connectivity," *AIAA paper*, Vol. 4128, 2003, pp. 2003.

[30] Landmann, B. and Montagnac, M., "A highly automated parallel Chimera method for overset grids based on the implicit hole cutting technique," *International Journal for Numerical Methods in Fluids*, Vol. 66, No. 6, 2011, pp. 778–804.

[31] Chan, W. M., "Enhancements to the Hybrid Mesh Approach to Surface Loads Integration on Overset Structured Grids," *AIAA Paper*, Vol. 3990, 2009.

[32] Kenway, G. K., Mishra, A., Secco, N. R., Duraisamy, K., and Martins, J. R. R. A., "An Efficient Parallel Overset Method for Aerodynamic Shape Optimization," *58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2017, p. 0357.

[33] Lyu, Z., Kenway, G. K., Paige, C., and Martins, J. R. R. A., "Automatic Differentiation Adjoint of the Reynolds-Averaged Navier–Stokes Equations with a Turbulence Model," *21st AIAA Computational Fluid Dynamics Conference*, San Diego, CA, Jul. 2013. doi:10.2514/6.2013-2581.

[34] Balay, S., Gropp, W. D., McInnes, L. C., and Smith, B. F., "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries," *Modern Software Tools in Scientific Computing*, edited by E. Arge, A. M. Bruaset, and H. P. Langtangen, Birkhäuser Press, 1997, pp. 163–202.

[35] Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H., "PETSc Web page," http://www.mcs.anl.gov/petsc, 2016.

[36] Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H., "PETSc Users Manual," Tech. Rep. ANL-95/11 - Revision 3.7, Argonne National Laboratory, 2016.

[37] Saad, Y. and Schultz, M. H., "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on scientific and statistical computing*, Vol. 7, No. 3, 1986, pp. 856–869.

[38] Gill, P., Murray, W., and Saunders, M., "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Journal on Optimization*, Vol. 12, No. 4, 2002, pp. 979–1006. doi:10.1137/S1052623499350013.

[39] Lyu, Z., Kenway, G. K. W., and Martins, J. R. R. A., "RANS-based Aerodynamic Shape Optimization Investigations of the Common Research Model Wing," *Proceedings of the AIAA Science and Technology Forum and Exposition (SciTech)*, National Harbor, MD, January 2014. doi:10.2514/6.2014-0567, AIAA 2014-0567.

[40] Brooks, T. R., Kennedy, G., and Martins, J. R. R. A., "High-fidelity aerostructural optimization of a high aspect ratio tow-steered wing," *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2016, p. 1179.

[41] Vassberg, J. C., DeHaan, M. A., Rivers, S. M., and Wahls, R. A., "Development of a common research model for applied CFD validation studies," *AIAA paper*, Vol. 6919, 2008, pp. 2008.

[42] Hascoët, L., Vázquez, M., and Dervieux, A., "Automatic differentiation for optimum design, applied to sonic boom reduction," *Computational Science and Its ApplicationsICCSA 2003*, 2003, pp. 976–976.

[43] Brodersen, O., "Drag prediction of engine-airframe interference effects using unstructured Navier-Stokes calculations," *Journal of aircraft*, Vol. 39, No. 6, 2002, pp. 927–935.

[44] Laflin, K. R., Klausmeyer, S. M., Zickuhr, T., Vassberg, J. C., Wahls, R. A., Morrison, J. H., Brodersen, O. P., Rakowitz, M. E., Tinoco, E. N., and Godard, J.-L., "Data summary from second AIAA computational fluid dynamics drag prediction workshop," *Journal of Aircraft*, Vol. 42, No. 5, 2005, pp. 1165–1178.

[45] Vassberg, J., Tinoco, E., Mani, M., Brodersen, O., Eisfeld, B., Wahls, R., Morrison, J., Zickuhr, T., Laflin, K., and Mavriplis, D., "Summary of the third AIAA CFD drag prediction workshop," *45th AIAA Aerospace Sciences Meeting and Exhibit*, 2007, p. 260.

[46] Lee, B. and Kim, C., "Aerodynamic shape optimization using discrete adjoint formulation based on overset mesh technique," *European Conference on Computational Fluid Dynamics*, 2006.

[47] Liao, W. and Tsai, H. M., "Aerodynamic shape optimization on overset grids using the adjoint method," *International journal for numerical methods in fluids*, Vol. 62, No. 12, 2010, pp. 1332–1356.

[48] Brezillon, J. and Dwight, R. P., "Applications of a discrete viscous adjoint method for aerodynamic shape optimisation of 3D configurations," *CEAS Aeronautical Journal*, Vol. 3, No. 1, 2012, pp. 25–34.

[49] Vassberg, J., Sclafani, A., and DeHaan, M., "A wing-body fairing design for the DLR-F6 model: a DPW-III case study," *23rd AIAA Applied Aerodynamics Conference*, 2005, p. 4730.

[50] Tinoco, E. N., Brodersen, O., Keye, S., and Laflin, K., "Summary of Data from the Sixth AIAA CFD Drag Prediction Workshop: CRM Cases 2 to 5," *55th AIAA Aerospace Sciences Meeting*, 2017, p. 1208.

[51] Spalart, P. and Allmaras, S., "A one-equation turbulence model for aerodynamic flows," *30th aerospace sciences meeting and exhibit*, p. 439.

[52] Coder, J. G., Pulliam, T. H., Hue, D., Kenway, G. K., and Sclafani, A. J., "Contributions to the 6th AIAA CFD Drag Prediction Workshop Using Structured Grid Methods," *55th AIAA Aerospace Sciences Meeting*, 2017, p. 0960.

[53] Dacles-Mariani, J., Zilliac, G. G., Chow, J. S., and Bradshaw, P., "Numerical/experimental study of a wingtip vortex in the near field," *AIAA journal*, Vol. 33, No. 9, 1995, pp. 1561–1568.

[54] Spalart, P. R., "Strategies for turbulence modelling and simulations," *International Journal of Heat and Fluid Flow*, Vol. 21, No. 3, 2000, pp. 252–263.

[55] Vassberg, J., Brodersen, O., Wahls, R., Zickuhr, T., Mavriplis, D., Tinoco, E., Mani, M., Levy, D., and Morrison, J., "Comparison of NTF Experimental Data with CFD Predictions from the Third AIAA CFD Drag Prediction Workshop," *26th AIAA Applied Aerodynamics Conference*, 2008.