AIAA SciTech
5-9 January 2015, Kissimmee, Florida
56th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference

Self-Designing Parametric Geometries

András Sóbester*
University of Southampton, Southampton, Hampshire, SO16 7QF, United Kingdom

The thesis of this paper is that script-based geometry modelling offers the possibility of building 'self-designing' intelligence into parametric airframe geometries. We show how sophisticated heuristics (such as optimizers and complex decision structures) can be readily integrated into the parametric geometry model itself using a script-driven modelling architecture. The result is an opportunity for optimization with the *scope* of conceptual design and the *fidelity* of preliminary design. Additionally, the proposed 'self-design' philosophy of using an integrated design heuristic to construct much of the geometry is a good mechanism for de-constraining the design space, as we can take the design variables as a starting point from which we generate a feasible design, wherever possible. We illustrate these ideas through the parametric geometry model of a twin-engined light aircraft.

I. Introduction

Most aircraft conceptual design processes still rely heavily on largely empirical models, derived from legacy design data. In particular, weight estimates are often obtained in this way in the conceptual phase of the design process. A typical such weight model will be a curve fit in a small number of dimensions, such as wing span, payload, range and other high level parameters.

There is much to commend this approach. First, it is extremely fast, as the computational cost of evaluating such models is usually negligible. Second, it is based on 'real' historical aircraft data. However, it has some very serious drawbacks too – here are three of the most significant:

- Its validity decays rapidly as we stray into the extrapolation domains of the curve fits: a weight model based on 'tube and wing' type airliner data will almost certainly deceive the conceptual design of an airframe of a more unusual topology (e.g., a blended wing body or an aircraft with over-the-wing engines).
- A related issue is that such models tend to only respond to changes in a small, fixed set of design variables, thus, once again, limiting the scope of the conceptual design exploration.
- Models based on legacy data will, by definition, be oblivious to new technologies. Typical blind spots for these models are aircraft with a substantially greater proportion of composite materials or *'more electric'* aircraft (not to mention extreme cases, such as *electrically powered* aircraft).

The solution is, of course, Multi-disciplinary Design Optimization (MDO) relying on numerical (or, perhaps, experimental) performance analysis and weight estimation. There is, however, a major stumbling block in the path of effective MDO deployment at the conceptual level. Multi-disciplinary analysis, of accuracy that is sufficiently high for the reliable guidance of an MDO process, requires *geometry* of comparable fidelity and detail; and that, at present, is a rare commodity.

Consider the following scenario. We have the parametric outer mould line (OML) description of an aircraft, which allows the exploration of a broad range of geometries. We may use a flow solver to compute the aerodynamic derivatives – we can then compute an objective and/or a (set of) constraint(s). However, this stage of the design process is all about understanding high level trade-offs and these are almost always multi-disciplinary.

Starting with the wing, we may ask: at what point does the extra structural mass outweigh the aerodynamic advantage of increasing the span? Should the wing be braced and, if so, approximately where should

*Associate Professor, Aeronautics, Astronautics and Computational Engineering Unit, SMAIAA.

the bracing point be located for the best balance between structural mass reduction and added pressure drag? Would a long, slender fuselage or a short, double decker provide the most efficient way to house passengers and cargo? In either case, what is the cost of increasing comfort by varying the available legroom? These are all questions that require some level of additional design detail before the necessary analysis can be completed.

Given infinite resources (human and computational) the ideal solution would be to design a complete, feasible aircraft (where possible) inside each putative outer mould line thrown up by the optimizer and compute all their performance metrics – clearly not affordable in all but the most trivial cases.

Short of this, however, one may consider *automating* some design tasks usually associated with preliminary design, thus producing the geometry required by the various disciplines and allowing sophisticated trade-off analysis at the conceptual design stage.

Of course, multi-disciplinary geometry models are not a new idea. What we propose here, however, is *automated geometrical design*, wherein the optimizer drives a set of high level design variables and a design recipe attempts to work with this baseline definition and make decisions on the values of large numbers of further variables. In order to guard against the *curse of dimensionality*, these lower level variables are not exposed to the optimization process.

At the heart of the practical implementation of this idea lies a fundamental decision of software engineering. A conventional hierarchical CAD geometry model does not, generally, lend itself to the self-design approach. Here we propose to integrate the design algorithms as tightly as possible into the fabric of the geometry model, by making the automatically designed subsystems part of the parametric geometry itself – in other words, the parametric geometry model incorporates the heuristics for equipping it with detail beyond that implied directly by the definition of the design variables. This is only practical through *scripted geometry modelling*.

For the example to be discussed in the next section we opted for *Rhinoceros 5/OpenNURBS* as the geometry engine and *Python* as the scripting language. The concepts discussed in this paper are independent of the scripting language and the geometry engine chosen, but there is much to commend this particular combination.

OpenNURBS offers the sophisticated surface geometry formulations that are the *sine qua non* of effective outer mould line design and Rhino's high performance graphics make it a very effective means of visualising and the analysing the design. As for Python as the scripting language that encodes the design recipes, its power lies in its portability, its large user community and, of course, in the fact that it is free[a]. It is also a convenient portal to numerical libraries[b].

II. AirCONICS – Aircraft Configuration Through Integrated Cross-Disciplinary Scripting

In order to illustrate the concept of self-designing geometries, we shall use a collection of parametric Python objects and methods designed for aircraft geometry generation in Rhinoceros, named *AirCONICS* (Aircraft Configuration Through Integrated Cross-Disciplinary Scripting). This freely available toolset[c] was designed following the principles advocated in Refs 1 and 2.

Of those principles, the most germane to the idea of self-designing geometries is probably that of building high level, aircraft surface specific objects ('object' in the 'in the object oriented programming' sense), which depend on a set of variables, as well as on a set of user-defined functions. The two most important features here are *geometry definition via scripting* and *functions as design variables*.

The user interface of a traditional geometry design tool at most levels of the design process is usually a menu system and a sometimes interactive graphical view of the object being designed (all CAD systems work in this way or at least have such a mode). Perhaps the powerful OpenVSP[3] is the most prominent such tool at present in aircraft engineering.

The alternative is the 'geometry as computer code' principle and a new wave of such tools is beginning to emerge – see, for instance, Hwang and Martins's GeoMACH[4] or Gagnon and Zingg's GENAIR[5] (the RAGE geometry engine[6] also offers a text file based means of defining how geometrical building blocks are

[a] As is the MacOS X version of Rhinoceros.

[b] That said, at the time of writing, the two main such libraries, *NumPy* and *SciPy*, are only supported by Rhino on some platforms.

[c] Available for download at *www.aircraftgeometry.codes*.

assembled). At the cost of a steeper learning curve, this approach exploits the almost limitless flexibility with which scripting-based systems can access collections of generic primitives (such as OpenGL, OpenNURBS, etc.). The engineer here is exposed to a lower level of the 'programming' (as opposed to 'drawing') of the geometry, an approach which enables more bespoke mathematical formulations. Most importantly though, from the point of view of the main thrust of this paper, scripting allows the ability to *weave engineering knowledge and judgement into the fabric of the parametric geometry model itself* and this is why in AirCONICS we adopted the scripting methodology.

The AirCONICS toolset also follows the other principle mentioned above, namely that it increases geometrical flexibility by including functions as arguments to geometrical objects. For instance, the generic lifting surface object within AirCONICS accepts traditional scalar arguments, such as span, root chord length, projected area (in itself a higher level variable, driving a set of lower level parameters through a built-in optimization process[d]), etc., but it also accepts user defined function arguments for features such as spanwise dihedral variation, spanwise airfoil shape variation, etc. In fact, the latter two are essential for the definition of a wing such as that of the aircraft shown in Figure 1, where the spanwise dihedral function allows us to define the transition from the wing to the vertical fin and the airfoil transition function defines the way in which the wing is gradually and smoothly de-cambered during this transition (the vertical 'fin' section has a symmetrical profile).

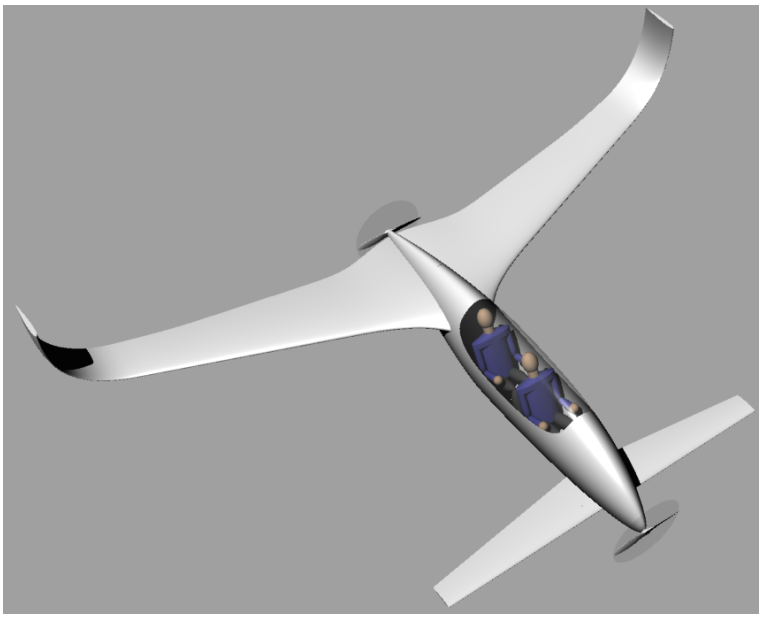

Figure 1. Two-seat, twin-engine canard, with a pusher and a tractor engine placed in the symmetry plane of the aircraft to eliminate yawing moments in case of loss of power.

III. An Illustrative Case Study

In order to illustrate the concept of a 'self-designing geometry', as outlined in Section I, let us consider the following design task. A 'late conceptual' level parametric geometry is required to represent a light aircraft with a canard layout in the sub-450kg maximum take-off weight category, powered by two rotary engines nominally rated at 50HP each. In order to ensure benign handling characteristics in case of the loss of an engine, the two engines are to be mounted in a tandem arrangement (this, and the choice of the canard layout, are assumed to be the outcomes of an earlier phase of the conceptual design process). Figure 1 shows an example of such an aircraft.

The only input to the geometry building process is a five element design vector defining the shape of the outer mould line of the fuselage of the light aircraft. We use the generic fuselage model included in AirCONICS, which is defined as having three sections: a tapering forward section, a parallel sided central

[d]An optimizer built into the fabric of the geometry to allow the self-design of the wing – a feature that can only be implemented in a scripted geometry context.

section and a tapering tail section (the profile shapes corresponding to each section are fixed). The design variables allow for the control of the relative extents of these sections. As illustrated in Figure 2, design variables one and two control the relative lengths of the nose and the tail section respectively; variables three, four and five control the scale of the geometry in the x, y and z direction.

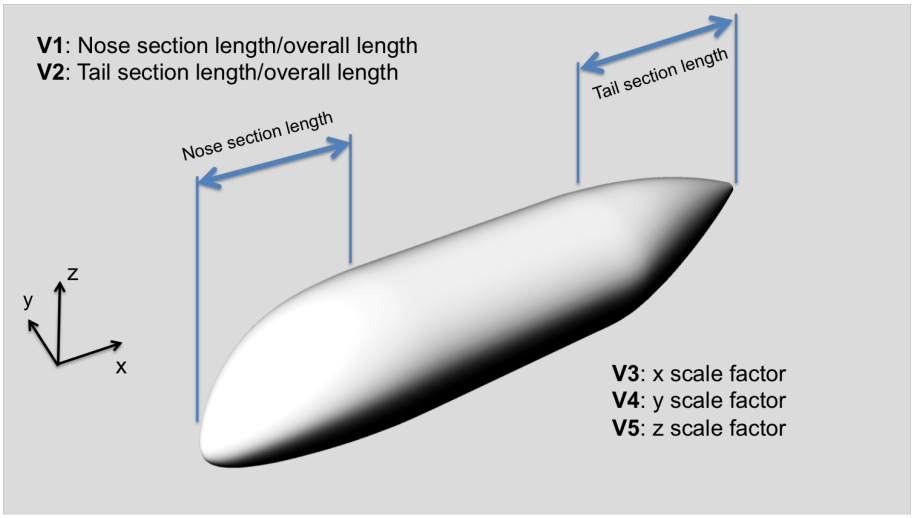

Figure 2. The five high level design variables of the fuselage geometry, which serves as the starting point of the self-design process.

Figure 3 depicts the optimal design workflow, including the place of this top level parametric geometry. Up to the point where variables $V1$ through $V5$, as produced by, for example, an optimizer, are converted into the outer mould line geometry of the fuselage, the process is the same as optimization loops centered on conventional geometry models.

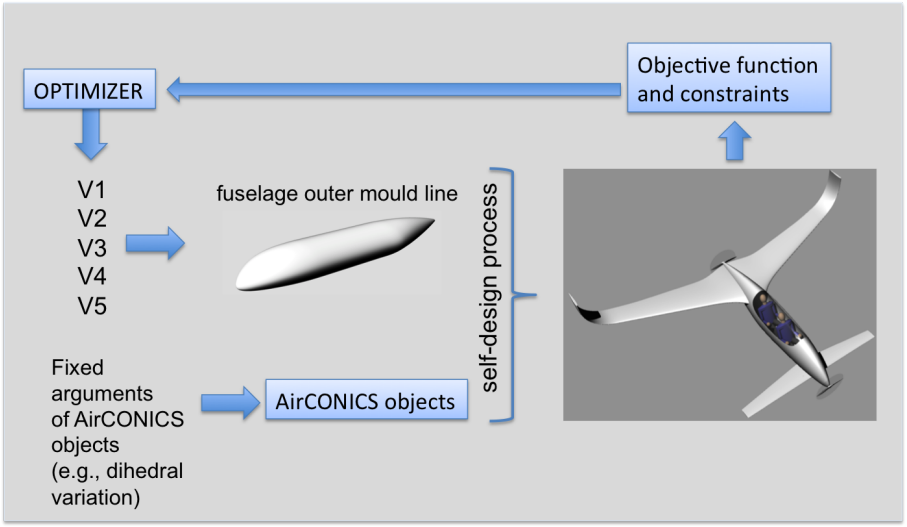

Figure 3. Optimization workflow built around the self-designing geometry of the light aircraft example.

As seen in Figure 3, this is where the proposed workflow deviates from the standard approach. Using AirCONICS objects (such as lifting surfaces and models of 95 percentile male humans to serve as the occupants of the aircraft) *all remaining features of the conceptual level model of the whole aircraft are automatically generated by the scripted parametric geometry model itself.* The Python code that accomplishes this for the light aircraft case study is, essentially, an implementation of the following **high level design rules** (refer to Figures 4 and 5):

1. The two occupants are to be 2020 projections of 95 percentile US males.

2. When seated, no body part of either occupant should be in contact with the inside surface of the fuselage.

3. If space allows, the pilot and the passenger should be seated side by side.

4. If only a tandem arrangement is possible (as in, for example, Figure 1), there should be a clearance between the passenger knees and the pilot's back of at least 0.1m.

5. Both occupants should be reclined in normal operation at 25 degrees to the vertical.

6. The tractor engine should be placed as near to the front of the fuselage as possible to minimize the required drive shaft length.

7. The pusher engine should be placed as near to the stern as possible to minimize the required drive shaft length.

8. The canard is to be positioned such that the quarter chord point of its generating airfoil should line up with the plane of the attachment flange of the tractor engine. This is to allow the implementation the structural philosophy depicted in Figure 5, wherein a multi-function forward structural hub acts as tractor engine support, canard attachment point and main forward frame.

9. A similar structural hub is to be placed at the back, with the two connected by a central main longeron at the lowest point of fuselage, in its symmetry plane (Figure 5).

10. In the interest of good visibility, the wing leading edge should intersect the fuselage aft of the passenger's head.

11. The trailing edge of the wing is to intersect the wing just forward of the pusher propeller hub assembly.

12. The combined projected area of the wing and the canard should be 12m^2 (this is assumed to result from an early conceptual design stage analysis of power to weight versus wing loading domain constraints). The starting canard/main wing area ratio is 1:3 – this can be subsequently fine tuned through a stability calculation module that can also be integrated into the geometry.

13. The canopy is to be sized and shaped to balance two competing needs: good visibility and structural efficiency (the larger the canopy, the better the visibility is likely to be, but the less efficient the semi-monocoque structure of the aircraft is likely to become).

14. In order to maximise visibility, for a given longitudinal station, the occupants are to be placed as high up as possible (without their heads contacting the canopy, of course).

15. The occupants are to be positioned as far forward as possible to maximize the space available for the wing root chord (the fuselage length being defined by the top level design variables).

Figure 6 shows a selection of instances of the geometry resulting from a range of design variable values (eight diverse $\{V1, \ldots, V5\}$ vectors). The diversity of these design vectors is reflected in the range of design decisions taken by the self-design component of the geometry model: note, for example, the variation in wing aspect ratios (driven by the varying amount of wing root space available on the fuselage), or the fact that the varying amount of lateral space determines the side by side or tandem arrangement of the occupants. While not visible in these renderings, each of these designs also implements all the rules listed above, so they will feature differently positioned engines, etc.

Clearly, it is possible that some of these aircraft end up being unfeasible and an additional constraint evaluation (see the top right box in Figure 3) will be required alongside the objective function to filter these out. That said, one of the advantages of building a design algorithm into the geometry construction process is that constraint violations can be largely designed out and the optimization process will then be able to drive the design simply through the objective function[e].

[e]This is somewhat akin to the old idea of including a geometry repair step into the optimization loop – we proposed such a system in Ref 7. The evolutionary computing community likes to draw a parallel here with *Lamarckian learning*.

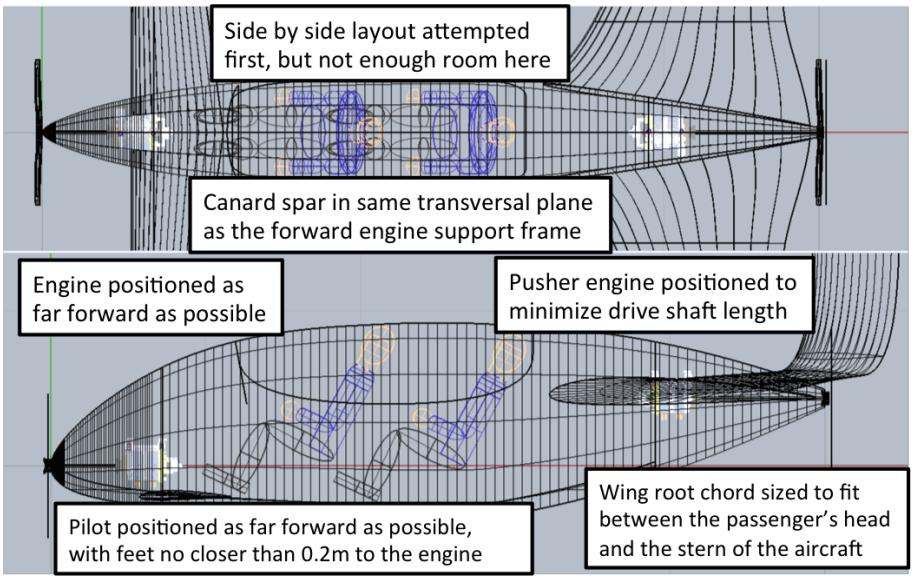

Figure 4. A selection of the high level design rationale embedded in the self-designing geometry of the light aircraft model.

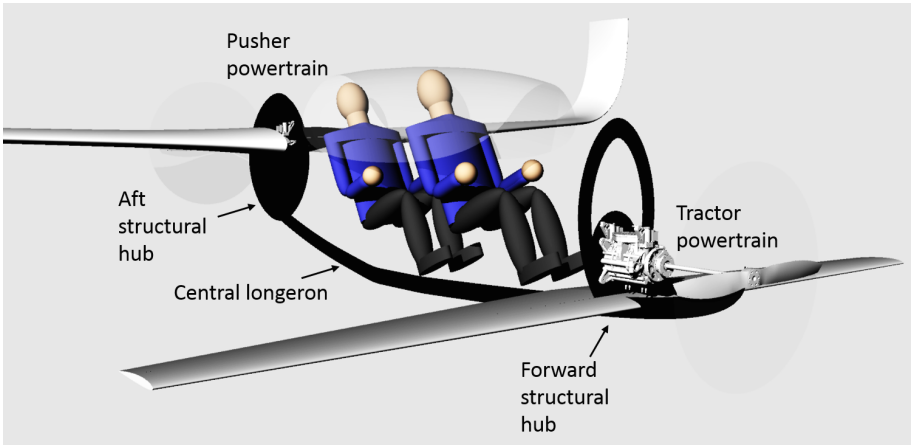

Figure 5. Sketch of the structural philosophy of the light aircraft concept at the centre of the 'self-designing' parametric geometry case study presented here.

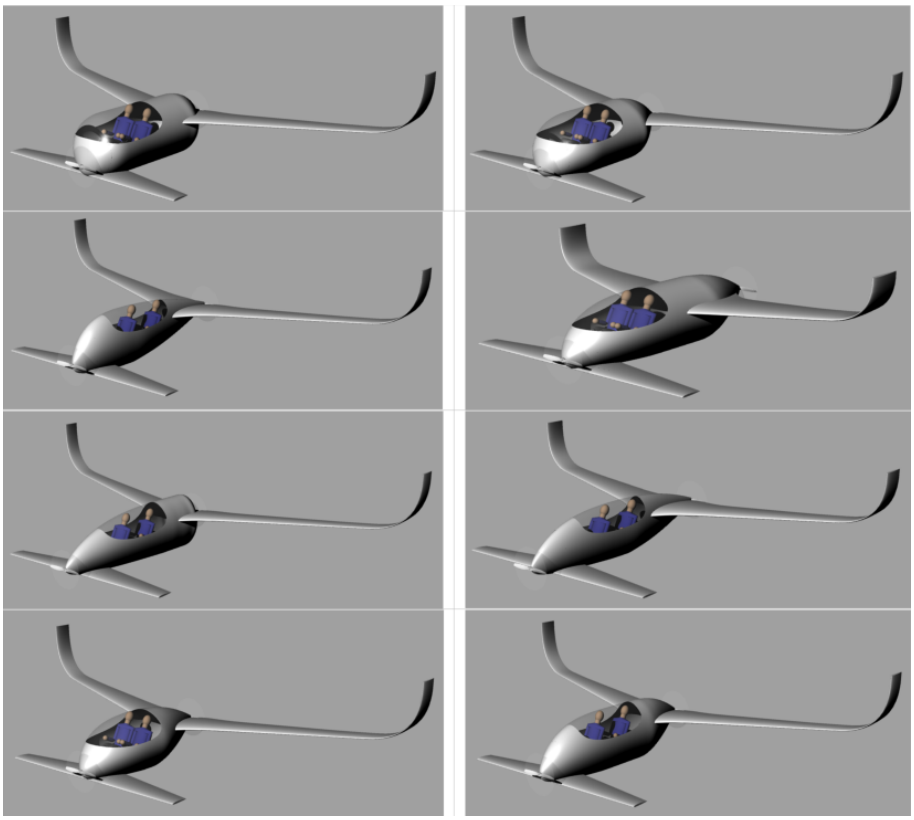

Figure 6. A selection of instances of the self-designing two-seat light aircraft geometry.

IV. Looking Towards the Later Stages of the Design Process

The exact scope, fidelity and level of detail of each of the classical stages of the design process varies from one design team to another or one product (class) to another, but the respective philosophies of these phases do not.

The *conceptual* design phase decides upon the layout (external and internal topology) and the working principles (e.g., type of propulsion system and airframe configuration to be used at various phases of the flight). Within the freedoms afforded by the selected topology, *preliminary* design selects the higher level design variables and freezes the outer mould line, as well as the major structural components and partitions the airframe amongst the systems claiming space within it. Finally, the *detail* design stage populates these partitions and fleshes out the fine detail, all the way up to tolerances and manufacturing process minutiae.

We need to recognize the disruptive nature of the geometry modelling process described here and one of the key aspects of this is that it need not, in principle, fit into the classical template above. While clear technological breaks are usually evident between the three main phases of the standard design process – most pertinently, they are usually conducted on three different geometry modelling tools and they each have their own (sometimes tacit) parameterization – here we could simply slide up and down a continuum of design process scope and comlexity.

The example we described here happens to be pitched at the late conceptual stage, but moving up the design process would require no paradigm shift, design hand-over or design tool set swap – simply an increase in the level of detail. For example, the routine that places the 'engine support frame' – a simple hoop here – up against the support flange of the engine, could be substituted like-for-like with a scripted frame design process that makes fastener catalogue choices and makes a series of decisions driven by simple stress calculations (or sophisticated Finite Element solves for that matter – not that hard to integrate into the Python script). This would not be easy, perhaps as complex a script as the entire geometry building code used here for the whole airframe, but its fundamental principle would be the same: a script that encodes a design recipe operating within the constraints defined by a high level variable set (in this case the five basic

outer mould line shape definition variables).

Incidentally, up to this point we have considered outer mould lines that automatically design the rest of an aircraft, but is this necessarily the best direction or is it worth asking the question: which part of the design do we parameterize and which part do we design automatically? Do we parameterize the outer mould line and create algorithms to 'fill it' with structure, payload, etc. or do we parameterize the packaging of the payload (whether it is an instrument or cargo bay or a passenger cabin) and build algorithms to 'wrap' an automatically generated outer mould line around it? Or is there a third way, a type of hybrid between the two? In any case, the driving principle remains that a high level optimizer operates on a relatively small set of variables and a design recipe builds the parts of the design that are not directly exposed to the optimization.

By this point, the reader will have noted a drawback in the 'self-designing geometries' philosophy, namely around the issue of design task complexity. Take the inclusion of a simple sub-system in the design, for instance, a canopy latching mechanism. Given a frozen general canopy design, integrating a latching system might be trivial, whereas writing a generic latch design code able to respond to all canopy shapes and sizes may be much harder. The payoffs of the latter approach could, however, make the effort worthwhile: the resulting design is far more likely to optimize whatever cost function is pursued, derivative designs will be very easy to generate and the code will forever serve as a complete and unequivocal design audit trail. Such bonuses are worth having even at the cost of major organisational disruption.

References

[1]Sóbester, A., "Four Suggestions for Better Parametric Geometries," *10th AIAA Multidisciplinary Design Optimization Conference, National Harbor, MD*, 2014.

[2]Sóbester, A. and Forrester, A. I. J., *Aircraft Aerodynamic Design: Geometry and Optimization*, Wiley, 2015.

[3]Hahn, A., "Application of Cart3D to Complex Propulsion-Airframe Integration with Vehicle Sketch Pad," *AIAA-2012-547*, 2012.

[4]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*, 2012.

[5]Gagnon, H. and Zingg, D. W., "Geometry Generation of Complex Unconventional Aircraft with Application to High-Fidelity Aerodynamic Shape Optimization," *AIAA 2013-2850*, 2013.

[6]Suwaratana, D. L. and Rodriguez, D. L., "A More Efficient Conceptual Design Process Using the RAGE Geometry Modeler," *AIAA 2011-159*, 2011.

[7]Sóbester, A. and Keane, A. J., "Supervised Learning Approach to Parametric Computer-Aided Design Geometry Repair," *AIAA Journal*, Vol. 44, No. 2, 2014/12/01 2006, pp. 282–289.