Universidade de Aveiro
2021

Rodrigo Rocha
Lopes da Fonseca

Algorithmic design for customizable products with additive manufacturing

Design algorítmico para produtos customizáveis
com fabrico aditivo

Universidade de Aveiro
2021

Rodrigo Rocha Lopes da Fonseca

Algorithmic design for customizable products with additive manufacturing

Design algorítmico para produtos customizáveis com fabrico aditivo

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecanica, realizada sob orientação científica de João Alexandre Dias de Oliveira, Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro, e de Victor Fernando Santos Neto, Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro.

Este trabalho foi apoiado pelos projetos UIDB / 00481/2020 e UIDP/00481/2020 - FCT - Fundação para a Ciência e a Tecnologia; e CENTRO-01-0145-FEDER-022083 - Programa Operacional Regional do Centro Portugal (Centro2020), no âmbito do Acordo de Parceria PORTUGAL 2020, através do Fundo Europeu de Desenvolvimento Regional.

agradecimentos / acknowledgements

Os meus sinceros agradecimentos ao meu orientador, Professor João Alexandre Dias de Oliveira, e ao meu co-orientador, Professor Victor Fernando Santos Neto, por me guiarem durante este processo, pela sua disponibilidade, acompanhamento, rigor, boa disposição, apoio e por me ajudarem a desenvolver um espírito crítico no desenvolvimento tanto neste processo como para a minha futura carreira.
Á minha mãe Natália, ao meu irmão Francisco e á minha madrinha Lúcia, os meus pilares, por me apoiarem no meu percurso pela Universidade, de se certificarem de que era feliz enquanto o fazia, e de, principalmente, me aturarem.
Aos meus amigos, tanto da Universidade como não, em especial ao Micael, ao José Carlos, ao Bruno, ao Filipe e ao Eduardo, pela companhia, apoio, por partilharem o seu conhecimento comigo, por me ajudarem a crescer como pessoa e como engenheiro. Por todas as aventuras durante o meu percurso académico, e por me fazerem sentir em casa, estando tão longe da minha.
Á minha namorada Clarisse, pelo apoio, carinho, dedicação, por me fazer querer ser melhor para mim mesmo, por estar na minha vida e por me fazer feliz.
A todos com quem cruzei caminhos durante a Universidade, mesmo os que, por pena minha, já não tenho contacto, por partilharem os momentos comigo e por me ajudarem a desenvolver como pessoa e engenheiro.

keywords

Algorithmic Design; Parametric Design; Generative Design; Additive Manufacturing; Customizable Products; Scripting; Fusion360

abstract

The constant development of computer technologies led to novel ways of designing products in order to adapt to an always changing industry and customer requirements. A contribution to this development is the greater of availability and quality of additive manufacturing technologies, which brought new possibilities of tailoring products to the customers needs, due to its digital nature, wide range of available materials, low cost, reduced production times and ability to manufacture geometries of increased complexity.

Two approaches, parametric and generative design, have gained traction in recent years. They seek to assist in the design process, by assisting with alternative exploration, and tailoring products to certain specifications, utilizing computing power as a medium to assist in the decision making process. A third approach, algorithmic design, has received attention in recent years mostly in the field of architecture. It proposes utilizing scripts, either through a text or visual-based programming language as algorithms, to form a basis for inferring knowledge, interfacing with other tools and simplifying the process of product customization through customized user interfaces.

In the present work, the concepts of parametric, generative and algorithmic design are reviewed in the light of engineering design, to ascertain the activities and processes which constitute them, as well as the relationship between them. With algorithmic design as the focus, the scripting capabilities of Autodesk's Fusion360 are used to explore shape creation and the possibility of added customization through custom user interfaces, while also inferring knowledge from additive manufacturing and outlying the benefits of integrating such an approach inside a software tool's ecosystem.

A novel methodology is proposed to address a design problem using algorithmic design, by creating drone models based on user inputs through a user interface. The methodology focuses on simplifying design exploration for the early stages of a typical engineering design process, and prepares the model for the final shape creation by integrating other software tools, namely, Autodesk's Generative Design.

This work scratches only the surface of algorithmic design, and does not go into deterministic approaches as much as possible, as a way to demonstrate the possibilities it entails without being limited by specific approaches. Therefore, it serves as a basis for future works which seek to explore alternatives or optimize solutions using the methodology as a basis.

palavras-chave

Design Algorítmico; Design Paramétrico; Design Generativo; Fabrico Aditivo; Produtos Customizáveis; Programação; Fusion360

resumo

O constante desenvolvimento de tecnologias computacionais originou novas maneiras de desenhar produtos para se adaptarem a uma industria em constante mudança. Uma contribuição para este desenvolvimento é a maior disponibilidade e qualidade de tecnologias de fabrico aditivo, que devido á sua natureza digital, grande variedade de materiais disponiveis, baixo custo, tempo de fabrico reduzido e capacidade de fabricar gemotries de elevada complexidade, proporcionaram novas possibilidades de adaptar produtos para as nessecidades do consumidor.
Duas abordagens, design paramétrico e design generativo, têm ganho relevância em anos recentes. Ambas propõem auxiliar no desenvolvimento de um design, ao auxiliar na exploração de alternativas e ao adaptar produtos a especificações do consumidor, utilizando poder de computação como intermediário para assistir no processo de tomar decisões. Uma terceira abordagem, design algorítmico, tem recebido atenção principalmente no ramo da arquitetura. Esta abordagem propõe utilizar scripts, tanto através de uma linguagem de programação visual ou de uma textual como algoritmos, os quais formam uma base para inferir conhecimento, ligar a outras ferramentas de software, e simplificar o processo de customização do produto através de interfaces de customização.
No presente trabalho, os conceitos de design paramétrico, generativo e algorítmico são revistos á luz de design em engenharia, para avaliar as atividades e processos que os constituem, assim como a relação entre eles. Com design algorítmico como foco, as capacidades de programação do Fusion360 da Autodesk são usadas para explorar a criação de formas e a possibilidade de adicionar customização através de interfaces de utilizador customizadas, ao mesmo tempo inferindo conhecimento sobre fabrico aditivo e realçando os benefícios desta abordagem quando integrada no ecossistema de uma ferramenta de software.
Uma metodologia é proposta para responder a problemas de design utilizando design algorítmico, criando modelos de drones baseados em inputs do utilizador através de uma interface. Esta foca-se em simplificar a exploração de alternativas nos primeiros estágios de um processo de design de engenharia típico, e prepara o modelo para a criação de forma final ao integrar outras ferramentas de software, nomeadamente, o Generative Design da Autodesk.
Este trabalho é apenas toca na superfície das abordagens de design algorítmico, evitando ao máximo abordagens determinísticas, como forma de demonstrar as suas possibilidades sem limitá-la a aplicações específicas. Desta forma, serve de base para futuros trabalho que procurem explorar alternativas ou otimizar soluções usando a metodologia como base.

Contents

List of Figures

Part I

Theoretical Foundation

Chapter 1

Introduction

1.1 Framework and Objectives

Algorithmic Design (AD) has been the subject of research within the field of architecture for some time, usually paired with the use of Parametric Design (PD) and Generative Design (GD). The relative novelty of the concept eventually led to some ambiguity in its definition, where the line between AD, PD and GD became blurred. What is most agreed upon is that AD involves a great deal of scripting to assist with the early stages of design exploration.

Scripted approaches have several benefits. In essence, they allow the user to customize the experience, enabling the design to be made customizable by an end user through the use of an interface, which will guide the user through the design process. Additionally, interfacing can also be done between software tools, allowing the designer to connect different tools within the same script for a more informed design process. Other benefits include the automation of modeling tasks, which can even be optimized for certain performance goals.

Having customizable designs means that each design may be different from the previous, which complicates the tooling required to manufacture them, as each variant has to be accounted for. To remedy this, Additive Manufacture (AM) is presented as a cost-effective way to produce customizable designs, due to its ability to manufacture complex shapes in a shorter amount of time, and the digital nature of the models required.

This work was conducted in order ascertain the true nature of an AD approach to a design problem, and its potential use in the field of engineering, while studying additive manufacturing as a way of producing customizable designs enabled by the scripting workflow.

1.2 Methodology

Initially, a literature review is conducted on the use of Computational Design in which PD, GD and AD are inserted, followed by definition proposals for each and their relationship to each other, as well as situating AD in the design process. Some software tools which are of interest to the exploration of AD's capabilities are also selected. Subsequently, additive manufacturing and customizable products are approached to showcase their relevance inside an AD approach. This is followed by experimentation and validation using a scripting environment inside a software tool relevant to engineering problems, but keeping the examples used as simple as possible as a means to demonstrate the capabilities of such an approach, without drifting into a specialized field. Lastly, a case study is selected to bring all components of this work together, showcasing the use of AD to solve a specific design problem, while integrating it with its environment.

1.3 Document Structure

Setting aside the present chapter, which contains the introductions and intentions of this work, the remainder of this dissertation is divided into seven chapters. To assist the reader in accessing the content within, this document was organized with the following structure:

- Chapter two — Contains the background for this work, and provides a distinction between Computer Aided Design (CAD) modeling and using it to infer on the design process. The definitions for the two more common approaches, PD and GD are also provided.

- Chapter three — Introduces AD with a literature review, defines it and its relationship with the other two approaches, as well as its place in the design process and a small selection of software tools appropriate for the application of AD.

- Chapter four — Discusses AM and its technologies with the intent of establishing a basis for its connection with an AD approach. Attention is also given to Design for Additive Manufacturing (DFAM), an important stage in the AM process, to identify how it could benefit from AD. Lastly, it discusses Customizable Products, the design of which can be heavily influenced by AD approaches.

- Chapter five — In this section, a programming language and software tool is chosen based on the tools described in Section three, followed by a demonstration of PD and GD approaches by resorting to extremely simple examples. Next, the process of scripting in a CAD software is described as a basis for the AD approach, again limited to simple geometries and pre-built examples. Some attention to the application of DFAM concepts is also given inside this chapter.

- Chapter six — A case study is performed. Instead of a specific application inside the field of engineering, a methodology is proposed for the use of an AD approach to solve a design problem. This methodology is then applied to drone building as a demonstration of problem tackling using a scripted basis.

- Chapter seven — The final comments on the PD, GD and AD concepts, as well as AM and Customization.

Chapter 2

Design Methods

Successful product design and development involves aspects of creativity, experimentation, field knowledge and problem solving to meet product requirements both aesthetically and functionally [Mountstephensand and Teo 2020]. These requirements are translated into measurable specifications, and are then used in the design process to iteratively generate, refine and test a wide range of solutions until a final design is chosen [Mountstephensand and Teo 2020]. It is difficult to find a single generic formula for the design process, as it is usually tailored to the field of its application. As such, this work considers an engineering design approach, which follows the process depicted in Figure 2.1, a generic approach to solving design problems using an engineering mindset. Of note is that in most depictions of the engineering design process, researching, exploring and testing solutions will consist in approximately half the total tasks.

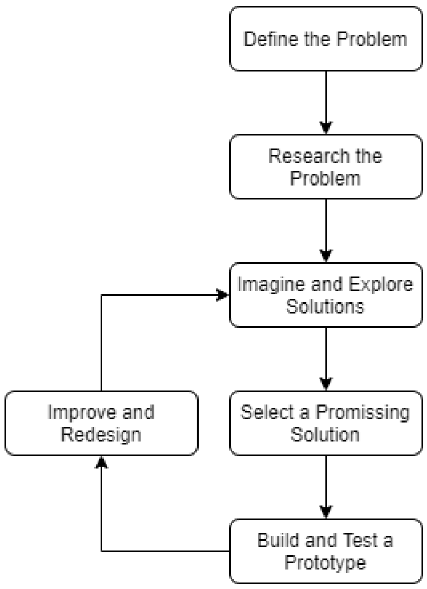

Define the Problem
Research the Problem
Imagine and Explore Solutions
Improve and Redesign
Select a Promissing Solution
Build and Test a Prototype

Figure 2.1: The followed engineering design process followed throughout this work, adapted from [of Colorado 2021].

[Mountstephensand and Teo 2020] divide the design process into two stages, conceptual and detailed. The conceptual stage generates the most design possibilities through exploration, experimentation and evaluation, while the detailed stage is dedicated to applying functional information and testing. Representing design solutions in these stages is naturally distinct. Concepts are traditionally hand-drawn, as Computer Aided Design (CAD) models require exact information and are generally slower and more laborious to design [Mountstephensand and Teo 2020, Dino 2012]. For this reason CAD models are confined to the detailed design stage, where a design has been selected and its physical properties are already known. In this way, CAD tools are only used as a way to increase efficiency and automate drafting (or modelling), as well as for design analysis [Dino 2012, Krish 2011, K. Terzidis and Srinivasan 2004]. Since designers dedicate a large portion of the development process to finding the necessary information for a particular design [Ye *et al.* 2008], it stands to reason that placing both design stages within the same design platform would allow for informed decisions during the design exploration process which would greatly reduce the time-to-market of a product, as well as its total development costs. Here enter Computational Design (CD) methodologies, in which the conceptual stage blends with the detailed design stage through the use of computational capabilities, specifically using CAD based software to explore design solutions.

2.1 The use of Computing Power in the Design Process

Terms such as CD and Digital Design (DD) have been used for some time to describe the use of computers in the design process, and the exponential increase to the number of scientific literature regarding the use of these terms without a proper definition has naturally led to ambiguity in their utilization [Caetano *et al.* 2020]. According to [Caetano *et al.* 2020], DD comprises the use of "computer tools in the design process" while CD "entails the use of computation to develop designs". [Ramos and Melgosa 2020] depict how CAD design is taught with a strong emphasis on DD, as mechanical engineering students are expected to be able to transfer a certain part design into the digital format as a part of their education. Modifying a design built manually through CAD features can be time consuming, as the designer performs constant changes and evaluations [Eltaweel and SU 2017]. Since latter stages of design, which correspond to a very small amount of the total development time, incur a significant amount of the development costs, there are benefits to optimizing the workflow on the earlier stages of design [Gardan and Schneider 2015].

Another issue is related to the ever increasing complexity of models. [Biedermann and Meboldt 2020] state that increasingly complex structures require more complex design tools for use with Additive Manufacturing (AM), and how this "manual, low-level process limits rapid and iterative design changes, as well as the [...] variety of design concepts". CD approaches have promising applications in fields requiring increasingly complex micro structures, such as biomimetic morphology, as geometries may be abstracted as they have an underlying algorithmic logic [Symeonidoua 2019]. Several approaches were developed to address these issues, aiming to automate and expand the design process. [Biedermann and Meboldt 2020] propose an automated design method based on object-oriented programming and parametric features, while [Oxman 2017] expanded by considering its use in the design process, rather than just a simple drawing tool.

Overall, the most complete definition for CD is given by [Caetano *et al.* 2020], who provides four activities for a design process to be considered computational design: automation of design features (be it by deduction or induction), adaptability towards multiple software, ease of incorporation of changes, and assisting the designer in form-finding processes. [Caetano *et al.* 2020]'s description provides the most complete scope of activities of CD present in recent literature, it will be the one used for the development of this dissertation.

Methods of improving and expanding the design process are ever-growing, but the most common terms in computational design literature are Parametric Design (PD) and Generative Design (GD), with a recent addition called Algorithmic Design (AD). Much like CD, instead of defining and establishing clear boundaries, it it much more appropriate to define the scope of activities which each approach entails.

2.1.1 Parametric Design

The conventional way of CAD modeling involves setting dimensions, constraints, orientation and many other variables to form a coherent shape. These variables are often considered as parameters which lead to the denomination of CAD tools as parametric software tools. An important distinction must then be made between what is considered PD and what is considered Parametric Modelling (PM). Early definitions of PD focus on the ability to model based on parameters which can be updated and visualized in real time [Caetano *et al.* 2020]. [Kolarevic 2004] expanded on this by considering parametrics as a way to create "an infinite number of similar objects", using parameters as a range of possibilities. This process allows different components of each part to be related, updating the model as a whole on each modification. [Zardo *et al.* 2019] refers to PD as "visual programming language tools" capable of creating complex geometries, improving workflow, allowing the analysis of several design possibilities and consequent selection of optimized solutions in the early stages of the design process. Similarly, [Dino 2012] regards PD as an algorithm based method to achieve a wide variety of solutions for design exploration. [Eltaweel and SU 2017] describe PD as a mathematical design where the design elements are related to each other using parameters that can generate complex geometries, which update in real-time by manipulating the parameters. Comparatively, PM is considered as the act of translating a specific design into a 3D using parameters, at a stage where design is more or less finalized and there is little to no need for studying other alternatives, therefore it does not constitute a CD approach.

The most recent definition of PD, from [Caetano *et al.* 2020], is "an approach that describes a design symbolically based on the use of parameters", removing associative geometry as a requirement of PD, but acknowledging its natural occurrence during the design process. This definition would invalidate the concept of CD, being instead more closely related to PM. To more closely fit within the CD concept, parameters are considered distinct from variables, as the later are values that change inside a system, while the former are entities that connect variables [Gunagama 2018]. Therefore, parametric design is considered as the use of parameters or constraints in the design process, allowing the real-time creation of several distinct design alternatives, iterative design refinement and simplified modification for parts of the design. It is an automated workflow, in which the constituent parts of the design are also related by parameters, allowing the design to be updated as a whole by modifying only one of its elements.

For example, updating an extrusion feature in CAD is a simple affair, but only if there are no other associated geometries to this feature. If such is the case, each geometry will have to be modified accordingly to correspond to the new extrusion feature. Using a PD approach to this problem, the modification to the extrusion feature would automatically equate to a change in the associated geometries. In this way, PD provides good design adaptability for complex designs with changing requirements [Dino 2012].

2.1.2 Generative Design

Generative Design (GD) is often mistakenly associated with a shape generation technique rather than an approach to design, mostly due to the improved capabilities of current engineering software tools such as Autodesk's Generative Design (AGD), which can give an organic look to a geometry, much like it was "grown" rather than created. However, GD is fact a design approach which aims to simplify the process of exploring alternative solutions to a design problem, relying on computing power to explore large amounts of viable solutions which the designer can then select from [Guidera 2011], rather than a defined process of giving shape to an object.

Several authors have provided their own insight on this topic, and most discrepancies occurs in the actual process of generating the alternatives. [Buonamici *et al.* 2020] defines GD as an approach that can generate a "series of plausible solutions for a design problem", by satisfying design constraints and maximizing a goal function. [Krish 2011] focuses on the creative design aspect, portraying GD as way of solving "creative design problems" brought by the subjectivity of each designer, and guiding them through a large number of design possibilities constrained by certain performance criteria. [Oh *et al.* 2019] states GD as a design exploration method "performed by typically varying design geometry parametrically and assessing the performance of output designs". This view is shared by [Dino 2012], which describes parametric modelling as a generative design tool, with a focus on encoding the design procedure rather than creating a physical object. [Krish 2011] considers GD as an approach where a generic model is created and its parameters varied in order to generate alternatives, then filtering these results based on their geometry, manufacturing and costs. Similarly, [Buonamici *et al.* 2020] proposes a similar method of GD: a generic CAD model is created and its dimensions stored in a design table. Value ranges are then set, randomized and used to produce a set number of design possibilities, based on the random variation of the stored values. Each design then has its performance evaluated by a previously set function. This function is an ambiguous way to refer to the application of filters to be able to narrow down solutions.

Far from being confined to parametric variation, other authors consider the use of algorithms in the GD process. Here, algorithms can be considered as a "step-by-step guide" on how to generate a solution, and as such has unlimited potential on shape generation approaches [Gunagama 2018]. For [Mountstephensand and Teo 2020], GD creates pieces or entire objects through algorithms, either interactively or automatically, while generating a large number of alternatives for exploration which a human designer could not achieve manually in a reasonable amount of time. Parallel to parametric approaches are evolutionary ones [Caetano *et al.* 2020]. These methods involve incorporating metaheuristic algorithms in the generation process, such as [Oh *et al.* 2019] propose a method using Topological Optimization (TO) as a "design generator" instead of the more common parametric approach, to develop parts merging engineering performance with good aesthetics. At some point, TO started to be named as a necessary component when discussing GD, primarily due to software tools such as Autodesk's Generative Design, but this addition is largely incompatible with the available literature.

In the end, as GD is a design process, much like PD, the shape generation process matters little in regards to the methodology. [Dino 2012] states that generative design focuses on encoding the design procedure, rather than creating an object. As such, be it created through an algorithmic or parametric base, GD is considered as an automated approach in which the goal is to generate a large number of design alternatives to assist with early design exploration. As an automated approach, GD allows the application of filters based on the physical characteristics (or metrics derived from them) of each solution to better aid with selecting a solution to further develop.

Intentionally blank page.

Chapter 3

Algorithmic Design

3.1 Literature Review and Definition

3.1.1 Literature Review

Algorithmic Design (AD) is found in literature as early as 2004, with a strong link to architecture. [K. Terzidis and Srinivasan 2004] portrays AD as new direction in architecture design. While previous utilization of computation power was limited to presenting or storing an idea already present in the designers mind, this new form of design sought to code the design intention by use of existing 3D software. [K. Terzidis and Srinivasan 2004] further refers to algorithmic design as a conceptual methodology for "the exploration of forms, structures and processes" usign mathematical or logical methods, while outlying the systems "consistency, structure, coherency, traceability and intelligence".

Another mention of algorithmic design as a "conceptual design method" is given by [Bukhari 2011], emphasizing its use in the earlier stages of the design process as a collaborative effort between human creativity and computational capability, and the benefits of "A rapid means of customising designs for local circumstances". [Zboinska 2015] outlines the lack of use of computational capabilities in the earlier stages of design exploration, and proposes algorithmic design based on parametric tools, allowing tasks "such as quick generation, exploration and evaluation of large design spaces, containing geometrically-complex solutions."

Through a comprehensive review of scientific texts, [Caetano *et al.* 2020] outlines the difficulty of differentiating algorithmic design from generative design, and proposes algorithmic design as "a design paradigm that uses algorithms to generate models", thus considering it generative. As proposed by [Caetano *et al.* 2020], the differentiating aspect of AD, when compared to GD, comes down to the strong correlation between the shape generating algorithms and its outputs, underlining its "finer degree of control" over certain aspects of the final shape. Unfortunately, this provides little insight to what an AD approach might consist of, and how it would differ over a GD approach in practice. [Martinho *et al.* 2020] offers a more practical application for AD: a script based tool which interfaces with CAD and analysis tools, although in this case modeling is done manually, with the script only serving as an interface which updates the model.

3.1.2 Algorithmic Design Definition

The literature on AD shows a distinct ambiguity on its application. While most authors agree on its use as a conceptual and design exploration tool [K. Terzidis and Srinivasan 2004,Bukhari 2011,Zboinska 2015], little attention is given to its actual process, resulting in confusion in regards to its application. As algorithms are sets of rules or instructions aimed at achieving a particular solution, one could argue that using a CAD tool would constitute AD [Caetano *et al.* 2020]. However, such processes have no input in the design process, and as such are excluded as AD. [K. Terzidis and Srinivasan 2004] explored AD possibilities using "constraint based stochastic search", which is based on GD processes, and which results in generally unpredictable outcomes. Such algorithms can be part of an AD process. For example, [Caetano *et al.* 2020] considers a program capable of modeling a building by "creating slabs, columns, beams, walls" and other elements to be AD. Hence, even if some of the components of a certain design are based on evolutionary approaches, each element of the design can be directly traced to a specific part of the algorithm, and is considered AD. Algorithmic ambiguity is not limited to GD processes, but PD ones as well. [Symeonidoua 2019] used parametric based algorithms to generate biomimetic surfaces for building skins, in which the generated skin could adapt to the complex shapes of the required buildings. Such examples contribute to the difficulty of defining AD in light of PD and GD processes. [Caetano *et al.* 2020] defines AD as a subset of GD and PD. Although AD does generate design possibilities, it does not require a performance evaluation of the finished design, and thus there can be AD without GD. In the same way, AD does not require a PD base, as design generation can be achieve through other processes other than parameter variation.

For the scope of the work, Algorithmic Design is defined as the use of algorithms, scripts or pieces of code created by the designer with the specific intent of solving a design problem, allowing the generation of designs which can be modified and tuned by the designer. By crafting algorithms specifically for a design goal, be they generative, parametric or otherwise, it provides the designer with control over the generated solution and allows iterative design refinement, modification and analysis, as every component of the design is linked to a specific piece of code and can be controlled through it. Much like what [Martinho *et al.* 2020] describes, such an approach is not limited to CAD modelling, and can be used to interface with other tools or programs to simplify or inform a design process. Additionally, like [Bukhari 2011] reported, controlling all aspects of design also facilitates the customization of the design for specific circumstances.

3.1.3 The Relation between AD, PD and GD

It is difficult to establish a clear line between each approach. In fact, when describing GD approaches, some use parameters to achieve variation, resulting in a merging between PD and GD techniques. The main degree of separation between these two techniques would be the automation GD requires. Similarly, an AD approach can consist of using parameters to automatically generate models while interfacing with different tools, which also uses PD and GD techniques. Conversely, these approaches can also be thought of separately: an AD approach can solely consist of interfacing between software tools to solve a design problem, or it can make use of parameter variation exclusively as a PD approach. The same applies to GD approaches, which need not be based in parameters to generate alternatives. Despite this, to single-handedly use one approach while excluding the other requires conscient effort, and is far more natural for techniques from different approaches to be used together during a design process. Therefore, it is more accurate to represent the domain of each of these approaches inside CD as illustrated in Figure 3.1, which represent sets of activities which are linked to each of the approaches, but are not limited to just one approach.

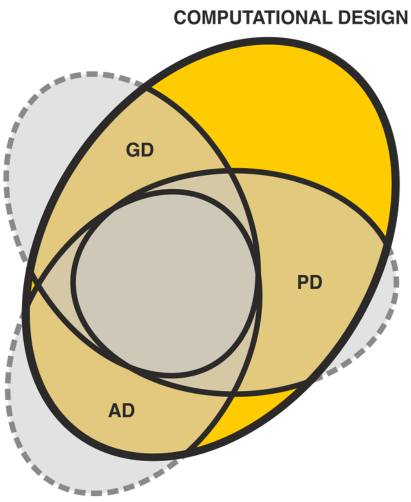


Figure 3.1: Proposed domains of AD, GD and PD withing Computational Design.

3.1.4 Integration in the Design Process

Scripting is the main activity of an AD approach. To best apply this type of approach, it is necessary to identify where in can be best applied. So far, this work identified three ways scripting can influence the design process: by directly building geometries, by interfacing between software tools, and by allowing for design customization. Although the focus of using CD techniques is in the early design process, AD need not be limited to it. In fact, the limitations of using AD during the design process are connected to the design complexity and the capabilities of the script to interface between the different tools. Theoretically, AD reaches its peak if the designer can interface between CAD, Computer Aided Engineering (CAE) and Computer Aided Manufacturing (CAM) tools, as well as the manufacturing and testing equipment. This application would contain the modelling, analysis and manufacturing inside a script tailored for the specific design problem. The main obstacle to such implementation is the fact that CAD, CAE and CAM software tools are proprietary, working with different file formats and sometimes different programming languages. In some cases, software tools that posses CAD, CAE and CAM do not allow interfacing between them, and creating geometries using only scripting can also be laborious if the design is complex. However, AD is well suited to automating design exploration. Having access to the features of such software tools through scripting can help automate and reduce the time expenditure of the early design stages. The creation of multiple scripts tailored for certain tasks, along with creating interfaces for direct manipulation of the design can also help reduce the total load on the designer. Lastly, the ability to integrate design customization means that designs which integrate CAD, CAE and customization can be made, further exarcebating the potential of such an approach.

3.2 Software Availability

In order to validate the AD concept within an engineering design process, software tools with a Application Programming Interface (API) and a well documented programming language are considered. This restriction was made to take advantage of the design capabilities of existing software tools such as CAD tools, but simpler approaches using programming by itself are also possible. Two tools with promising API's were considered: Autodesk's Fusion 360 (or Fusion for short), and Rhinocerous (also known as Rhino 3D or simply Rhino). There are many more CAD software tools which would be appropriate for testing AD, but the two described above are the most accessible at the time of writing, and present interesting capabilities which will be described shortly. This section describes these tools and their potential when using an AD approach.

3.2.1 Fusion 360

Fusion is cloud-based software tool, combining CAD, CAE, and CAM in a single product, along with freeform modeling, sculpting, parametric features, material and manufacturing data [Pradhan 2019]. It is available with a Free or Educational License, although Free users have limited access to features, and use of CAE features requires at least an educational license.

Being a cloud service, Fusion allows for several users to collaborate on projects no matter their location. As all information is automatically stored in the cloud, along with the ability of integrating manufacturing and design in a single tool, it allows concept exploration and prototyping to be much faster than its competitors [Suhada *et al.* 2018]. Fusion also features an API [Pradhan 2019] based on Python, which is a high-level programming language, focusing on ”ease of use and reliability”, and with vast libraries on multiple subjects [Meurer *et al.* 2017]. Research has been conducted on using Python to overcome engineering challenges, such as analysis and optimization of electrical power systems [Thurner *et al.* 2018] and adaptive truss layout optimization [He *et al.* 2019], which demonstrate Python's as a valuable asset in any engineers tool set.

One of Fusion's included features is Autodesk's Generative Design (AGD) which, according to [Autodesk 2020], is a tool that assists in creating, testing and evaluating the best possible outcomes for a specific design scenario. AGD uses algorithms to automate the generation of a large number of solutions which can be further refined by the designer. The process uses two types of geometries, preserve and obstacle, to identify geometries to keep and zones to avoid, respectively. The user can then specify loads and constraints, as well as objectives (minimize mass or maximize stiffness), manufacturing methods and materials in order to optimize the design. AGD is used to generate optimized solutions for a particular design solution from which the designer can then filter based on a different number of parameters. As a GD tool, it isn't meant specifically for design exploration, rather to explore optimized solutions to a specific problem, but its use of algorithms as its basis for creating shape, while using parameters to better define the design problem shows value as a tool to implement with an AD approach. Since scripts can be made to interact with AGD inside Fusion, it presents a good candidate for a software tool for also combining PD and GD, where design exploration is carried out through algorithmic means, and the final design is ran through AGD.

3.2.2 Rhinocerous 3D

Rhinocerous 3D (Rhino) is a 3D modeling software tool based on Non-Uniform Rational B-Splines (NURBS) [Lee and Song 2021]. Its ability to produce complex surfaces attracted great attention in the field of architecture [Guidera 2011] and is much expanded through its plugins, some of which include environment simulation [Lee and Song 2021]. Besides the availability of scripting with Python and RhinoScript, that makes Rhino an interesting tool for an algorithmic approach is Grasshopper, a plugin which provides a novel way of interacting with geometry, allows for environmental simulation, lighting and energy optimization through the use of additional components [Lee and Song 2021]. It is a visual programming interface [Lee and Song 2021] which references geometry in Rhino, allowing a different control over the geometry and real time updating of any changes made to it [Guidera 2011].

Using Grasshopper involves determining initial parameters and connecting them to "blocks" to generate a final output, which lead to the use of Grasshopper as a PD tool. These "blocks" represent algorithmic operations which can be customized by the designer, and several "blocks" can be connected to each other to further add complexity to the design. Figure 3.2 is an example of a geometry created using extrude and offset operations in the Grasshopper environment [Guidera 2011].

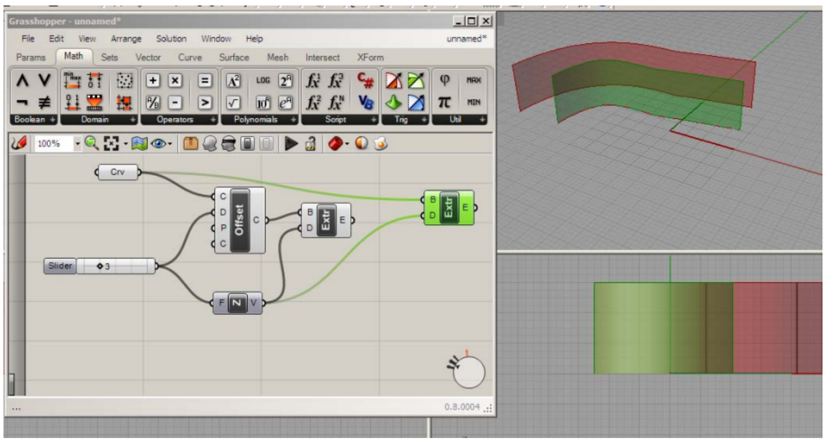

Grasshopper - unnamed*
File Edit View Arrange Solution Window Help
Params Math Sets Vector Curve Surface Mesh Intersect XForm
Boolean Domain Operators Polynomials Script Trig Util
Crv
Offset
Extr
Extr
Slider
3
F N V
0.8.0004

Figure 3.2: An example of a Grasshopper program with its respective output [Guidera 2011]

Although Grasshopper uses a visual interface compared to the text based one from Fusion, both seek to interact with the tool using algorithms. These follow a simple structure, in which an input is passed through a function to obtain a desired output. The functions can be mathematical equations or sets of rules, which allows for not only shape creation, but also for task automation. Arguably, interacting with Grasshopper is easier due to its visual nature, and each block can be modified by resorting to Python or RhinoScript, but text based scripting provides more freedom to the designer, as well as the possibility of creating their own plugins to interface with other tools. In the end, the choice to use either is dependant on designer preference.

Intentionally blank page.

Chapter 4

Additive Manufacturing and Customizable Products

The inherent digital nature of Algorithmic Design (AD) approaches, along with their focus on design exploration, present a few challenges for manufacturing. Since an AD approach can obtain several solutions which can have varying degrees of complexity and may require prototyping and testing, it is essential to select an appropriate manufacturing method. To better understand how solutions obtained from AD approaches can be effectively produced, Additive Manufacturing (AM) is looked at as a digital based method that can satisfy the requirements of such an approach.

Additionally, AD approaches are based on scripting an thus customizable by nature. Combined with a manufacturing technique like AM, AD approaches can simplify the process of building customizable products and producing them in a timely and cost-effective manner.

4.1 Prototyping and the Birth of Additive Manufacturing

When attempting to develop a product, it is often necessary to build and test a prototype. These serve a multitude of functions, including (but not limited to) functional testing, concept exploration, customer feedback, and evaluation of design fidelity [Camburn *et al.* 2017]. Traditional manufacturing techniques can be classified in four distinct groups: Subtractive, such as machining, which removes layers of material to create geometry, and are usually automated; Joining, such as welding; Dividing, such as cutting or shearing; Transformative, in which the mass of the object remains unchanged, such as forging [Abdulhameed *et al.* 2019] or hydro-forming. Subtractive processes are common for prototyping, but require large amounts of energy and have high "buy-to-fly" ratios, (the relationship between the mass of the starting billet to the finished part) [Watson and Taminger 2018], while the increase in part complexity leads to higher complexity of the manufacturing equipment, which in turn limits the use of Transformative and dividing processes.

Originally developed for prototyping, a fifth class, Additive Manufacturing (AM), allows for greater energy efficiency and lower material consumption [Watson and Taminger 2018]. Parts manufactured through this method allow production of complex geometries from a selection of tailored materials while reducing material waste [Bikas *et al.* 2016]. Contrary to Subtractive manufacturing, this process consists in joining materials by adding successive layers, based on a pre-existing 3D model, using 3D printers [Abdulhameed *et al.* 2019, Zafar and Zhao 2020].

While initially confined to rapid prototyping, AM quickly transitioned to rapid tooling, creating tools for traditional manufacturing methods. Advances in technology as well as the greater range of materials available mean that multi-material, fully functional end user products are now possible. The constant decrease in the price of 3D printers made the technology available to all business sizes, and some large electronic retailers provide dedicated 3D printing services. As 3D printers became more common for low-volume production, the unit cost reduced accordingly [Holmström *et al.* 2016], which led to a newfound success in home fabrication, as large retailers started selling 3D printers and other associated components [Rayna and Striukova 2016]. Recent research on AM technologies focus on innovation and customization, while also showing promise in the optimization of factory operations, by containing production in a single build envelop and therefore greatly reducing tooling costs [Holmström *et al.* 2016].

Aside from increased complexity of geometries, AM allows for the manufacture of non-removable assemblies and the use of multiple materials during printing to optimize the parts performance. New possibilities also come with new limitations, and AM processes must take into account factors such as temperature control, build orientation for improved mechanical performance and built time reduction, support generation and removal to ensure correct fabrication and machine limitations [Salem *et al.* 2020]. The material selection is wide and ever increasing, with metals polymers, concrete, composites and even edible materials such as chocolate [Mwema and Akinlabi 2020]. Common applications for AM technologies include medical industries (implants, functional body parts, biotech), construction (components or entire buildings [García-Alvarado *et al.* 2021]), as well as food, automotive, research and aerospace industries [Wickramasinghe *et al.* 2020].

The structural complexity which AM allowed led to a natural integration of TO into the AM design process. AM removes the constraints of traditional manufacture method, while also creating "lightweight, high-performance" parts [ZHU *et al.* 2021].

4.1.1 Quick Overview of AM Technologies

AM technologies can be classified in several different ways. [Abdulhameed *et al.* 2019] uses the initial material state to separate different technologies. This is a common description of AM technologies, as the initial material state will dictate which process can be used for manufacturing. Following this logic, Figure 4.1 depicts a selection of materials and associated process of AM.

The choice of AM technology is dependant of weighing their advantages and disadvantages with of the required design specifications, along with material and mechanical considerations, such as impact resistance, ease of printing, heat resistance and visual quality, among others. [Wickramasinghe *et al.* 2020] overviews some of the methods present in Figure 4.1, but as they are far too many to go into detail, and the intricacies of each are not the subject of this work. Instead, Fused Deposition Modeling is used as a reference example of what a designer should consider when designing a part with an AM method in mind.

Fused Deposition Modeling, also known as FDM, is the most common extrusion based technology for AM, being low technology, easy to operate, and cheap to maintain and resupply [Medellin-Castillo and Zaragoza-Siqueiros 2019]. In this process, string shaped

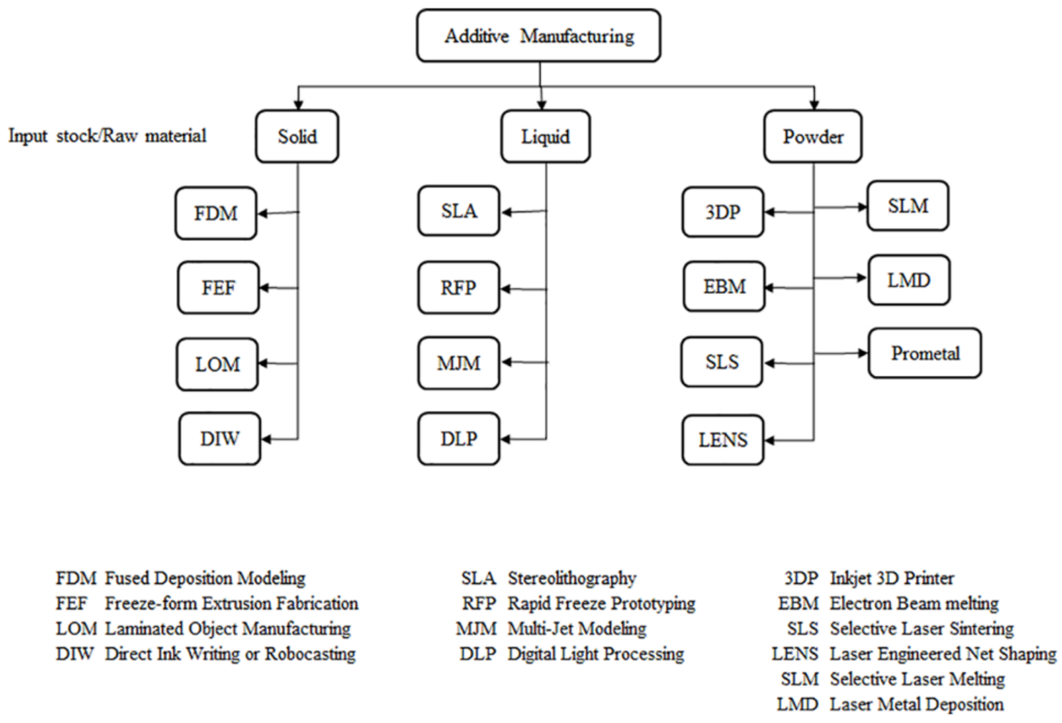

Additive Manufacturing
Input stock/Raw material
Solid
Liquid
Powder
FDM
FEF
LOM
DIW
SLA
RFP
MJM
DLP
3DP
EBM
SLS
LENS
SLM
LMD
Prometal
FDM Fused Deposition Modeling
FEF Freeze-form Extrusion Fabrication
LOM Laminated Object Manufacturing
DIW Direct Ink Writing or Robocasting
SLA Stereolithography
RFP Rapid Freeze Prototyping
MJM Multi-Jet Modeling
DLP Digital Light Processing
3DP Inkjet 3D Printer
EBM Electron Beam melting
SLS Selective Laser Sintering
LENS Laser Engineered Net Shaping
SLM Selective Laser Melting
LMD Laser Metal Deposition

Figure 4.1: Summary of several AM technologies, separated by initial material state and associated technology, adapted from [Abdulhameed *et al.* 2019]

solid material (filament) is passed through a metal nozzle which melts and deposits it layer by layer into a build plate. Multiple nozzles can be used if the printer and slicer support it, for cases in which multi material or support material is needed, such as soluble supports [Wickramasinghe *et al.* 2020]. Common materials include PLA, ABS, PETG, Nylon, composite materials, carbon fiber [Wickramasinghe *et al.* 2020], and even flexible materials, wood and chocolate. FDM is also the most prevalent in home fabrication, as printers became small enough to fit on desktops.

4.1.2 The Manufacturing Process

Producing a part for AM follows a digital based methodology: Initially, a digital model of the part is required. These can be created using 3D modeling software (such as CAD) or using 3D scanning technologies [Rayna and Striukova 2016]. In this first phase Computer Aided Engineering (CAE) can be used to simulate the material response to certain types of loads and constraints [Wickramasinghe *et al.* 2020], optimizing the geometry through fast iterations, with AM allowing for the production of the resulting complex geometry [Remache-Vinueza *et al.* 2021]. The model is then converted into a triangle mesh which can be stored into an .STL file format, then passed through a ”slicer” software, which generates a layered tool-path and converts it into commands for the 3D printer [Steuben *et al.* 2016]. The ”slicer” allows for customization of most printer parameters, such as layer height, temperature, and part orientation, depending on the AM process. After the part is prepared, the manufacture process occurs autonomously and requires little supervision (and some 3D printers come prepared with monitoring

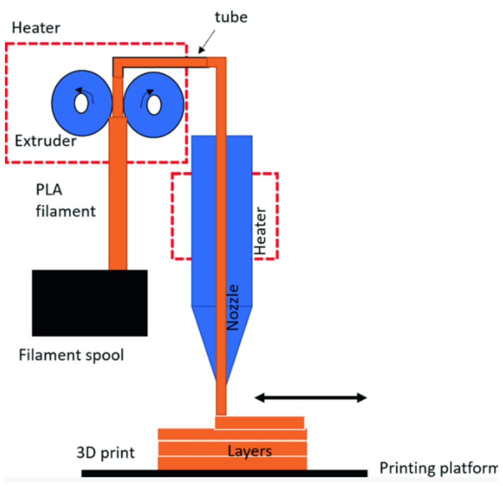


Figure 4.2: Schematic of the functional components of the FDM process [Mwema and Akinlabi 2020]

equipment to facilitate supervising work over distances). When the parts are finished printing, they are removed, and excess material is discarded and additional processing (such as sanding or polishing) is done, if required [Vaneker *et al.* 2020].

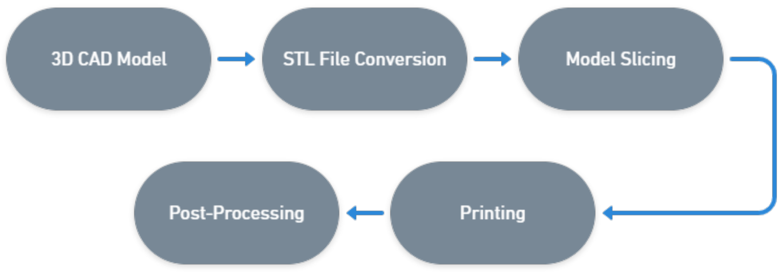


Figure 4.3: Generic process of design for AM, although some variation can occur from technology to technology.

4.2 Design for Additive Manufacturing

As with traditional manufacturing methods, AM also benefits from designing a product with insight from the desired AM process. The increase in product complexity and client expectations, both of quality and modularity, necessitates for a manufacturing process that responds accordingly [Salem *et al.* 2020].

Designing without insight from the manufacturing process can incur extra costs during development if the product were to fail during manufacture, or if it required modification in the later stages. With the continuous advancement of computer technologies, came the possibility of creating a symbiotic relationship between design and manufacturing. [Gebisa and Lemu 2017]

This process, called Design for Manufacturing (DFM), attempts to reduce cost and complexity of a part, simplifying the process of its manufacture by minimizing the complexity of manufacturing operations and reducing the number of tight tolerances [Durakovic 2018]. A DFM methodology contains rules, guidelines and tools which aim to assist in creating a design adapted for a specific manufacturing process, while acknowledging the importance of having such considerations early in the design process [Medellin-Castillo and Zaragoza-Siqueiros 2019].

Design for Additive Manufacturing (DFAM) is based on the same principles as DFM (and as such, is an integrated methodology), but seeks only to optimize a product for a specific AM process. Products optimized in this way can be produced more reliably, require less manufacturing time and are more cost effective [Vaneker *et al.* 2020], as they have fewer and customizable parts, are more cost effective and have simplified assembly procedures, when compared to parts manufactured without DFAM [Durakovic 2018]. The goals for DFAM are the same as those for DFM, although the unique capabilities of AM presents new challenges as each AM technology has its own constraints which must be accounted for [Medellin-Castillo and Zaragoza-Siqueiros 2019].

[Salem *et al.* 2020] separates the information necessary for a DFAM methodology into four categories to successfully manufacture optimized parts for a specific AM machine:

- Material knowledge — Considering material shrinkage, constraints and anisotropy, as well as removing unnecessary geometries and defining their orientation.
- Process knowledge — Printer specifications (such as layer thickness for the machines resolution, and other material considerations, such as purity and temperature) and build orientation.
- Product knowledge — Adding functional information to the model and apply Topology Optimization (TO).
- Procedural knowledge — Optimizing production, reducing material and time costs, and accounting for the production volume.

4.2.1 DFAM Methodologies

Several frameworks have been proposed over the years, although the relatively recent acknowledgement of AM as a production technology, the varied production methods which are optimized differently and lack of education in the subject result in no single, generic framework for all design situations [Vaneker *et al.* 2020], [Medellin-Castillo and Zaragoza-Siqueiros 2019]. [Salem *et al.* 2020] proposes a five step methodology for DFAM:

- Client Specifications — Converting requirements and constraints given by the client into workable data, such as dimensions, functional surfaces, tolerances and loads.
- Functional Domain — Functional surfaces and volumes are created taking into account the necessary manufacturing tolerances (such as minimum thickness), which may vary depending on part orientation and the chosen process.
- Process Domain — Integrating the process details and constraints and choosing the appropriate material, based on machine and part constraints, such as part orientation and thermal considerations, to ensure the functional part in the previous process can be manufactures successfully;

- Optimization — Using TO to optimize the part volume, reduce weight, material usage, cost and operation times.

- Simulation — The manufacturing process, as well as the mechanical properties of the part are simulated to ensure adequate manufacture of the part.

Each AM technology has different considerations in regards to the Functional and Process domains. The FDM process is notorious for needing supports for overhanging parts that are over a 45º angle, while others dispense with supports entirely [Wickramasinghe *et al.* 2020], but the methodology is maintained throughout the different processes. Other geometric considerations must be taken into account, such as part size, minimum wall thickness and build orientation, which can influence the final part's visual and mechanical characteristics, and therefore cost and production time [Medellin-Castillo and Zaragoza-Siqueiros 2019].

The digital nature of the AM process allows the use of algorithms to assist in automating such procedures. [Briard *et al.* 2020] proposes an automated methodology based on GD combined with TO, providing better insight on possible solutions for the final part design, while also maintaining a similar methodology to the one described above. The methodology involves constant looping during each step to ensure that the selected designs are optimized before continuing to the next step, and can be summarized in four steps:

- Translation — Converting all part data into the required GD software data (such as dimensions, loads and tolerances).

- Initialization — Optimization, refinement and analysis using GD tools.

- AM Guidelines — Integrating AM constraints and production optimization based on material and process.

- Refinement — Further refinement of obtained solutions, and final analysis.

Using GD techniques allows the designer to obtain unconstrained models in the initialization step, and looping the process allows the designer to return to an earlier model, if the next iteration wasn't satisfactory. The same process applies to the next two steps, where only new information is added (such as manufacture constraints in AM Guidelines), and further looping is performed to guarantee the best possible result.

In practice, automating such methodologies requires key points of each process to be addressed. [Zwier and Wits 2016] utilizes an algorithm to find the optimal build orientation and minimize the support volume needed, by using the convex hull principle and comparing it to the parts mesh model. The algorithm provides several solutions for part orientation, and utilizes the best one to orientate the part in the build plate, as demonstrated in Figure 4.4

In summary, although there is no single accepted DFAM baseline methodology, their principles remain the same, and the digital nature of the AM process provides the necessary framework to automate several steps in these methodologies using an algorithmic approach.

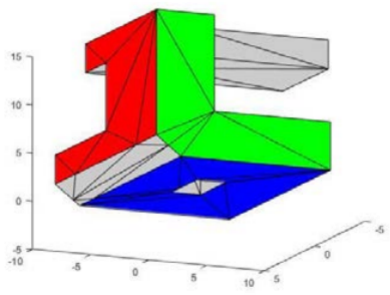


Figure 4.4: Possible part orientations obtained from the algorithm, red being most suited, followed by green and then blue [Zwier and Wits 2016].

4.3 Customizable Products

A customizable product is a product designed with input from the user, and which can have its features or properties modified after the product has already seen its first use, allowing the user to change the products performance to meet their specifications [Colombo *et al.* 2020]. It is distinct from product personalization, as in this case a product is tailored to a specific customer, often without their input, while customization implies that the customer has had input in the product development process [Deradjat and Minshall 2017]. A common example for personalization is marketing, in which costumer information is used to produce advertisement targeted to a specific consumer demographic [Pallant *et al.* 2020].

[Pallant *et al.* 2020] show an increasing preference for the majority of U.S.A. online shoppers to acquire customized products or services, revealing how customizing adds value to a product, which can be sold at higher prices and with costumers accepting longer wait times. Many large companies allow some form of customization for their products, such as Nike, which allows a consumer to customize their sneaker, and Dell which offers a large level of customization (with regard to other brands) when choosing a desktop computer or laptop. However, applying a customization model to large volumes of production, know as mass customization, requires a deep level of understanding of the underlying market [Bernhardt *et al.* 2007]. The shoe company Shoes of Prey, which started making fully customizable shoes based on customer specification in 2009, entered liquidation in 2019 due to the high cost of the customization model used, as well as the "inability to achieve mass market customer adoption" [Pallant *et al.* 2020].

Additive Manufacture technologies present a new way of tackling the high costs of manufacturing a customized product [Deradjat and Minshall 2017]. A new trend, Direct Digital Manufacturing (DDM) investigates the use of AM technology for the local manufacture of customized goods [Srinivasan *et al.* 2018]. AM's high degree flexibility is one of the main reasons for its use with customizable products [Srinivasan *et al.* 2018]. As AM allows manufacturing of end-user products, allied with an ability to work solely of 3D models, it presents a faster alternative to producing such a product. Additionally, the reduced tooling costs, the ability to work with small batch sizes without extra costs, the use of multiple materials and reduced need to keep inventory (as some parts can be produced on demand) are other reasons why AM has great potential in producing mass customized products [Srinivasan *et al.* 2018].

Displayed in Figure 4.5 is a 7-inch tablet developed by [Stanciu *et al.* 2019], complete with manufacturing process, which allows users to modify any of its six modules to suit their needs. Such modules include, but are not limited to, a bluetooth speaker, a flashlight, additional battery capacity, and several cosmetic options. Included are also several manufacturing options for the tablet frame.

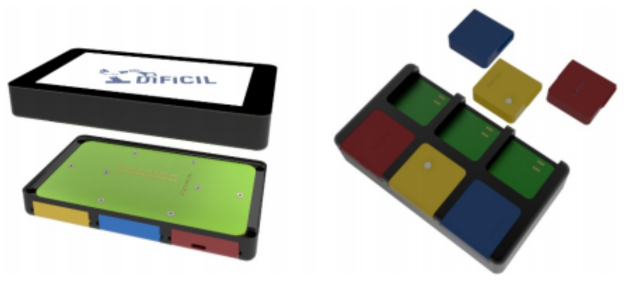


Figure 4.5: Customizable tablet with replaceable modules, as proposed by [Stanciu *et al.* 2019].

When developing customized products, it is also important to dedicate resources to the interface in order to guide the user through the creation process, as demonstrated by [R. Pescaru and Oancea 2017]. Attention should be given to how the information is displayed, and which should be present, as functionality is an important part of a customizable product [Felfernig *et al.* 2001]. As such, it is important to not saturate the user with information, and a careful selection of information should be undertaken.

Part II

Exploration

Chapter 5

Exploration and Validation

With the concepts now discussed, this chapter will focus on validating each approach using simple examples, and exploring AD's capabilities and its presence in currently available software tools. Initially, the CAD tool and programming language are selected, followed by testing shape creation using Parametric, Generative and Algorithmic techniques, and ending with use of existing software tools to help solve design problems.

5.1 Tool and Language Selection

The choice between Fusion and Rhino comes down to two factors: the type of license available, and the feature set provided through scripting. Fusion is a full fledged CAD software, with several extras such as Topology Optimization (TO) or Autodesk's Generative Design (AGD), and is more adequate for solving engineering problems, as Fusion's solid based models also allow for design tolerances and general dimensions to be applied to models, which make it ideal for functional models. The text-based Application Programming Interface (API) used by Fusion is based on Python, which potentially allows for integration of other tools inside the same script due to Python's large and growing community which provide free and paid plugins. As for Rhino, its comprehensive surface based modeling, although powerful, is less adequate for solving engineering problems. The main interest in Rhino for this work comes from Grasshopper, with its visual programming interface, which can also be customized with python.

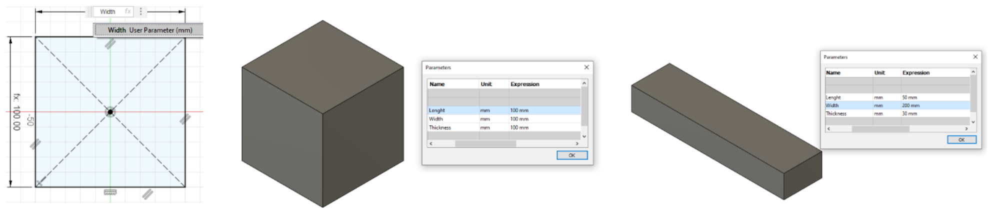
The choice was made to use Fusion, as it has an educational license which grants access to its comprehensive analysis tools and is more adequate for solving engineering design problems.

5.2 Exploring Parametric and Generative Approaches

Experimentation is carried out on Fusion to test the validity of the parametric and generative techniques to better understand how to use them to solve design problems. The initial tests were dedicated to establishing a set of activities which can be identified as either Parametric Design (PD) or Generative Design (GD), and highlighting some gray areas that occur in their interaction. Testing was done resorting to simple geometries, as the focus of these approaches is not how its shape is generated, but how a design problem is tackled with each approach.

5.2.1 Creating Objects Parametrically

To model a rectangular hexahedron in a parametric way, its 3 dimensions are used as parameters. Fusion allows for lists to be created through its “Change Parameters” option, and each dimension assigned a variable, *length*, *width*, and *thickness* with a base value of 100mm. A sketch is then drawn using *length* and *width* as dimensions, and extruded with *thickness* to form a perfect cube. The presence of a *fx* prefix when sketching in Fusion indicates that this dimension is set by a parameter. By altering the parameters directly, the cube alters shape accordingly in real time. In contrast, modifying shape traditionally requires modifying the initial sketch and extrude feature to achieve the same result. Furthermore, these parameters can be related to each other, modifying the model as a whole with only one parameter. In Figure 5.1, the thickness is set at half the sum of the other two dimensions, and the overall shape can be manipulated by modifying its *length* and *width*.

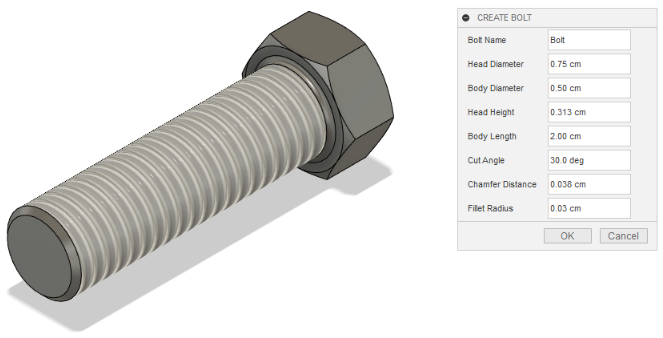


Name	Unit	Expression
Lenght	mm	100 mm
Width	mm	100 mm
Thickness	mm	100 mm

Name	Unit	Expression
Lenght	mm	50 mm
Width	mm	200 mm
Thickness	mm	30 mm

Figure 5.1: Parametric variation of a rectangular parallelepiped’s dimensions.

The same approach can used with more complex shapes. Figure 5.2 shows Bolt.py, one of Fusion’s inbuilt examples, which creates a bolt based on inputs by the user, and allows real time modification of its shape. The methodology is the same as the previous example, where the user controls key parameters to obtain design variation, instead of resorting to manual modification of the model.



Bolt Name	Bolt
Head Diameter	0.75 cm
Body Diameter	0.50 cm
Head Height	0.313 cm
Body Length	2.00 cm
Cut Angle	30.0 deg
Chamfer Distance	0.038 cm
Fillet Radius	0.03 cm

Figure 5.2: Parametric bolt created using the API, along with interface for real time modification. From Fusion’s built-in library.

Although the previous examples are simple in nature, they showcase the parametric way of approaching a design problem: selecting key parameters from the multiple design variables, achieving design variation according to the designer’s intent to use as parameters. This type of workflow simplifies the process of modifying the model by having it adapt to changes immediately, while also establishing relationships between multiple components to ensure the model adapts correctly.

5.2.2 Generative Thinking

Exploring the generative approach can be done by expanding the parametric example into a fully automated routine. Multiple variations of the initial cube can be achieved by automatically varying the desired parameters and displaying each outcome to the user. Fusion lacks this feature as a built-in, but the same effect is achieved by using Fusions API. Firstly, the basic cube shape is programmed in python, and configuring one if its dimensions as a *height* parameter, randomly generated within a 50 mm to 500 mm range. These basic instructions are then placed in a cycle to generate 100 variations with a pseudo-random *height*. Running the script returns each iteration displayed to the user as a separate body. Notably, the process of creating a cube based solely on programming is much more laborious compared to the traditional method, but allows automating the generation of multiple alternatives. The result is depicted in Figure 5.3.

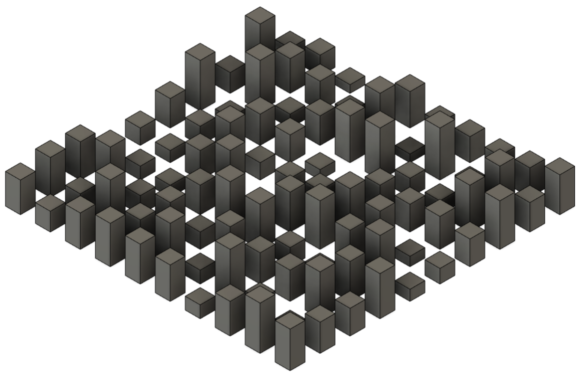


Figure 5.3: Generative solution set with height variation, showing 100 results.

Generative approaches benefit greatly from automation, as filters can be applied to help select alternatives with the right characteristics. Fusions API allows access to all attributes of the object body, including its material, so results can be tailored to fit any parameter imposed by the designer. Mathematical equations can also be used, as long as its output corresponds to a valid attribute of the object body, such as height, mass or length. For demonstrative purposes, the example in Figure 5.4 was obtained by generating and filtering out all objects with a volume greater than 2500 cm^3.

In the example from Figure 5.4, a parameter was set within a range to obtain variation. However, parameters aren't limited to variables which describe the geometry, they can also be mathematical equations. In Figure 5.5, the parameter that controls the object height was changed instead to follow the curve of an exponential function with each iteration. Much like the previous example, some solutions were filtered out based on their volume.

These examples were both parametric and generative in nature, as the object creation was achieved through parameter variation and several results were generated and filtered. As it requires the use of the API, and therefore was scripted, it also constitutes an

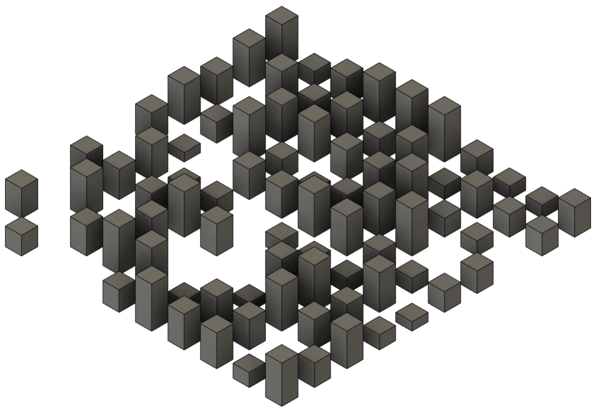


Figure 5.4: A different Generative solution set in Figure 5.3, with results with a volume of over 2500 cm^3 removed from the solution space.

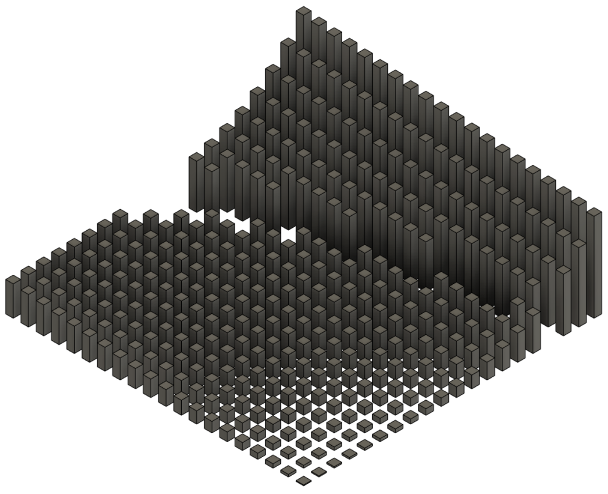


Figure 5.5: Generative solution set where each object's height follows an exponential function, with some results filtered based on their volume.

algorithmic approach. To note that in the previous examples, each geometry represents a a possible solution for the designer to choose, rather than a single model made from smaller geometries.

5.3 Working with the API

Programming shapes through scripting is analogous to the process used to generate a cube for the generative testing, but, in this section, the process of creating shapes and automating tasks is explored in-depth. Complex shapes are increasingly difficult to create while relying solely on algorithmic descriptions when compared to using the GUI due to the need of re-running the script with every modification. However, there are benefits for using a scripted approach, namely in task automation, the ability to integrate pertinent design knowledge through mathematical equations, or assist in exploring solutions through automatic model generation for conceptual work. Hence, the focus of the this section is in programming scripts that can be interactive with the designer, either automating or providing insight about the model.

Two options for programming are available. The first, Scripts, selects and executes a single script. Depending on the script, inputs may be required for it to function properly, but it can also be ran independently from the user. The second option is Add-ins, which can be loaded automatically when Fusion starts and used to create commands tailored to assist the user in certain situations. Add-ins are best suited to assist during the design process as they remain in Fusions toolbar and are easier to access.

Not all features can be accessed using the API, most notably, access to other modules inside Fusion such as Topology Optimization (TO)[1] or Autodesk's Generative Design (AGD), and to a select few features including Mirror and the measure tool is restricted, as depicted in Figure 5.6. This means that to make use of some features from Fusion's library an intermediary step between running the script and obtaining a result is done manually. Features with full functionality can be freely accessed through scripting, while those with limited functionality may not be as complete as with manual interaction.

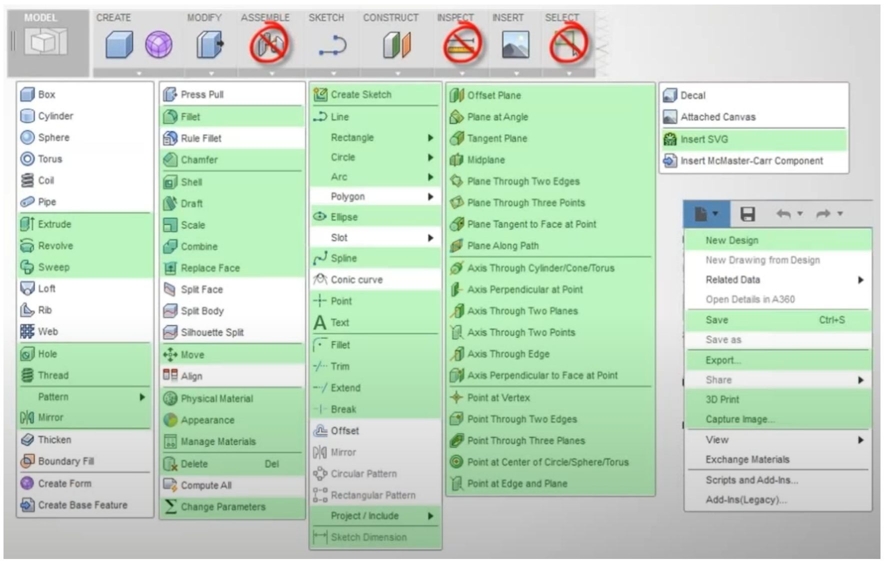


Figure 5.6: Sketch feature availability (in green) when using the API [Erinks 2015]

5.3.1 Exploring Algorithmic Shape Generation

Tackling a problem using a scripted approach allows sketches to be drawn based on mathematical equations. Alternatively to expressing profile sketches as separate points in space, these can be expressed as functions, either 2D or 3D, and used to create very intricate surfaces. In the next example, several splines are expressed as multiples of an exponential function and expanded into the third dimension to give a slanted look to the splines. Running this in a loop creates several splines with a increasingly steeper incline. From here, the shape is developed further by adding Perpendicular splines to create a weave effect in the object as seen in Figure 5.7. These can be swept using a

[1]The Topology Optimization module is named Shape Optimization inside Fusion360, which is not entirely correct in the scope of structural optimization

profile and set to automatically fillet edges, creating rods as is seen in Figure 5.8, but other features such as lofting are available.

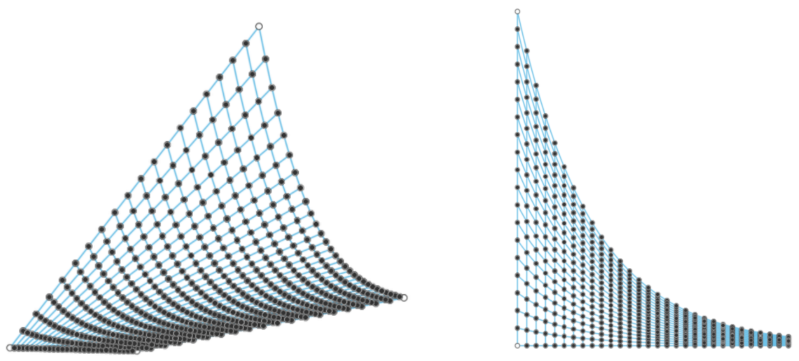


Figure 5.7: Spline generation based on a exponential function through algorithmic means.

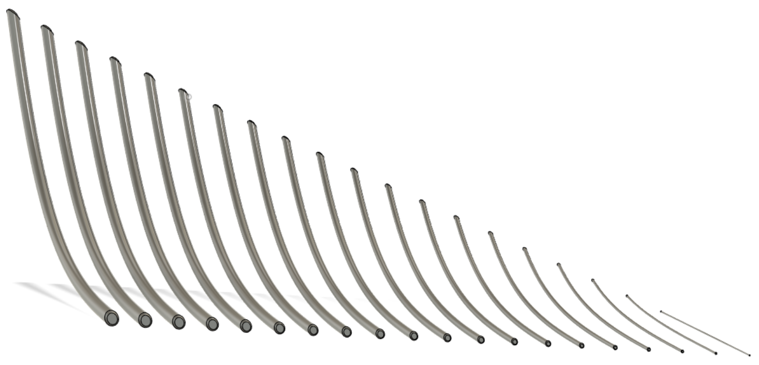


Figure 5.8: Rods created using splines, using the sweep feature.

Both examples were created with parametric characteristics in mind. The number of splines, their length, the number of spline fitting points and the strength of the curvature are used as parameters to modify the final shape produced by the script.

As each spline fitting point corresponds to a point defined by its spatial coordinates, the positional information from the exponential function can instead be used to place objects and generate abstract shapes. The example from Figure 5.9 was created from the spline examples, but instead of creating a spline point, a cube was placed at each spline location, using exponential functions for the splines. It shares some similarities to the generative solution set from Figure 5.5, without solution filtering. Instead, each geometry is evaluated and automatically color coded according to its total area, using the API to interact with the Appearances module of Fusion.

The same procedure can be used to extract relief information using height maps. Each pixel of a height map usually contains contains displacement (or height) data which can be translated into a mesh object using the API, providing a reliable reproduction of the surface topography captured by the image, and allowing images to be converted into meshes. These maps are usually black and white, but can also be in gradients, with lighter colors representing a protuberance and darker colors representing depressions Fig-

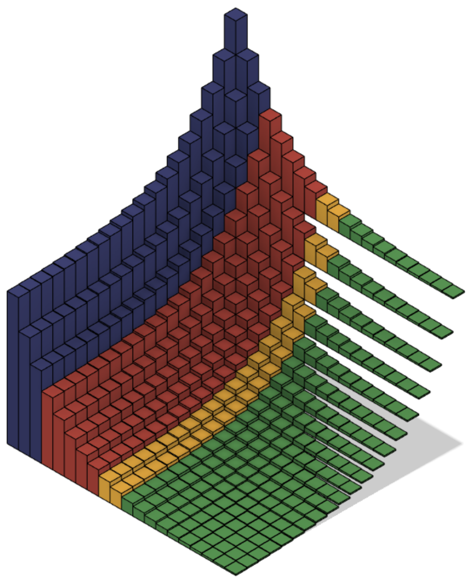


Figure 5.9: Geometries placed following an exponential function, and color coded based on their area. Green represents geometries under 600 cm^2, yellow between 600 and 1000 cm^2, red between 1000 and 3000 cm^2, and blue are over 3000 cm^2.

ure 5.10 depicts one add-in example, `Image2Surface.py`, a script which automatically converts any height maps into a 3D shape, either a mesh or a surface [Kellner 2016].

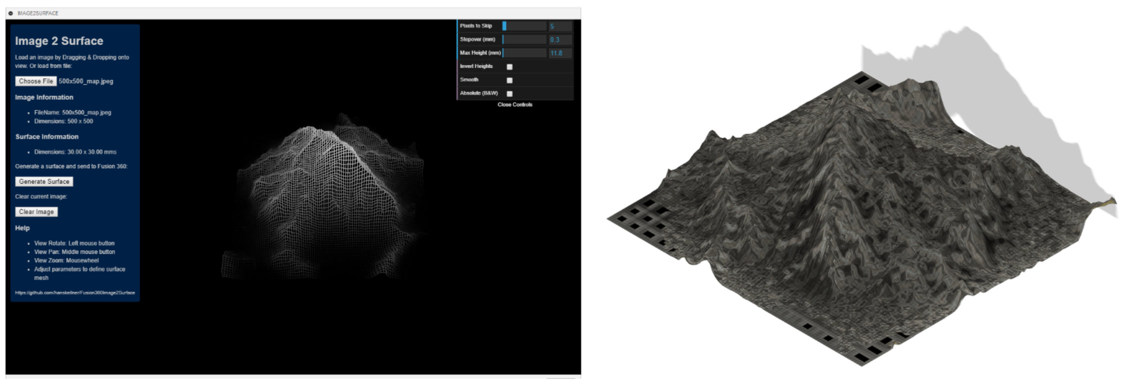


IMAGE2SURFACE

Image 2 Surface

Load an image by Dragging & Dropping onto view. Or load from file:

Choose File 500x500_map.jpeg

Image Information

- FileName: 500x500_map.jpeg
- Dimensions: 500 x 500

Surface Information

- Dimensions: 30.00 x 30.00 mms

Generate a surface and send to Fusion 360:

Generate Surface

Clear current image:

Clear Image

Help

- View Rotate: Left mouse button
- View Pan: Middle mouse button
- View Zoom: Mousewheel
- Adjust parameters to define surface mesh

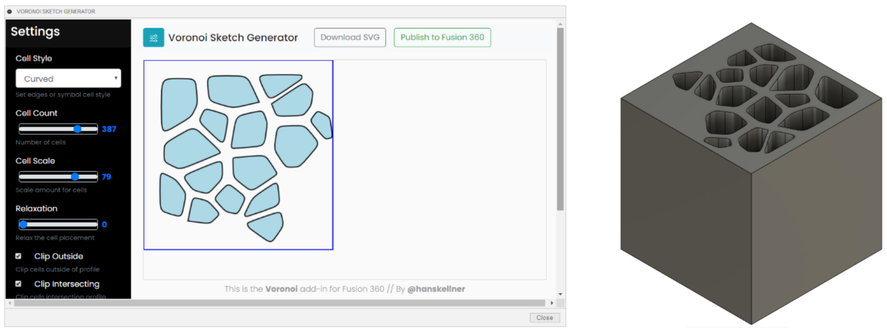
https://github.com/hanskellner/Fusion360Image2Surface

Pixels to Skip 5
Stepover (mm) 0.3
Max Height (mm) 11.8
Invert Heights ☐
Smooth ☐
Absolute (B&W) ☐
Close Controls

Figure 5.10: Mesh surface generation based on a terrain height map.

Texture maps can also be used to generate or edit shape. Instead of using the height data, in this example it is used to create profiles and apply them to a sketch selected by the user. The example from Figure 5.11 automatically generates profiles fitting the selected sketch profile. The profile shape, size, and number can be modified by the user before publishing the design to fusion. Once in fusion, the generated sketches will behave as any other sketch. The generated profiles were then used to cut a pre-built cube.

It is clear that using AD has a steep learning curve, necessitating programming background and API knowledge, although applying an AD perspective to certain examples can be beneficial when attempting to automate specific design tasks and inferring

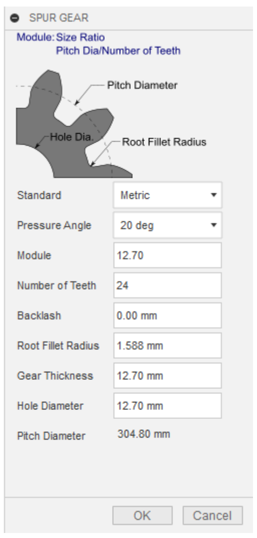


Cell Style: Curved
Cell Count: 387
Cell Scale: 79
Relaxation: 0
☑ Clip Outside
☑ Clip Intersecting

Figure 5.11: Automated voronoi sketch generation applied as a cut feature [Kellner 2016]

mathematical knowledge upon them, rather than attempting to create a fully completed design using just AD. However, features that require modification or customization can be built using AD principles, as it provides an interface to make alterations, as well as simplifying the modification process.

Figures 5.10 and 5.11 used elaborate interfaces created with the API. These are essential for enabling the customization of the design, as they enable the user to interact with the model in a simple way through the use of parameters, reducing the time needed to perform modifications when compared to manually updating the script and running it again. Simpler interfaces can be made, much like in Figure 5.12 from the included add-in Spur Gear. Aside from the options for the interface shown, others are available, including text boxes, sliders, radio buttons and even images, allowing the user not only customization of the design, but also the ability to tailor the interface for the desired design exploration.

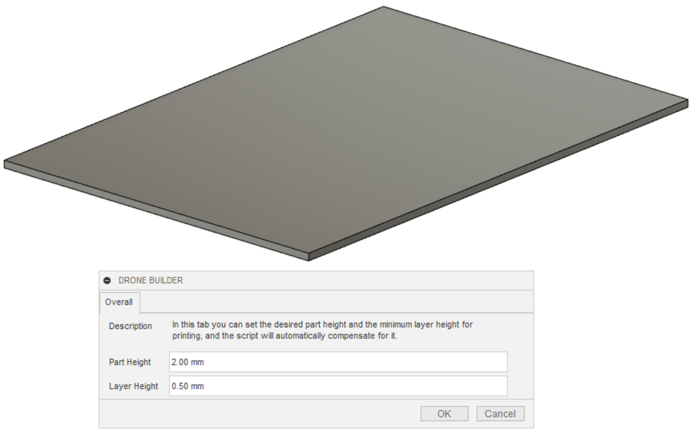


Standard	Metric
Pressure Angle	20 deg
Module	12.70
Number of Teeth	24
Backlash	0.00 mm
Root Fillet Radius	1.588 mm
Gear Thickness	12.70 mm
Hole Diameter	12.70 mm
Pitch Diameter	304.80 mm

Figure 5.12: Interface for the included add-in Spur Gear, which creates gears based on user inputs.

5.3.2 Applying DFAM Concepts

Additive Manufacture (AM) benefits greatly from automation, hence building rules for AM into a script will adapt geometries for specific manufacturing techniques. These are dependant of the manufacturing process and the complexity of the design. For example, parts for Fused Deposition modelling (FDM) benefit from no overhangs to limit the construction of supports and therefore reduce print time. Complex scripts can be built which can orientate the part to reduce the amount of supports, such as described in sub-section 4.2.1, but these are complex programs. A simple automation is the control of the part height in rengards to the layer height used by the printer. Each printer has a layer height defined in the slicer settings, which can be adjusted to account for several factors, such as nozzle diameter. Although not strictly necessary, if the layer height is set for a given, the height of the part should be a multiple of this to ensure the finished products printing tolerances are correct. Figure 5.13 depicts one such example: a script which creates a flat plate with a customization interface to allow the user to set its height and a desired layer height. The value the user inputs is automatically set as close to a multiple of the layer height as possible to ensure correct printing.



Description: In this tab you can set the desired part height and the minimum layer height for printing, and the script will automatically compensate for it.

Part Height: 2.00 mm

Layer Height: 0.50 mm

Figure 5.13: A flat plate with customizable height and layer height, where the part height is automatically adjusted to be a multiple of the layer height to improve printing tolerances.

5.4 Exploring Built-in Tools

Besides interacting with Fusion and creating geometry using scripts, there is also interest in its built-in tools, namely for their engineering applications, for the possibility that they might also include activities related to AD, and for the possibility of interacting with it through scripting. The tool selected for this is Autodesk's Generative Design (AGD), a module focused on alternative generation for specific design problems. AGD's inputs include loads, constraints, manufacturing methods, materials and a design goal. Although namely a Generative Design (GD) approach, it is a good candidate for interfacing using scripts since it is included in the Fusion environment, and therefore can constitute part of an Algorithmic Design (AD) approach.

5.4.1 AGD Methodology

The example chosen to test its design capabilities is a sample from the built in library in Fusion, a "Motorcycle Triple Clamp", depicted in Figure 5.14.

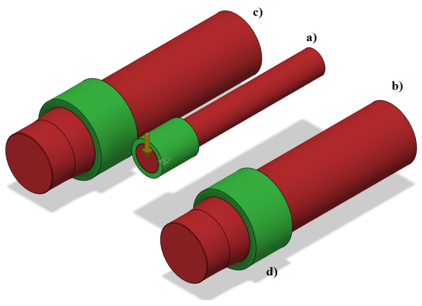


Figure 5.14: The prepared CAD model for the Motorcycle Triple Clamp example. a) Chassi connecting rod; b) Right suspension arm; c) Left suspension arm; d) Preserved geometry during the generation process.

In this example, the goal is to design a clamp between the chassis linkage point and the suspension arms of a motorcycle. Colored red are the chassis pickup points (FIG a) as well as the suspension arms (Fig b and c), indicating they are obstacle geometry in which material cannot be placed. Green geometry indicates that it is to be preserved during the creation process, and represents the initial geometry from which the algorithm will use to create shape. The central preserved ring has its constraints set to fixed, and a load case of 4500 N in the -Oz direction applied (perpendicular to the length of the obstacle geometry represented by the yellow arrow). Two additional bodies are placed vertically as obstacle geometry, one at each end of the preserved geometries, to guarantee that the finished solutions do not extend past this region.

The manufacturing methods selected were 2 axis cutting, 2.5, 3 and 5 axis milling, as well as additive and an Unrestricted fabrication method. AGD does not place limitations on the type of material used for a manufacturing method. As the amount of design results is dependant of the number manufacturing processes and materials used, the materials were chosen while taking into account their possibility of manufacture, but should merely be seen as illustrative of the entire process:

- 2 axis cutting and milling — Titanium 6Al-4V, Stainless Steel, Bronze, Brass, and Aluminium 7075, Aluminium 6061-O.
- Additive — ABS Plastic, PET Plastic, PC Plastic, Nylon 6, Titanium 6Al-4V, Aluminium and Poliamide.
- Unrestricted — Aluminium 6061-O, Titanium 6Al-4V, Stainless Steel, Bronze, and Brass.

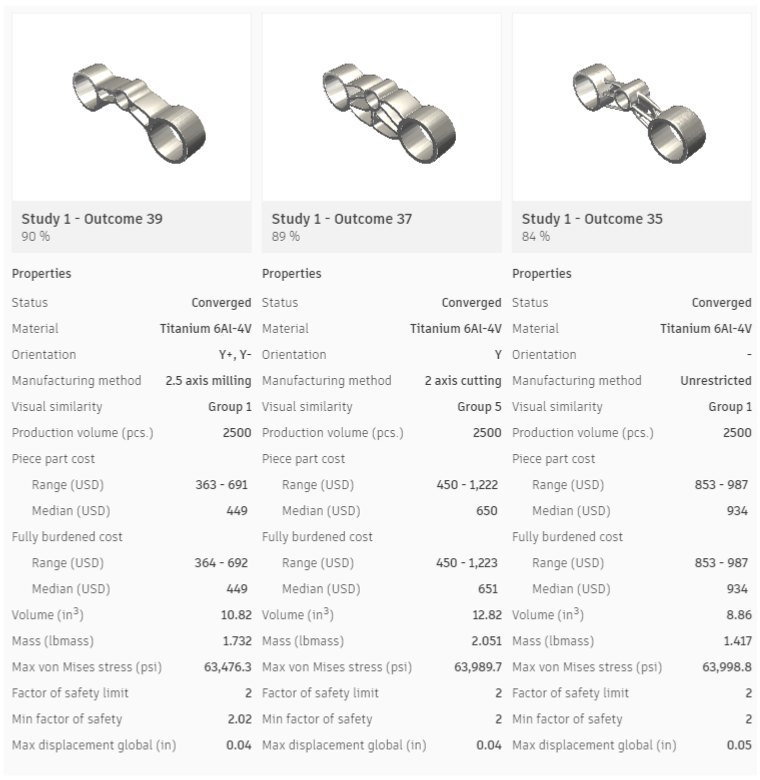
The materials for the 2-axis cutting and all the milling alternatives provided are identical, while Additive included thermoset plastics, and the Unrestricted method dispensed with Aluminium 6061-O. It is important to note that some materials are not ideal for the situation or for the manufacture method, but the interest of this example is to explore AGD's capabilities and to generate the most possible solutions for this design problem, they were still kept as possibilities. Fusion also provides cost estimates based on production volume, but does so to a limited number of materials available, and plastics such as ABS or PET are not among these. However, to provide more realistic materials for the Additive process, and since the library of additive materials for plastics is extremely limited, they were still included in the example.

The next step involves selecting the objective for the creation process, and AGD allows the user to select two goal functions, either minimize mass or maximize stiffness. The user can then set constraints for them, either the safety factor for the minimize mass goal function, or safety factor and mass target for the maximize stiffness goal function. In the interest of simplifying the process, the goal was set to minimize mass with the default safety factor. The example can now be run through Autodesk's cloud service, which frees the designer from using their own machine to compute results, while at the same time leaving the designer dependant of the cloud service's ability to process the request.

Once complete, the process yielded 48 total outcomes, called “studies”. Each outcome went through several iterations until it either converged, or the process terminated. AGD evaluates how well suited each result is to the design conditions, represented by a percentile recommendation under each result. These results can be filtered and sorted using a different number of available metrics, including volume, mass, material, and maximum displacement, as well as selecting which manufacture methods will be displayed at one time. The best recommended 3 studies of this process, selected from the titanium and stainless steel studies are displayed in Figure 5.15. Each study provides detailed information about its cost, process and mechanical properties, along with the study recommendation for their application.

To better visualize the totality of studies performed by AGD, the designer can select scatter plot view, and define the axis to be any of the same metrics available. Figure 5.16 contains a scatter plot with 8 studies, with mass as the X-axis and maximum displacement as the Y-axis, with the ”thumbs-up” icon representing a recommended study. Hovering over each plot displays information similar to Figure 5.15 without having to access another screen. For 3D viewing of the model, AGD can also display a stress reference map, providing insight on its mechanical properties, as depicted in Figure 5.17.

Taking a closer look at the solution creation process, although the final shape appears to have been ”grown” from the initial preserved geometry, the process actually consists in creating a structure that covers the rough shape of the part and avoids the defined obstacle geometry. To this Fusion applies a layer of Topological Optimization (TO), with further iterations refining the shape until the termination criteria is achieved, however the algorithm does not account for buckling, which may result in unrealistic results in pure compression or tension situations [University 2021]. At its core, AGD uses a level-set approach which is based on surface area, so additional outcomes can be achieved by simply varying the area of the preserved geometries [University 2021]. The total amount of outcomes obtained will vary depending on how the process is customized, but one can generally achieve a large solution space with relatively optimized parts for different manufacturing methods. This generative process avoids the deterministic approach of TO to finding the optimal solution, and instead focuses on displaying a wide arrange of options, but still reaps the benefits of TO by optimizing for different sets of conditions given by the user.



	Study 1 - Outcome 39	Study 1 - Outcome 37	Study 1 - Outcome 35
	90 %	89 %	84 %
Properties			
Status	Converged	Converged	Converged
Material	Titanium 6Al-4V	Titanium 6Al-4V	Titanium 6Al-4V
Orientation	Y+, Y-	Y	-
Manufacturing method	2.5 axis milling	2 axis cutting	Unrestricted
Visual similarity	Group 1	Group 5	Group 1
Production volume (pcs.)	2500	2500	2500
Piece part cost			
Range (USD)	363 - 691	450 - 1,222	853 - 987
Median (USD)	449	650	934
Fully burdened cost			
Range (USD)	364 - 692	450 - 1,223	853 - 987
Median (USD)	449	651	934
Volume (in^3)	10.82	12.82	8.86
Mass (lbmass)	1.732	2.051	1.417
Max von Mises stress (psi)	63,476.3	63,989.7	63,998.8
Factor of safety limit	2	2	2
Min factor of safety	2.02	2	2
Max displacement global (in)	0.04	0.04	0.05

Figure 5.15: Filtered results containing some of the parts manufactured using only stainless steel and titanium, sorted by recommendation.

In summary, AGD performs not only within the concept of a GD software, but also with an algorithmic base. However, it cannot be considered as a fully AD software, as the control given to the designer is rather limited. Although AGD does generate a considerable amount of alternative solutions for the proposed constraints, if the designer wants to modify the part to accommodate different types of suspensions with different dimensions, he will have to perform these modifications manually, before beginning the process all over again. In this way, AGD does not truly promote early stage design exploration, rather generating shapes for a single solution provided by the designer.

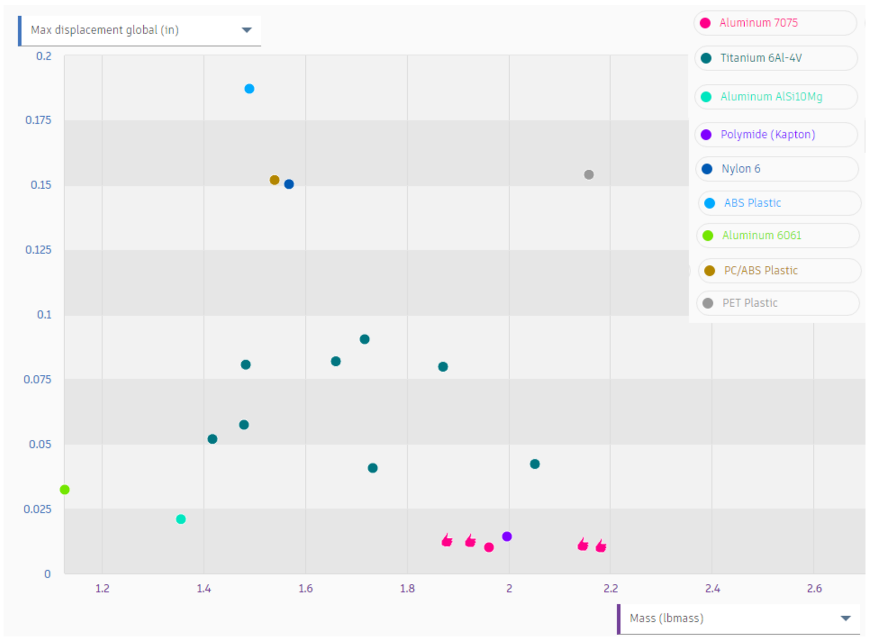


Figure 5.16: Scatter plot.

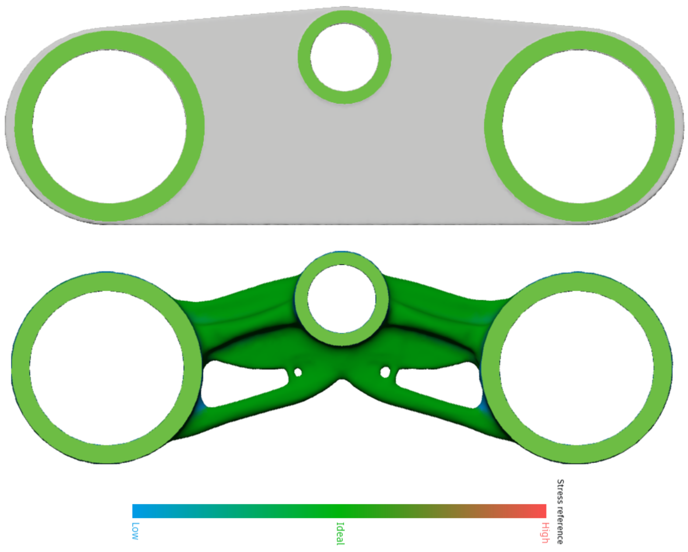


Figure 5.17: Top view of Outcome 42 showing the first iteration of the creation process with the preserved geometry shown in green (Top), and the stress map of the last iteration (Bottom).

Intentionally blank page.

Chapter 6

Case Study

6.1 Proposing a Methodology

To demonstrate an AD approach to a design problem, a practical application was developed using the methodology proposed in Figure 6.1. This process is to be applied in the early design stages, when it is necessary to explore different design alternatives, and as such is geared towards reducing the overall time spent exploring designs by automating menial tasks, as well as inferring knowledge relating to a specific products design, while providing a degree of expandability based on a good coding workflow. Due to its scripted nature, such approach allows for interfacing with other software tools, be they in the same ecosystem or not. It is of note that the particular object of this case study is not to develop a final product, rather to demonstrate the use of the methodology as a generic approach to a design problem.

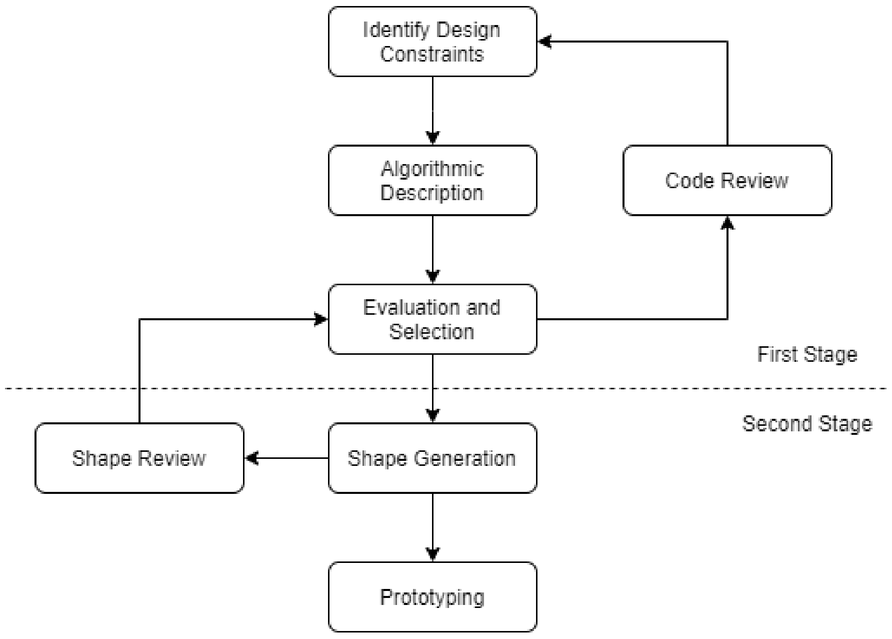


Figure 6.1: Proposed methodology for solving design problems using Algorithmic Design.

The proposed methodology is divided in two stages. The first stage is the most important, as it involves creating a script based on previously identified rules which describe the design object, as well as an interface to allow for direct manipulation of the design. Since the script will control the design exploration, it should be tailored in order to achieve the desired results, although it is always possible to perform further improvements to the script. The second stage is dedicated to generating the final shape and prototyping, which is not directly linked to design exploration. However, since such an approach results in several design alternatives, it is always possible to explore alternatives at this stage by combining several different solutions. The first stage comprises four steps:

- **Step 1: Identifying Design Constraints** — In this task, all important information relating to the design is collected. Selecting components, identifying their main dimensions and defining characteristics, key parameters which will allow for the desired design variation, as well as a design goal. A manufacture process is selected, along with a tool for shape generation in the second stage. After a tool is selected, any additional components necessary for its use are also identified. For example, using AGD as a second stage tool will necessitate the creation of preserve and obstacle geometry which needs to be identified before hand.

- **Step 2: Algorithmic Description** — After collecting all necessary information, a generic model of the design object is created, along with the customization interface which will allow the designer to edit the model. This interface should contain the parameters the designer believes to be necessary to achieve design variation in the intended manner. At the same time, it should be clear and guide the designer through the design exploration by limiting or advising about the desired parameters. Geometry generated at this stage should be simple in nature, mostly due to the long time needed to code complex shapes. The generic model is only a mockup of the general component layout, along with all other created geometries.

- **Step 3: Evaluation and Selection** — The actual design exploration stage, and a central step for this methodology. The script is run, and the designer uses the customization interface to explore different design alternatives. Running the script multiple times should also allow for side by side comparisons between several design alternatives. Alternatively, a GD dimension can be added by automatically generating the results based on a goal function. Other design considerations, initially missed or now apparent, can be added to the script to further enhance the exploration stage. After exploring alternatives, a design is selected to proceed to the second stage and the final shape generation.

- **Step 4: Code Review** — While exploring different alternatives in Step 3, other design considerations may be identified, or it may be made apparent that the alternatives generated are less than ideal. In this step, the newly identified constraints are added to the script to help better tailor the alternative generation.

Once the obtained alternatives are satisfactory, the second stage can begin. Since it is possible to have several different alternatives in this stage, this stage can create great variety by matching different components. This stage includes three steps:

- **Step 4: Shape generation** — The final object shape is created using the tool specified in step one of the first stage. There is no limitation on what this tool can be. It can be manual modelling based on the the Mockup stage, an AGD or TO based generation, another scripted approach created by the designer, or even other software tools which can use the exported model created in the first stage. Since the script was tailored for this process, it is important that this choice is made early in the first stage. If the results obtained for the object's shape is unrealistic or undesired, it is recommended to review and return to Step 3 in order to generate better solutions.

- **Step 5: Shape Review** — If the final shape is not satisfactory, it is also recommended to review and identify the possible problems, then return to Step 3 to attempt to generate better alternatives. At this point, if no other promising alternatives can be generated, the script should be reviewed.

- **Step 6: Prototyping** — With the final shape generated, all that is left is to produce the model with the selected manufacturing method to test the prototypes functionality and its manufacture feasibility.

6.2 Case Study

The case study selected applies the proposed methodology to custom drone construction, specifically on multicopters. These Unmanned Aerial Vehicles (UAV) are know for their agility and versatility which makes them suitable for use in areas of difficult access [Rible *et al.* 2020, Carney *et al.* 2021]. Applications include search and rescue, surveillance [Carney *et al.* 2021], topography, agriculture, sports [Vepa and Sagar 2019], cargo transport and forest fire detection [Ononiwu *et al.* 2016], and building inspection [Eiris *et al.* 2021, Donghai *et al.* 2021], among others.

Figure 6.2 contains some of the necessary components to build a quadcopter, with the exception of the battery. Since the frame is used to attach all components, it is chosen as the design object. Several authors have tackled the issue of frame building for drones. [Bright *et al.* 2021] sought to optimize an existing frame by using generative design, while others used FEM analysis to produce optimized frames [Vepa and Sagar 2019, Wei *et al.* 2015]. These examples all focus on building a frame for chosen components, but they do not allow for exploration of different combinations. Quadcopter frames alone can have a number of configurations in relationship to its arms [Bright *et al.* 2021], and components, although standard, can vary quite a bit in size and mounting location. The goal is then to create a frame which support the main components, as well as arms which support the motor and propellers which interface with the frame. However, it should be noted that the main focus is achieving design variation, rather than following a deterministic approach to be able to select the best possible solution.

The wide range of hobby-level drone components make it an excellent candidate for application of a customizable interface, where the user can input the specifications of each component to achieve different solutions. Thus, to automate design exploration, a customizable frame that can produce a large number of alternatives is created using Python. Design modularity is fundamental, so to better use Python's wide spread availability, the code developed for this example features several quality-of-life improvements to assist with adding further modules down the line.

The script generates design alternatives for combinations of drone components, as

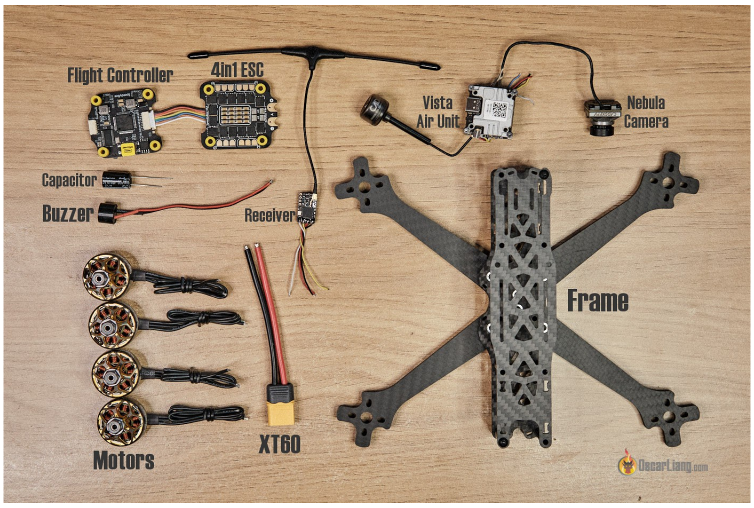


Figure 6.2: Main components of a quadcopter drone [Liang 2021].

well as their mounting points and functional surfaces, and utilizing manufacturing considerations for the FDM manufacturing process. In the interest of demonstrating the capabilities of an AD approach, three methods were used for the final stage: an AGD approach to generate the arms, an algorithmic approach, consisting of script based shape generation for the frame, and a manual approach to add the final details mostly for aesthetic purposes. While the number of main components can vary depending on the intended application, there are seven which are essential for any drone build:

- Motors — Each provides a rated amount of thrust for a specific propeller, while the number of motors will dictate the maximum lifting capacity and the overall shape of the frame.
- Propeller — Matched with specific motors, they provide the needed thrust for lifting.
- Battery and power distribution cables - Provides the power necessary for the motors and on-board electronics. Its capacity will dictate total flight time, which can be calculated by using the rated power for each motor.
- Propellers - Matched with specific motors, they provide the needed thrust for lifting.
- Electronic Speed Controller (ESC) — Allows for controlling motor speed individually.
- Flight Control Unit (FCU) — The main control unit of the drone, it manages all other electronic components, as well as user inputs to enable drone flight. Contains components such as gyroscopes and accelerometers to enable altitude and attitude control. FCU's can vary greatly in complexity, and some can be configured for automatic functions such as path following. Some FCU's also come in four-in-one configuration, in which the ESC and FCU are ”stacked” on top of each other to reduce required space.

- Radio transmitter and antenna — Responsible for receiving inputs from a transmitter on the ground to allow for drone control.

Quadcopters are the only type of drone available with "stacked" FCU and ESCs. Other types of multicopter require a separate ESC for each motor, which is then interfaced to the FCU. To simplify the design process, no particular action was taken for these types of ESCs as they can be attached anywhere, although they are most commonly attached to the arms.

6.3 Script Development

6.3.1 Design Constraints

Much like other design processes, the first step involves identifying which components are necessary, their initial mounting location, physical dimensions, general design considerations and predicted loads on the system, with most components being provided with detailed schematics which can be used to extract their characteristics. Since the most complex method of shape generation is AGD, it is also necessary to create obstacle and preserve geometry, which will be used for the algorithmic and manual approach as well.

After identifying the required components, parameters are selected to allow for design variation. In this case, as the script is intended to work with primitive shapes, each component is simplified into a shape that will contain it. For example, motors and propellers are simplified to cylinders, while the battery and FCU are represented by rectangular parallelepipeds. Each components dimensions are converted to parameters, as well as variables indicating their location in relation to others, which allows for customizing each individual component for the users needs.

The components provide a general layout upon which to build the frame. Figure 6.3 represents a possible layout for a quadcopter from which was used for initial parameter extraction. Three main parameters control the number of motors (and, with this, the number of arms), the arm length measured from the center of the frame, and in the case of quadcopters, the angle to the horizontal. For other types of multicopter there is little to gain in controlling the angle directly, as these focus on stability and as such should have their motors equally spaced. Together with the component related parameters, they control the main design variation. To add further detail, other geometries are considered, mainly the support points for the motors, the arm connection to the frame, along with supports for the components within it. These geometries hold the secondary purpose of serving as obstacle and preserve geometry for AGD latter in the second stage. In addition, several holes were made to be able to mount the motors, FCU and the frame-arm connection using screws.

6.3.2 Algorithmic Description

With all parameters extracted, a generic model is built, as well as the customization interface to interact with them. Since the model is intended as a representation, rather than a final product, all geometric shapes were kept as simple as possible. The resulting model is depicted in Figure 6.4, with each body being color coded to facilitate identification. Green bodies are the frame sections, blue bodies correspond to the frame

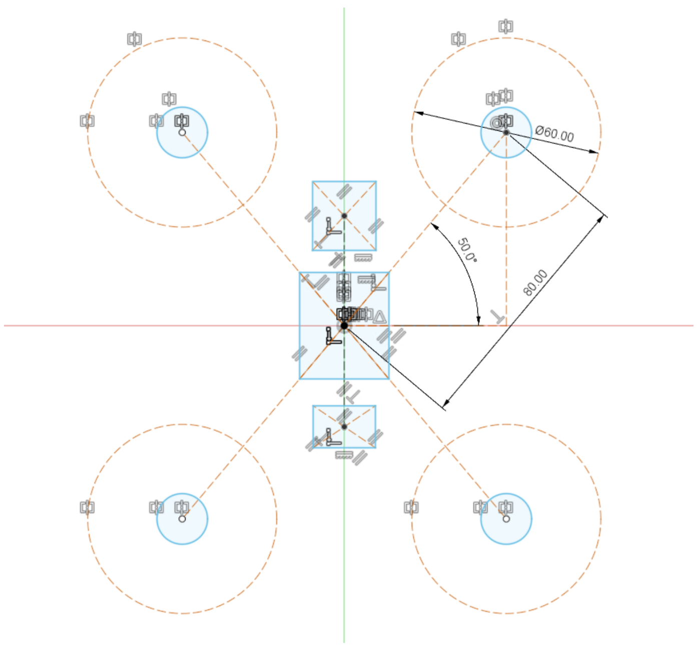


Figure 6.3: Example sketch representing the quadcopter layout (with some components) which the frame is based upon, based on the layout from [Liang 2021].

and motor supports, yellow are the motors and red are the propellers, and the standard components are default gray. For illustration purposes, a propeller body and the top frame section were hidden.

In the interest of facilitating 3D printing for prototyping, the frame was separated into a top and bottom sections, which are flat with no overhangs to avoid the use of supports when printing. They differ only in the mounting holes for the FCU which are present in the bottom frame, but absent from the top frame. Separation is achieved through frame supports (in blue) which are also the attachment points for the arms, providing clearance for the components between frame sections.

Interfacing with each parameter is done through a customization interface, which is separated into tabs, each with a small description for ease of use, depicted in Figure 6.5. The first tab contains the main parameters related to the frame, with the four following tabs controlling each component corresponding to each component, as well as general frame dimensions and mounting holes.

Due to its scripted nature, it is possible to improve the interface further by having the script respond to incorrect inputs, or provide warnings for conflicting geometries, in order to further guide the user.

6.3.3 Evaluation and Selection

With the script complete, the exploration process can begin. After some experimenting, a quadcopter solution was selected, depicted in Figure 6.6. It was selected due to its symmetric nature, which necessitates only one-fourth of the final design to be completed to arrive at the final solution.

The solution followed common guidelines for drone building. Starting with the frame

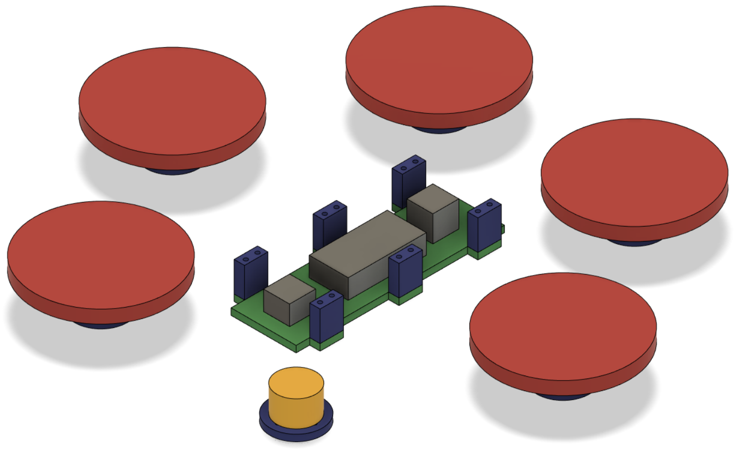
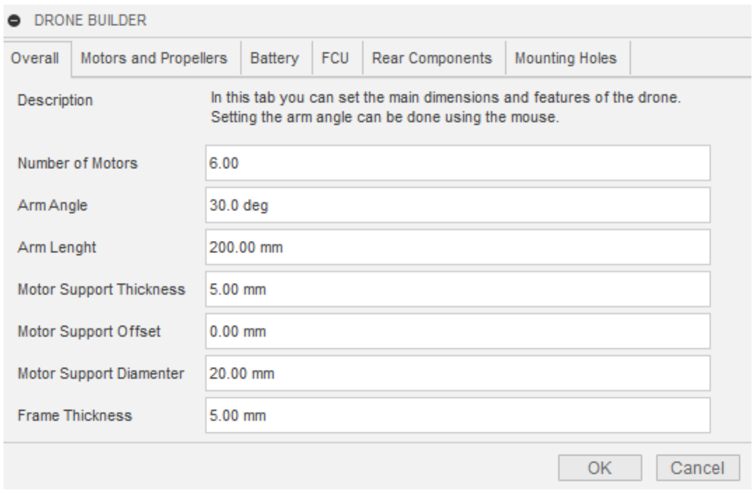


Figure 6.4: Generic model for a hexacopter obtained from all the extracted parameters.



Description	In this tab you can set the main dimensions and features of the drone. Setting the arm angle can be done using the mouse.
Number of Motors	6.00
Arm Angle	30.0 deg
Arm Lenght	200.00 mm
Motor Support Thickness	5.00 mm
Motor Support Offset	0.00 mm
Motor Support Diamenter	20.00 mm
Frame Thickness	5.00 mm

Figure 6.5: The interface built to interact with the model.

size, which is measured from tip to tip of each motor, the maximum propeller size corresponds to half the the frame size to avoid propeller interface. For multicopters above four motors, this rule isn't particularly accurate, but serves as a basis. Motors are then chosen based on propeller size, with ESC's and batteries chosen based on motor power requirements. The FCU is then chosen to connect all these components. The frame and its supports are automatically sized by the script to accommodate the chosen components. The intricacies of the calculations needed for drone design are omitted from this process as it is not the intended goal of this case study. As such, the solutions are not an optimized way of drone building and should only be seen as a design approximation, but serve the purpose of allowing unorthodox drone designs. Most importantly, the interface allows for quick and easy editing, while taking into account component relationships, which ensures that the whole model will react to changes in a logical way.

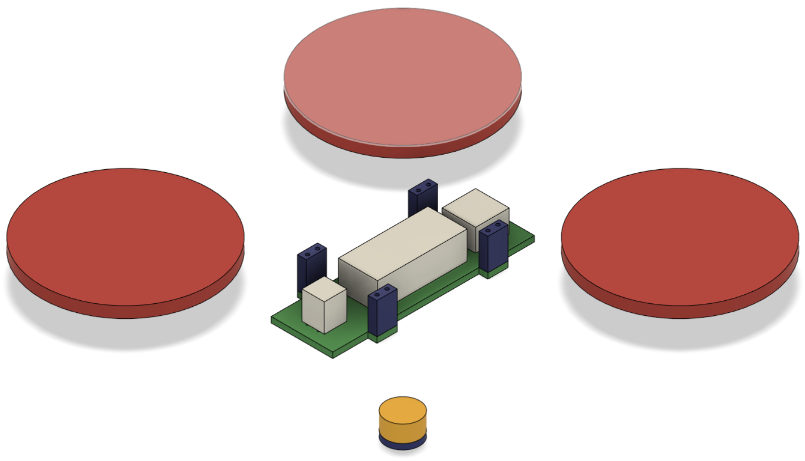


Figure 6.6: Selected model for further refinement.

6.4 Second Stage

After arriving at the desired solution, it it put through further development throughout the second stage. At this stage the model is far too simple to be functional, so this stage generates the final shape before prototyping. The frame sections will be generated through a mix of algorithmic (scripts) and manual approaches, while the arms will be created using AGD. Thanks to the models symmetry, only one arm needs to be designed to arrive to the final model.

6.4.1 Creating Finished Models

Starting with the frame sections, they were prepared for application of the Voronoi script from Section 5.4.1. At this point, further design alternatives can be created by creating several complete models an matching components between them, before selecting a final version. The process of preparing the sections consisted of filleting desired locations and creating a boundary sketch within the frame section to avoid interfering with areas containing mounting holes. After the frame was prepared, the Voronoi script was applied several times to achieve several solutions, two of which are depicted in Figure 6.7.

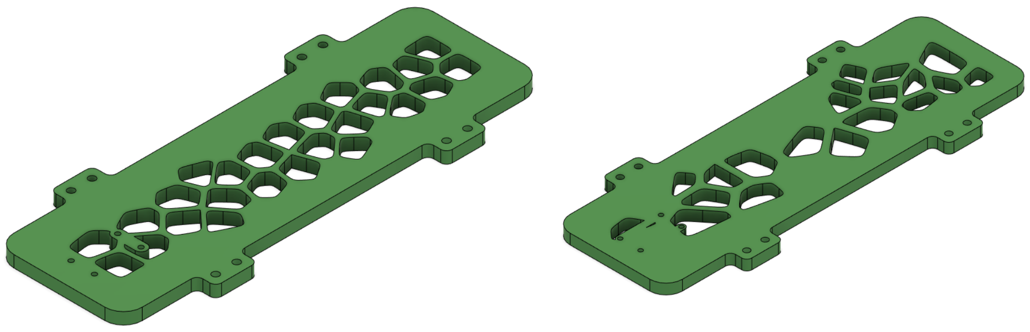


Figure 6.7: Two solutions for the frame section, after application of the Voronoi Script.

This process served three main purposes: to reduce the total amount of material and time needed to print the frame sections while still retaining some structural strength, to make the model more aesthetically pleasing, and to create more alternatives before picking a final design. Once the frame was complete, an arm was created from the motor and frame supports (in blue), using the process described in Section 5.4.1, making use of the geometries built with the script and color coded in blue. Although AGD is meant to obtain a range of solutions for a specific structural problem, in this case it was rather used to generate several geometries to choose from. Three visually unique solutions are presented in Figure 6.8.

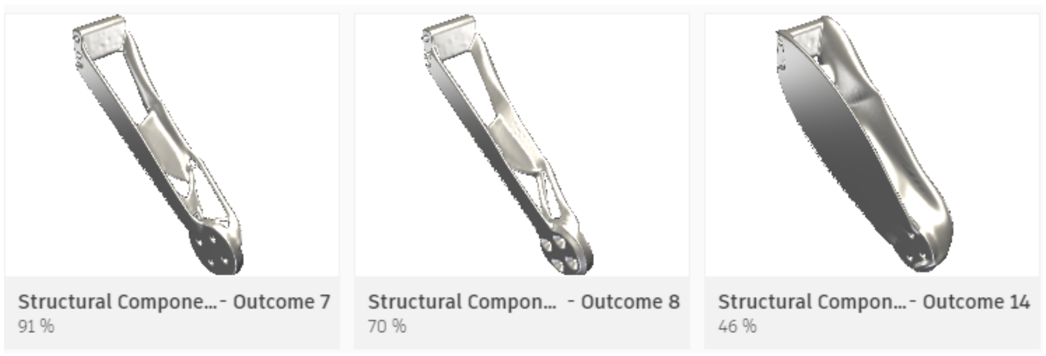

Structural Compone...- Outcome 7
91 %
Structural Compon... - Outcome 8
70 %
Structural Compon...- Outcome 14
46 %

Figure 6.8: Three unique solutions obtained from AGD, ordered by recommendation.

Lastly, the final models can be assembled from these parts. Although the frame sections were ready for assembly, some artifacts resulted from the AGD process, necessitating for some post-processing before final assembly. This consisted of reopening the screw holes which were closed during shape generation, as well as refining contact surfaces to guarantee proper assembly. Due to the different alternatives generated for each part, several finished models can be obtained by simply switching parts. One result depicted in Figure 6.9 has the arms built following an unrestricted manufacturing method, while the one depicted in Figure 6.10 was created based on Additive Manufacturing (AM). For illustrative purposes all other components were removed to provide a clearer visualization.

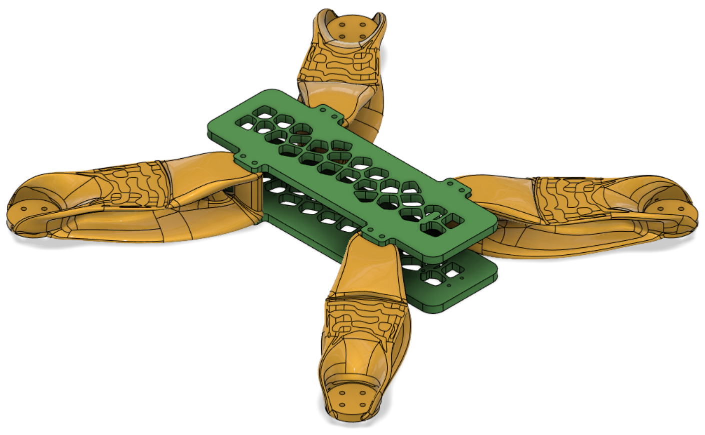


Figure 6.9: An assembled model containing both frames and all arms.

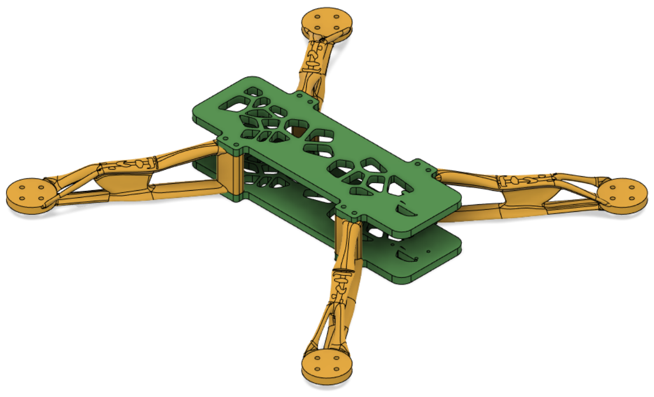


Figure 6.10: Another possible solution with a different set of arms and a different voronoi profile applied.

6.4.2 Prototyping

The final step in the proposed AD methodology is prototyping the final model. Both frame sections are flat with few details, but each arm is a complex geometry, which is where AM is advantageous. Resorting to AM technologies in cases such as these allows timely fabrication of the arm structure using only one machine. The arms and frame sections were specifically designed with this AM in mind, specifically FDM, as it is the most common method. To demonstrate how these parts are fabricated, Ultimaker Cura (shortened to Cura) was used to slice them and prepare them for printing.

Throughout the development of this case study, the subject of engineering a product was deliberately avoided in order to be able to focus on exploring design alternatives, rather than producing the best possible solution. However the use AGD and AM allows for objective comparison of solutions in terms of their functionality, hence the arm solutions obtained with AGD from Figure 6.9 and Figure 6.10 are compared. Regarding the frame sections, these were designed to be easily printed without the use of support structures, and as such present no significant challenge for this type of manufacturing., with a printing time of under two hours per section.

The first arm solution, now sliced, is depicted in Figure 6.11, with the support structures represented in cyan, the walls in red, the filling in yellow, and with a total printing time of seven hours. Although this specific arm solution was not obtained for use with AM (rather, this solution is for an unrestricted method), Cura shows that it is still possible to manufacture such a complex geometry, however it necessitates the extensive use of support structures which are not ideal, but produce a functional model in the end.

As for the other arm solution, this time optimized (through AGD) for AM, and depicted in Figure 6.12 with the same color coding. This solution should be as functional as the one presented in Figure 6.11, but requires far less supports and as such has a final printing time of just two hours. Notably, the underside of the arm geometry, which contacts the building surface of the printer, is now perfectly flat to improve adhesion and reduce the amount of supports needed.

Since the goal has always been to generate alternative solutions, any of the finished

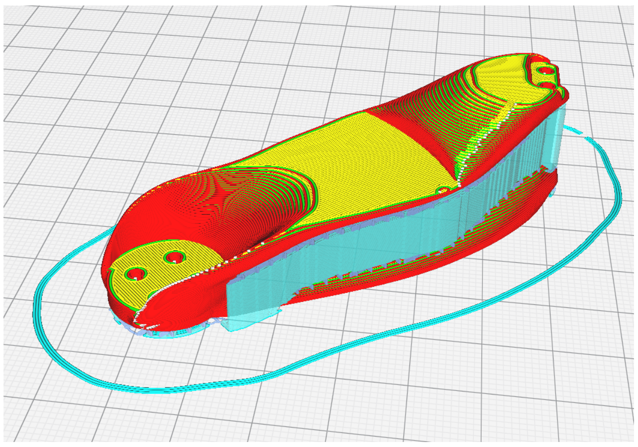


Figure 6.11: Sliced arm geometry in the Cura environment.

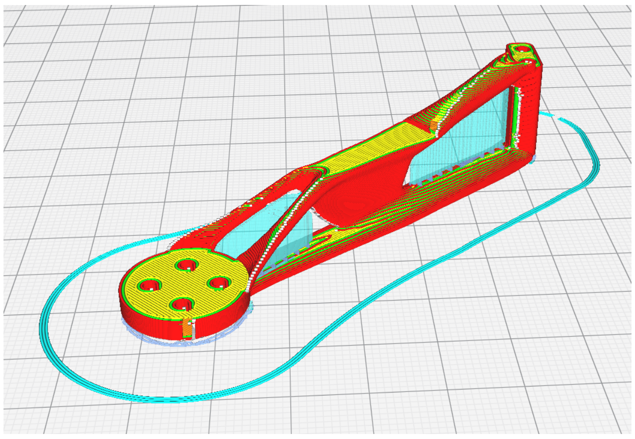


Figure 6.12: Alternative arm geometry, sliced in the Cura environment.

models could be used to develop a final product, but since they were created for design exploration rather than engineered to solve a problem, not much can be said about their actual functionality. To improve these models, other structural considerations should be taken into account when using AGD, and additional care with manufacturing methods and materials should be taken. Despite their possibly lackluster performance, both solutions achieve their goal of actually being possible to manufacture. However, if a decision had to be made, AGD provides most of the information needed to be able compare both solutions, such as mass, maximum stress, predicted cost and maximum displacement. With Cura providing information about each models manufacturing time and cost, plus the customization inherent in a slicer software, the designer has tools to assist in making an informed selection of solutions throughout the process.

6.5 Script Additions

One of the strengths of using an AD based approach is the ability to tailor the scripts to better explore design alternatives. The script used for this case study was built on a parametric basis, using parameters to control the intended design variation while maintaining the capability of manually altering them, resulting in a combination of both PD and AD. To better portray the capabilities of scripting inside a CAD tool like Fusion, the initial script was modified to include a generative process together with the PD and AD approaches already present.

6.5.1 Generative Design Variation

To achieve the desired GD process with the same parametric base, the main parameters relating to frame dimensions, the number of arms, their length and their angle to the horizontal, are now calculated automatically by the script, instead of being manually introduced by the user. All remaining parameters relating to off-the-shelf components, like the battery or FCU are kept as user inputs so as to serve for a basis for the design.

The result is depicted in Figure 6.13, with four total design outcomes from which the designer can choose from. For each solution presented, the angle and length of each arm is randomized to achieve the desired design variation. However, the arm length was made a function of the propeller diameter to ensure proper clearance between the propellers and the central frame sections.

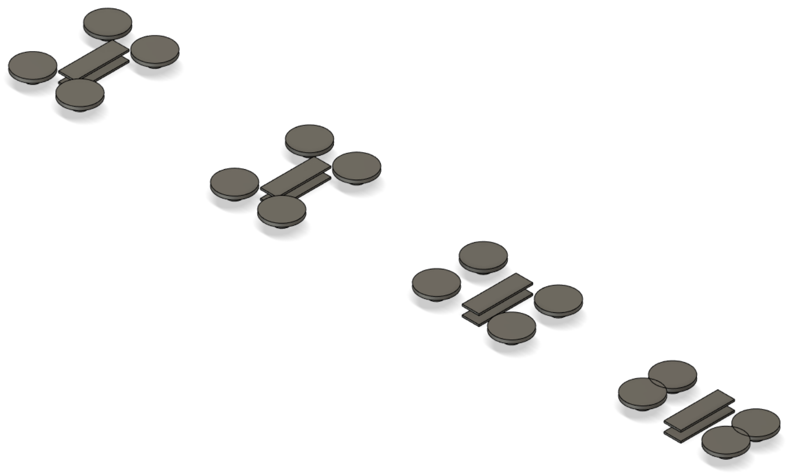


Figure 6.13: Generative modification to the script, returning four alternatives based on user defined components, which the designer can then choose from.

Randomizing is one of many ways design variation can be achieved using this approach. Using functions as a basis to achieve variation, or having the script iterate until it achieves an optimum value for these parameters. This parameter optimization is not limited to GD approaches, as the original script is also a good candidate to its application. However, the goal was to explore different design solutions, rather than optimize results for a specific case, so no optimization was included in the script.

6.6 Discussion

Creating a scripted tool for design exploration proved to have its challenges, the main of which was establishing the connection between scrip and actual model. Locating each body in spatial coordinates is simple when dealing with simple and few geometries, but once more geometries were added, it became difficult to keep track of geometric relationships and the coordinates associated to them. This also complicates the process of designing complex shapes, hence the reason why shape generation is relegated to other tools. However, once the actual script was developed, it largely simplified the process of exploring solutions, in which a design can be edited easily to meet changing requirements through the design process. This process is further expedited by the use of the interface, which although laborious to build, is easily expanded with new variables and edited to provide other ways of altering the design, such as direct manipulators.

The time benefit in following the proposed methodology when compared to a completely manual approach is dependant of several factors, such as the design complexity, the designers previous coding experience, and the intended use of the script. The nature of complex designs will define its ease of implementation, as reducing a geometry to mathematical equations with splines or using primitive geometries proves easier than manually editing. Another factor is the scripts intended use. If it is only required for a single use case, and the design is complex in nature, this methodology is not ideal. However, building a tool which supports expandability, either by the designer or other parties, using a well known programming language such as Python, can improve upon such a tool with much more integrated knowledge.

Throughout the script design process, a certain similarity was recognized to programs such as Grasshopper. In Grasshopper, geometries are created by feeding parameters through rule sets which will perform different tasks. Much in the same way, when scripting for use within Fusion, functions were created to simplify the process of creating geometries. Since geometries were simple cylinders and paralelipipeds, this required only two functions which received parameters such as cartesian coordinates, extrusion height, and type of operation (such as join, cut or new body). These functions serve as the rule sets used by Grasshopper, as they take in parameters to perform certain tasks, further reinforcing the use of scripting as an Algorithmic approach.

Lastly, following this methodology revealed that not only did it became possible to generate alternatives easily, but also greatly simplified the process of evaluating and selecting these alternatives. The result is several finished models with some degree of modularity, further expanding the alternatives made available to the user.

6.7 Future Developments

Expandability was always the focus of the case study, so to provide better insight to what could be done to improve the drone building tool in particular, this section will cover some of the immediate works that could be done to provide an improved design exploration experience.

6.7.1 Design Library

There is a limited number of components available for drone builds. As such, it is entirely possible to create a library within the script to store the values relating to each component, made simpler if this tool is divulged within a community which can work to improve it. This way, the user could simply select their desired component from a list instead of manipulating the values directly. Adding to the component list, and taking advantage of several users using the tool, the same could be done to the frame parts generated during the design process, further increasing the total designs available for exploration.

6.7.2 Algorithmic Geometries

Since the final shape generation did not utilize an algorithmic or scripted procedure, an additional future work would be to contain this stage within the script itself. The exploration process would then be fully contained within a single tool, removing the necessity of resorting to external means for shape creation and allowing a more finalized view of the model during design exploration. Other additions can include geometries for fine tuning the AGD process. The case study within this work utilized only the necessary geometries for its use to maintain the scripts simplicity, but it can be improved by further adding geometry to the model solely for this purpose.

6.7.3 Including Design Considerations

The general guidelines used to build a drone can be directly implemented into the script. Since the main goal was to simply generate alternatives and show the potential of such an approach, other functionalities were left out during development. For future developments, instead of leaving all decisions to the designer, some of the parameters would be automatically calculated by the script, based on rules for drone design.

Chapter 7

Final Remarks

7.1 Parametric, Generative and Algorithmic Approaches

This work was started to evaluate the concept of Algorithmic Design (AD) approaches when applied to engineering design, a topic relatively recent in architecture, while at the same time understanding its relevance within Additive Manufacturing (AM), in order to create truly customizable products. From the literary review, two design approaches stood out as interconnected with the use of AD: Parametric Design (PD) and Generative Design (GD).

PD often generated confusion, as Computer Aided Design (CAD) software tools are generally parametric, hence one could argue that all CAD models were made using PD. However, here the distinction between variable and parameter is important: a variable is any value which represents a specific measure, while a parameter is a variable deliberately chosen to introduce variation into the design, usually introduced with relationships to other parameters to ensure proper model adaptation without errors. Therefore, a PD approach is different from CAD modelling, as its purpose is to explore and not to transcribe an already finalized designed into the digital workspace.

The other approach with relation to AD is GD. Here several designs are generated at the same time for the user to select from. Autodesk's Generative Design (AGD) makes for an excellent example, generating several part solutions based on structural loads and constraints, with a layer of Topological Optimization (TO) to optimize the design, and then allows the user to filter the design outcomes to reach the best solution. However, software tools such as AGD, as great as they may be for engineering applications, may cause users to perceive GD as an approach which requires TO, when the reality is that GD approaches seek only to automate the process of generating several solutions.

When compared to GD, AD provides more freedom of choice. AD approaches are based on algorithms, in this case scripts, which can be specifically tailored towards assisting with early design exploration. They allow for design automation, while providing unique ways of interacting with the design. Due to their scripted nature, AD approaches can also include parametric and generative processes, while also integrating important information into the design through mathematical functions or optimizations. However, AD approaches suffer from a steep learning curve, not only do they require programming experience, but they also complicate the creating of complex shapes using only scripted methods. Therefore, AD approaches are useful in most stages of design, either to automate tasks or for design exploration, but should be used mainly as a early stage exploration tool when designing complex 3D models.

7.2 On Additive Manufacturing and Customization

Much like what was demonstrated with the case study, AD approaches allow for customization of the design. By setting parameters as a way to achieve design variation and placing them in an interface, the designer has real time control over the design aspects and can tailor it for any number of situations. Moreover, this type of approach allows the creation of multiple design alternatives in a relative short amount of time.

To take advantage of the ability to customize the designs, Additive Manufacturing (AM) was used to ensure that parts could be manufactured in a timely manner, and to allow for a never-ending customization of the design. Furthermore, by integrating scripts with other tools, complex, organic-like structures can be created, which AM is capable of producing at reduced costs when compared to other more traditional methods such as multi-axis machining. Tool integration can help offset the difficult process of accurate shape creation using scripts, but it is in itself a complex procedure, which might require interfacing between two or more different software tools. Notwithstanding, even if tool integration is not possible automatically, scripts can also help create the necessary conditions for the use of such tools, even if manual input is required. This was the process followed for the case study, which not only generated fairly complex results for the drone arms, but also optimized them for use with 3D printers, greatly reducing total print time and cost.

Lastly, what is most notable about an AD approach, besides its ability of constantly updating and improving the scripts, is the ability of integrating it with other approaches to facilitate or improve the design, resulting in a more complete development process. Hence, CD approaches, such as the ones described throughout this document, must not be thought as isolated inside product design, but as a way to enhance the development process with different techniques.

Bibliography

[Abdulhameed *et al.* 2019] O. Abdulhameed, A. Al-Ahmari, W. Ameen and S. H. Mian. Additive manufacturing: Challenges, trends, and applications. *Advances in Mechanical Engineering*, 11(2), 2019.

[Autodesk 2020] Autodesk. Demystifying Generative Design. `https://damassets.autodesk.net/content/dam/autodesk/www/solutions/generative-design/autodesk-aec-generative-design-ebook.pdf`, 2020. Accessed: 2021-07-14.

[Bernhardt *et al.* 2007] D. Bernhardt, Q. Liu and K. Serfes. Product customization. *European Economic Review*, 51(6):1396–1422, 2007.

[Biedermann and Meboldt 2020] M. Biedermann and M. Meboldt. Computational design synthesis of additive manufactured multi-flow nozzles. *Additive Manufacturing*, 35, 2020.

[Bikas *et al.* 2016] H. Bikas, P. Stavropoulos and G. Chryssolouris. Additive manufacturing methods and modeling approaches: A critical review. *International Journal of Advanced Manufacturing Technology*, 83(1-4), 2016.

[Briard *et al.* 2020] T. Briard, F. Segonds, and N. Zamariola. G-DfAM: a methodological proposal of generative design for additive manufacturing in the automotive industry. *International Journal on Interactive Design and Manufacturing*, 14(3), 2020.

[Bright *et al.* 2021] J. Bright, R. Suryaprakash, S. Akash and A. Giridharan. Optimization of quadcopter frame using generative design and comparison with DJI F450 drone frame. *IOP Conference Series: Materials Science and Engineering*, 1012, 2021.

[Bukhari 2011] F. A. Bukhari. A Hierarchical Evolutionary Algorithmic Design (HEAD) System for Generating and Evolving Building Design Models. 2011.

[Buonamici *et al.* 2020] F. Buonamici, M. Carfagni, R. Furferi, Y. Volpe and L. Governi. Generative design: An explorative study. *Computer-Aided Design and Applications*, 18(1), 2020.

[Caetano *et al.* 2020] I. Caetano, L. Santos and A. Leitão. Computational design in architecture: Defining parametric, generative, and algorithmic design, 2020.

[Camburn *et al.* 2017] B. Camburn, V. Viswanathan, J. Linsey, D. Anderson, D. Jensen, R. Crawford, K. Otto and K. Wood. Design prototyping methods: State of the art in strategies, techniques, and guidelines. *Design Science*, 3, 2017.

[Carney *et al.* 2021] R. Carney, M. Chyba, C. Gray, G. Wilkens and C. Shanbrom. Multi-agent systems for quadcopters. *Journal of Geometric Mechanics*, 0(0), 2021.

[Colombo *et al.* 2020] E. F. Colombo, N. Shougarian, K. Sinha, G. Cascini and O. L. de Weck. Value analysis for customizable modular product platforms: theory and case study. *Research in Engineering Design*, 31(1), 2020.

[Deradjat and Minshall 2017] D. Deradjat and T. Minshall. Implementation of rapid manufacturing for mass customisation. *Journal of Manufacturing Technology Management*, 28(1), 2017.

[Dino 2012] I. Dino. Creative design exploration by parametric generative systems in architecture. *Metu Journal of the Faculty of Architecture*, 29(1), 2012.

[Donghai *et al.* 2021] L. Donghai, X. Xietian, J. Chen and L. Shuai. Integrating Building Information Model and Augmented Reality for Drone-Based Building Inspection. *Journal of Computing in Civil Engineering*, 35(2), 2021.

[Durakovic 2018] B. Durakovic. Design for additive manufacturing: Benefits, trends and challenges. *Periodicals of Engineering and Natural Sciences*, 6(2), 2018.

[Eiris *et al.* 2021] R. Eiris, G. Albeaino, M. Gheisari, W. Benda and R. Faris. InDrone: a 2D-based drone flight behavior visualization platform for indoor building inspection. *Smart and Sustainable Built Environment*, 2021.

[Eltaweel and SU 2017] A. Eltaweel and Y. SU. Parametric design and daylighting: A literature review, 2017.

[Erinks 2015] B. Erinks. Get started with Fusion 360 API, 2015. Accessed: 2021-04-20.

[Felfernig *et al.* 2001] A. Felfernig, G. Friedrich and D. Jannach. Conceptual modeling for configuration of mass-customizable products. *Artificial Intelligence in Engineering*, 15(2), 2001.

[García-Alvarado *et al.* 2021] R. García-Alvarado, G. Moroni-Orellana and P. Banda-Pérez. Architectural evaluation of 3d-printed buildings, 2021.

[Gardan and Schneider 2015] N. Gardan and A. Schneider. Topological optimization of internal patterns and support in additive manufacturing. *Journal of Manufacturing Systems*, 37, 2015.

[Gebisa and Lemu 2017] A. W. Gebisa and H. G. Lemu. Design for manufacturing to design for Additive Manufacturing: Analysis of implications for design optimality and product sustainability. *Procedia Manufacturing*, 13, 2017.

[Guidera 2011] S. Guidera. Conceptual design exploration in architecture using parametric generative computing: A case study. In *ASEE Annual Conference and Exposition, Conference Proceedings*, 2011.

[Gunagama 2018] G. M. Gunagama. Generative Algorithms in Alternative Design Exploration. *SHS Web of Conferences*, 41, 2018.

[He *et al.* 2019] L. He, M. Gilbert and X. Song. A Python script for adaptive layout optimization of trusses. *Structural and Multidisciplinary Optimization*, 60(2), 2019.

[Holmström *et al.* 2016] J. Holmström, M. Holweg, S. H. Khajavi and J. Partanen. The direct digital manufacturing (r)evolution: definition of a research agenda. *Operations Management Research*, 9(1-2), 2016.

[K. Terzidis and Srinivasan 2004] J. Isorna K. Terzidis and V. Srinivasan. Algorithmic Design: a Paradigm Shift in Architecture? 2004.

[Kellner 2016] Hans. Kellner. Fusion 360 Image 2 Surface Add-In. `https://github.com/hanskellner/Fusion360Image2Surface`, 2016.

[Kolarevic 2004] B. Kolarevic. Architecture in the digital age: Design and manufacturing. 2004.

[Krish 2011] S. Krish. A practical generative design method. *CAD Computer Aided Design*, 43(1), 2011.

[Lee and Song 2021] K. S. Lee and H. K. Song. Automation of 3D average human body shape modeling using Rhino and Grasshopper Algorithm. *Fashion and Textiles*, 8(1), 2021.

[Liang 2021] Oscar Liang. How to Build an FPV Drone Tutorial (DJI FPV System). `https://oscarliang.com/build-fpv-drone-dji/`, 2021.

[Martinho *et al.* 2020] H. Martinho, I. Pereira, S. Feist and A. Leitão. Integrated Algorithmic Design in Practice. *Anthropologic: Architecture and Fabrication in the cognitive age - Proceedings of the 38th eCAADe Conference - Volume 1*, 1(September), 2020.

[Medellin-Castillo and Zaragoza-Siqueiros 2019] H. I. Medellin-Castillo and J. Zaragoza-Siqueiros. Design and Manufacturing Strategies for Fused Deposition Modelling in Additive Manufacturing: A Review, 2019.

[Meurer *et al.* 2017] A. S. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. T. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, S. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman and A. Scopatz. SymPy: Symbolic computing in python. *PeerJ Computer Science*, 2017(1), 2017.

[Mountstephensand and Teo 2020] J. Mountstephensand and J. Teo. Progress and challenges in generative product design: A review of systems, 2020.

[Mwema and Akinlabi 2020] F. M. Mwema and E. T. Akinlabi. Basics of Fused Deposition Modelling (FDM). In *SpringerBriefs in Applied Sciences and Technology*. 2020.

[of Colorado 2021] University of Colorado. Engineering Design Process. `https://www.teachengineering.org/populartopics/designprocess`, 2021. Accessed: 2021-08-25.

[Oh *et al.* 2019] S. Oh, Y. Jung, S. Kim, I. Lee and N. Kang. Deep generative design: Integration of topology optimization and generative models. *Journal of Mechanical Design, Transactions of the ASME*, 141(11), 2019.

[Ononiwu *et al.* 2016] G. Ononiwu, O. Onojo, O. Ozioko and O. Nosiri. Quadcopter Design for Payload Delivery. *Journal of Computer and Communications*, 04(10), 2016.

[Oxman 2017] R. Oxman. Thinking difference: Theories and models of parametric design thinking. *Design Studies*, 52, 2017.

[Pallant *et al.* 2020] J. Pallant, S. Sands and I. Karpen. Product customization: A profile of consumer demand. *Journal of Retailing and Consumer Services*, 54, 2020.

[Pradhan 2019] R. S. Pradhan. INDUSTRIAL DESIGN AND CLOUD-BASED 3D VISUALIZATION TOOL – AUTODESK FUSION 360. *International Journal of Engineering Applied Sciences and Technology*, 04(04), 2019.

[R. Pescaru and Oancea 2017] G. R. Pescaru, P.Kyratsis and Oancea. Software tool used for automated design of customizable product. In *MATEC Web of Conferences*, Vol. 137, 2017.

[Ramos and Melgosa 2020] B. Ramos and C. Melgosa. Cad learning in mechanical engineering at universities. *Computer-Aided Design and Applications*, 18(1), 2020.

[Rayna and Striukova 2016] T. Rayna and L. Striukova. From rapid prototyping to home fabrication: How 3D printing is changing business model innovation. *Technological Forecasting and Social Change*, 102, 2016.

[Remache-Vinueza *et al.* 2021] B. Remache-Vinueza, K. Dávila-Cárdenas and M. Zapata. CAD/CAE Tools and Additive Manufacturing to Reduce the Impacts of Critical Equipment Shutdown on Production Planning. 2021.

[Rible *et al.* 2020] G. P. Rible, N. A. Arriola and M. Ramos. Modeling and implementation of quadcopter autonomous flight based on alternative methods to determine propeller parameters. *Advances in Science, Technology and Engineering Systems*, 5(5), 2020.

[Salem *et al.* 2020] H. Salem, H. Abouchadi and K. El Bikri. Design for additive manufacturing. *Journal of Theoretical and Applied Information Technology*, 10(19), 2020.

[Srinivasan *et al.* 2018] R. Srinivasan, V. Giannikas, D. McFarlane and A. Thorne. Customising with 3D printing: The role of intelligent control. *Computers in Industry*, 103, 2018.

[Stanciu *et al.* 2019] S. G. Stanciu, G. Sergiu, R. E. Petrusel and B. C. Pîrvu. Development Overview of a Smart Customizable Product. *ACTA Universitatis Cibiniensis*, 70(1), 2019.

[Steuben *et al.* 2016] J.C. Steuben, A. Iliopoulos and J. G. Michopoulos. Implicit slicing for functionally tailored additive manufacturing. *CAD Computer Aided Design*, 77, 2016.

[Suhada *et al.* 2018] R. T. Suhada, S. Ariyanti, A. V. Fajar and A. Komalasari. TRAINING AUTODESK FUSION 360 FOR TEENAGE OF SENIOR HIGH SCHOOL GRADUATES IN IMPROVING ABILITY IN DISASTERS. *ICCD*, 1(1), 2018.

[Symeonidoua 2019] I. Symeonidoua. Epidermis: Algorithmic Design Based on Biomimetic Morphology. *Nexus Network Journal*, 21(1), 2019.

[Thurner *et al.* 2018] L. Thurner, A. Scheidler, F. Schafer, J. H. Menke, J. Dollichon, F. Meier, S. Meinecke and M. Braun. Pandapower - An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems. *IEEE Transactions on Power Systems*, 33(6), 2018.

[University 2021] Autodesk University. Fusion 360 Introduction to Generative Design, 2021. Accessed: 2021-08-21.

[Vaneker *et al.* 2020] T. Vaneker, A. Bernard, G. Moroni, I. Gibson and Y. Zhang. Design for additive manufacturing: Framework and methodology. *CIRP Annals*, 69(2), 2020.

[Vepa and Sagar 2019] K. S. Vepa and N. V. S. S. Sagar. Design, optimization and manufacturing of quad copter frame using FDM. *International Journal of Engineering and Advanced Technology*, 8(5), 2019.

[Watson and Taminger 2018] J. K. Watson and K. M. B. Taminger. A decision-support model for selecting additive manufacturing versus subtractive manufacturing based on energy consumption. *Journal of Cleaner Production*, 176, 2018.

[Wei *et al.* 2015] P. Wei, Z. Yang and Q. Wang. The Design of Quadcopter Frame Based On Finite Element Analysis. In *Proceedings of the 3rd International Conference on Mechatronics, Robotics and Automation*, Vol. 15, 2015.

[Wickramasinghe *et al.* 2020] S. Wickramasinghe, T. Do and P. Tran. FDM-Based 3D printing of polymer and associated composite: A review on mechanical properties, defects and treatments, 2020.

[Ye *et al.* 2008] X. Ye, H. Liu, L. Chen, Z. Chen, X. Pan and S. Zhang. Reverse innovative design - an integrated product design methodology. *CAD Computer Aided Design*, 40(7), 2008.

[Zafar and Zhao 2020] M. Q. Zafar and H. Zhao. 4D Printing: Future Insight in Additive Manufacturing, 2020.

[Zardo *et al.* 2019] P. Zardo, L. A. Ribeiro and A. Q. Mussi. APLICAÇÕES DE BIM E DESIGN PARAMÉTRICO PARA EFICIÊNCIA ENERGÉTICA DAS EDIFICAÇÕES: UMA ANÁLISE DE APLICAÇÕES PRÁTICAS. *Arquitetura Revista*, 15(2), 2019.

[Zboinska 2015] M. A. Zboinska. Hybrid CAD/E platform supporting exploratory architectural design. *CAD Computer Aided Design*, 59, 2015.

[ZHU *et al.* 2021] J. ZHU, H. ZHOU, C. WANG, L. ZHOU, S. YUAN and W. ZHANG. A review of topology optimization for additive manufacturing: Status and challenges, 2021.

[Zwier and Wits 2016] M. P. Zwier and W. W. Wits. Design for Additive Manufacturing: Automated Build Orientation Selection and Optimization. In *Procedia CIRP*, Vol. 55, 2016.

Appendix A

Code Examples

In order not to saturate the main body of this document with code, all examples used for the case study are presented here. This appendix describes how to replicate the script to achieve the same result as the one obtained in the case study.

A.1 Building the Interface

One of the main features of this case study was the interface built to interact with the model, which was done using event handlers. The command created event handler is is triggered by running the script, and is used to create the interface itself and define its default values, depicted in Listing A.1. Although the one created is simple, there is a large range of options for taking inputs or showing information. This section of code is extensive as it requires for all of the chosen parameters to be identified and named.

```

import adsk.core, adsk.fusion, traceback
import math

default_arm_l = 20
default_arm_n = 6
default_angleValue = 30 * (math.pi / 180)
default_mot_sup_off = 0
default_mot_sup_d = 2
default_mot_sup_t = 0.5
default_frame_sup_l = 7
default_frame_sup_t = 1
default_frame_sup_r = 2
default_frame_t = 0.5
default_frame_sup_d = 2
default_prop_d = 10.16
default_motor_d = 1.5
default_motor_h = 2
default_batt_l = 6
default_batt_w = 3
default_batt_t = 1.5
default_fcu_l = 2
default_fcu_w = 2
default_fcu_h = 1.5
default_fcu_dist = 2
default_fcu_mh_l = 2
```

```
default_fcu_mh_w = 2
default_fcu_mh_r = 0.2
default_motor_mh_n = 4
default_motor_mh_bd = 1.4
default_motor_mh_d = 0.3
default_frame_mh_n = 3
default_frame_mh_bd = 1.2
default_frame_mh_d = 0.3
default_rear_bat_d = 1
default_rear_l = 2
default_rear_w = 2
default_rear_h = 2

# global set of event handlers to keep them referenced for the duration
    of the command
handlers = []
app = adsk.core.Application.get()
if app:
    ui = app.userInterface

newComp = None

class BoltCommandCreatedHandler(adsk.core.CommandCreatedEventHandler):
    def __init__(self):
        super().__init__()
    def notify(self, args):
        try:
            cmd = args.command
            cmd.isRepeatable = False
            onExecute = BoltCommandExecuteHandler()
            cmd.execute.add(onExecute)
            onExecutePreview = BoltCommandExecuteHandler()
            cmd.executePreview.add(onExecutePreview)
            onDestroy = BoltCommandDestroyHandler()
            cmd.destroy.add(onDestroy)
            # keep the handler referenced beyond this function
            handlers.append(onExecute)
            handlers.append(onExecutePreview)
            handlers.append(onDestroy)

            #define the inputs
            inputs = cmd.commandInputs

            # Creating all necessary tabs.
            # These refer each to its own component, to avoid just making
     a giant list
            tabCmdInput1 = inputs.addTabCommandInput('tab_1', 'Overall')
            tabCmdInput2 = inputs.addTabCommandInput('tab_2', 'Motors and
     Propellers')
            tabCmdInput3 = inputs.addTabCommandInput('tab_3', 'Battery')
            tabCmdInput4 = inputs.addTabCommandInput('tab_4', 'FCU')
            tabCmdInput5 = inputs.addTabCommandInput('tab_5', 'Rear
    Components')
            tabCmdInput6 = inputs.addTabCommandInput('tab_6', 'Mounting
    Holes')


            # Get child inputs
            tab1ChildInputs = tabCmdInput1.children
            tab2ChildInputs = tabCmdInput2.children
            tab3ChildInputs = tabCmdInput3.children
            tab4ChildInputs = tabCmdInput4.children
            tab5ChildInputs = tabCmdInput5.children
            tab6ChildInputs = tabCmdInput6.children

            #############################################
                        # Default Inputs
            #############################################

            init_arm_angle = adsk.core.ValueInput.createByReal(
    default_angleValue)
            init_arm_lenght = adsk.core.ValueInput.createByReal(
    default_arm_l)
            init_arm_n = adsk.core.ValueInput.createByReal(default_arm_n)
            init_mot_sup_off = adsk.core.ValueInput.createByReal(
    default_mot_sup_off)
            init_mot_sup_d = adsk.core.ValueInput.createByReal(
    default_mot_sup_d)
            init_mot_sup_t = adsk.core.ValueInput.createByReal(
    default_mot_sup_t)
            init_frame_sup_l = adsk.core.ValueInput.createByReal(
    default_frame_sup_l)
            init_frame_sup_t = adsk.core.ValueInput.createByReal(
    default_frame_sup_t)
            init_frame_sup_r = adsk.core.ValueInput.createByReal(
    default_frame_sup_r)
            init_frame_t = adsk.core.ValueInput.createByReal(
    default_frame_t)
            init_prop_d = adsk.core.ValueInput.createByReal(
    default_prop_d)
            init_motor_d = adsk.core.ValueInput.createByReal(
    default_motor_d)
            init_motor_h = adsk.core.ValueInput.createByReal(
    default_motor_h)
            init_batt_l = adsk.core.ValueInput.createByReal(
    default_batt_l)
            init_batt_w = adsk.core.ValueInput.createByReal(
    default_batt_w)
            init_batt_t = adsk.core.ValueInput.createByReal(
    default_batt_t)
            init_fcu_l = adsk.core.ValueInput.createByReal(default_fcu_l)
            init_fcu_w = adsk.core.ValueInput.createByReal(default_fcu_w)
            init_fcu_h = adsk.core.ValueInput.createByReal(default_fcu_h)
            init_fcu_dist = adsk.core.ValueInput.createByReal(
    default_fcu_dist)
            init_motor_mh_n = adsk.core.ValueInput.createByReal(
    default_motor_mh_n)
            init_motor_mh_bd = adsk.core.ValueInput.createByReal(
    default_motor_mh_bd)
            init_motor_mh_d = adsk.core.ValueInput.createByReal(
    default_motor_mh_d)
            init_frame_mh_n = adsk.core.ValueInput.createByReal(
    default_frame_mh_n)
            init_frame_mh_bd = adsk.core.ValueInput.createByReal(
    default_frame_mh_bd)
```

```
        init_frame_mh_d = adsk.core.ValueInput.createByReal(
default_frame_mh_d)
        init_default_fcu_mh_w = adsk.core.ValueInput.createByReal(
default_fcu_mh_w)
        init_default_fcu_mh_l = adsk.core.ValueInput.createByReal(
default_fcu_mh_l)
        init_default_fcu_mh_r = adsk.core.ValueInput.createByReal(
default_fcu_mh_r)
        init_rear_bat_d = adsk.core.ValueInput.createByReal(
default_rear_bat_d)
        init_rear_l = adsk.core.ValueInput.createByReal(
default_rear_l)

        #############################################
                    # Tab 1 Inputs - Overall
        #############################################

        # Create a read only textbox input.
        tab1ChildInputs.addTextBoxCommandInput('readonly_textBox_1',
'Description', 'In this tab you can set the main dimensions and
features of the drone. Setting the arm angle can be done using the
mouse.', 2, True)

        # Create integer slider input with two sliders and a value
list
        # Defines the number of motors
        tab1ChildInputs.addValueInput('arm_n', 'Number of Motors', ''
, init_arm_n)


        # Create angle value input
        # Defines the angle of the drone arm to its frame support
point
        tab1ChildInputs.addAngleValueCommandInput('angleValue', 'Arm
Angle', init_arm_angle)

        # Define Arm Lenght.
        tab1ChildInputs.addValueInput('arm_l', 'Arm Lenght', 'mm',
init_arm_lenght)

        # Define Motor support thickness.
        #tab1ChildInputs.addValueInput('mot_sup_thic', 'Motor Support
 Thickness', 'mm', adsk.core.ValueInput.createByReal(0.0))
        tab1ChildInputs.addValueInput('mot_sup_t', 'Motor Support
Thickness', 'mm', init_mot_sup_t)

        # Define Motor support height offset.
        tab1ChildInputs.addValueInput('mot_sup_off', 'Motor Support
Offset', 'mm', init_mot_sup_off)

        # Define Motor support Diamenter.
        tab1ChildInputs.addValueInput('mot_sup_d', 'Motor Support
Diamenter', 'mm', init_mot_sup_d)

        # Define Frame Thickness.
        tab1ChildInputs.addValueInput('frame_t', 'Frame Thickness', '
mm', init_frame_t)

```

```

        #############################################
                # Tab 2 Inputs - Motors and Props
        #############################################

        # Create a read only textbox input.
        tab2ChildInputs.addTextBoxCommandInput('readonly_textBox_2', 'Description', 'Here you can set the values relating to motor and propeller dimensions.', 2, True)

        # Define Propeller Diameter.
        tab2ChildInputs.addValueInput('prop_d', 'Propeller Diameter', 'mm', init_prop_d)

        # Define Motor Diameter.
        tab2ChildInputs.addValueInput('motor_d', 'Motor Diameter', 'mm', init_motor_d)

        # Define Motor Height.
        tab2ChildInputs.addValueInput('motor_h', 'Motor Height', 'mm', init_motor_h)


        #############################################
                # Tab 3 Inputs - Battery
        #############################################

        # Create a read only textbox input.
        tab3ChildInputs.addTextBoxCommandInput('readonly_textBox_3', 'Description', 'Here you can set the values relating to battery dimensions.', 2, True)

        # Define Battery Lenght.
        tab3ChildInputs.addValueInput('batt_l', 'Battery Lenght', 'mm', init_batt_l)

        # Define Battery Width.
        tab3ChildInputs.addValueInput('batt_w', 'Battery Width', 'mm', init_batt_w)

        # Define Battery Thickness.
        tab3ChildInputs.addValueInput('batt_t', 'Battery Thickness', 'mm', init_batt_t)


        #############################################
                # Tab 4 Inputs - FCU
        #############################################

        # Create a read only textbox input.
        tab4ChildInputs.addTextBoxCommandInput('readonly_textBox_4', 'Description', 'Here you can set the values relating to FCU dimensions.', 2, True)

        # Define FCU Lenght.
        tab4ChildInputs.addValueInput('fcu_l', 'FCU Lenght', 'mm', init_fcu_l)

        # Define FCU Width.
        tab4ChildInputs.addValueInput('fcu_w', 'FCU Width', 'mm', init_fcu_w)

        # Define FCU Height.
        tab4ChildInputs.addValueInput('fcu_h', 'FCU Height', 'mm', init_fcu_h)

        # Define FCU Distance to Battery.
        tab4ChildInputs.addValueInput('fcu_dist', 'FCU Distance to Battery', 'mm', init_fcu_dist)

        ############################################
                # Tab 5 Inputs - Rear Components
        ############################################

        # Create a read only textbox input.
        tab5ChildInputs.addTextBoxCommandInput('readonly_textBox_5', 'Description', 'Here you can set the values relating to the Rear components.', 2, True)

        # Define Distance to battery.
        tab5ChildInputs.addValueInput('rear_bat_d', 'Distance to battery', 'mm', init_rear_bat_d)

        # Define Rear component lenght.
        tab5ChildInputs.addValueInput('rear_l', 'Rear component lenght', 'mm', init_rear_l)

        # Define Rear component lenght.
        tab5ChildInputs.addValueInput('rear_w', 'Rear component width', 'mm', init_rear_l)

        # Define Rear component lenght.
        tab5ChildInputs.addValueInput('rear_h', 'Rear component height', 'mm', init_rear_l)


        ############################################
                # Tab 6 Inputs - Mounting Holes
        ############################################

        # Create a read only textbox input.
        tab6ChildInputs.addTextBoxCommandInput('readonly_textBox_6', 'Description', 'Here you can set the values relating to the mounting hole position, available for the frame, motors and FCU.', 2, True)

        # Define Motor Mounting Hole Number.
        tab6ChildInputs.addValueInput('motor_mh_n', 'Number of Motor Mounting Holes', '', init_motor_mh_n)

        # Define Motor Mounting Hole Bounding Circle Diameter.
        tab6ChildInputs.addValueInput('motor_mh_bd', 'Motor Mounting Hole Bounding Circle Diameter', 'mm', init_motor_mh_bd)

        # Define Motor Mounting Hole Diameter.
        tab6ChildInputs.addValueInput('motor_mh_d', 'Motor Mounting Hole Diameter', 'mm', init_motor_mh_d)
```

```

            # Define Motor Mounting Hole Diameter.
            tab6ChildInputs.addValueInput('frame_mh_d', 'Frame Mounting 
    Hole Diameter', 'mm', init_frame_mh_d)

            # Define FCU mounting hole size
            tab6ChildInputs.addValueInput('fcu_mh_r', 'FCU Mounting Hole 
    Radious', 'mm', init_default_fcu_mh_r)

            # Define FCU mounting x
            tab6ChildInputs.addValueInput('fcu_mh_w', 'FCU Mounting Hole 
    Ox position', 'mm', init_default_fcu_mh_w)

            # Define FCU mounting y
            tab6ChildInputs.addValueInput('fcu_mh_l', 'FCU Mounting Hole 
    Oz position', 'mm', init_default_fcu_mh_l)

        except:
            if ui:
                ui.messageBox('Failed:\n{}'.format(traceback.format_exc()
    ))
```

Listing A.1: Creating the interface using event handlers.

With the interface built, it it now necessary to have the model update to each form input, which is done through a command execute handler, as depicted in Listing A.2. Every time the user inputs a new value, the input is identified and passed to the Bolt() function, where shape creation is done. Additionally, a destroy command handler is also defined to close the interface when the program is terminated.

```

class BoltCommandExecuteHandler(adsk.core.CommandEventHandler):
    def __init__(self):
        super().__init__()
    def notify(self, args):
        try:
            unitsMgr = app.activeProduct.unitsManager
            command = args.firingEvent.sender
            inputs = command.commandInputs

            bolt = Bolt()
            for input in inputs:
                if input.id == 'arm_l':
                    bolt.arm_l = unitsMgr.evaluateExpression(input.
    expression, "mm")
                elif input.id == 'angleValue':
                    bolt.angleValue = unitsMgr.evaluateExpression(input.
    expression, "deg")
                elif input.id == 'arm_n':
                    bolt.arm_n = int(inputs.itemById('arm_n').value)
                    #bolt.angleValue = unitsMgr.evaluateExpression(input.
    expression, "")
                elif input.id == 'mot_sup_thic':
                    bolt.mot_sup_thic = unitsMgr.evaluateExpression(input
    .expression, "mm")
                elif input.id == 'mot_sup_off':
                    bolt.mot_sup_off = unitsMgr.evaluateExpression(input.
    expression, "mm")
```

```
                elif input.id == 'mot_sup_d':
                    bolt.mot_sup_d = unitsMgr.evaluateExpression(input.
    expression, "mm")
                elif input.id == 'mot_sup_t':
                    bolt.mot_sup_t = unitsMgr.evaluateExpression(input.
    expression, "mm")
                elif input.id == 'frame_sup_l':
                    bolt.frame_sup_l = unitsMgr.evaluateExpression(input.
    expression, "mm")

                %(...)%

            bolt.buildBolt();
            args.isValidResult = True

        except:
            if ui:
                ui.messageBox('Failed:\n{}'.format(traceback.format_exc()
    ))

class BoltCommandDestroyHandler(adsk.core.CommandEventHandler):
    def __init__(self):
        super().__init__()
    def notify(self, args):
        try:
            # when the command is done, terminate the script
            # this will release all globals which will remove all event
    handlers
            adsk.terminate()
        except:
            if ui:
                ui.messageBox('Failed:\n{}'.format(traceback.format_exc()
    ))
```

Listing A.2: Handler responsible for receiving inputs and updating the code real time, as well as its destroy handler responsible for closing down on program termination.

A.2 Building the Model

With the interface completed, and with all parameters collected, all that is left is the Bolt() function, in which the actual shape creation takes place. To simplify the process of building each shape, three functions were created: two to perform extrude, cut, join and create new bodies, one for center-point rectangles and another for center-point circles, and a function to create new components automatically. Each function, displayed in Listing A.3, needs the object coordinates and the desired extrude height, along with a `mode` variable which controls the desired operation, either cut, extrude or new body.

```

def createCylinder(Comp, x, y, z, radious, extrude_height, mode):

    # Set mode to 0 to cut instead of extrude
    sketches = Comp.sketches
    sketch = sketches.add(Comp.xZConstructionPlane)
    #sketchlines = sketch.sketchCurves.sketchLines
    circles = sketch.sketchCurves.sketchCircles
```

```python
    extrudes = Comp.features.extrudeFeatures
    body = circles.addByCenterRadius(adsk.core.Point3D.create(x, y, z), 
   radious)
    prof = sketch.profiles.item(0)
    extrude_dist = adsk.core.ValueInput.createByReal(extrude_height)
    if mode == 1:
        extrude_motor = extrudes.addSimple(prof, extrude_dist, adsk.
   fusion.FeatureOperations.NewBodyFeatureOperation)
    else:
        extrude_motor = extrudes.addSimple(prof, extrude_dist, adsk.
   fusion.FeatureOperations.CutFeatureOperation)

    return extrude_motor

# This function creates blockout components such as battery, FCU, antenna
   , receiver
def createBlocks(Comp, center, corner, height, mode):
    sketches = Comp.sketches
    sketch = sketches.add(Comp.xZConstructionPlane)
    sketchLines = sketch.sketchCurves.sketchLines
    extrudes = Comp.features.extrudeFeatures

    block_sketch = sketchLines.addCenterPointRectangle(center, corner)
    prof_block = sketch.profiles.item(0)
    extrude_dist_block = adsk.core.ValueInput.createByReal(height)
    if mode == 1:
        extrude_motor =  extrudes.addSimple(prof_block, 
   extrude_dist_block, adsk.fusion.FeatureOperations.
   NewBodyFeatureOperation)
    elif mode == 0:
        extrude_motor =  extrudes.addSimple(prof_block, 
   extrude_dist_block, adsk.fusion.FeatureOperations.CutFeatureOperation)
    elif mode == 2:
        extrude_block = extrudes.addSimple(prof_block, extrude_dist_block
   , adsk.fusion.FeatureOperations.JoinFeatureOperation)

# This function creates new components
def createNewComponent():
    # Get the active design.
    product = app.activeProduct
    design = adsk.fusion.Design.cast(product)
    rootComp = design.rootComponent
    allOccs = rootComp.occurrences
    newOcc = allOccs.addNewComponent(adsk.core.Matrix3D.create())
    return newOcc.component
```

Listing A.3: Functions created to easily create rectangular hexahedrons and cylinders.

A class is then created to set all variables. Since this is a repetitive process, only part of the code is shown in Listing A.5. Inside the class, the actual function to build the drone is buildDrone(), in which all components are created based on the functions from A.3.

```python

class Bolt:
    def __init__(self):
        # Default values
        self._arm_l = default_arm_l
        self._arm_n = default_arm_n
        self._angleValue = default_angleValue
        self._mot_sup_off = default_mot_sup_off
        self._mot_sup_d = default_mot_sup_d
        self._mot_sup_t = default_mot_sup_t

        %(...)%

    #properties
    @property
    def arm_l(self):
        return self._arm_l
    @arm_l.setter
    def arm_l(self, value):
        self._arm_l = value
    #########
    @property
    def arm_n(self):
        return self._arm_n
    @arm_n.setter
    def arm_n(self, value):
        self._arm_n = value
    #########
    @property
    def angleValue(self):
        return self._angleValue
    @angleValue.setter
    def angleValue(self, value):
        self._angleValue = value
    #########
    @property
    def mot_sup_off(self):
        return self._mot_sup_off
    @mot_sup_off.setter
    def mot_sup_off(self, value):
        self._mot_sup_off = value
    #########

    %(...)%

    def buildDrone(self):

        # Get a reference to an appearance in the library.
        lib = app.materialLibraries.itemByName('Fusion 360 Appearance
    Library')
        libYellow = lib.appearances.itemByName('Plastic - Matte (Yellow)'
    )
        libRed = lib.appearances.itemByName('Plastic - Matte (Red)')
        libGreen = lib.appearances.itemByName('Plastic - Matte (Green)')
        libBlue = lib.appearances.itemByName('Plastic - Matte (Blue)')

        global motorComp
        global propComp
        global mot_sup_Comp
        global arm_sup_Comp
        global frame_sup_Comp
        global batt_Comp
        global fcu_Comp
```

```python
        global rear_Comp
        motorComp = createNewComponent()
        propComp = createNewComponent()
        mot_sup_Comp = createNewComponent()
        arm_sup_Comp = createNewComponent()
        frame_sup_Comp = createNewComponent()
        batt_Comp = createNewComponent()
        fcu_Comp = createNewComponent()
        rear_Comp = createNewComponent()

        for x in range(1, self.arm_n+1):
            if self.arm_n == 4:
                arm_angle = self.angleValue *57
                if x == 1:
                    arm_angle =  self.angleValue * 180/math.pi
                elif x == 2:
                    arm_angle =  180 - self.angleValue * 180/math.pi
                elif x == 3:
                    arm_angle =  180 + self.angleValue * 180/math.pi
                elif x == 4:
                    arm_angle =  - self.angleValue * 180/math.pi
            else:
                arm_angle = x*360/self.arm_n



            arm_width = self.arm_l*math.cos(math.radians(arm_angle))
            arm_height = self.arm_l*math.sin(math.radians(arm_angle))

            frame_sup_width = self.frame_sup_l*math.cos(math.radians(arm_angle))
            frame_sup_height = self.frame_sup_l*math.sin(math.radians(arm_angle))

            # Create top and bot frame
            # Check for max frame size:
            front_l = self.batt_l/2 + self.fcu_l + self.fcu_dist + 1
            rear_l = self.batt_l/2 + self.rear_l + self.rear_bat_d + 1

            if front_l > rear_l:
                frame_corner = front_l
            else:
                frame_corner = rear_l

            top_frame_h = 0.0
            # Check for the tallest component:
            if self.batt_t >= self.fcu_h and self.batt_t >= self.rear_h:
                top_frame_h = self.batt_t + 0.3
            elif self.fcu_h >= self.batt_t and self.fcu_h >= self.rear_h:
                top_frame_h = self.fcu_h + 0.3
            elif self.rear_h >= self.batt_t and self.rear_h >= self.fcu_h:
                top_frame_h = self.rear_h + 0.3

            # Create extensions for the frame
            # Set extension size based on screw hole size
            ext_size = 2*self.frame_mh_d + self.batt_w/2 + 1.2
            ext_l = frame_sup_height + self.frame_mh_n * self.frame_mh_d
            center_point_sup_bot = adsk.core.Point3D.create(0, frame_sup_height, 0)
            corner_point_sup_bot = adsk.core.Point3D.create(ext_size, ext_l , 0)
            createBlocks(frame_sup_Comp, center_point_sup_bot, corner_point_sup_bot, -self.frame_t,1)

            center_point_sup_top = adsk.core.Point3D.create(0, frame_sup_height, top_frame_h)
            corner_point_sup_top = adsk.core.Point3D.create(ext_size, ext_l , top_frame_h)
            createBlocks(frame_sup_Comp, center_point_sup_top, corner_point_sup_top, self.frame_t,1)

            # Create motor component
            createCylinder(motorComp,arm_width,arm_height,self.mot_sup_t + self.mot_sup_off, self.motor_d, self.motor_h, 1)

            # Create propeller component
            createCylinder(propComp,arm_width,arm_height,self.motor_h + self.mot_sup_t + self.mot_sup_off, self.prop_d/2, 1, 1)

            # Create motor support
            createCylinder(mot_sup_Comp,arm_width,arm_height, self.mot_sup_off, self.mot_sup_d, self.mot_sup_t, 1)

            # Coloring components for easier identification
            # this cycle runs through all bodies in a component
            for k in range(0, motorComp.bRepBodies.count):
                motor_body = motorComp.bRepBodies.item(k)
                prop_body = propComp.bRepBodies.item(k)
                motor_sup_body = mot_sup_Comp.bRepBodies.item(k)
                #arm_sup_body = arm_sup_Comp.bRepBodies.item(k)
                frame_sup_body = frame_sup_Comp.bRepBodies.item(k)
                motor_body.appearance = libYellow
                prop_body.appearance = libRed
                motor_sup_body.appearance = libBlue
                # setting colors for frame supports
                frame_sup_body.appearance = libGreen



            # After the motors and frame supports are created, create the mounting holes for the motors
            for h in range(1,int(self.motor_mh_n+1)):
                # Calculating necessary variables
                screw_angle = h*360/self.motor_mh_n# (h-1)*90+arm_angle
                screw_ex_height = self.mot_sup_t + self.mot_sup_off
                screw_z = self.mot_sup_off

                pos_hole_width = self.motor_mh_bd*math.cos(math.radians(screw_angle)) + arm_width
                pos_hole_height = self.motor_mh_bd*math.sin(math.radians(screw_angle)) + arm_height

                createCylinder(mot_sup_Comp, pos_hole_width, pos_hole_height, screw_z, self.motor_mh_d/2, screw_ex_height,0)

```

```python


    # Create frame
    center_point_bot_frame = adsk.core.Point3D.create(0, 0, 0)
    corner_point_bot_frame = adsk.core.Point3D.create(self.batt_w/2 + 1, frame_corner, 0)
    createBlocks(frame_sup_Comp, center_point_bot_frame , corner_point_bot_frame , -self.frame_t,2)


    center_point_top_frame = adsk.core.Point3D.create(0, 0, top_frame_h)
    corner_point_top_frame = adsk.core.Point3D.create(self.batt_w/2 + 1, frame_corner, top_frame_h)
    createBlocks(frame_sup_Comp, center_point_top_frame , corner_point_top_frame , self.frame_t,2)



    # Create the arm supports
    for x in range(1, self.arm_n+1,2):

        if self.arm_n == 4:
            arm_angle = self.angleValue *57
            if x == 1:
                arm_angle =  self.angleValue * 180/math.pi
            elif x == 2:
                arm_angle =  180 - self.angleValue * 180/math.pi
            elif x == 3:
                arm_angle =  180 + self.angleValue * 180/math.pi
            elif x == 4:
                arm_angle =  - self.angleValue * 180/math.pi
        else:
            arm_angle = x*360/self.arm_n

        frame_sup_width = self.frame_sup_l*math.cos(math.radians(arm_angle))
        frame_sup_height = self.frame_sup_l*math.sin(math.radians(arm_angle))
        ext_l = frame_sup_height + self.frame_mh_n * self.frame_mh_d
        center_point_sup_bot = adsk.core.Point3D.create(0, frame_sup_height, 0)
        corner_point_sup_bot = adsk.core.Point3D.create(ext_size, ext_l , 0)

        support_hole_x = ext_size - 1.5*self.frame_mh_d

        createBlocks(mot_sup_Comp, center_point_sup_bot, corner_point_sup_bot,top_frame_h,1)

        createCylinder(mot_sup_Comp, support_hole_x, frame_sup_height-self.frame_mh_d*1.5, -self.frame_t, self.frame_mh_d/2, 30,0)
        createCylinder(mot_sup_Comp, support_hole_x, frame_sup_height+self.frame_mh_d*1.5, -self.frame_t, self.frame_mh_d/2, 30,0)
        createCylinder(mot_sup_Comp, -support_hole_x, frame_sup_height-self.frame_mh_d*1.5, -self.frame_t, self.frame_mh_d/2, 30,0)
        createCylinder(mot_sup_Comp, -support_hole_x,
frame_sup_height+self.frame_mh_d*1.5, -self.frame_t, self.frame_mh_d
/2, 30,0)

    for k in range(0, mot_sup_Comp.bRepBodies.count):
            motor_sup_body = mot_sup_Comp.bRepBodies.item(k)
            motor_sup_body.appearance = libBlue


    # Cut the supports to fit the frame
    createBlocks(frame_sup_Comp, center_point_bot_frame ,
corner_point_bot_frame , top_frame_h, 0)

    # Last step is creating other components such as battery, FCU,
antenna, receiver
    # Create center and corner points for the battery
    center_point_batt = adsk.core.Point3D.create(0, 0, 0)
    corner_point_batt = adsk.core.Point3D.create(self.batt_w/2, self.
batt_l/2,0)

    # Create rear components
    center_point_rear = adsk.core.Point3D.create(0, -(self.batt_l/2 +
 self.rear_l/2 + self.rear_bat_d), 0)
    corner_point_rear = adsk.core.Point3D.create(self.rear_w/2, -(
self.batt_l/2 + self.rear_l + self.rear_bat_d), 0)
    createBlocks(rear_Comp, center_point_rear, corner_point_rear,
self.rear_h,1)

    # Create battery
    createBlocks(batt_Comp, center_point_batt, corner_point_batt,
self.batt_t,1)

    # Create center and corner points for the FCU
    center_point_fcu = adsk.core.Point3D.create(0, self.batt_l/2 +
self.fcu_l/2 + self.fcu_dist, 0)
    corner_point_fcu = adsk.core.Point3D.create(self.fcu_w/2, self.
batt_l/2 + self.fcu_l + self.fcu_dist, 0)

    # Create FCU
    createBlocks(fcu_Comp, center_point_fcu, corner_point_fcu, self.
fcu_h,1)

    # Cutting the holes for FCU mounting
    createCylinder(fcu_Comp,self.fcu_mh_w/2, (self.batt_l/2 + self.
fcu_l/2 + self.fcu_dist) - self.fcu_mh_l, 0, self.fcu_mh_r, -self.
frame_t,0)
    createCylinder(fcu_Comp,self.fcu_mh_w/2, (self.batt_l/2 + self.
fcu_l/2 + self.fcu_dist) + self.fcu_mh_l, 0, self.fcu_mh_r, -self.
frame_t,0)
    createCylinder(fcu_Comp,-self.fcu_mh_w/2, (self.batt_l/2 + self.
fcu_l/2 + self.fcu_dist) - self.fcu_mh_l, 0, self.fcu_mh_r, -self.
frame_t,0)
    createCylinder(fcu_Comp,-self.fcu_mh_w/2, (self.batt_l/2 + self.
fcu_l/2 + self.fcu_dist) + self.fcu_mh_l, 0, self.fcu_mh_r, -self.
frame_t,0)
```

Listing A.4: The beginning of the class function, where each parameter is converted into a property.

The last remaining piece is the run function, which is responsible for calling all other functions at the script start, depicted in Listing

```python

def run(context):
    try:
        product = app.activeProduct
        design = adsk.fusion.Design.cast(product)
        # This changes the model so it's no longer capturing the parametric history.
        # increases speed and declutters history
        design.designType = adsk.fusion.DesignTypes.DirectDesignType
        if not design:
            ui.messageBox('It is not supported in current workspace, please change to MODEL workspace and try again.')
            return
        commandDefinitions = ui.commandDefinitions
        #check the command exists or not
        cmdDef = commandDefinitions.itemById('Drone Builder')
        if not cmdDef:
            cmdDef = commandDefinitions.addButtonDefinition('Drone Builder',
                    'Drone Builder',
                    'Drone Builder.',
                    './resources') # relative resource file path is specified

        onCommandCreated = BoltCommandCreatedHandler()
        cmdDef.commandCreated.add(onCommandCreated)
        # keep the handler referenced beyond this function
        handlers.append(onCommandCreated)
        inputs = adsk.core.NamedValues.create()
        cmdDef.execute(inputs)

        # prevent this module from being terminate when the script returns, because we are waiting for event handlers to fire
        adsk.autoTerminate(False)
    except:
        if ui:
            ui.messageBox('Failed:\n{}'.format(traceback.format_exc()))
```

Listing A.5: The beginning of the class function, where each parameter is converted into a property.