1-1-2005

Preliminary design of unmanned aircraft using genetic algorithms and data mining

Daniel J. Neufeld
Ryerson University

PRELIMINARY DESIGN OF UNMANNED AIRCRAFT USING GENETIC ALGORITHMS AND DATA MINING

by

Daniel J. Neufeld
B.Eng. (Aerospace Engineering)
Ryerson University (Canada), 2005

A thesis
presented to Ryerson University
in partial fulfillment of the
requirement for the degree of
Master of Master of Applied Science
in the Program of
Mechanical Engineering.

Toronto, Ontario, Canada, 2005

Declaration

Abstract

PRELIMINARY DESIGN OF UNMANNED AIRCRAFT USING GENETIC ALGORITHMS AND DATA MINING

Daniel J. Neufeld
Master of Applied Science
Graduate Department of Mechanical Engineering
Ryerson University
2005

Aircraft design is a complex process involving multiple co-dependent design variables and many design decisions. For commercial aircraft design, this difficulty is offset somewhat by the wealth of knowledge available. Observing existing designs has provided useful empirical relationships and insights for the designer to apply, yielding a relatively well defined problem. The wide variety of configuration possibilities, mission profiles, and the relative lack of historical data leave the problem of unmanned aerial vehicle (UAV) design less defined. The purpose of this research was to develop a robust optimization package for UAV design using data mining to aid configuration decisions and to develop empirical relationships applicable to a wide variety of mission profiles. An optimization software package was developed using a Genetic Algorithm (GA) and Data Mining. The algorithm proved successful in carrying out the preliminary design phase of a number of test cases similar to existing UAVs. Designs produced by the algorithm promise improved performance flight performance relative to existing systems, and reduced development time when compared with conventional design methodology. Future work will introduce high fidelity analysis to the framework developed in this research.

Dedication

To Helen Neufeld

Acknowledgements

I would like to thank Jordan Whyte for his advice on programming and putting together a well structured database, Caleb Gerlach for the loan of many of his computer science textbooks, Sue Erickson for her assistance with Latex, and Dr. Joon Chung, for his advice, support, and patience.

Contents

List of Symbols

a	–	Acceleration
AR	–	Aspect Ratio, defined as the ratio of the square of wing span to planform area
BHP	–	Brake Horsepower
C	–	Conventional tail
CFD	–	Computational Fluid Dynamics
C_D	–	Drag coefficient
$C_{D,0}$	–	Zero lift drag coefficient
$C_{L,max}$	–	Maximum lift coefficient prior to stall
$C_{L,design}$	–	Airfoil design lift coefficient
C_L	–	Lift coefficient
$C_{L,0}$	–	Lift coefficient at zero angle of attack
$C_{M,0}$	–	Moment coefficient at zero angle of attack
CR	–	Close Range UAV category
D	–	Drag
e	–	Oswald efficiency factor
η	–	Propeller efficiency factor
GA	–	Genetic Algorithm
Γ	–	Vortex magnitude
H	–	H tail with twin boom arrangement
HALE	–	High Altitude Long Endurance UAV category
JATO	–	Jet Assisted Take off
k	–	Induced drag constant

LADP	–	Low Altitude Deep Penetration UAV category
LALE	–	Low Altitude Long Endurance UAV catgory
L/D_{max}	–	Maximum lift to drag ratio
λ	–	Taper ratio, defined as the wing tip chord length to root chord length
MALE	–	Medium Altitude Long Endurance UAV category
Micro	–	Very small UAV category for aircraft under 5kg
Mini	–	Small UAV category for aircraft under 30kg
MR	–	Medium Range UAV category
MRE	–	Medium Range Endurance UAV category
ρ	–	Atmospheric density
SFC	–	Specific Fuel Consumption
SR	–	Short Range UAV category
S	–	Planform Area, defined as the projection area of the wing
TSFC	–	Thrust Specific Fuel Consumption
θ	–	Non-dimensional co-ordinate along the wing
T	–	Thrust
UAV	–	Unmanned Aerial Vehicle
V	–	V tail
V_{stall}	–	The stalling velocity of an aircraft
V_{max}	–	The maximum velocity of an aircraft
W	–	Weight

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Traditional aircraft design methodology requires the combined effort of expert designers from several disciplines to arrive at a satisfactory design complying to a set of mission requirements [3]. It is often a lengthy and expensive process. Consequently, aircraft designers have relied on past design trends and empirical equations developed over decades of manned aviation to facilitate the design process. The design of Unmanned Aerial Vehicles (UAV) adds additional complexity because UAV design is a recent development with far less historical guidance to draw upon. UAVs are not bound by many of the constraints that apply to manned aircraft. Removing the 'human factor' eliminates many constraints, leaving UAV design a less well defined problem. Empirical sizing and weight equations developed for manned aircraft preliminary design are not necessarily applicable over the broad spectrum of sizes and configuration possibilities for UAVs. Designs range from inexpensive hand launched units to expensive and highly complex aircraft such as the General Atomics Predator shown in figure 1.1.

Most research in aircraft design optimization is focused on a specific subset of design possibilities according to a need at hand. This research was to develop an optimization package that is adaptable, capable of handling designs ranging from small hand launched aircraft to large unmanned combat aircraft by using Genetic Algorithms (GAs) and data mining. GAs operate on Darwin's theory of natural selection to "breed" superior designs from a pool of random designs. They can be defined as a"population based model that uses selection and recombination operators and mutation operators to generate new sample points in a search space[4]." Work on GA techniques began as a means

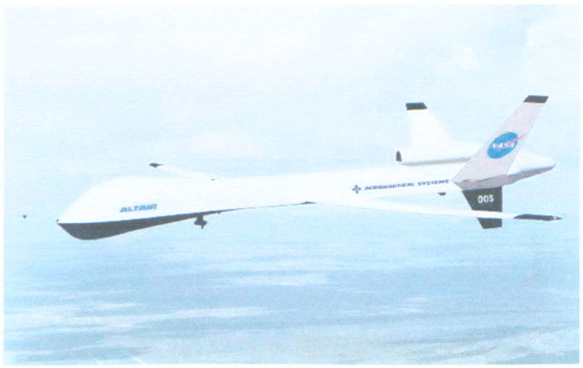


Figure 1.1: General Atomics Predator B [1].

of modeling biological evolutionary systems in the 1960s. This work was not initially intended to be applied to artificial systems, but eventually gave rise to the concept of evolutionary programming, a technique for solving finite state problems (problems having a finite number of possible solutions)[5]. Similar to GAs, evolutionary programming introduced the concept of carrying populations of solution trials and selecting the fittest examples for use in subsequent populations. However, not until Holland's work in 1975 was the concept of crossover breeding of population members introduced[6, 7]. Bethke's work in 1981 firmly established the application of GAs to function optimization[8]. Recent developments in GA research have led to algorithms capable of handling multiple optimization objectives simultaneously. Recent research has further improved GA performance by introducing self adaptation of mutation rates, population size, and crossover schemes. GAs have proved to be efficient optimization schemes for aircraft design optimization [3, 9, 10, 11, 12, 13]. GAs are capable of handling problems with discontinuities, discreet and continuous design variables, and multiple objective functions [12]. Additionally, GAs do not require gradient information to proceed. For these reasons, GAs are becoming increasingly popular as a numerical optimization scheme [12, 13].

Data mining is a term that emerged in the early 1990s. It has its roots in classical statistics but has long been applied to Artificial Intelligence (AI) and heuristics. Advances in computer technology gave rise to the concept of machine learning - an evolution of AI, combining heuristics and statistical analysis. It is used to discover hidden trends in information databases from which decisions can be made [14]. Data mining is most commonly employed in business applications but recently it has proved effective in scientific applications as well [10, 11]. In this research, data mining is applied to address the complexity and diversity of UAV design by providing an artificially intelligent decision model developed by automatic examination of a database of existing UAV designs.

1.2 Research Objectives

Interest in UAVs has grown substantially in recent years due to their improving capabilities and low cost relative to manned aircraft. Much emphasis and effort in current UAV research is focused on avionics, command and control, and airspace integration, leaving airframe design optimization as an afterthought. It is important that advances in these systems are matched with cost effective airframe designs optimized for efficient mission operation. This research describes the development of a computer assisted conceptual design package capable of categorizing existing UAV designs, assisting configuration decisions using database driven methodology, and designing new aircraft optimized for a set of mission goals using data mining and GAs. This research is restricted to the study of fixed wing UAVs.

1.3 Unmanned Aerial Vehicles

To fully appreciate the motivation for UAV research, it is pertinent to consider their historical and present implications. The following is an outline of the history of unmanned aircraft systems, a review of their current applications, and the unique challenges they present to the aircraft designer. The study of UAV design is vital as the "UAV market is expected to grow dramatically by 2020" in both commercial and military roles [15].

When one considers modern UAV systems such as the RQ-1 Predator, it is easy to mistake the invention of unmanned aircraft as a very recent development. Certainly, recent political circumstances have made clear the need for UAV aircraft and the corresponding surge in popular interest is not surprising. However, unmanned aircraft have served in various applications since the First World War; the first being remotely piloted versions of manned aircraft used for target drones [16]. The Second World War gave rise to more fervent attempts to develop unmanned aircraft yielding moderately successful designs such as the German V1: an unpiloted aircraft equipped with a rudimentary mechanical guidance system consisting of a gyroscope and timer and tipped with a small warhead. The United States Air Force experimented with aircraft that were designed to be flown by a pilot into a stable condition near a target. The pilot would bail out of the aircraft and control would be taken over by pilots in nearby aircraft by radio [17]. UAVs played effective roles in the Vietnam War. Then called drones, UAVs were used for photo reconnaissance and to troll for surface to air missile radar frequencies. These missions were usually secretive, and the mainstream awareness and acceptance of UAVs was not achieved until the first Gulf War. UAVs played a critical role in surveillance and were touted in news broadcasts, bringing awareness to the general public [16]. The past decade has seen major growth in UAV research. Modern electronics allow for fully autonomous flight. Modern UAV systems range from small hand launched remote controlled aircraft to large multi million dollar stealth aircraft. Figure 1.2 is a collection of typical UAVs from Second World War to the modern era. With such emphasis and effort being placed on UAV research for both civil and military applications, clearly UAVs have advantages over the manned aircraft they are replacing.

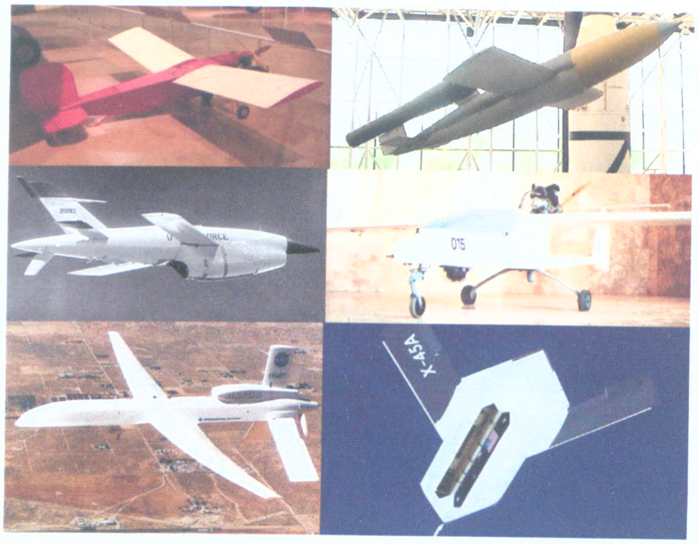


Figure 1.2: UAVs in History: 1930 - Present[2] [1].

The motivation for UAV research stems from the need for vehicles that can fly missions that are too dull, dirty or dangerous for manned aircraft. Jones describes the infamous shoot down of Gary Powers' U2 spy plane in 1960 as the "genesis event for the UAV [16]." The desirability of a pilotless spy plane was made clear by the subsequent political disaster brought about by the downed pilot's confession to Soviet authorities. Aside from the obvious benefit of operating in dangerous circumstances without human risk, UAVs can provide unwavering attention for the duration of flights sometimes in excess of forty hours - well beyond the capability of humans. In such missions, "by removing human factors issues from the aircraft, its performance can be enhanced in many ways[18]." Another, and perhaps even more crucial advantage of UAV systems, is cost. With the Cold War long over, defense budgets have dwindled [16]. For manned aircraft design, much effort is invested in systems designed to protect the pilot. Armour, ejection seats, life support systems, and visual avionics systems add to the cost and complexity of modern combat aircraft. UAVs are not bound by the constraint of protecting a human occupant, and can be designed to be smaller, lighter, and cheaper than their manned counterparts.

Currently, the majority of UAV systems are designed to conduct military surveillance in various scenarios. Some, like the AeroVironment Pointer are small and relatively inexpensive aircraft carried by backpack and hand launched. These are used for short endurance, low altitude missions at close range. Others, like the AAI Shadow are of an intermediate level of complexity and are rail launched in the field by specially equipped vehicles. Capable of endurance on the order of ten hours, these aircraft can be fully autonomous. Highly complex and expensive UAVs such as the RQ-1 Predator must be operated from airfields and are capable of high altitudes and endurance exceeding two days carrying highly sophisticated imaging and avionics equipment. Another common role for UAVs is the aerial target. These aircraft are used to train surface to air missile crews and gunnery by providing a realistic yet somewhat expendable target. Like surveillance UAVs, there is much diversity observed in the ranks of target UAVs - some being hand launched craft for small arms training, other being capable of near supersonic speeds for surface to air missile testing. Applications are not limited, however, to military roles. There are a wide variety of civil roles being considered for UAVs. DeGarmo and Nelson envision scenarios where UAVs are employed for tasks such as border patrol, telecommunications, high altitude imagery, media and traffic reporting, law enforcement, and others areas where having continuous aerial observation is crucial [15]. Additionally, tedious missions currently carried out by manned aircraft, such as coastal surveillance and forest fire spotting, are being considered for UAV application[19].

The design of UAVs is a unique challenge. It is easy to assume that because UAVs lack pilots and passengers, the designer is relieved of constraints such as the airworthiness requirements and safety factors. This is not the case. Any UAV designed to operate in open airspace over inhabited areas must be designed to operate safely. Operating a UAV inside airspace crowded with other aircraft and over inhabited land requires that the UAV must be sufficiently reliable to pose little risk to people and property on the ground and other aircraft in the air. Airspace integration is the focus of significant work by NASA and other government agencies [20]. A second challenge designers face is developing command and control systems. Remotely piloted UAVs require ground control stations from which the pilot can operate the aircraft. Imagery must be streamed from the UAV to the ground station without interruption, otherwise the aircraft would be lost. Autonomous UAVs have complex control systems and intelligent behavior. They must be able to adapt to situations such as inclement weather and possible in-flight collisions with other aircraft. A third challenge, and the one that this research is intended to address is diversity. A commercial aircraft designer has a long history of successful designs to build from. New commercial aircraft are nearly always developments of previous designs. The designer knows that the final outcome of the aircraft will consist of swept wings fixed to a tube like fuselage, large enough to safely accommodate some number of passengers, with turbofan engines fixed either by pylons underneath the wing for larger aircraft, or fixed aft on the fuselage for smaller aircraft. This design has been developed and refined over sixty years and many examples of this philosophy can be observed in the fleets of all major manufacturers including Airbus, Boeing, Bombardier, and Embraer. All these designs are optimized to minimize cost while flying over the commercial routes for which they are intended at high subsonic speeds. UAV design is not nearly as straightforward. Designs ranging from small hand launched vehicles to large high subsonic stealth aircraft are not all optimized to operate at similar speeds, altitudes, or to perform the same task as in commercial aircraft design. A relatively high percentage of UAVs are unconventional designs by manned aircraft standards. Many have twin booms, pusher propellers, or canard wings and 'V' tails. This complexity is in part due to tremendous diversity in mission requirements. Since commercial and civil aircraft are nearly always designed to carry some payload from one point to another as efficiently as possible, it is not surprising that many commercial and civil aircraft, though designed and optimized by different companies, are very similar. UAV designs are optimized with completely different and diverse goals in mind. Some designs minimize size, while some minimize weight. Others maximize endurance at extremely high altitudes, while some maximize endurance at very low altitudes. Many do not take off and land in the conventional manner, but are launched by rail, hand, or rocket. Some are recovered by parachute, while some are expendable and not recovered at all. Figure 1.3 is a collection of typical UAVs showing the diversity in design philosophy.

This complexity has led to a standardized system of UAV classification. Table 1.1 shows the definitions of these classification categories and their abbreviations[20]. Underlying all of these goals is the general objective also found in commercial and civil aircraft design - minimizing cost. This research was undertaken to address the complexity of

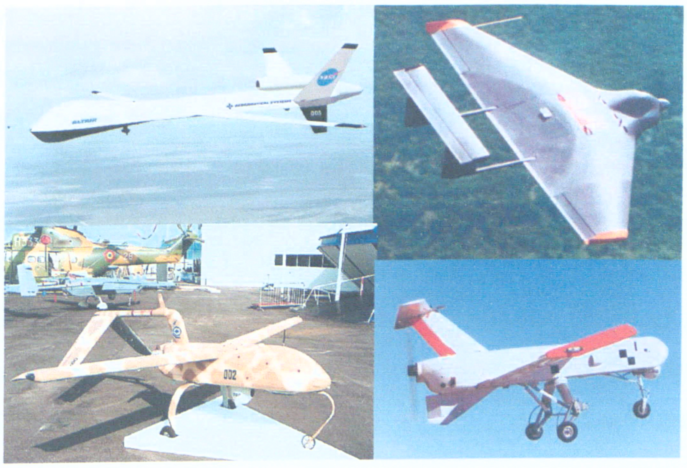


Figure 1.3: Diversity in UAV designs [1].

UAV Category	Acronym	Range (km)	Altitude (m)	Endurance (h)	Takeoff Mass (kg)
Micro	Micro	<10	250	1	<5
Miniature	Mini	<10	150-300	<2	<30
Close Range	CR	10-30	3000	2-4	150
Short Range	SR	30-70	3000	3-6	200
Medium Range	MR	70-200	5000	6-10	1250
Medium Range Endurance	MRE	>500	8000	10-18	1250
Low Altitude Deep Penetration	LADP	>250	50-9000	0.5-1	350
Low Altitude Long Endurance	LALE	>500	3000	>24	<30
Medium Altitude Long Endurance	MALE	>500	14000	24-48	1500
High Altitude Long Endurance	HALE	>2000	20000	24-48	12000

Table 1.1: UAV Classification Table

UAV design. It highlights the development of a conceptual design optimization software package. This package combines database driven artificial intelligence for aiding configuration decisions as well as an airframe optimizer for design refinement developed to handle diverse mission objectives and constraints. While this research does not contain a direct costing model for UAV design, it does address the need for cost reduction by greatly speeding preliminary design of UAV airframes.

Canadian interest in UAVs has grown recently. Canadian geography and topography present challenges that UAVs are particularly suited to meet. Canada is a vast country featuring the longest undefended border in the world, coastlines bordering on three oceans, and vast areas of wilderness with little or no population. Additionally, Canada has a very low population for its size and consequently a small armed force, making border and coastline patrol a very difficult challenge. The cost of adequate manned patrols over so vast an area is daunting. UAVs can address these challenges because they can operate autonomously, with multiple vehicles being under the command of one ground station.

Tedious missions such as forest fire spotting can be undertaken by UAV systems with a greater degree of efficiency than manned flights. Also, patrolling the Northwest Passage has become an important issue since its status as Canadian waters is under dispute. Operating manned aircraft in the hostile wilderness of northern Canada is certainly a daunting prospect, whereas designing a high endurance UAV system capable of operating in the extreme conditions would allow for better coverage from fewer permanent bases. These concepts, however, depend on the establishment of an infrastructure to control and communicate with the aircraft and integrate them safely into the airspace and is the subject of much ongoing research[20]. Canada's role as a peacekeeping nation has recently

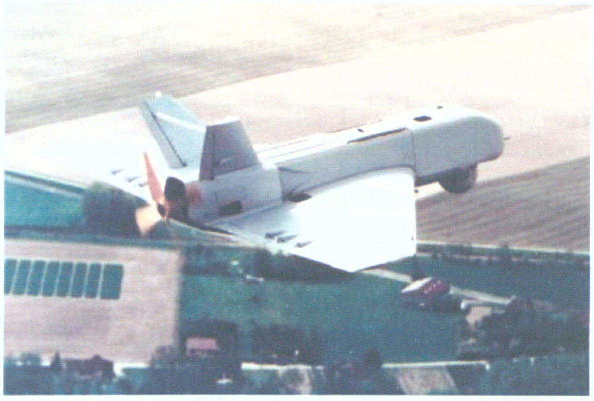


Figure 1.4: SAGEM Sperwer [1].

led to the purchase of battlefield surveillance UAVs for deployment in Afghanistan. The system is a French designed aircraft and ground station called the SAGEM Sperwer. It is classified as a Medium Range UAV and has an endurance capability of 6 hours[21]. and is shown in figure 1.4. There have been several successful Canadian designed UAVs, the most famous being the Bombardier/Canadair CL series UAVs. Although never fully adopted by the Canadian armed forces, the CL series UAVs are enjoying "continued success in use by both France and Germany[21]." The CL-227 UAV is shown in figure 1.5.

1.4 The UAVOpt Package

The software developed in this research is called UAVOpt. It was developed in the Java programming language. It is capable of interacting with decision models produced by a public domain data mining package called Weka, developed by Witten et al.[14]. Addi-

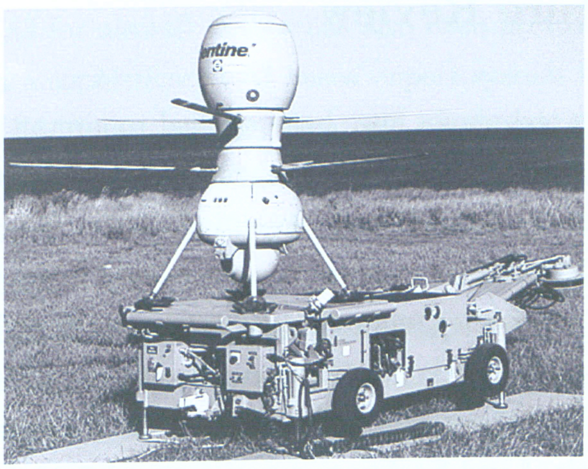


Figure 1.5: CL-227 Sentinal [1].

tionally, UAVOpt interacts with an engine and airfoil database compiled in the Microsoft Access format. It contains performance analysis modules, database query modules, user interface modules, and a genetic algorithm optimizer. The algorithm was designed to handle several different objective functions including weight minimization, range maximization, and endurance maximization. It can provide advise to the user in qualitative design decisions including landing gear type, engine type, engine location, and aircraft configuration layout by examining a UAV specification database. The algorithm can provide the user with conceptual designs complying with a set of user defined constraints, optimized for maximum performance in a user selected objective. Details on the development and operation of UAVOpt is discussed in later chapters.

1.5 Literature Review

Design optimization techniques have been applied to aircraft design for some time with generally positive results. Some research such as the work conducted by Perez et al[3]. has focused more on the feasibility of the optimization scheme itself than the fidelity of the analysis used. Bartholomew defined a set of three fidelity levels for describing fluid and structural analysis as they relate to optimization packages[22]. The first, called Level 1, covers analysis founded upon empirical equations. Level 2 is an intermediate level and can involve basic beam theory for structural analysis and vortex panel methodology for aerodynamic analysis. Level 3 refers to computational fluid dynamics and finite element analysis. Most previous aircraft optimization schemes have relied on level 1 methodology. These low fidelity methods are not, however, without merit. Low fidelity does not necessarily mean poor accuracy. It can be more precisely defined as a lack of assurance that the real world is being faithfully simulated [23]. Markish proposed level 1 techniques for studying the interaction between engineering design and financial impact, and developed useful an optimization tool[24]. Additionally, Raymer's work relies upon "sophisticated implementations of classical methods[13]." Raymer's research compared the effectiveness of several optimization schemes in addition to genetic algorithms. Raymer's optimization algorithm was designed to take an aircraft concept developed by the user and tweak the design to optimize the results. This was done by relying heavily on rigorously developed statistical analysis methods rather than computational methods for aerodynamic and structural design. Zang et al. proposed a novel approach, avoiding the long computational time that CFD analysis entails by using primarily low fidelity methods and turning to CFD for occasional corrections[25]. However, with the increasing availability of powerful computers and with the continued development and refinement of high fidelity methods, as asserted by Giesing, the design optimization industry is rapidly moving toward these methods[23]. In fact, Chung et al. developed an optimization algorithm that relies on high fidelity CFD solution entirely[9]. This research aims to eliminate level 1 techniques wherever possible by relying heavily on database component selection for engine and airfoil data instead of low fidelity empirical engine sizing and airfoil design. Traditional performance equations were discarded in favor of a simulation approach. Level 1 methods were only used for weight analysis, and this should be considered a temporary stand in, as the vision for this project includes incorporation of full structural design.

The application of AI for aircraft design has also been previously explored. In 1991, Nah successfully tested a sophisticated AI based expert system for aiding configuration of commercial aircraft[26]. Later work by Rentema combined AI based reasoning with a solid modeling tool as a computer assisted design package[27]. Subsequently, AI for aircraft design has been largely replaced by design optimization. Expert systems are very complex, built upon hard coded reasoning that must be physically entered by expert aircraft designers, leaving such systems unable to adapt to new trends. While the GA optimization algorithm can replace the expert system when it comes to decisions that have clear numeric advantages, it cannot carry out logical decisions that have little or no influence on directly quantifiable aircraft performance characteristics. Most design optimization research takes a basic design layout and refines and optimizes that layout, previously chosen by an aircraft designer. While design optimization, as in Raymer's research, has already been used to design UAVs, no attempt has yet been made to design a system capable of offering configuration decision assistance that addresses the tremendous diversity in design philosophy observed in UAV designs today. This research attempts to combine the advantages of design optimization with a reasoning system capable of offering "artificially expert" advice on configuration decisions faced by the designer, specifically for UAV design. The pace of UAV growth renders the concept of an expert system useless since such a system would quickly become obsolete. To address this, a new approach was taken. Decision advice is generated by data mining of UAV specification database, generating decision trees to aid configuration layout problems. Data mining, having its roots in a number of fields including statistics, AI, and information theory, "has been applied with significant success in a number of areas[28]." Other research has incorporated data mining in aerospace related problems. Letourneau et al. tested a data mining system for predicting aircraft component failure[10]. Liu employed data mining to aid the development of flight control systems[29]. However, to the author's knowledge, the usage of data mining to aid preliminary design is novel.

Chapter 2

Data Mining

2.1 Overview

Data mining is defined as the extraction of hidden and useful information from databases. It is a broad term that can apply to a wide variety of methodologies, including well known techniques like regression analysis[14, 30]. It usually applies to automated or computer assisted techniques and is often referred to as machine learning and is widely used in AI applications "that address the formulation and modification of theories as a result of observations[31, 32]."

A major goal of this research was to provide a UAV designer with decision advice. Other research involved developing an expert system as a decision model to accomplish this goal[26]. Expert systems are a subset of AI and are developed by consultation of an expert in the field to which the system is to be applied. For the case of aircraft design, an aerospace engineer would try to envision all the possible decisions that must be made during the preliminary design phase and how they are affected by various performance targets and applications the aircraft will be required to fulfill. These design decisions may include the choicc among turboprop, piston, or jet engines. The expert designer knows the advantages and disadvantages of each engine type and where they are best applied and can suggest a set of rules for selecting the engine type. These rules can be encoded as if - then statements such as: if speed $<$ Mach 0.5 then use piston engines, if speed $>$ Mach 0.5 and $<$ Mach 0.8, then use turboprop engines, and finally if speed $>$ Mach 0.8, use jet engines. Despite the fact that this is perhaps an oversimplified example, it becomes clear that the development of expert systems is a painstaking process. The complex interaction between design decisions, their justification, and their influence on other aspects of design must be modeled accurately covering as broad a base as possible to handle diverse mission goals. As a consequence, expert system design is a complex and lengthy process. The end result can justify the effort for applications such as medicine, civil engineering, and fields where rapid change in philosophy is not common.

For the case of UAV design, new designs are constantly emerging and new applications are being regularly envisioned. Any attempt to build an expert system for aiding the development of UAVs would be subject to almost immediate obsolescence and would require constant updating and revision to remain useful. This research proposes data mining of a UAV specification database as an alternative to the expert system for aiding design decisions that are not readily quantifiable. The advantages of this approach are obvious, as an automated data mining algorithm can rapidly construct complex decision trees and new data can easily be added to the specification database. As a consequence, it is easy for the system to remain current and applicable. Such a system would be akin to consulting an expert who can instantly recall any UAV built in exacting detail, and advise on decisions applicable to new designs based on that knowledge. As previously mentioned, data mining is a term having broad applicability. For this research, the type of data mining employed is called the decision tree, and will be introduced later. First, a description of the specification database to which these methods are applied is necessary.

2.2 The Database

The database compiled for this research contains four tables. The first is a UAV specification table containing detailed information on many existing UAV designs. The categories and a sampling of several entries are shown in table 2.1. Most of the categories shown are self explanatory, but clarification of several of the more ambiguous categories is warranted. The full database is available on CD.

Under the Mission heading, what is referred to here is the primary mission goal of the UAV. In this database, attributes under the Mission heading can become either of the following: surveillance, multirole, civil, target, and combat. The surveillance category contains aircraft that are primarily designed to loiter about a target area for a period of time, sending visual information back to the ground station. The multirole category contains UAVs designed to perform adequately for several mission types. Target UAVs are craft designed to serve as targets for surface-to-air missile tests and gunnery or small arms training. Combat UAVs are designed to carry and deploy ordinance to

ID	1	2	3
Name	AAI Shadow 200	AAI Shadow 400	...
Mission	Surveillance	Multirole	...
Classification	MR	MR	...
Launch	Conventional	Conventional	...
Landing Gear	Fixed	Fixed	...
Engine Location	Pusher	Pusher	...
Engine Type	Piston	Piston	...
Engine Number	1	1	...
Tail Type	H	H	...
Empty Weight (kg)	91	Unknown	...
Payload Weight (kg)	27	45	...
Fuel Weight (kg)	29	Unknown	...
Gross Weight (kg)	149	220	...
Wing Area (m^2)	2.14	Unknown	...
Wing Span (m)	3.89	4.3	...
Aspect Ratio	7.07	Unknown	...
Fuselage Length (m)	3.4	Unknown	...
Fuselage Height (m)	0.91	Unknown	...
Maximum Velocity (km/h)	228	222	...
Service Ceiling (m)	4575	4755	...
Range (km)	125	185	...
Endurance (h)	6	8	...

Table 2.1: UAV Database Sample

a target. Values under the Launch heading refer to the takeoff method of the UAV and can take on one of either conventional, rail, catapult, rocket assisted, vertical, or hand. The conventional takeoff category refers to launch by rolling takeoff on a runway. Rail launched UAVs takeoff under their own power from a field deployed rail structure. Catapult launch refers to a rail system with a pneumatic or spring mechanism designed to rapidly accelerate the UAV. Rocket assisted takeoff systems are comprised of a small rocket motor fixed to the UAV to aid the acceleration of the craft on takeoff. Hand launched UAVs are launched by the operator, who must pick up and throw the aircraft. The software developed in this research simulates conventional takeoff only. However, UAVs can be designed with the other takeoff methods by setting a stall speed constraint appropriate for the desired method. For example, a hand launched UAV can be designed by setting reasonable weight and sizing constraints and a stall speed constraint such that an operator can easily impart sufficient velocity to the craft. Similarly, the stall speed constraint can be set to apply to a specific catapult or rocket system the designer has in mind. Vertical takeoff refers to UAVs that can takeoff vertically under their own power. The Classification heading refers to the performance classification group of which the UAV is a member. A chart of this standard classification scheme is given in table

1.1, adapted from UAVs: A Vision of the Future [33]. The Landing Gear category describes the type of landing gear (if any) the UAV has and can take on any one of the following attributes: fixed, retractable, or none. The Engine Location category describes the location and orientation of the UAV's power plant. Attributes can take on one of the following values: tractor (engines having front mounted propellers) , pusher (engines having rear facing propellers), or wing mounted. The Tail Type category refers to the configuration of the horizontal and vertical stabilizers and can take on one of the following values: H (for twin boom arrangements) V, T, conventional, or canard. The remaining database categories are numerical and their meaning is evident with no further explanation.

ID	1	2	3
Name	Rotax 914UL	Honeywell TPE331	...
Type	Piston	Turboprop	...
Length (mm)	581	1168	...
Width (mm)	575	660	...
Height (mm)	409	660	...
Weight (kg)	64	174.6	...
Power (BHP) or Thrust (lb)	115	940	...
SFC (lbm/hr/hp) or TSFC	0.54	0.5	...
Flat Rate Altitude (m)	4877	4400	...

Table 2.2: Engine Database Sample

The second table in the database is an engine specification table. A truncated table is shown in table 2.2. Again, the meaning of most of the categories is readily apparent. The SFC or TSFC category refers to specific fuel consumption or thrust specific fuel consumption. The former is a measure of the ratio of engine fuel consumption per hour per horsepower produced and applies to piston or turboprop engines. The latter is a measure of the ratio of engine fuel consumption per hour per pound of thrust and applies to jet engines.

The third table in the database is a collection of airfoil specification data pre-calculated using computational fluid dynamics. The basic geometric information about each airfoil is stored in the airfoil table shown in figure 2.3. The aerodynamic data is stored in a sub-table for each entry and contains data relevant to a number of different Reynolds number regimes. An example of these tables is shown in table 2.4. The primary table of the airfoil database contains constants that do not change with conditions, such as naming conventions and geometric properties. The sub-table includes aerodynamic properties of the airfoil for a number of different Reynolds numbers.

In the main table, the Maximum Thickness Ratio category refers the the ratio of the airfoil maximum thickness to chord length. The Location of Maximum Thickness is expressed in percentage of chord length. The Maximum Camber category refers to the maximum distance between the chord line (the line from the airfoil leading edge to the trailing edge) and the mean line (a curve exactly bisecting the airfoil thickness from leading to trailing edge). The Reference Perimeter and the Reference Tip Area represent the ratio of the airfoil perimeter to chord and the ratio of tip area to chord.

ID	1	2	3
Name	S1020	N0009	...
Usage	Wing	Empennage	...
Designer	Selig	NACA	...
Max Thickness Ratio	15.08	9	...
Location of Max Thickness	35	29.7	...
Max Camber	5.04	0	...
Location of Max Camber	50.9	0	...
Reference Perimeter	2.0594	2.0239	...
Reference Tip Area	0.0987	0.0618	...

Table 2.3: Airfoil Database Sample

Reynolds Number	100000	150000	...
Cl_{design}	1.4497	1.4643	...
L/D_{max}	52.3357	62.8455	...
$C_{L,0}$	1.151	1.1636	...
$C_{M,0}$	-0.2644	-0.2665	...
Lift Slope	5.502279	6.050557	...
$C_{L,max}$	1.915	2.1645	...

Table 2.4: Airfoil Aerodynamic Properties Sub-Table

The first category of the sub-table is the Reynolds number at which the CFD tests were carried out. When the GA calls for airfoil properties, it will query the database for properties corresponding to the Reynolds number regime in which the aircraft under study is intended to operate. The $C_{L,design}$ category is the lift coefficient at which the airfoil was designed to operate. The L/D_{max} category is the maximum lift to drag ratio the airfoil can achieve. The $C_{L,0}$ category contains the lift coefficient value produced by the airfoil at an angle of attack of zero. The $C_{M,0}$ category is the moment coefficient of the airfoil at zero angle. The Lift Slope is a measure of the airfoil lift coefficient to angle of attack expressed in $1/rad$. The $C_{L,max}$ category is the maximum lift coefficient the airfoil can operate at before stalling. It should be noted that these are all two-dimensional airfoil properties, that is, properties calculated based on the assumption of an infinite wing, considering none of the spillage effects observed on finite wings. Accounting for the effects of spillage on three-dimensional wings is a topic discussed later.

The final database table is an user input database. The table contains a list of input criteria to be filled in by the aircraft designer running the algorithm. Basic performance goals and constraints are entered for analysis by the algorithm and can be selected when the software is executed. These parameters are shown in table 2.5. The meaning of the input criterion are readily evident with no further explanation. One thing should be noted regarding the performance target input criteria. If a particular target for a proposed aircraft is desired, such as maximum velocity, the target value can be entered, and the algorithm will not strive to produce aircraft exceeding that target. However, if it is desired to design the fastest possible aircraft while meeting the other requirements, a very large (and unattainable) number can be entered. The algorithm will function, but will never cease striving to produce a faster aircraft, ensuring maximization of the airspeed constraint. Alternatively, if airspeed is not an important factor for the aircraft's intended mission, the entry can be set to zero, leaving no incentive for the algorithm to produce fast aircraft, allowing it to focus on other parameters. The decision making methodology applied to the UAV database is introduced in the following section.

ID	1	2	...
Name	RQ7-A Test	Gnat Test	...
Target Speed (m/s)	63	72	...
Target Weight (N)	892	2492	...
Maximum Allowable Fuel Weight (N)	280	1893	...
Avionics Weight (N)	69	120	...
Payload Weight (N)	265	628	...
Payload Volume (m^3)	0.1	0.5	...
Target Ceiling (m)	4575	7620	...
Target Takeoff Distance (m)	100	1200	...
Design Load Factor	2.5	2.5	...

Table 2.5: Input Database Sample

2.3 Decision Trees

Decision trees, also known as classification and regression trees (CART) were popularized in Breiman's work in 1983 [34]. This method is capable of searching historical data and constructing decision models with which new data can be classified [35]. For the case of UAV design, these decision trees are generated by examination of a database of UAV design specifications with the purpose of aiding the configuration decisions of new UAV designs. Decision tree algorithms are expressed recursively[14]. The algorithm searches through all possible variables and values by brute force through the entire database looking for the best split - an attribute around which the population can best be divided into two parts, minimizing incorrectly categorized instances such that both halves are as close to homogeneous as possible with respect to the splitting attribute. For the case of engine type selection, as shown in figure 2.1, the first splitting attribute was speed. Aircraft having a maximum speed capability less than 370 km/h are almost always powered by piston engines. Those having maximum speeds over 370 km/h are almost always powered by something else - in this case, either turboprop or jet engines. This process is repeated, acting on each of the two new branches. Clearly, the piston branch is very close to homogeneous, with only 6.7% of the members of this group incorrectly classified, so no further split is undertaken. For the other branch, however, there still remains significant representation from both jet and turboprop engine catagories, requiring another split. Speed again was selected (this is coincidental, often attributes other than the first are chosen) as the optimal splitting attribute - aircraft that have maximum speeds greater than 370 km/h and speeds less than a new split point, 444 km/h, tend to have turboprop powerplants. Those having speeds greater than 444 km/h usually have jet engines. As shown in figure 2.2, the new branches, turboprop and jet, are close to homogeneous. Not surprisingly, the turboprop branch has some representation from both piston engines and jet engines, as the capabilities of turboprop engines place them into a narrow niche between piston and jet engines. Engine type selection happend to be a very simple example. However, decision trees, as observed in others generated for this research, can be much more complex. With the basic approach to decision tree construction introduced, the mathematics behind the algorithm is briefly summarized.

The data mining software used for this research was Weka, a public domain package developed by Witten et al. [14]. The software has several data mining algorithms available, but for this research, study was restricted to the use of the J48 decision tree algorithm, an extended implementation of Quinlan's C4.5 algorithm [36]. The mathematical basis of this algorithm lies with the concept of entropy as defined in equation (2.1) where I_E is a measure of "goodness of split" and $f(i,j)$ is a function of correctly

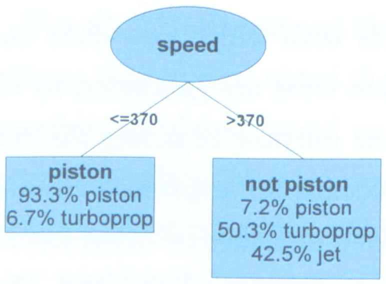


Figure 2.1: First Split

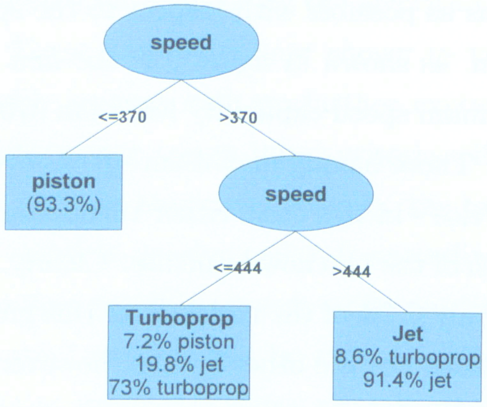


Figure 2.2: Second Split

classified instances, i, and incorrectly classified instances, j.

$$I_E(i) = -\sum_{j=1}^{m} f(i,j) \log f(i,j) \tag{2.1}$$

The J48 algorithm defines the entropy value as a function of data members having a certain attribute and the total number of data entries. This definition is given in equation (2.2) where the value T refers to the total number of data points and T_j refers to data points having the attribute under study.

$$Entropy(T) = -\sum_{j=1}^{m} \frac{|T_j|}{|T|} \log \frac{|T_j|}{|T|} \tag{2.2}$$

It is a measure of uniformity and is used to determine which attribute is the best attribute about which to divide the population. If the user wishes, for example, to know whether to design the aircraft with a V tail or a conventional tail, the first attribute that provides the best split could be related to the UAV category, dividing the population of data into two groups, the first being, for example, MR UAVs and the second, not MR

UAVs. The entropy is determined by adding up the number of aircraft in each group that have V tails, then the number with conventional tails. If the groups are relatively homogeneous, and no other attribute provides a better split between V and conventional tailed aircraft, then the algorithm is free to continue to the next node. Operating recursively, this is how the J48 algorithm constructs a decision tree. There are additional complexities that allow the algorithm to "prune" the decision tree into reasonable size while preserving model accuracy. Additionally, the J48 algorithm constructs many decision trees due to the large number of permutations possible. The tree having the minimum entropy is saved and the rest, discarded.

The advantages of this technique are numerous. The J48 algorithm, like most decision tree algorithms, can "easily handle both numerical and categorical variables[13]." The algorithm selects only the most relevant variables, not requiring anyone to theorize as to which variables are significant or not. They can handle missing data points and outliers - an important feature since complete data is difficult to find. Additionally, they are very easy to interpret[35]. Decision trees were generated using the J48 algorithm to aid a number of design decisions. The implementation and results of this analysis is discussed in the next section.

2.4 Implementation

What would be referred to as unconventional in conventional aircraft design can be observed frequently in UAV aircraft. For example, the twin boom pusher arrangement is somewhat of an oddity in conventional aircraft and represents a tiny fraction of the total aircraft population. V tailed aircraft and canard aircraft are also very uncommon. However, V tailed designs and twin boom designs account for a large proportion of the population in the UAV database compiled for this research. In fact, pusher propeller arrangements are even more frequently observed than tractor arrangements. The designers of these aircraft clearly understood the advantages of these formerly unique arrangements toward the application for which their aircraft were designed. Therefore, using data mining to establish a pattern in these design choices as they relate to the UAV design goals and mission requirements is a useful exercise and is used in this research to aid these configuration decisions. To carry out the analysis, it was presumed that the designer has a set of mission goals and performance targets in mind. These include the following.

- Target airspeed
- Target altitude
- Target range
- Desired launch mode (hand, conventional, rail...)
- Required payload
- Target aircraft weight

The decision trees are formulated based on the known desired attributes and can advise the designer with regards to the following design choices.

- Landing gear type (fixed, retractable, skid, or none)
- Engine type (jet, turboprop, or piston)
- Engine location (pusher, tractor, or wing)
- Empennage type (H, V, T, conventional, or canard)

The results of the decision tree algorithm are shown in figures 2.3, 2.4, 2.5, and 2.6. The logic of these decision trees is readily apparent. Refer to the engine type decision tree in figure 2.5. Clearly, speed is the determining factor when it comes to engine selection. Engines having speeds less than 370 km/h strongly correlate to a piston engine selection. Those having speeds between 444 km/h and 370 km/h correlate to turboprops with a probability of 0.73, due to the overlap of piston and jet capabilities in this region. The aircraft having speeds in excess of 444 km/h strongly correlate to the selection of jet engines. More complex decision trees such as figure 2.6 and figure 2.3 are harder to follow, but the logic that is readily apparent in the simple engine selection example holds for these as well.

In these figures, the numeric percentages shown refer to the percentage of correctly classified instances in that block. The remaining fraction of the members are all incorrectly classified attributes in the block.

The engine location decision tree, shown in figure 2.4, determines the placement of the engine. Tractor refers to a propeller arrangement with a front mounted engine and

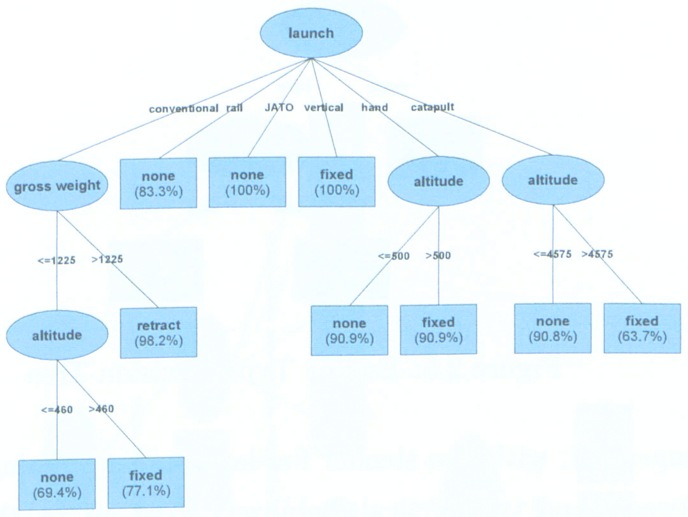


Figure 2.3: Landing Gear Decision Tree

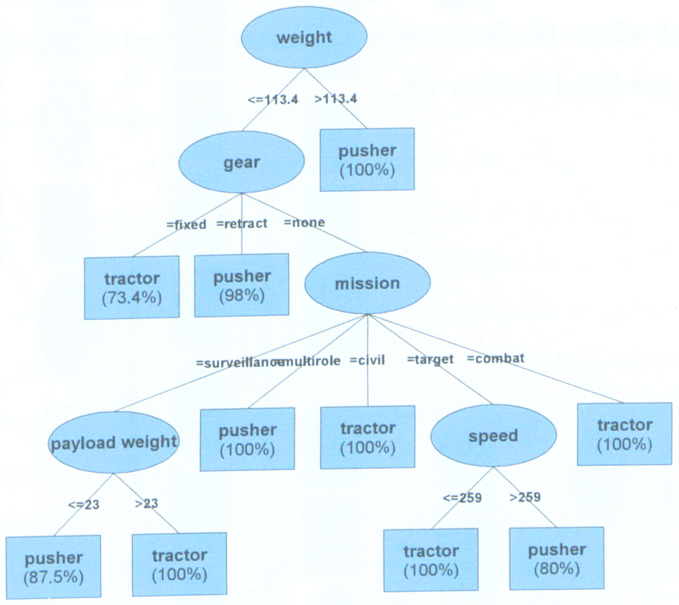


Figure 2.4: Engine Location Decision Tree

propeller. This arrangement is not applicable to jets. Pusher refers to aft mounted engines and propellers. Jet engines can take on this attribute.

The tail type decision tree, shown in figure 2.6, determines the empennage type. In the figure, C refers to conventional tails, where the vertical and horizontal stabilizer are fixed to the fuselage and mounted aft. V refers to the V tail arrangement, where the horizontal and vertical stabilizers are combined into two wing surfaces mounted at high dihedral angles with coupled controls, providing both yaw and pitch control. H refers to a

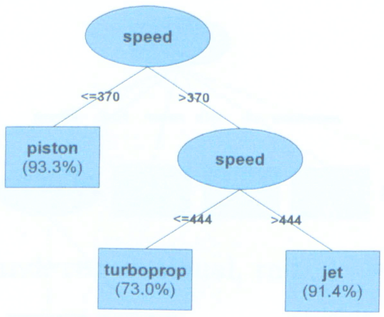


Figure 2.5: Engine Type Decision Tree

twin boom arrangement, with two slender fuselage booms running aft with a horizontal stabilizer in between, and two vertical stabilizers fixed aft on the booms. T refers to an arrangement where the vertical stabilizer is mounted aft on the fuselage and the horizontal stabilizer is fixed to the top of the vertical stabilizer. A canard arrangement refers to a layout where the horizontal stabilizer is fixed to the front of the fuselage and the main wings are fixed further aft.

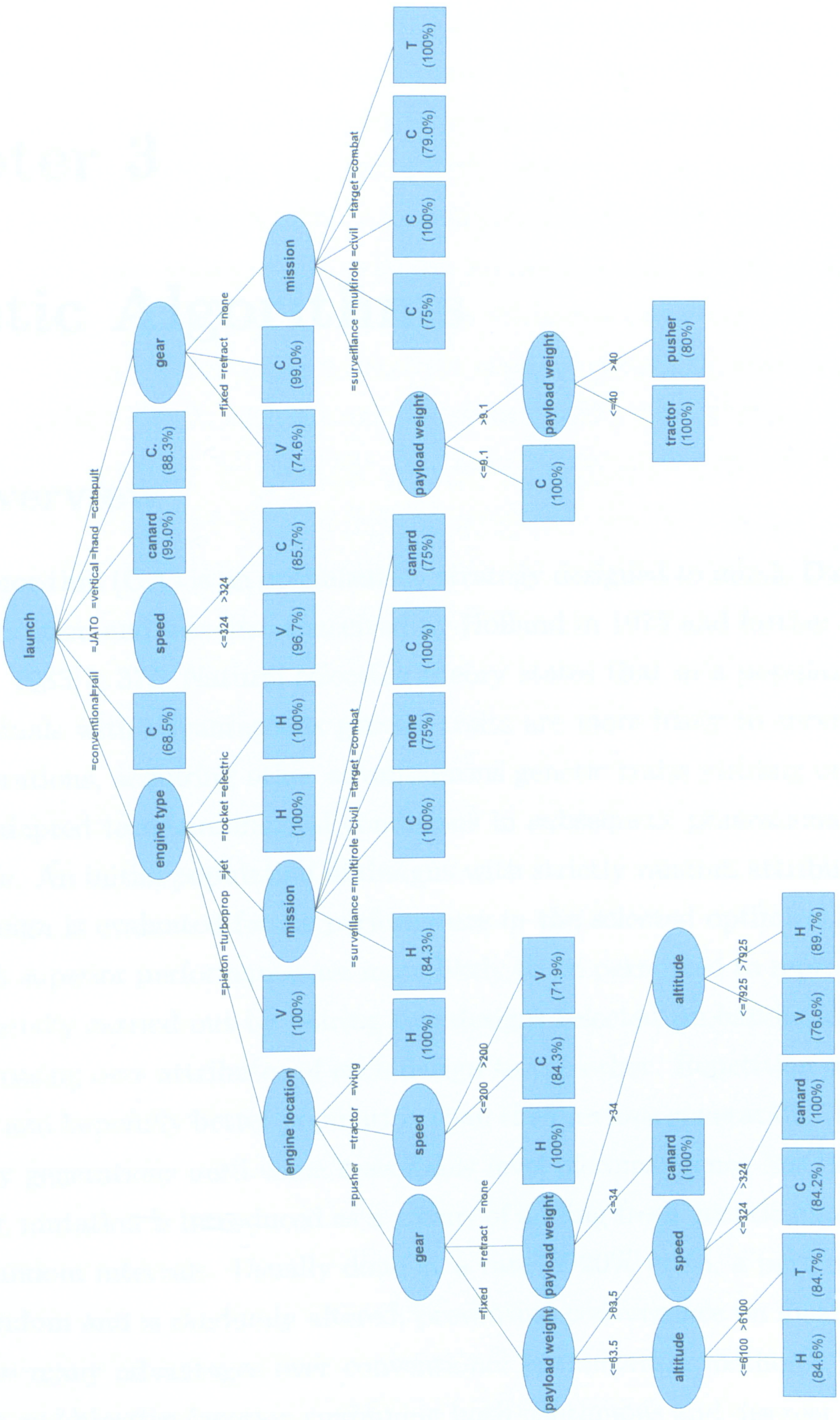


Figure 2.6: Tail Type Decision Tree

Chapter 3

Genetic Algorithms

3.1 Overview

A genetic algorithm (GA) is an optimization strategy designed to mimic Darwin's theory of natural selection and was first conceived by Holland in 1975 and further developed by Goldberg in 1983[7, 37]. Natural selection theory states that in a population of organisms, individuals with advantageous genetic traits are more likely to survive and breed further generations, acquiring other advantageous genetic traits yielding organisms that are better adapted to environmental conditions in subsequent generations. GAs follow this principle. An initial population of designs with strictly random attributes is created and each design is evaluated for its performance in the selected optimization objective. Designs with superior performance are more likely to be permitted to reproduce. Reproduction is usually carried out by pairing two designs selected probabilistically based on rank, and crossing over attributes of each design to the other. Repetition of this process yields a new and hopefully better population than the previous generation. This is carried out for many generations until some reasonable level of convergence has been observed. Additionally, mutation is introduced as a means of adding fresh genetic material into the scheme at random intervals. Usually done at a rate of 20% or so, a particular design is picked at random and is randomly altered, preventing convergence on local optimua[12].

GAs have many advantages over conventional optimization methods. Firstly, GAs can optimize an objective function containing both continuous and discreet variables[12]. Secondly, GAs are less prone to convergence on local optima, a useful trait for optimizing functions that have complex shapes and many constraints [13]. Additionally, GAs can find useful solutions for objective functions containing too many design variables for traditional methods to be practical. Since this research is focused on a database driven approach to design, there are several discreet variables under study such as index references to a parts database. In addition to the discreet variables, there are continuous variables such as wing span under consideration as well. GAs were the clear choice for the optimization scheme since an objective function having both discreet and continuous variables is unsuitable for gradient optimization methods. GAs, however, do present the algorithm designer with significant challenges. Diverse methods for handling constraints, for evaluating fitness and ranking, for selecting and crossing over breeding pairs, for penalizing constraint violation and for carrying out mutation have been studied. This has left GA designers with many options available, making the design of the algorithm itself a study in optimization. This has led to the concept of self-adaptation, where these parameters change when certain trends in the solution emerge. Despite these issues, GAs have been successfully employed for many optimization problems and are known to be robust, reliable, and efficient. Successful application of GAs in aircraft design can be observed in Raymer's work, where GAs are used to optimize a number of different aircraft concepts[13]. Obayashi used a GA optimization algorithm to improve wing planform configurations[12]. Applications of GAs are certainly not limited to aircraft design. Yong proposed a GA system for optimizing cancer treatment[38] et al. and Pires et al. applied a GA to robotic control systems[38, 39]. Clearly, the advantages offered by the GA optimization scheme have led to the successful application of the technique toward diverse problems in many fields.

3.2 GA Implementation

For this research, a GA was developed to optimize UAV designs with the objective of either maximizing flight endurance or minimizing weight. The objective function in any optimization problem is the function that defines a primary variable that the designer wishes to maximize or minimize and is often subjected to a number of constraints. In this case, the objective function is to maximize endurance, or minimize weight subject to performance requirements and is a function of design variables that define the shape and dimensions of the aircraft, the power plant used, and the airfoil selected. In order to reduce computational time, the number of design variables was limited to six key parameters, thought to be the minimum practical number of variables needed to build an airplane as discussed in Raymer's work[13]. The design variables were defined as the following.

- Airfoil
- Engine
- Fuselage Length
- Wing Span
- Aspect Ratio
- Taper Ratio

The objective function formulation is given by equation (3.1) where E is aircraft endurance and W is aircraft weight, both being functions of the chosen design variable values including the airfoil, engine, fuselage length denoted by fl, wing span denoted by b, aspect ratio denoted by AR, and taper ratio denoted by λ respectively.

$$\begin{aligned} max(E) &= f(airfoil, engine, fl, b, AR, \lambda) \\ min(W) &= f(airfoil, engine, fl, b, AR, \lambda) \end{aligned} . \tag{3.1}$$

This objective function is subject to geometric and performance constraints shown in equation (3.2) where $Lmax$, $Bmax$, and $ARmax$ are maximum length, span, and aspect ratio constraints input by the user, respectively. $Vmin$, $TOdist$, $Wmax$, and $Hmin$ are the minimum velocity requirement, the maximum allowable takeoff distance, the maximum allowable weight, and the minimum required altitude achieved by the design. These performance constraints are also defined by the user.

$$\begin{gathered} Length <= Lmax \\ Span <= Bmax \\ AspectRatio <= ARmax \\ MaximumSpeed >= Vmin \\ TakeoffDistance <= TOdist \\ Weight <= Wmax \\ Ceiling >= Hmin \end{gathered} \tag{3.2}$$

Two of these variables are discreet. In this research, rather than using empirical "rubber engine sizing" and primitive airfoil analysis, engines and airfoils are selected from a database of actual specifications. The engine size, weight, and output do not need to be guessed based on empirical methods. Traditional methods are dubious at best for this application since they were developed before the prevalence of UAVs and do not allow for easy distinction between the power plant choices common among UAV designs. The engine database for this research contains specification information for electric, jet, rocket, piston, and rotary engines. The airfoil variable simply is a database index that points to pre-calculated Computational Fluid Dynamics (CFD) solutions for existing airfoil designs. This reduces computation time while not adversely affecting the accuracy of the airfoil analysis by oversimplification. For the logic of the GA procedure to work on something like a database entry, the entries must be sorted based on a parameter, in this case, engine power output, or maximum lift coefficient. Because of this, in the case of the engine database, engines of higher index have clear and logical distinction from those with lower index values - namely, the higher the value, the higher the power output. This is necessary for there to be a logical progression from one generation to the next; otherwise, engine and airfoil selection would be effectively random. These variable are expressed as floating point (double) numbers between 0 and 1, representing the fraction between the first entry index number and the last, rounded to the nearest integer. The remaining four variables are continuous and are also expressed as floating point (double) numbers between 0 and 1, they represent the fraction between defined variable limits that the variable has taken on but are not rounded. For the case of aspect ratio, the limits are defined as between 5 and 20, since aspect ratios lower than 5 cannot be accurately analyzed with the methodology used in this research and wings having aspect ratios greater than 20 become too slender to be feasible. This floating point representation of the six variables forms a design variable vector and is not a typical GA representation. Design variables are usually represented by binary numbers, and the GA crossover function operates at the binary level, swapping digits. Obayashi asserts that the vector approach is a "more natural" representation of real function optimization[12].

With the objective function defined, the next step in the algorithm is to initialize a population from which offspring can be derived. After some experimentation, the algorithm developed in this research seemed to perform best when the population size was fairly large: over 100 and as high as 1000. These populations are simply matrices of random real numbers between 0 and 1, with the number of columns equaling the number of design variables, 6 in this case, and the number of rows equal to the population size. These random numbers are converted into design variable values by finding the distance they represent between the variable limits defined earlier. Once the initial population of design variables is initialized, aircraft are designed around them and tested for fitness, or rather, tested for their relative quality according to their objective function results, either being weight or endurance in this research. This step is called fitness evaluation and is discussed in detail in later chapters. Ultimately, the random aircraft designs are ranked according to their objective function performance and constraint violation. Like many aspects of GA design, the handling of constraint violation is not an exact science. A penalty function was used in this research to ensure constraint satisfaction. The penalty function is a function of the difference between performance of a design and a constraint and is given by equation (3.3) where k is a gain factor set by the user to assign priority to a particular constraint, R is the required value and A is the achieved value.

$$P = e^{(-k(R-A)^2)} \tag{3.3}$$

If, for example, an aircraft was found to have a maximum altitude much lower than the altitude constraint specifies, the penalty function yields the decimal multiplier P by which the fitness value is scaled. A bell shaped penalty function was used in this research because this shape allows for designs that come very close to the constraint values to be not so heavily penalized that they have no likelihood of reproduction. This is important because often an optimum design will lie exactly on one or more constraints. The best performers are assigned an 80% chance of reproduction and the worst designs are assigned a 20% chance, ensuring that no genetic material is lost with absolute certainty. These values were selected after rigorous testing of the GA. Allowing the best designs to have a 100% breeding probability often resulted in premature convergence because the first child generation would often contain too many designs based on the same parents with little diversity. Assigning a 0% breeding probability eliminates the potential for breeding between good and poor designs. This is undesirable since cases may arise where pairing designs with low fitness can breed good designs. The selection process simply scans through the population, selecting individuals in a random, but weighted manner such that their selection probabilities are matched. Individuals can be selected multiple times or not all. Two arrays, each half the size of the population, comprised of selected individuals, are built up in this manner. Members of the two arrays are then crossed over, each producing two offspring to form a new population of equal size to the original. Crossover functions usually occur at the binary level. As shown in figure 3.1, parts of the binary expression defining a design variable value are exchanged with that of another design variable yielding a new value. Crossover functions serve as the model by which genetic material is passed from parents to the child population.

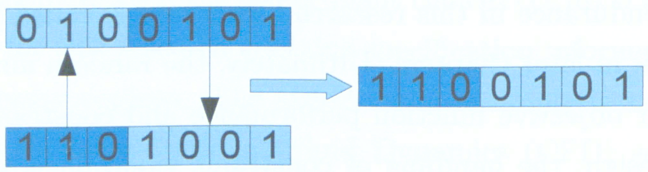


Figure 3.1: Binary Crossover

In this research, a real number crossover was used since the design variables under study include both real number values and integer values. The crossover function was defined in Obayashi and is given in equation 3.4[12].

$$\begin{cases} Child1 = (ran1)Parent1 + (1 - ran1)Parent2 \\ Child2 = (1 - ran1)Parent1 + (ran1)Parent2 \end{cases} \tag{3.4}$$

To prevent premature convergence on local optima in the solution space, a random mutation operator was introduced. This has the effect of introducing new genetic material into the population and prevents stagnation of the design convergence. A mutation rate of 20% was found to be satisfactory by rigorous testing of the algorithm. The mutation operator simply changes one or more design variable values carried by a member of the child population to random numbers. If the algorithm has converged on a local optimum point, the addition of new genetic material provides opportunity for designs to emerge from that location, evolving the population towards regions of higher fitness. The algorithm tracks the fittest member of the population, displaying the corresponding fitness value. New generations are created until there is no significant change in the solution from one generation to the next.

Chapter 4

Algorithm Description

4.1 Overview

The algorithm consists of a user input form followed by a data mining module responsible for making configuration decisions and a genetic algorithm to carry out the design optimization. The data mining algorithm results (in the form of decision trees) are pre-calculated using the Weka data mining software package and saved in the database so no actual data mining computation occurs in the algorithm loop, saving computation time. These decision trees only need to be regenerated when significant changes are made to the aircraft database. A simplified algorithm block diagram is shown in 4.1.

4.2 Fitness Evaluation Module

The fitness evaluation block of the genetic algorithm consists of two libraries of interacting software modules, the role of which is to design new aircraft around each unique set of design variables and calculate their estimated performance. Each library is responsible for handling different aspects of the aircraft design layout and performance analysis. The following section describes the function of each code group and the methodology and assumptions used. The first group is called 'UAVPart' and contains six program blocks representing each significant component a given aircraft possesses. The second is called 'UAVPerformance' and contains performance estimation code.

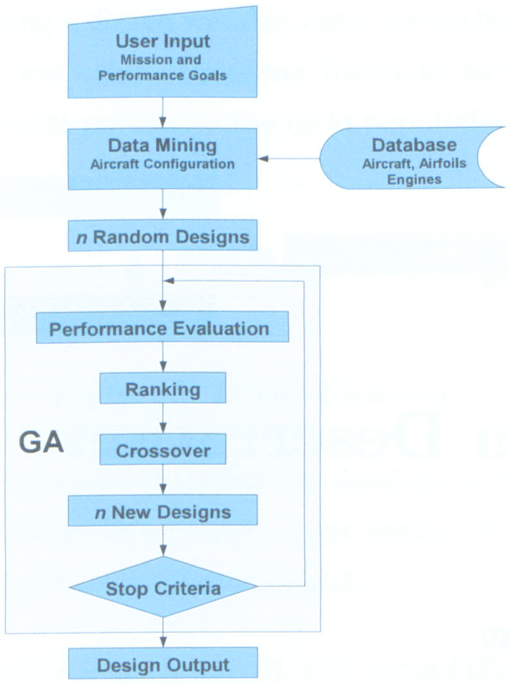


Figure 4.1: Simplified Algorithm Block Diagram

4.2.1 UAVPart Library

The UAVPart library is responsible for designing and assembling the aircraft based on the design variables and decisions undertaken earlier.

- Engine
- Airfoil
- Wing
- Fuselage
- Empennage
- Aircraft

The airfoil analysis block, represented by figure 4.2, handles the two dimensional airfoil analysis. Rather than resorting to low fidelity estimates or computationally expensive airfoil optimization, the airfoil analysis is a database driven system. Many popular airfoil profiles were compiled into a database. Key airfoil properties, shown in figure 4.2, were computed using computational fluid dynamics. These results are pre-calculated and stored in the airfoil database for a number of different Reynolds number values. The airfoil algorithm, given an index indicated by the airfoil design variable and an operating Reynolds number, returns the required two dimensional airfoil properties.

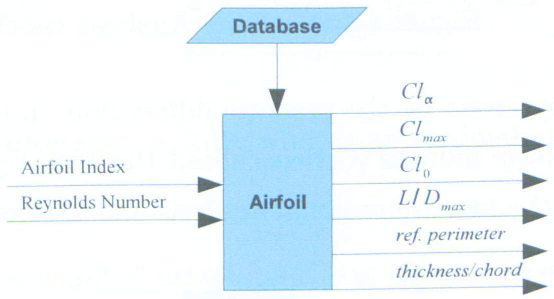


Figure 4.2: Airfoil Analysis Block

Like the airfoil block, the engine block, represented by figure 4.3, submits performance data to the algorithm by querying a database of actual engine specifications and dimensions rather than empirical equations. This database driven approach removes errors of assumption associated with traditional "rubber engine sizing" methodology and easily allows for many different and unusual types of engines, such as rotary and electric, to be considered.

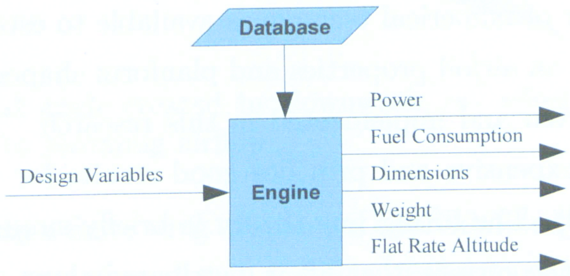


Figure 4.3: Engine Analysis Block

The fuselage design block, represented by figure 4.4 assembles the fuselage geometry such that the required payload volume is accommodated. Additionally, the fuselage is sized to accommodate the engine. The basic fuselage layout is selected by the data mining algorithm, leading to several possible geometries including twin or single boom, and pusher or tractor configurations.

The wing analysis block, shown in figure 4.5 accounts for the aerodynamic effects on a three-dimensional wing. Two dimensional airfoil analysis fails to account for air spillage

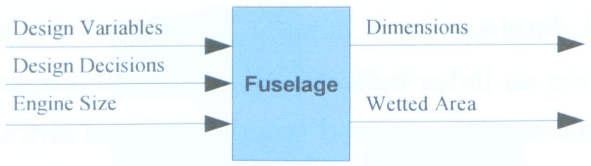


Figure 4.4: Fuselage Analysis Block

over the tip of a wing caused by the pressure differential on the upper and lower surface of the wing. This spillage induces vorticies about the wing tips and leads to drag that is not accounted for in the two-dimensional analysis and is referred to as induced drag.

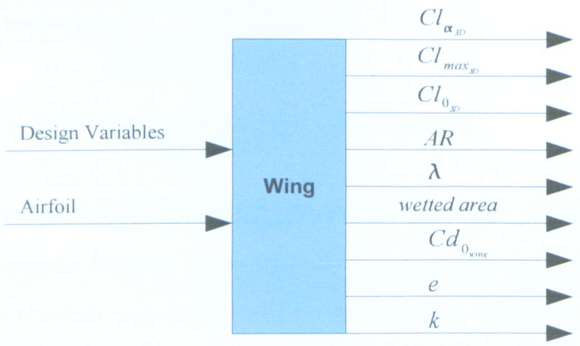


Figure 4.5: Wing Analysis Block

The magnitude of this drag is affected by the planform configuration of the wing. There are a number of numerical techniques available to estimate the magnitude of the induced drag based on airfoil properties and planform shapes. The Prandtl Lifting Line technique was selected and implemented in this research. This technique is relatively computationally inexpensive and provides good results for wings having aspect ratios greater than five[40]. The lifting line theory is briefly summarized below. Lifting line theory begins with the surmise that lift is distributed along a wing such that it is maximum at the middle of the span and zero at the tips as shown in the left diagram in figure 4.6. Prandtl's theory replaces the wing and lift curve with a series of vorticies along a line, increasing then decreasing in magnitude proportional to spanwise location.

The fundamental equation of the Prandtl lifting line theory was developed by assessing the influence of each vortex on any point beyond the lifting line. The reader may refer to Anderson where the full derivation is available. Equation (4.1) is the fundamental equation of the Prandtl lifting line theory. Applied to a finite wing of span length b, it states that "the geometric angle of attack is equal to the sum of the effective angle plus the induced angle of attack - shown below in figure 4.7[40]. Here, Γ is the strength of

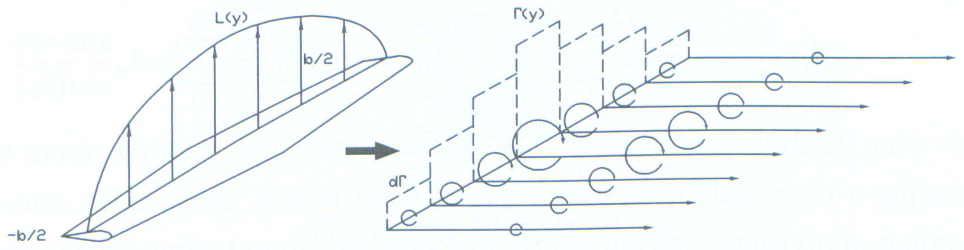


Figure 4.6: Lift distribution over a finite wing is approximated by a line of vorticies of strength Γ

the fluid vortex, α is the angle of attack, and y is the distance along the lifting line to the vortex in question.

$$\alpha(y_0) = \frac{\Gamma(y_0)}{\pi V_\infty c(y_0)} + \alpha_{L=0}(y_0) + \frac{1}{4\pi V_\infty}\int_{-b/2}^{b/2} \frac{(d\Gamma/dy)dy}{y_0 - y} \tag{4.1}$$

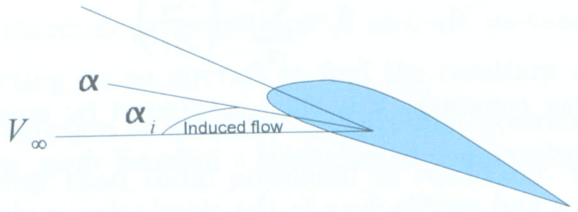


Figure 4.7: The induced angle created by downwash, α_i, effectively reduces the wing angle of attack relative to incoming airflow

Fluid circulation along a finite wing of span b is approximated by a Fourier sine series. Spanwise locations y are transformed to a function of the non dimensional parameter, θ expressed by $y = -\frac{b}{2}\cos\theta$ and the circulation function is given by equation (4.2).

$$\Gamma(\theta) = 2bV_\infty \sum_{1}^{N} A_n \sin(n\theta) \tag{4.2}$$

Combining this result with (4.1) yields equation 4.3. Clearly, with incoming angles of attack known, as well as the two dimensional airfoil properties gathered from the airfoil analysis block, equation (4.3) contains the unknown parameters A_n which can be calculated for each of N divisions along the wing by solving the resulting system of equations.

$$\alpha(\theta_0) = \frac{4b}{C_{L,\alpha}c(\theta_0)}\sum_{1}^{N} A_n \sin(n\theta_0) + \alpha_{L=0}(\theta_0) + \sum_{1}^{N} nA_n \frac{\sin(n\theta)}{\sin(\theta_0)} \tag{4.3}$$

The wing analysis block solves this equation over nine stations along the span of a wing having a previously determined airfoil, span length, taper ratio, and aspect ratio. The number of divisions was limited to nine because, upon testing, increases beyond nine did not significantly improve or change results and greatly increased the computational demands and time required for the algorithm to complete.

Upon calculating the A_n constants, the desired parameters shown in figure 4.5 are computed beginning with the Oswald Efficiency Factor, e given in equation 4.4.

$$e = \frac{1}{(1+\delta)} \tag{4.4}$$

Here, δ is a non-dimensional parameter given by 4.5, expressed in terms of the constants A_n, computed by equation (4.3).

$$\delta = \sum_{2}^{N} n \left(\frac{A_n}{A_1}\right)^2 \tag{4.5}$$

The induced drag constant, k is then computed by equation (4.6). This result is considered in all performance calculations - induced drag, given by (4.7), is computed and added to friction and profile drag in the classic drag polar equation (4.8) where AR is the aspect ratio of the wing and e is the Oswald efficiency factor.

$$k = \frac{1}{\pi ARe} \tag{4.6}$$

$$C_{D,i} = kC_L^2 \tag{4.7}$$

$$C_D = C_{D,0} + kC_L^2 \tag{4.8}$$

Having computed the aerodynamic properties, the geometric properties (including total wetted area, frontal area, and tip area) are calculated.

The aircraft block, shown in figure 4.8 is a construct of the parts previously discussed, containing a wing, engine, fuselage, and an empennage part. The remaining properties needed before carrying out performance estimates is the zero lift drag coefficient, $C_{D,0}$,

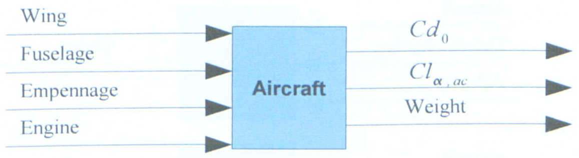


Figure 4.8: Aircraft Block

and an estimate of the aircraft weight. The drag coefficient is estimated using the component drag buildup method outlined by Raymer[13]. The weight is estimated by a series of semi-empirical equations adjusted to fit the database trends found relating weight to aircraft type and geometry.

4.2.2 UAVPerformance Library

The performance library carries out performance estimates of proposed designs. The software developed in this research has modules for estimating takeoff, climb, endurance, and flight envelope performance. These performance estimates are based upon time integration of the two dimensional equations of aircraft motion. This is achieved by computing the forces acting on an aircraft to find the resultant acceleration values and conducting numeric integration to solve for velocity and position values. This method can achieve higher fidelity than those presented in Anderson, where the equations of motion are simplified enough to provide direct solutions[40]. However, this comes at the expense of computation time.

The thrust module, represented by figure 4.9 is responsible for relaying the thrust force value to the various performance modules based on some input conditions. For a piston engine, with thrust being a strong function of shaft power, as well as incoming air velocity, the model is given by equation (4.9), where η_{pr} is a propeller efficiency factor, P is engine power, and V_∞ is the incoming air velocity.

$$T = \frac{\eta_{pr} P}{V_\infty} \tag{4.9}$$

For jet and rocket engines, the thrust is assumed to be constant with incoming air velocity.

The takeoff analysis module, shown in figure 4.10, simulates an aircraft as it rolls from an initial velocity of zero until it has exceeded stall speed by a sufficient margin to leave the ground safely. The forces acting on the aircraft during takeoff roll are computed, the

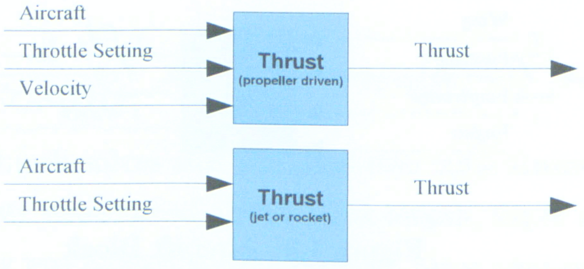


Figure 4.9: Thrust Block

residual forces divided by the aircraft mass yield the lateral acceleration. These forces are reassessed many times per second. Between each three second interval, the aircraft is assumed to have a constant acceleration. The change in velocity during this time, Δt, is then computed followed by the change in position, δs. Time is allowed to advance until the aircraft velocity exceeds the stall speed of the aircraft sufficiently to allow it to safely leave the ground.

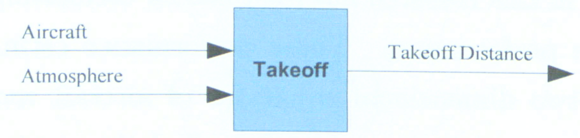


Figure 4.10: Takeoff Block

During the takeoff roll, the forces acting on the aircraft include the thrust produced by the power plant, rolling friction between the wheels and ground, lift created by the wing, and drag. These forces are illustrated in figure 4.11. A summation of these forces yields the acceleration value shown in equation (4.10), with the value μ representing the surface friction coefficient.

$$a = T - D - \mu(W - L) \tag{4.10}$$

Translating this equation to known coefficient values computed by the UAVPart library discussed above yields equation (4.11).

$$a = T - \frac{1}{2}\rho V^2 S(C_{D,0} + kC_L^2) - \mu(W - \frac{1}{2}\rho V^2 SC_L) \tag{4.11}$$

Acceleration is computed for a zero initial velocity and then recalculated every three seconds until liftoff is achieved. Velocity change, ΔV is computed with the assumption

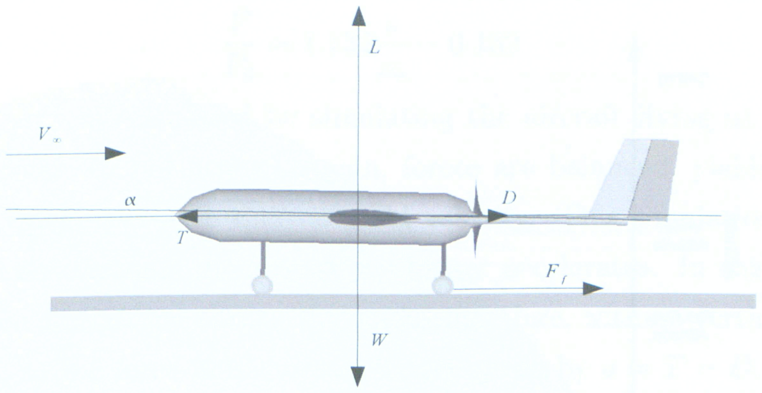


Figure 4.11: Takeoff Free Body Diagram

that acceleration is constant over the time interval, and is given as $\Delta V = a\Delta t$. Position change, ΔS, is then given by $\Delta S = V\Delta t$. Summation of these quantities over the time span of the takeoff roll yields final takeoff distance. The flight envelope block calculates the altitudes and velocities at which the aircraft is capable of operating. Beginning at sea level, the flight envelope block calculates the stall speed of the aircraft and the maximum speed. The altitude of the aircraft is advanced and the process is repeated until the stall speed is equal to the maximum speed of the aircraft, representing the absolute maximum attainable altitude that can be achieved. Flight envelopes are often represented graphically as shown in figure 4.13[41].

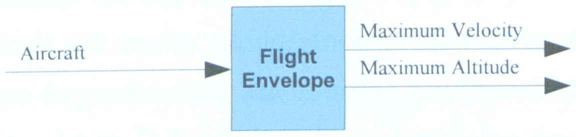


Figure 4.12: Flight Envelope Block

The stall speed of the aircraft is given by equation (4.12), where the term ρ represents the atmospheric density surrounding the aircraft.

$$V_{Stall} = \sqrt{\frac{2W}{\rho V^2 S}} \tag{4.12}$$

As altitude increases, the air density, ρ, decreases. In this research, density change is assumed to follow the International Standard Atmosphere (ISA) model. As the atmosphere thins out with altitude, the dynamic pressure $q = \frac{1}{2}\rho V^2$ at a constant airspeed

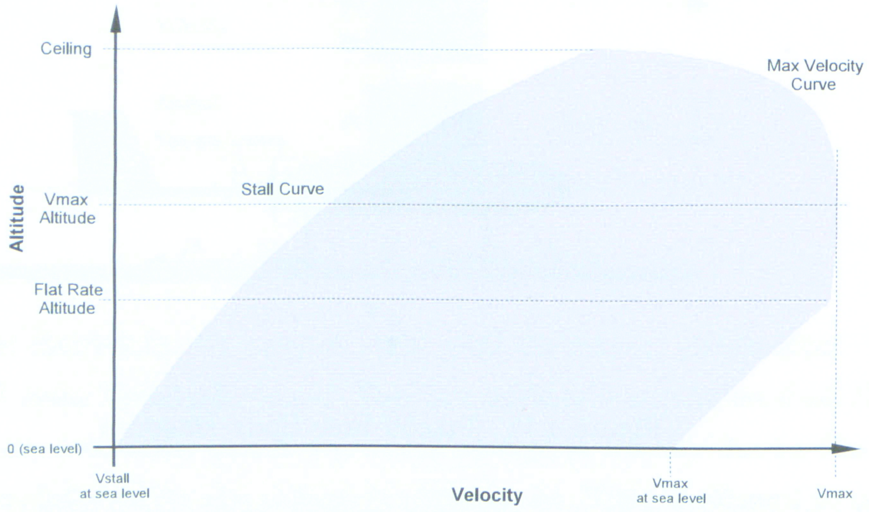


Figure 4.13: Example of a Flight Envelope Graph

decreases, meaning that there are less aerodynamic forces acting on an aircraft at a given speed at high altitudes than at low altitudes. This would seem to indicate that the aircraft should fly faster as altitude increases. However, the power output of air breathing engines is also proportional to atmospheric pressure. As intake pressure decreases, engine output at full power is reduced. This effect is modeled by equation (4.13), presented by Anderson [40], where P is the power produced by the engine at altitude and P_0 is the power produced by the engine at sea level. Equation (4.13) is an empirical equation based on atmospheric measurements, where the decimal numbers are statistical constants. Some air breathing engines are turbocharged such that they can maintain constant power output as altitude increases until the turbocharger's capacity is reached. The altitude at which this occurs is referred to as the flat rate altitude of the engine. Referring to figure 4.13, the stall speed of the aircraft is shown by the left outer curve. As shown, the aircraft stall speed increases with altitude because of the reduced air density. The maximum velocity of the aircraft is defined by the outer right curve. With increasing altitude, the aircraft will be able to fly faster until the engine flat rate altitude is reached. The maximum speed then decreases until the stall curve intersects with the maximum velocity curve, defining the maximum altitude at which the aircraft can operate. The effect of turbo charging is clearly shown in figure 4.13. Aircraft having turbochargers gain maximum velocity performance with increasing altitude until the flat rate altitude is reached, after which velocity performance begins to decrease.

$$\frac{P}{P_0} = 1.132\frac{\rho}{\rho_0} - 0.132 \tag{4.13}$$

Maximum velocity is calculated by simulating the aircraft flying at stall speed and advancing the throttle to full power. Again, forces are balanced, yielding acceleration values which are integrated over time to yield velocity. This simulation is carried out in three second intervals until the aircraft no longer accelerates. In this case, the only forces acting on the aircraft are thrust, lift, and drag. Here, with no vertical acceleration, lift equals weight, so the force imbalance is simply given by $a = T - D$. The free body diagram for this case is shown in figure 4.14. Substituting to write in terms of known coefficients yields equation (4.14).

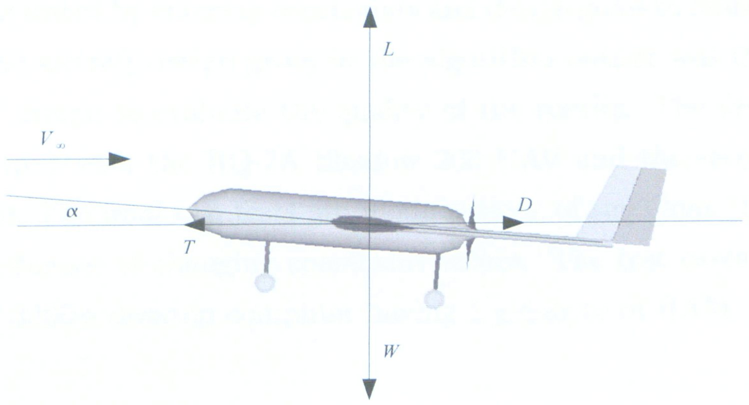


Figure 4.14: Steady Level Flight Free Body Diagram

$$a = T - \left[C_{D,0} + k\left(\frac{2W}{\rho V^2 S}\right)^2\right] \tag{4.14}$$

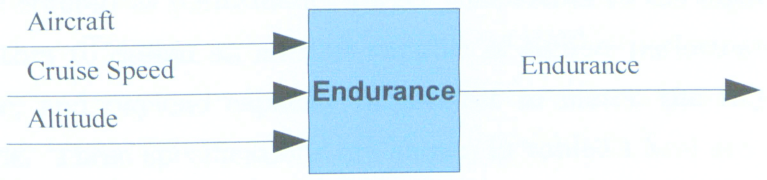


Figure 4.15: Endurance block

The endurance block, shown in figure 4.15, is responsible for estimating the loiter performance of the aircraft. Traditional methods for computing range assume that the aircraft is maintaining the optimal coefficient ratio for endurance in piston powered aircraft, $(\frac{C_L^{3/2}}{C_D})_{max}$. This method was rejected since it requires an unrealistic flight program that constantly adjusts power and airspeed to sustain the maximum ratio as fuel burns off and consequently yields optimistic results. Instead, a time integration simulation approach was used as before. The program simply holds a constant power setting that yields a low airspeed that exceeds the aircraft stall speed by a suitable margin both with maximum and minimum fuel loads. The simulation maintains this speed at the required cruise altitude, deducting the fuel consumed at each time step, δt until the fuel supply is exhausted. At the time of writing, the UAVOpt software can be set to maximize endurance or minimize weight. Due to its modular design, simple modifications can allow for optimization of any of the performance or geometric constraints.

Chapter 5

Results

The algorithm was tested by entering constraints and design goals to match several typical UAV systems. The aircraft design given in the algorithm output was then compared to the existing UAV design to evaluate the quality of the results. The first test compares the algorithm output with the RQ-7A Shadow 200 UAV and the second, the General Atomics Gnat 750. The final two tests are modifications of the Gnat 750 design carried out to test the influence of changing constraint values. The test cases were run on an AMD Athlon XP 3200+ desktop computer having 1 gigabyte of RAM.

5.1 Test One - Lightweight Medium Range

The RQ-7A Shadow 200 is classified as a medium range (MR) UAV, and has typical capabilities compared to other UAVs in this popular class. MR UAVs are the single largest group represented in the UAV database compiled for this research, providing a suitable vehicle against which to test the algorithm. The performance specifications of the RQ-7A were entered as performance target constraints to the algorithm in order to cause the algorithm to design an aircraft capable of similar performance, with weight, velocity, altitude, and payload capacity targets set to match the RQ-7A while maximizing endurance. These specifications are shown in table5.1 and are compared to the algorithm output. The target constraints were assigned a moderate weight such that the algorithm was permitted to undershoot these goals if the benefits to the objective function (maximizing endurance) are large enough.

The algorithm was permitted to run for 150 generations carrying a population of 200 aircraft. The convergence history plot shown in figure 5.1 shows definite convergence

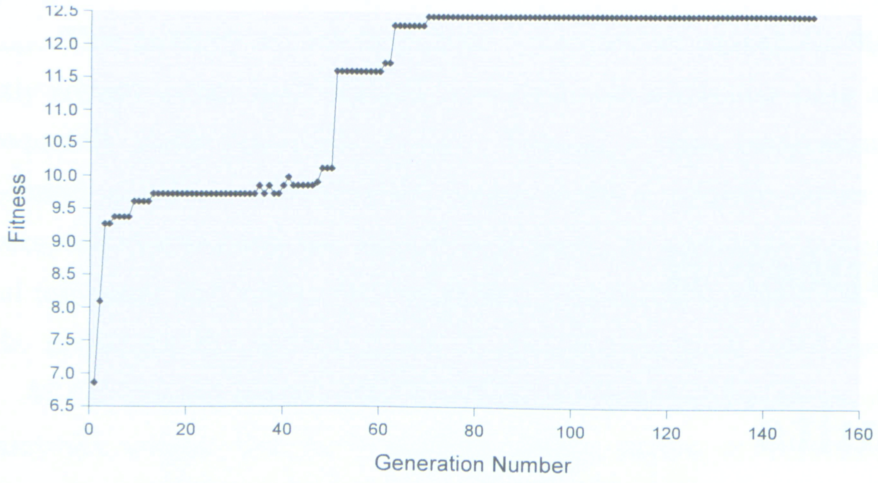
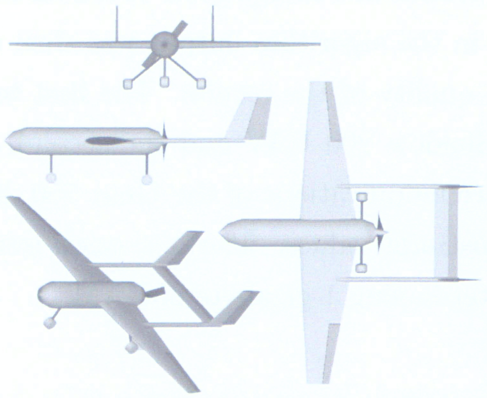


Figure 5.1: Test 1 Convergence History


(a) Test 1 Aircraft

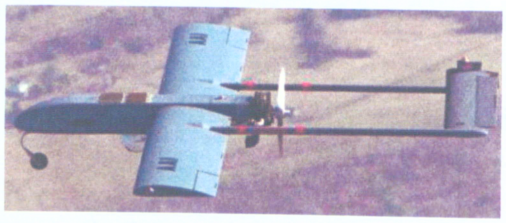
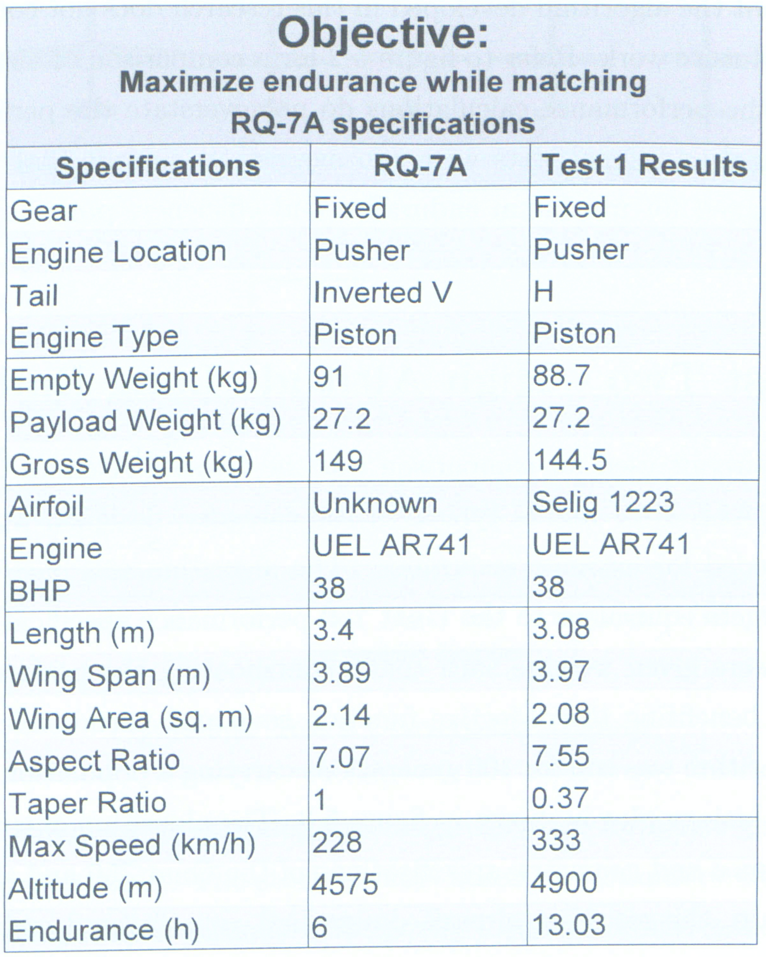

(b) RQ7-A Shadow

Figure 5.2: Test 1 Results: (a) Resulting Aircraft (b) The RQ7-A Shadow UAV

toward higher fitness values and appears to level out beyond 850 generations. Slightly better results may have been achieved if the algorithm had been permitted to run longer, but at great cost to computational time. For GAs, convergence is not something that can be explicitly proved, and it may be found that many generations go by with little or no improvement, then a sudden jump to a higher fitness level occurs. With this being the case, the user must become accustomed to the behavior of the GA to understand when convergence can be expected. This was done by running many trials. It was found that when carrying a population size of 100 members, good convergence usually occurred before 100 generations and very little change was ever observed after 200 generations.

Based on the data mining results, the aircraft was configured with a piston engine in



Objective:
Maximize endurance while matching RQ-7A specifications

Specifications	RQ-7A	Test 1 Results
Gear	Fixed	Fixed
Engine Location	Pusher	Pusher
Tail	Inverted V	H
Engine Type	Piston	Piston
Empty Weight (kg)	91	88.7
Payload Weight (kg)	27.2	27.2
Gross Weight (kg)	149	144.5
Airfoil	Unknown	Selig 1223
Engine	UEL AR741	UEL AR741
BHP	38	38
Length (m)	3.4	3.08
Wing Span (m)	3.89	3.97
Wing Area (sq. m)	2.14	2.08
Aspect Ratio	7.07	7.55
Taper Ratio	1	0.37
Max Speed (km/h)	228	333
Altitude (m)	4575	4900
Endurance (h)	6	13.03

Table 5.1: Test 1 Results

a pusher layout with an "H" style tail and fixed landing gear. The comparison chart in table5.1 indicates that the test aircraft would have superior endurance than the RQ7-A at greater maximum speed and altitude. This is achieved by optimizing the wing design. The algorithm selected a taper ratio of 0.37 which, according to lifting line theory, is close to the optimum planform efficiency of a single tapered wing, coming within a few percentage points of the performance of elliptical wings. Also, the test aircraft has a higher aspect ratio than that of the RQ7-A. Interestingly, the algorithm selected the UEL AR741 engine, which is precisely the same power plant as that of the RQ7-A. Endurance capabilities exceed that of the RQ7-A by a factor of 2, a surprisingly large margin. Much of this margin can be attributed to the test aircraft's superior aerodynamic configuration discussed earlier. With its rectangular wings, simple inverted V tail, and square fuselage, it is apparent that the RQ7-A was designed to maximize simplicity rather than efficiency, something that the algorithm developed in this research does not consider but should be addressed in future work. Refer to figure 5.2 for a comparison of aircraft geometries. To ensure that the performance calculations do not overstate the performance of the test aircraft, several additional tests were run against an aircraft that has the appearance of being designed for maximum endurance and efficiency, providing a better basis for comparison.

5.2 Test Two - High Altitude Long Endurance

The second aircraft used for comparison was the General Atomics Gnat 750. It is classified as a MALE UAV and is typical of the category, designed with the aerodynamic efficiency needed for extreme endurance. The algorithm was once again set with performance targets equivalent to the Gnat 750 performance specifications. As before, the constraints were given weights such that undershooting the performance targets while significantly benefiting the objective function remained permissible. To obtain the result, the algorithm was run for 100 generations carrying a population of 200 aircraft. The convergence history plot is shown in figure 5.3. The algorithm output is compared with the performance and geometric specifications of the Gnat 750 and shown in table5.2.

Once again, the resulting aircraft design had an estimated endurance capability in excess of the existing aircraft under comparison. The configuration selected by the data mining algorithm was a V-tail design with a piston engine in pusher configuration and retractile landing gear. Refer to figure 5.4 for a comparison of the Gnat 750 to a drawing of the algorithm's proposed design. In this case, the superior endurance of the test aircraft can be attributed to its engine selection. The Gnat 750 is powered by a Rotax 582 producing 64hp while the test aircraft has a 51hp UEL AR801, a much smaller and lighter engine. The test aircraft has a shorter wingspan and lower aspect ratio, leading to a lighter structural weight, allowing the aircraft to perform adequately with the smaller engine. However, the superior endurance of the test aircraft comes altitude capability, which is inferior to the Gnat 750. Evidently, the weight applied to the altitude constraint was such that the benefits of the superior endurance afforded by the smaller engine outweighed the ability to reach the altitude target. If desired, constraint weights can be assigned very high values, effectively eliminating the possibility of undershooting design goals. This is not advisable, however. Doing so may eliminate potentially successful designs that come close to the design goals. Additional tests were carried

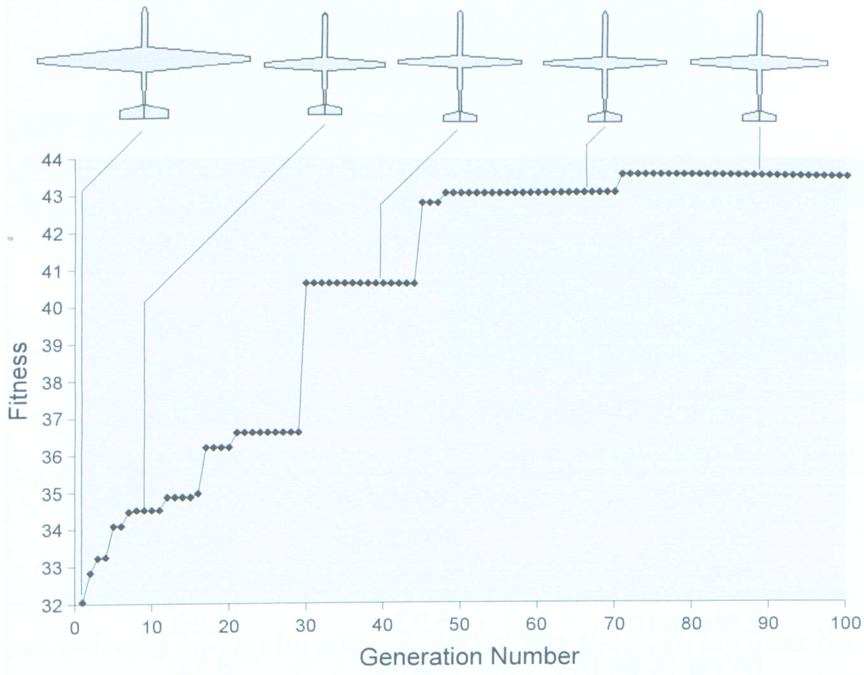
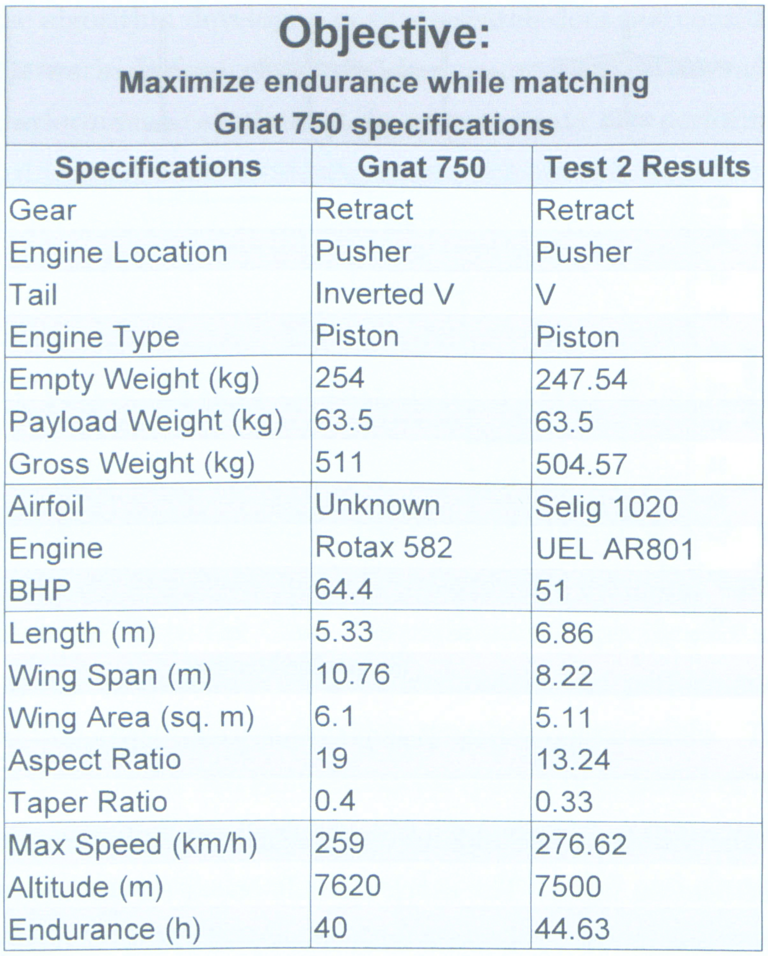


Figure 5.3: Test 2 Convergence History

out to study the influence of assigning more aggressive performance goals to the basic Gnat 750 specification input. The stock aircraft based on the unaltered Gnat 750 UAV specifications is compared to aircraft with mostly identical design goals except for one key parameter under study for each of the tests. The first is an attempt at matching the Gnat 750 performance while minimizing weight. The second attempts to maximize endurance while having the target capability of flying at 320 km/h, much higher than the stock test aircraft. The third test also maximizes endurance but attempts to reach the target capability of an 8500 m altitude ceiling.

Figure 5.5 shows the results of this test. Clearly, the algorithm successfully adapted to these altered design goals. The weight minimization test yielded an aircraft weighing significantly less than the stock aircraft. The optimum design has a significantly smaller wing span and area than that of the stock aircraft, effectively reducing the structural weight of the aircraft. It is powered by a 51 hp engine, slightly smaller and lighter than the engine powering the stock aircraft. The design is slightly inferior in endurance and altitude performance, but had as superior maximum speed - all consequences of the smaller wing design and smaller engine. The second test, with an increased velocity constraint, yielded an aircraft with significantly narrower and shorter wings than the stock aircraft, typical of high speed aircraft designs. Both the wing span and wing area

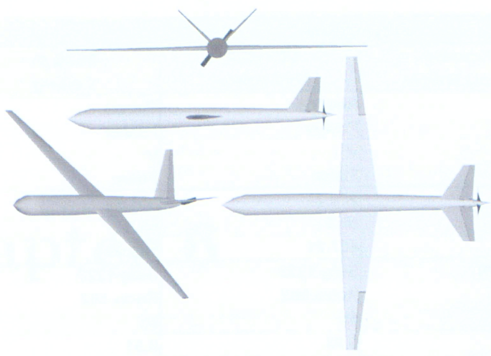


Objective:
Maximize endurance while matching Gnat 750 specifications

Specifications	Gnat 750	Test 2 Results
Gear	Retract	Retract
Engine Location	Pusher	Pusher
Tail	Inverted V	V
Engine Type	Piston	Piston
Empty Weight (kg)	254	247.54
Payload Weight (kg)	63.5	63.5
Gross Weight (kg)	511	504.57
Airfoil	Unknown	Selig 1020
Engine	Rotax 582	UEL AR801
BHP	64.4	51
Length (m)	5.33	6.86
Wing Span (m)	10.76	8.22
Wing Area (sq. m)	6.1	5.11
Aspect Ratio	19	13.24
Taper Ratio	0.4	0.33
Max Speed (km/h)	259	276.62
Altitude (m)	7620	7500
Endurance (h)	40	44.63

Table 5.2: Test 2 Results

is smaller than the stock aircraft by a significant margin and the aspect ratio is much higher. These factors yielded a more efficient wing design for high speed flight. The optimized aircraft is powered by the 65 hp Rotax 582 engine, significantly more power than what was required for the stock aircraft, enabling the design to reach a maximum altitude of 7600 m. Not surprisingly, the extra power and speed comes at a significant cost to the endurance. At under 33 hours, it is significantly less capable than the 44 hours achieved by the stock aircraft. The final test had the design goal of 8500 m altitude capability. The optimized aircraft has a long and narrow wing design, similar to that of the stock aircraft. However, the aircraft has significantly more power available than the stock aircraft. The 65 hp Rotax was selected, being both more powerful and heavier than the 51 hp engine powering the stock aircraft. The aircraft proved capable of flight at an altitude of 8100 m, slightly undershooting the 8500 m target. This occurred because an


(a) Test 2 Aircraft

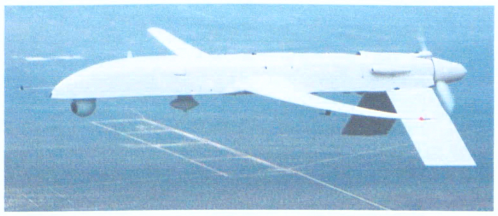

(b) General Atomics Gnat 750

Figure 5.4: Test 2 Results: (a) Resulting Aircraft (b) The Gnat 750 UAV

aircraft capable of reaching the final 400 m would have likely required a larger engine, and would have reduced the endurance objective too much to surpass the fitness of the slightly less capable aircraft having the 65 hp engine.

5.3 X-Plane Simulation

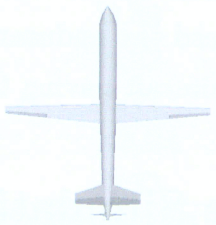
To test the feasibility of the algorithm results, they were modeled in X Plane, a commercially available flight simulator for personal computers. The X Plane simulator has a realistic flight model that computes aircraft flight characteristics by applying the blade element theory to the aircraft geometry, solving for forces and accelerations. Figure 5.6 shows the Test 1 aircraft X Plane model. While a direct comparison of performance between the algorithm output and X Plane is not easily done because of the difficultly of matching conditions precisely, the agreement between the algorithm estimated performance and the X Plane simulation were generally very close. The static stability of the test aircraft was verified as well[42].

Specifications	Objective			
	Match Gnat 750 Specs	Minimize Empty Weight	320 km/h Max Speed	8500 m Ceiling
Gear	Retract	Retract	Retract	Retract
Engine Location	Pusher	Pusher	Pusher	Pusher
Tail	V	V	V	V
Engine Type	Piston	Piston	Piston	Piston
Empty Weight (kg)	247.54	149.01	200.21	251.99
Payload Weight (kg)	63.5	63.5	63.5	63.5
Gross Weight (kg)	504.57	406.01	457.21	508.99
Airfoil	Selig 1020	Selig 1020	Selig 1223	Selig 1223
Engine	UEL AR801	L-550E	Rotax 582	Rotax 582
BHP	51	50	65	65
Length (m)	6.86	6.80	6.93	6.91
Wing Span (m)	8.22	6.43	6.15	7.89
Wing Area (sq. m)	5.11	3.08	2.54	5.00
Aspect Ratio	13.24	11.26	14.91	12.47
Taper Ratio	0.33	0.30	0.32	0.29
Max Speed (km/h)	276.62	292.67	321.95	291.3
Altitude (m)	7500	7400	7600	8100
Endurance (h)	44.63	31.42	32.86	31.7

(a) Comparison

(b) Stock (Case 2)

(c) Decreased Weight


(d) Increased Speed

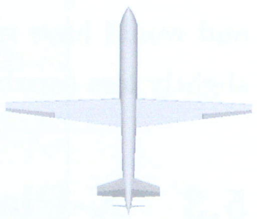
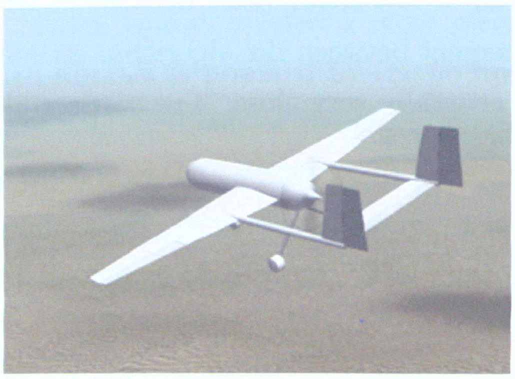

(e) Increased Altitude

Figure 5.5: These figures highlight the influence of modifications to the design requirements. Figure (b) is the result from the Gnat 750 comparison case. Figures (c), (d), and (e) are modifications of this case, each with altered requirements as shown and figure (a) is a specification comparison chart.


(a) X Plane Model of Test 1

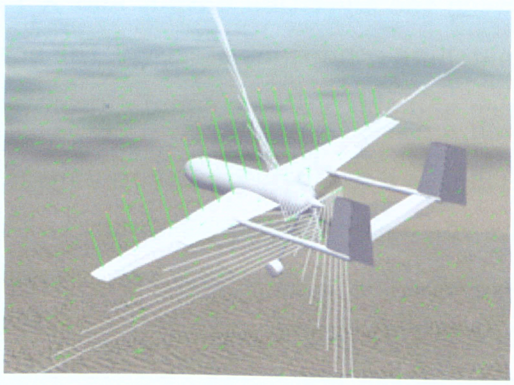

(b) Test 1 with visible flight model

Figure 5.6: X Plane Model

Chapter 6

Conclusion

The software developed in this research was shown to be capable of designing efficient aircraft, capable in theory of outperforming current designs with similar capabilities. Data mining proved capable of aiding configuration decisions for a wide variety of design possibilities, but in some instances it was hampered by insufficient data in the more obscure UAV categories. Improved performance analysis would be of great benefit. The solution of the two dimensional equations of motion, while superior to the simplified, directly solvable equations, does not account for dynamic instability and control coupling. Should a user desire to build and fly an aircraft designed by the algorithm, they would have to conduct much additional analysis to ensure that the aircraft is dynamically stable, and they may find that the empennage sizing must be altered. Since control coupling (the influence, for example, of rudder on the roll axis, and ailerons on the yaw axis) is not considered in two dimensional analysis, it is not possible to size control surfaces except by "rule of thumb" methods that do not globally apply to all design configuration possibilities available to the UAV designer. Propeller analysis is also an area of weakness in this research and may account for some of the apparent optimism observed in the performance analysis. Propellers were assumed to be continuously operating at maximum efficiency. This is not possible in reality, particularly for fixed pitch propellers. This research was aimed at conceptual aircraft design and did not carry out structural design. This necessitated a statistical weight analysis method. The equations were borrowed from statistical relationships used for civil, military, and commercial aircraft and scaled to fit the UAV database wherever, possible. While the equations follow the right curve - increasing weight for long, high aspect ratio wings and slender structures - they may not always provide a realistic estimate and may contribute to the optimistic performance results.

Another factor worthy of note was the limitations to the UAV database. With over 200 entries, there was sufficient data available for certain UAV categories to have significant representation, allowing for meaningful patterns to be exposed by the data mining algorithm. However, UAV categories pushing the envelope - either small, fast, high, or otherwise exceptional, do not have sufficient representation in the database for meaningful results to be extracted, making it necessary to limit algorithm tests covered in this report to areas of strength. This issue can be resolved with continued expansion of the database. Despite these concerns, the algorithm proved to be an excellent framework upon which a full multidisciplinary optimization package can be built.

The goal of this research was to design a robust optimization package for UAV conceptual design. A unique database driven approach was taken. Data mining was used to extract knowledge from a database of existing UAV systems to build empirical decision models to aid aircraft layout and configuration decisions. A genetic algorithm optimizer was developed with the ability to handle continuous variables, such as wing sizing, as well as discrete variables including indexing actual engine and airfoil specifications from a parts database. A variety of performance constraints were considered with the objective function being selectable between weight minimization or endurance maximization. Performance evaluation was carried out by time integration of the two dimensional equations of motion.

The algorithm was tested by entering performance data from existing UAV designs as performance targets for new designs. In all cases, the algorithm was able to design a concept aircraft with performance equal or superior to the aircraft under comparison, proving the validity of the algorithm framework established in this research. Limitations to be addressed in future work included the need for expansion of the UAV database, the weakness of the weight and propeller analysis, the lack of structural analysis, and the lack of dynamic performance and stability considerations. The algorithm developed in this research is a promising framework for a full multidisciplinary UAV design optimization package and continued development will increase its formidable capability and performance.

6.1 Future Work

Despite the success of this research, considerable improvements can be made upon the framework of the algorithm and are summarized below.

- Greatly expanding the UAV specification database
- Incorporating structural design
- Adding avionics and control component selection
- Allowing additional design variables and objective functions
- Generate a cost model using data mining
- Add ability to suggest existing UAV systems to the user
- Add the ability to handle multiple objective functions simultaneously
- Expand the algorithm to a true Multidisciplinary Design Optimization system by
  - Adding structural analysis
  - Improving aerodynamic computation
  - Handling 3d aircraft equations of motion
  - Studying aerodynamic and structural interaction

Appendix A

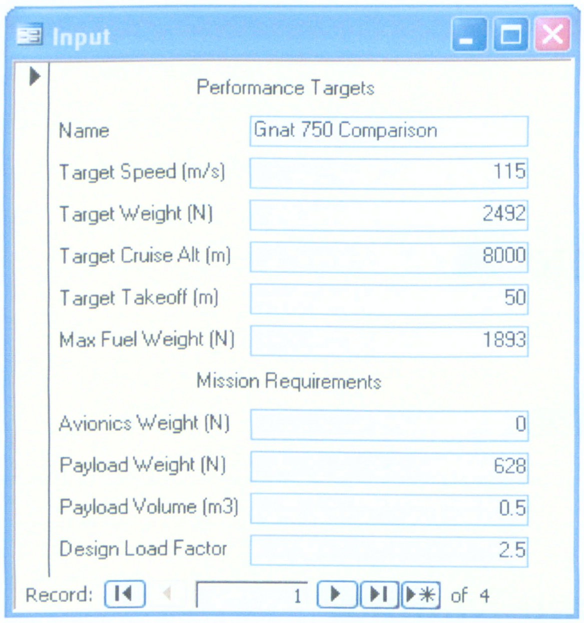
Appendix

A.1 Program Manual

The software developed in this research is called UAVOpt. It was programmed in Java and requires the Java 5.0 virtual machine to run. This package is public domain and available from www.java.com and must be installed prior to using UAVOpt. Additionally, the UAVOpt database was developed using Mircrosoft Access 2000 and requires Access 2000 or newer for modifying database contents. The data mining package used in this research was Weka 3.4. It is an open source Java based data mining environment and is available from www.cs.waikato.ac.nx/ml/weka/.

UAVOpt can be installed into any directory. The package contains two files, the database file, UAV.mdb, and the compiled software library, UAVOpt.jar, which can be executed on any computer with Java virtual machine 5.0 installed. Before running, the database must be mounted as a data source in the computer operating system. In Microsoft Windows, this is accomplished from the Administrative Tools - Data Sources entry in the system control panel. The database must be mounted as an ODBC database and named UAVDB. Detailed information on how to add data sources in Windows, Linux, or Macintosh can be found in the operating system manuals.

With Java 5.0 installed and the UAV.mdb database correctly mounted, input cases must be defined before running UAVOpt. This is done by opening UAV.mdb with Microsoft Access or compatible database utilities. The cases to be analyzed by UAVOpt must be entered into the "Input" form, shown in figure A.1. As many cases can be set up as desired. Entries are saved immediately when entered and are immediately available from the UAVOpt menu for analysis.



Performance Targets

Name	Gnat 750 Comparison
Target Speed (m/s)	115
Target Weight (N)	2492
Target Cruise Alt (m)	8000
Target Takeoff (m)	50
Max Fuel Weight (N)	1893

Mission Requirements

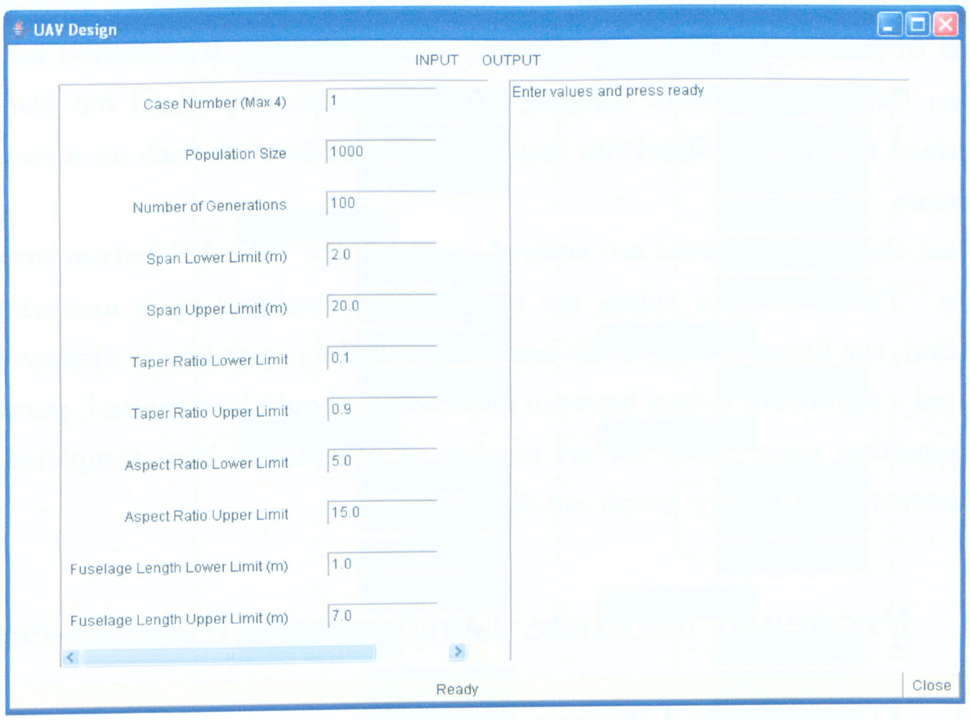
Avionics Weight (N)	0
Payload Weight (N)	628
Payload Volume (m3)	0.5
Design Load Factor	2.5

Record: 1 of 4

Figure A.1: UAVOpt Input Form

Test cases can be assigned names for reference in the "Name" dialog box. The next four dialog boxes are performance targets for the UAV to be designed. Target values can be entered and the genetic algorithm will strive to meet or exceed the targets entered. If the targets are met or surpassed, the GA will no longer seek to improve those areas. If it is desired for a parameter to be maximized, entering a very large number into the dialog box will ensure that the GA will never cease striving to improve that aspect. Likewise, if it is desired for a parameter to be minimized, such as weight, simply entering zero will ensure that the GA will never stop attempting to improve that aspect. The fuel weight dialog box allows the user to cap the maximum allowable fuel load assigned to an aircraft. This value must be greater than zero. This ensures that the GA will not try to improve endurance simply by adding more fuel indefinitely, and will instead improve aircraft efficiency. The remaining dialog boxes define the mission requirements of the UAV such as payload requirements and structural load factors at which the UAV is intended to safely operate. With one or more cases defined, UAVOpt can be executed and cases can be selected and analyzed. Running UAVOpt in Microsoft Windows is done simply by clicking the UAVOpt.jar file.

The main menu, shown in figure A.2, is displayed immediately upon execution of UAVOpt. A number of dialog boxes are immediately visible and contain recommended default values. Input cases defined earlier are selected by entering an index number



Case Number (Max 4): 1
Population Size: 1000
Number of Generations: 100
Span Lower Limit (m): 2.0
Span Upper Limit (m): 20.0
Taper Ratio Lower Limit: 0.1
Taper Ratio Upper Limit: 0.9
Aspect Ratio Lower Limit: 5.0
Aspect Ratio Upper Limit: 15.0
Fuselage Length Lower Limit (m): 1.0
Fuselage Length Upper Limit (m): 7.0

Enter values and press ready

Figure A.2: UAVOpt Main Menu

into the "Case Number" dialog box. The "Population Size" box sets the size of the aircraft population carried by the GA. Increasing the population number beyond 1000 will severely increase computational time with little benefit. Reducing the population size decreases computational time, but usually increases the number of generations required for convergence. The "Number of Generations" dialog box sets the generation number at which the GA will terminate. Higher numbers will increase computational time but may be required for convergence. Testing has shown that with population sizes of 1000, convergence is usually observed after 1000 generations, with answers coming within 90% of that after only 100 generations. The law of diminising returns applies, as good answers become available in as few generations as 100, while complete convergence cannot be assumed before at least 1000 generations. The remaining dialogue boxes set up the limits of the continuous design variables. The discreet design variables (engine and airfoil index) are set automatically based on the number of database entries. These can usually remain at their default values, but if there are geometric constraints such as a wingspan or fuselage length constaint on a design case, entering the limits here will ensure compliance to such requirements. Setting the aspect ratio limit lower than 5 is not recommended since UAVOpt's 3d wing analysis loses accurcy for low aspect ratio wings.

For best results, the limits should be defined realistically and be as wide as practically possible to ensure algorithm convergence. If, for example, it is desired for the UAV to reach an altitude of 9000 m carrying a large amount of payload but the wingspan is constrained to 1 m, the algorithm may not converge because such an aircraft would not be possible.

When the desired values are entered, pressing the "Ready" button executes the algorithm. The user input values are immediately displayed upon execution. For each generation, the fitness rating of the best design in the population is displayed as they are computed - significant delays between generations should be expected, particularly with large popultion sizes. After the set number of generations have completed, the results and a convergence history graph are displayed.

A.2 Program Module Listing and Pseudo-code

A.2.1 UAV-Math Library

The UAVMath library is responsible for carrying out advanced mathematical operations not available in the standard Java libraries. Some were custom written by the author and some are based on public domain math libraries. Description of the public domain libraries used is limited to the classes used in this research.

Matrix

- Source: Part of the JAMA public domain matrix library for Java
- Input
  - A one or two dimensional array, A.
- Procedure
  - Builds a matrix data structure..
- Output
  - getMatrix() - Returns the matrix object, A.
- Usage

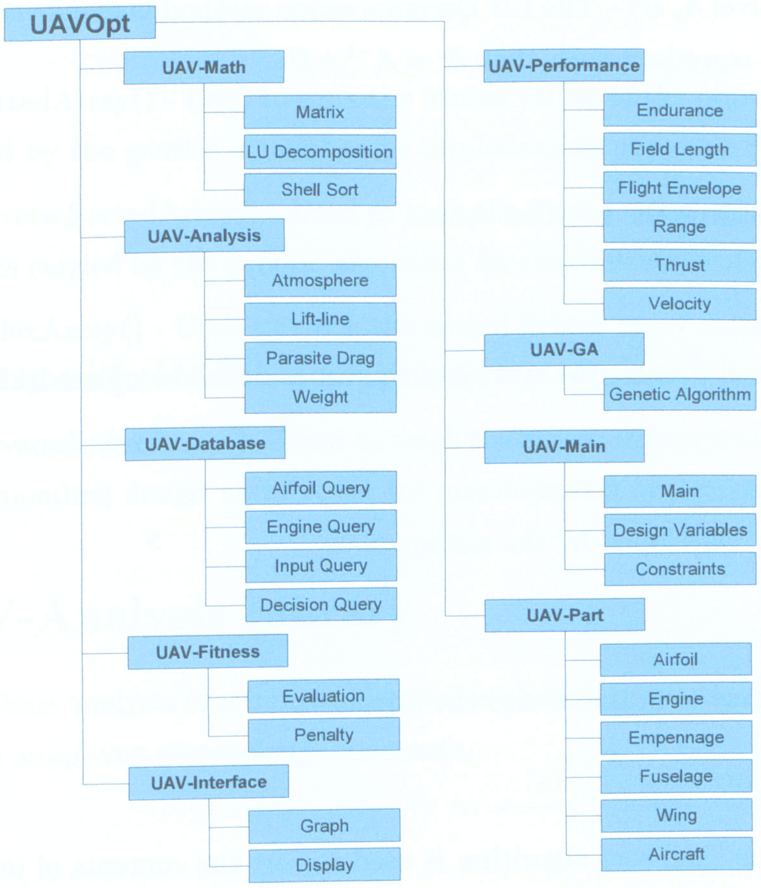


Figure A.3: UAVOpt Library Structure

  - Used by the LU Decomposition class for solving systems of linear equations.

LU-Decomposition

- Source: Part of the JAMA public domain matrix library for Java
- Input
  - An n by n matrix, A and an array of length n, B.
- Procedure
  - The LU Decomposition method is employed to conduct matrix operations including solutions of linear equations, and determinants.
- Methods

  - solve(A, B) - The LU Decomposition method is employed to solve the system of equations such that $X = A^{-1} * B$.

- Output
  - Returns the solution vector, X.

- Usage
  - Used to solve the system of equations defined by Prandtl Lifting Line analysis.

Shell-Sort

- Source: Developed by the author.

- Input
  - Array A of size n and the integer n.

- Procedure
  - The shell sort algorithm is used to sort the contents of input array, A.

- Methods
  - solve() - Instructs the algorithm to carry out the sort.

- Output
  - getSortedArray() - Returns an array, B, containing the contents of array A sorted from smallest to largest.
  - getReverseSortedArray() - Returns an array, B, containing the contents of array A sorted from largest to smallest
  - getIndexArray() - Returns an array containing integer index numbers that indicate the position of the sorted values in B to where they can be found in A when B is sorted from lowest to highest.
  - getReverseIndexArray() - Returns an array containing integer index numbers that indicate the position of the sorted values in B to where they can be found in A when B is sorted from highest to lowest.

- Usage
  - getSortedArray() - Used to rank the fitness values of the population of designs carried by the genetic algorithm for minimization objective functions.
  - getReverseSortedArray() - Used to rank the fitness values of the population of designs carried by the genetic algorithm for maximization objective functions.
  - getIndexArray() - Used to track the sorted fitness array values to their corresponding design variable set for minimization objectives.
  - getReverseIndexArray() - Used to track the sorted fitness array values to their corresponding design variable set for maximization objectives.

A.3 UAV-Analysis Library

This library contians analysis functions to determine aircraft properties such as drag and weight as well as supplying atmospheric constants.

Atmosphere

- Source: Developed by the author.
- Input
  - A real number (double), h, representing the aircraft altitude in meters.
- Procedure
  - The International Standard Atmosphere (ISA) model is used to calculate atmospheric temperature, density, sound speed, and pressure as a function of altitude.
- Methods
  - solve() - Instructs the algorithm to carry out the calculation.
- Output
  - getDensity() - Returns a real number, d, representing the atmospheric density at altitude h in kg/m^3.

- getDensityRatio() - Returns a real number, *dratio*, representing the ratio of the atmospheric density at altitude h to the density at sea level.
- getPressure() - Returns a real number, p, representing the atmospheric pressure at altitude h in N/m^2.
- getPressureRatio() - Returns an array containing integer index numbers that indicate the position of the sorted values in B to where they can be found in A when B is sorted from highest to lowest.

- Usage
  - getSortedArray() - Used to rank the fitness values of the population of designs carried by the genetic algorithm for minimization objective functions.
  - getReverseSortedArray() - Used to rank the fitness values of the population of designs carried by the genetic algorithm for maximization objective functions.
  - getIndexArray() - Used to track the sorted fitness array values to their corresponding design variable set for minimization objectives.
  - getReverseIndexArray() - Used to track the sorted fitness array values to their corresponding design variable set for maximization objectives.

Lift-Line

- Source: Developed by the author.
- Input: Real numbers
  - AR, representing the aspect ratio of an aircraft wing.
  - $Span$, representing the wing span.
  - TR, representing the wing taper ratio
  - $LiftSlope$, representing ratio of lift coefficient to angle of attack of the two dimensional airfoil.
  - $a0$, representing the zero angle of attack lift coefficient of a cambered airfoil.
- Procedure

  - The Pandtl Lifting Line method is used to compute three dimensional airfoil properties. The method involves dividing the wing into segments, building a system of linear equations. The solution is computed by building matrix objects and calling the LU Decomposition class.

- Methods

  - solve() - Instructs the algorithm to carry out the calculation.

- Output: Returns real numbers.

  - getCl() - Returns Cl, representing the lift coefficient of the three dimensional wing.
  - getCla() - Returns Cla, representing the three dimensional ratio of lift coefficient to angle of attack.
  - getCd() - Returns Cd, representing the induced drag coefficient of the three dimensional wing.
  - getAi() - Returns Ai, representing the induced angle of attack.
  - gete() - Returns e, the Oswald efficiency factor.

- Usage

  - getCl() - Used to compute the lift coefficient of the wing when the angle of attack is known.
  - geCla() - Used to compute the lift slope of the wing to be used internally for lift and drag computation in all of the performance estimation modules.
  - getCd() - Used to compute the induced drag of the wing when the angle of attack is known.
  - getAi() - Used to compute the induced angle of attack, required to find the true maximum lift coefficient of the three dimensional wing.
  - gete() - Used to compute the induced drag constant, k, relating induced drag to lift coefficient, required for all performance estimation modules.

Parasite Drag

- Source: Developed by the author.
- Input: Part data types.
  - *Wing*, representing the aircraft wing part.
  - *Fuselage*, representing the aircraft fuselage part.
  - *Empennage*, representing the aircraft empennage part.
- Procedure
  - The component drag buildup method is employed to estimate the parasite drag of the aircraft. This is a semi-empirical method based on the profile, thickness, roughness, and wetted area of the aircraft and Reynolds number in which each part is subjected to.
- Methods
  - solve() - Instructs the algorithm to carry out the calculation.
- Output: Returns real numbers.
  - getCd0() - Returns $Cd0$, representing the parasite drag coefficient of the aircraft.
- Usage
  - getCd0() - Is a component of the drag model of each aircraft used in all performance estimation modules.

Weight

- Source: Developed by the author.
- Input: Part data types and the constraint data type.
  - *Wing*, representing the aircraft wing part.
  - *Fuselage*, representing the aircraft fuselage part.
  - *Empennage*, representing the aircraft empennage part.

  - *Engine*, representing the aircraft engine part.

- Procedure

  - A statistical method of weight estimation scaled to fit the UAV database is used to estimate aircraft weight based on the dimensions of each part. Payload and fuel weight, taken from the constraint object is added, as well as the engine weight from the engine part.

- Methods

  - solve() - Instructs the algorithm to carry out the calculation.

- Output: Returns real numbers.

  - getCargoWeight() - Returns $CargoWeight$, representing the payload weight the aircraft is required to carry in N.
  - getEmptyWeight() - Returns $EmptyWeight$, representing the aircraft empty weight in N.
  - getFuelWeight() - Returns $FuelWeight$, representing the maximum fuel weight the aircraft carries in N.
  - getGrossWeight() - Returns $GrossWeight$, representing the aircraft maximum takeoff weight in N.

- Usage

  - All weight estimates used for performance estimation
  - getEmptyWeight() - Used for fitness comparison between aircraft designs.

A.3.1 UAV-Database

The UAV-Database library is responsible for sending query commands to the database.

Airfoil-Query

- Source: Developed by the author.

- Input: A String and a real number.

  - *SortVar*, a string representing the airfoil variable by which to sort the query results.
  - *Rex*, a real number representing the Reynold's regime for which airfoil performance data is to be retrieved.

- Procedure

  - A database query is executed to retrieve airfoil specifications from the database. The information is stored in an array for rapid retrieval.

- Output: Returns an integer index number and real numbers.

  - getAirfoilID(index) - Returns *AirfoilID*, an integer index value pointing to the airfoil location in the database.
  - getRefTipArea(index) - Returns *RefTipArea*, representing the airfoil tip area to chord length ratio.
  - getRefPerimeter(index) - Returns *RefPerimeter*, representing the airfoil perimeter to chord length ratio.
  - getMaxThicknessRatio(index) - Returns *MaxThicknessRatio*, representing the ratio of airfoil thickness to chord.
  - getLDmax(index) - Returns *LDmax*, representing the maximum lift to drag ratio of the airfoil.
  - getCl0(index) - Returns *Cl*0, representing the zero angle of attack lift of the airfoil.
  - getCm0(index) - Returns *Cl*0, representing the airfoil moment coefficient.
  - getLiftSlope(index) - Returns *LiftSlope*, representing the ratio of lift coefficient to angle of attack of the airfoil.
  - getClmax() - Returns *Clmax*, representing the airfoil maximum lift coefficient prior to stall.

- Usage

  - getAirfioilID(index) - Used to trace the airfoil data to its location in the database for reference.

  - getRefTipArea(index) - Used by the weight and drag modules to compute the wetted area of the wing tips.
  - getRefPerimeter(index) - Used by the weight and drag modules to calculate the wing wetted area.
  - getMaxThicknessRatio(index) - Used by the weight analysis module as a variable in the statistical equations.
  - getLDmax(index) - Used as a reference to for comparison to other airfoils.
  - getCl0(index) - Used by the lifting line module for building the lift and drag model.
  - getCm0(index) - Used by the empennage model as part of the static stability analysis.
  - getLiftSlope(index) - Used by the lifting line module for building the lift and drag model.

Engine-Query

- Source: Developed by the author.
- Input: A String and a real number.
  - *SortVar*, a string representing the airfoil variable by which to sort the query results.
  - *Type*, a string representing the engine type; jet, piston, turboprop, or electric.
- Procedure
  - A database query is executed to retrieve engine specifications from the database. The information is stored in an array for rapid retrieval.
- Output: Returns an integer index number and real numbers.
  - getEngineID(index) - Returns *EngineID*, an integer index value pointing to the airfoil location in the database.
  - getLength(index) - Returns *Length*, representing the engine length in *mm*.
  - getWidth(index) - Returns *Width*, representing the engine width in *mm*.

- getHeight(index) - Returns *Height*, representing the engine height in *mm*.
- getWeight(index) - Returns *Weight*, representing the engine weight in *N*.
- getPowerSL(index) - Returns *Psl*, representing the maximum power produced by the engine at sea level.
- getCm0(index) - Returns *Cl0*, representing the airfoil moment coefficient.
- getLiftSlope(index) - Returns *LiftSlope*, representing the ratio of lift coefficient to angle of attack of the airfoil.
- getClmax(index) - Returns *Clmax*, representing the airfoil maximum lift coefficient prior to stall.

- Usage
  - getAirfioilID(index) - Used to trace the airfoil data to its location in the database for reference.
  - getRefTipArea(index) - Used by the weight and drag modules to compute the wetted area of the wing tips.
  - getRefPerimeter(index) - Used by the weight and drag modules to calculate the wing wetted area.
  - getMaxThicknessRatio(index) - Used by the weight analysis module as a variable in the statistical equations.
  - getLDmax(index) - Used as a reference to for comparison to other airfoils.
  - getCl0(index) - Used by the lifting line module for building the lift and drag model.
  - getCm0(index) - Used by the empennage model as part of the static stability analysis.
  - getLiftSlope(index) - Used by the lifting line module for building the lift and drag model.

Input-Query

- Source: Developed by the author.
- Input: None.

- Procedure
  - A database query is executed to retrieve saved constraint and performance targets set by the user for algorithm input.
- Output: Returns an integer number and real numbers.
  - getAvionicsWeight(index) - Returns $Wavionics$, representing the required weight of avionics to be carried by the UAV in N.
  - getNumEntries() - Returns an integer index n, representing the number of test cases defined in the input database table.
  - getFuelWeight(index) - Returns $Wfuel$, representing the required maximum allowable fuel weight in N.
  - getPayloadVolume(index) - Returns $PayloadVol$, representing the required payload volume in m^3.
  - getPayloadWeight(index) - Returns $Wpayload$, representing the payload weight in N.
  - getTargetCruiseAlt(index) - Returns $ReqCruiseAlt$, representing the operating altitude the UAV is intended to reach in m.
  - getTargetSpeed(index) - Returns V, representing the target maximum velocity of the UAV in m/s.
  - getTargetTakeoffDistance(index) - Returns $TOdist$, representing the target maximum allowable takeoff field length of the UAV in m.
  - getTargetWeight(index) - Returns $Wmax$, representing the maximum allowable empty weight of the UAV.
- Usage
  - getNumEntries(index) - Used by the user interface.
  - getTargetFuelWeight(index) - Sets the limit of fuel.
  - getPayload Volume(index) - Used by the fuselage module to ensure that space is available for the payload.
  - getPayloadWeight(index) - Used by the weight analysis module to compute aircraft gross weight.

- getTargetCruiseAlt(index) - Used by the penalty module to penalize designs having maximum altitudes below the target.
- getTargetSpeed(index) - Used by the penalty module to penalize designs having maximum speeds below the target.
- getTargetTakeoffDistance(index) - Used by the penalty module to penalize designs having takeoff distances above the target.
- getTargetWeight(index) - Used by the penalty module to penalize designs having empty weights above the target.

Decision-Query

- Source: Developed by the author.
- Input: A design variable object and a constraint object.
- Procedure
  - A database query is executed to retrieve saved decision tree information. The design criterion and contraints are used to search for configuration decisions applicable to aircraft with similar requirements.
- Output: Returns string parameters.
  - getTailType() - Returns *TailType*, representing the empennage type selected. Can be H, V, T, conventional, or canard.
  - getEngineType() - Returns *EngType*, representing the engine type selected. Can be piston, turboprop, or jet.
  - getLandingGearType() - Returns *GearType*, representing the landing gear type selected. Can be fixed, retractible, or none.
  - getEngineLocation() - Returns *EngLocation*, representing the engine location selected. Can be pusher, tractor, or wing mounted.
- Usage
  - getTailType() - Used by the empennage module to determine tail sizing and by the fuselage to determine whether twin booms are needed or not.

  - getEngineType() - Used by the engine query module to restrict the database query to the correct type of engine for the application under consideration.
  - getLandingGearType() - Used by the drag model to account for reduced drag for the case of retractible or no gear and increased drag for conventional gear.
  - getEngineLocation() - Used by the empennage model for center of gravity calculations affecting static stability.

A.3.2 UAV-Fitness

The UAV-Fintness library contians modules responsible for controlling fitness evaluation and penalty function computation.

Evaluation

- Source: Developed by the author.
- Input: A design variable, constraints, airfoil query, and engine query object.
  - *DV*, representing the design variable set.
  - *Constraints*, representing the constraint set.
  - *FoilQuery*, representing the airfoil query data.
  - *EngineQuery*, representing the engine query data.
- Procedure
  - Assembles the aircraft and controlls the execution of the UAV-Part library, the UAV-Analysis library, and the UAV-Performance library as well as the penalty function module to determine the fitness of a UAV towards a given objective funtion.
- Methods
  - solve() - Instructs the algorithm to carry out the calculation.
- Output: An aircraft object, and real numbers.
  - getAircraft() - Returns *aircraft*, representing a fully assembled aircraft data type.

- getMaxVelocity() - Returns $Vmax$, representing the maximum velocity the aircraft can attain in m/s.
- getMaxEndurance() - Returns $GearType$, representing the endurance capabilities of the aircraft in s.
- getRange() - Returns R, representing the maximum flight distance the aircraft can achieve before refuelling in m.
- getTakeoffDistance() - Returns $TOdist$, representing the takeoff distance achieved by the aircraft in m.
- getFitness() - Returns $Fitness$, representing the penalized fitness rating of aircraft performance towards a given objective function.

- Usage
  - getAircraft() - Used by the main program module to report the specifications of the aircraft.
  - getMaxVelocity() - Used by the main program module to report the specifications of the aircraft.
  - getMaxEndurance() - Used by the main program module to report the specifications of the aircraft.
  - getRange() - Used by the main program module to report the specifications of the aircraft.
  - getTakeoffDistance() - Used by the main program module to report the specifications of the aircraft.
  - getFitness() - Used by the genetic algorithm module for ranking the design population.

Penalty

- Source: Developed by the author.
- Input: Real numbers.
  - $RequiredValue$, representing a required performance target.
  - $AchievedValue$, representing the performance achieved by the UAV.

- K, representing the weight to be applied to the penalty.

- Procedure
  - Assigns a penalty multiplier based on the difference between required and achieved performance and a weight factor. The penalty function is a bell curve whose "steepness" can be adjusted by changing the weight of the penalty.

- Methods
  - Penalty() - Instructs the algorithm to carry out the calculation.

- Output: A real number.
  - getPenalty() - Returns *Penalty*, representing a penalty factor.

- Usage
  - getPenalty() - Used by the evaluation module. The performance of a UAV towards the objective function is multiplied by the penalty factor to reduce fitness in the event that all constraints are not met.

A.3.3 UAV-Performance

The UAV-Performance library is responsible for estimating the UAV's capabilities in terms of altitude, speed, range, endurance, and field length.

Endurance

- Source: Developed by the author.

- Input: An aircraft and an atmosphere object.
  - *Aircraft*, representing the aircraft object to be analyzed.
  - *Atmosphere*, representing the atmosphere object containing the conditions at which the UAV endurance is to be computed.

- Procedure

  - The two dimensional equations of aircraft motion are solved over time intervals at a mean average velocity near the minimum drag speed of the UAV. The simulation is halted when the fuel is exhausted.

- Methods
  - Solve() - Instructs the algorithm to carry out the calculation.
- Output: A real number.
  - getEndurance() - Returns *Endurance*, the aircraft endurance performance in s.
- Usage
  - getEndurance() - Used by the evaluation module to when determining the aircraft's fitness rating.

FieldLength

- Source: Developed by the author.
- Input: An aircraft and an atmosphere object.
  - *Aircraft*, representing the aircraft object to be analyzed.
  - *Atmosphere*, representing the atmosphere object containing the conditions at which the UAV takeoff performance is to be computed.
- Procedure
  - The two dimensional aircraft equations of motion are solved over time steps from a velocity of zero, with maximum throttle until the aircraft achieves liftoff.
- Methods
  - Solve() - Instructs the algorithm to carry out the calculation.
- Output: A real number.
  - getTakeoffDistance() - Returns *TOdist*, the aircraft takeoff distance in m.

- Usage
  - getTakeoffDistance() - Used by the evaluation module when determining the aircraft's fitness rating.

Flight-Envelope

- Source: Developed by the author.
- Input: An aircraft object.
  - *Aircraft*, representing the aircraft object to be analyzed.
- Procedure
  - The two dimensional aircraft equations of motion are solved for the UAV's stall speed and maximum speed at altitude intervals by computing the stall speed directly, then allowing the UAV to accelerate over time until velociy becomes constant, representing the maximum speed at that particular altitude. The maximum altitude is found when the stall speed and maximum speed are equivalent.
- Methods
  - Solve() - Instructs the algorithm to carry out the calculation.
- Output: Real numbers.
  - getMaxAltitude() - Returns $hmax$, the UAV's absolute ceiling in m.
  - getMaxVelocity() - Returns $Vmax$, the UAV's maximum velocity in m/s.
  - getMaxVelocityAlt() - Returns $hVmax$, the altitude at which maximum velocity is achieved in m.
- Usage
  - getMaxAltitude() - Used by the evaluation module when determining the aircraft's fitness rating.
  - getMaxVelocity() - Used by the evaluation module when determining the aircraft's fitness rating.

- getMaxVelocityAlt() - Used when reporting the completed UAV's specifications.

Range

- Source: Developed by the author.
- Input: An aircraft and an atmosphere object.
  - *Aircraft*, representing the aircraft object to be analyzed.
  - *Atmosphere*, representing the atmospheric conditions at which the UAV's range is to be evaluated.
- Procedure
  - The two dimensional equations of aircraft motion are solved at time intervals at a mean average airspeed close to the maximum lift to drag ratio. The simulation halts and reports the range achieved when the fuel supply is exhausted.
- Methods
  - Solve() - Instructs the algorithm to carry out the calculation.
- Output: Real numbers.
  - getRange() - Returns R, the UAV's maximum range, m.
- Usage
  - getRange() - Used by the evaluation module when determining the aircraft's fitness rating.

Velocity

- Source: Developed by the author.
- Input: An aircraft and an atmosphere object and real numbers.
  - *Aircraft*, representing the aircraft object to be analyzed.
  - *Atmosphere*, representing the atmospheric conditions at which the UAV's range is to be evaluated.

  - *FuelPercent*, representing the fraction of the total fuel load carried by the aircraft.
  - *ThrottlePercent*, representing the throttle setting at which velocity is to be computed.

- Procedure
  - The two dimensional equations of aircraft motion are solved by allowing the aircraft to accelerate from stall speed until the velocity becomes constant, indicating the maximum velocity at the specified conditions and throttle setting.

- Methods
  - Solve() - Instructs the algorithm to carry out the calculation.

- Output: Real numbers.
  - getVelocity() - Returns V, the velocity achieved in m/s.

- Usage
  - getVelocity() - Used by the range and endurance simulations.

Thrust

- Source: Developed by the author.

- Input: An aircraft and atmosphere object and real numbers.
  - *Aircraft*, representing the aircraft object to be analyzed.
  - *Atmosphere*, representing the atmospheric conditions at which thrust.
  - *ThrottlePercent*, representing the throttle setting at which velocity is to be computed.

- Procedure
  - For piston and turboprop engines, thrust is computed as a function of velocity, atmosphere and throttle setting. For electric engines, thrust is computed as a function of throttle and velocity. Jet engine thrust is computed as a function of atmosphere. Engine performance is penalized for the effects of reduced air density at higher altitudes and temperatures as necessary.

- Methods
  - Solve(throttle, speed, atmosphere) - Instructs the algorithm to carry out the calculation for piston or turboprop engines (a function of speed, atmosphere, and throttle).
  - SolveJet(atmosphere) - Instructs the algorithm to carry out the calculation for jet engines (a function of throttle).
  - SolveElectric(throttle, speed) - Instructs the algorithm to carry out the calculation for electric engines (a function of speed and throttle).
- Output: Real numbers.
  - getThrust() - Returns T, the thrust in N.
- Usage
  - getThrust() - Used by all UAV performance modules.

A.3.4 UAV-GA

The UAV-GA library contains the genetic algorithm module.

GA

- Source: Developed by the author.
- Input: A design variable, display, and constraints object and integer values.
  - DV, representing the design variable object.
  - $Constraints$, representing the constraints object.
  - $GenerationNumber$, an integer representing the total number of generations to be computed.
  - $PopSize$, an integer representing the population size to be carried.
  - N, an integer representing the total number of design variables.
  - $Display$, representing the display interface object used for reporting progress.
- Procedure

  - A population of random design variable values are generated by a random number algorithm on the interval of [0,1]. These are used to set the design variable object containing the scaling factors to compute the corresponding design varible values. The fitness module is called with the design variable object. A fitness value is returned. Breeding probabilities are assigned based on the fitness ratings and breeding occurs between random pairs, weighted towards the fitter individuals. A new population is produced and the process is repeated until the number of generations previously set is reached.

- Methods
  - Solve() - Instructs the algorithm to begin.
- Output: A design variable object.
  - getOptDesignVars() - Returns *OptDV*, the set of optimum design variables found by the GA.
- Usage
  - getOptDV() - Used by main module to report algorithm results.

A.3.5 UAV-Main

The UAV-Main library contains the main program module responsible for calling the user interface, receiving input, calling the algorithm components, and reporting output. It also contains the design variable and constraints classes.

Main

- Source: Developed by the author.
- Input: User input including the following.
  - Case Number - User selected number representing the index for the input case to be run.
  - Popultion Size
  - Number of Generations

  - Span Lower Limit
  - Span Upper Limit
  - Taper Ratio Lower Limit
  - Taper Ratio Upper Limit
  - Aspect Ratio Lower Limit
  - Apsect Ratio Upper Limit
  - Fuselage Length Lower Limit
  - Fuselage Length Upper Limit

- Procedure
  - Calls the user interface module, collects and stores user input and sets up the design variables and constraints object accordingly, calls the database modules to retrieve the required database information, calls the genetic algorithm, and reports the results to the user.

- Methods
  - Main() - Executes the program.

- Output: A copy of the user input for reference, final aircraft specifications, and a convergence history graph is displayed on the user interface.
  - Endurance
  - Maximum Speed
  - Maximum Altitude
  - Engine
  - Power
  - Airfoil
  - Span
  - Fuselage Length
  - Taper Ratio
  - Aspect Ratio

  - Root Chord Length
  - Tip Chord Length
  - Planform Area
  - Empty Weight
  - Gross Weight
  - Drag Profile
  - Empennage Type
  - Engine Type
  - Landing Gear Type
  - Engine Location

- Usage
  - Is the main controlling module, calling all other functions as needed to carry out the procedure.

Design-Variables

- Source: Developed by the author.
- Input: None directly.
- Procedure
  - Contians the design variable limits and accepts design variable ratios on the interval [0,1] and calculates for retrieval the true design variable values.
- Methods
  - setAspectRange(ARR[]) - Sets an array of length 2 containing the allowable aspect ratio range.
  - setEngineRange(EIR[]) - Sets an array of length 2 containing the allowable engine index range.
  - setFoilRange(FIR) - Sets an array of length 2 containing the allowable airfoil index range.

  - setFuseRange(FLR) - Sets an array of length 2 containing the allowable fuselage length range.
  - setSpanRange(SR) - Sets an array of length 2 containing the allowable wing span range.
  - setTaperRange(TRR) - Sets an array of length 2 containing the allowable taper ratio range.
  - setDesignVarRatios(DVarray) - Sets an array of the design variable ratio values on [0,1]

- Output: Integer indexes and real numbers.
  - getAR() - Returns AR, representing the aspect ratio design variable value.
  - getEngineIndex() - Returns EI, an integer index pointing to the selected engine.
  - getFoilIndex() - Returns FI, an integer index pointing to the selected airfoil.
  - getFuseLength() - Returns $FuseLength$, representing the fuselage length design variable value.
  - getTaperRatio() - Returns TR, representing the taper ratio design variable value.
  - getWingSpan() - Returns $Span$, representing the span design variable value.
- Usage
  - Used by the fitness evaluation module to scale the decimal design variable ratios used by the GA to actual design variable values.

Constraints

- Source: Developed by the author.
- Input: None directly.
- Procedure: Accepts, stores, and distributes the design constraints.
  - Contians the constraints set by the user.
- Methods

  - setAvionicsWeight(AvionicsWeight) - Sets the weight of avionics systems to be carried by the UAV.
  - setCargoVolume(CargoVolume) - Sets the cargo volume to be carried by the UAV.
  - setCargoWeight(CargoWeight) - Sets the weight of the cargo to be carried by the UAV.
  - setCruiseAltitude(CruiseAltitude) - Sets the target altitude the UAV must attain.
  - setCruiseSpeed(CruiseSpeed) - Sets the target speed the UAV must attain.
  - setDesignLoadFactor(nmax) - Sets the limit load factor required for the UAV.
  - setFuelWeight(FuelWeight) - Sets the allowable limit on fuel load.
  - setTakeoffDistance(TOdist) - Sets the target maximum takeoff distance allowable for the UAV.
  - setTargetWeight(TargetWeight) - Sets the target maximum empty weight of the UAV.

- Output: Returns real numbers representing the variables listed above.
  - getAvionicsWeight() - Returns $AvionicsWeight$.
  - getCargoVolume() - Returns $CargoVolume$.
  - getCargoWeight() - Retruns $CargoWeight$.
  - getCruiseAltitude() - Returns $CruiseAltitude$.
  - getCruiseSpeed() - Returns $CruiseSpeed$.
  - getDesignLoadFactor() - Returns $nmax$.
  - getFuelWeight() - Returns $FuelWeight$.
  - getTakeoffDistance - Returns $TOdist$.
  - getTargetWeight - Returns $TargetWeight$.

- Usage
  - Used by the fitness evaluation function for computing penalty factors.

A.4 UAV-Part

The UAV-Part library handles the setup of each component of the aircraft can contain all data relevant to the part for distribution to other modules.

Airfoil

- Source: Developed by the author.
- Input: An Airfoil-Query object.
  - F_{Query}, representing the airfoil query object containing airfoil specification data.
- Procedure
  - Gets all relevant airfoil data pertaining to the particular airfoil under study from the airfoil query object and stores the data for distribution.
- Methods
  - Airfoil()
- Output: Real numbers.
  - getCl0() - Returns $Cl0$, representing the zero angle of attack lift coefficient.
  - getClalpha() - Returns Cla, representing the lift coefficient to angle of attack ratio.
  - getCm0() - Returns $Cm0$, representing the airfoil moment coefficient.
  - getClmax() - Returns $Clmax$, representing the airfoil maximum lift coefficient.
  - getLDmax() - Returns $LDmax$, representing the airfoil maximum lift to drag ratio.
  - getRefPerimeter() - Returns $RefPerimeter$, representing the perimeter to chord length ratio of the airfoil.
  - getRefTipAera() - Returns $RefTipArea$, representing the airfoil area to chord ratio.
- Usage

- getCl0() - Used to build the aircraft lift model.
- getClalpha() - Used to build the aircraft lift model.
- getCm0() - Used in tail plane sizing by the empennage part.
- getClmax() - Used to build the lift model.
- getLDmax() - Used for a comparison reference.
- getRefPerimeter() - Used for computing wing wetted area by the wing part module.
- getRefTipAera() - Used for computing wing tip wetted area by the wing part module.

Engine

- Source: Developed by the author.
- Input: An Engine-Query object.
  - Eng_{Query}, representing the engine query object containing engine specification data.
- Procedure
  - Gets all relevant engine data pertaining to the particular engine under study from the engine query object and stores the data for distribution.
- Methods
  - Airfoil()
- Output: Real numbers.
  - getFlatRateAltitude() - Returns *FlatRateAlt*, representing the altitude beyond which the engine loses power.
  - getLength() - Returns *Length*, representing the engine length in *mm.*
  - getWidth() - Returns *Width*, representing the engine width in *mm.*
  - getHeight() - Returns *Height*, representing the engine height in *mm.*

- getPowerSL() - Returns PSL, representing a piston or turboprop power produced at sea level in W.
- getThrustSL() - Returns $ThrustSL$k, representing jet thrust produced at sea level in N.
- getSFC() - Returns SFC, representing the specific fuel consumption for piston or turboprop engines in $lbm/hp/hr$.
- getTSFC() - Returns $TSFC$, representing the thrust specific fuel consumption for jet engines in $lbm/lbf/hr$.
- getWeight() - Returns $Weight$, representing the engine weight in N.

- Usage
  - getFlatRateAltitude() - Used for performance calculations. Instructs the thrust module to not penalize power produced by air breathing engines below the flat rate altitude.
  - getLength() - Used by the fuselage part module for sizing and by the empennage module for center of gravity calculations.
  - getWidth() - Used by the fuselage part module for sizing.
  - getHeight() - Used by the fuselage part module for sizing.
  - getPowerSL() - Used by the thrust module.
  - getThrustSL() - Used by the thrust module.
  - getSFC() - Used by the endurance and range performance modules.
  - getTSFC() - Used by the endurance and range performance modules.
  - getWeight() - Used by the weight analysis module for aircraft empty weight estimation and the empennage module for center of gravity calculations.

Fuselage

- Source: Developed by the author.
- Input: An Engine object and real numbers.
  - *Engine*, representing the engine part object.

- $CargoVol$, representing the cargo volume to be stored in the fuselage.
- $FuseLength$, representing the fuselage length design variable.

- Procedure
  - Builds a fuselage geometry such that the engine and cargo are accommodated, and smoothly tapers toward the tail. Layout is built according to the method call of Canard, Conventional, or TwinBoom. Computes frontal area and total wetted area and mass centroid position.

- Methods
  - Canard(Engine, CargoVol, FuseLength)
  - Conventional(Engine, CargoVol, Fuselength)
  - TwinBoom(Engine, CargoVol, Fuselength)

- Output: Real numbers.
  - getBayLength() - Returns $BayLength$, representing the length of the cargo bay in mm.
  - getBayWidth() - Returns $BayWidth$, representing the cargo bay width in mm.
  - getBayHeight() - Returns $BayHeight$, representing the cargo bay height in mm.
  - getBoomLength() - Returns $BoomLength$, representing the length tapered portion of the fuselage or the length of the twin boom portion for twin boom layouts in mm.
  - getCentroid() - Returns $FuseCentroid$, representing the geometric centroid of fuselage.
  - getFrontalArea() - Returns $FrontalArea$, representing the frontal area of the fuselage in mm^2.
  - getWettedArea() - Returns $Swet$, representing the fuselage total wetted area in mm^2.
  - getFuseLength() - Returns $Lfuse$, representing the total fuselage length.

- Usage
  - getBayLength() - Used by the empennage module for center of gravity estimation.
  - getBayWidth() - Can be displayed in algorithm output for reference.
  - getBayHeight() - Can be displayed in algorithm output for reference.
  - getBoomLength() - Can be displayed in algorithm output for reference.
  - getCentroid() - Used by the empennage module for center of gravity estimation.
  - getFrontalArea() - Used by the drag analysis module for parasite drag estimation.
  - getWettedArea() - Used by the drag analysis module for parasite drag estimation.
  - getFuseLength() - Used by the drag analysis module for parasite drag estimation and by the fuselage module for empennage sizing.

Wing

- Source: Developed by the author.
- Input: An airfoil object and real numbers.
  - $Airfoil$, representing the airfoil part.
  - $WingSpan$, representing the wing span design variable.
  - TR, representing the taper ratio design variable.
  - AR, representing the aspect ratio design variable.
- Procedure
  - Calls the Prandtl lifting line module to compute three dimensional wing aerodynamic properties, builds the wing gemoetry and computes wetted and frontal areas, and stores the wing lift model and geometry for distribution as needed.
- Methods
  - Wing()

- Output: Real numbers.
  - getAR() - Returns AR, representing the wing aspect ratio.
  - getTR() - Returns TR, representing the wing taper ratio.
  - getSpan() - Returns $Span$, representing the wing span length in m.
  - getPlanformArea() - Returns $PlanformArea$, the wing planform area in m^2.
  - getCl03d() - Returns $Cl03d$, the lift at zero angle of attack for the three dimensional wing.
  - getClalpha3d() - Returns $Cla3d$, representing the lift coefficient to angle of attack ratio attack for the three dimensional wing.
  - getCm0() - Returns the wing moment coefficient.
  - gete() - Returns the oswald efficiecy factor.
  - getk() - Returns the induced drag constant.
  - getFrontalArea() - Returns $FrontalArea$, representing the frontal area of the wing in m^2.
  - getWettedArea() - Returns $Swet$, representing the wing total wetted area in m^2.
- Usage
  - getAR() - Stored for output and reference.
  - getTR() - Stored for output and reference.
  - getSpan() - Stored for output and reference.
  - getPlanformArea() - Used by the UAV-performance library for aircraft performance estimation.
  - getCl03d() - Used by the UAV-performance library for aircraft performance estimation.
  - getClalpha3d() - Used by the UAV-performance library for aircraft performance estimation.
  - getCm0() - Used by the empennage part for empennage sizing and static stability.

    - gete() - Stored for output and reference.
    - getk() - Used by the UAV-performance library for aircraft performance estimation.
    - getFrontalArea() - Used in the drag estimation module.
    - getWettedArea() - Used in the drag estimation module.

Empennage

- Source: Developed by the author.
- Input: A wing, fuselage, and engine object.
    - *Wing*, representing the wing part.
    - *Fuselage*, representing the fuselage part.
    - *Engine*, representing the engine part.
- Procedure
    - Assembles the parts in their proper location and uses static stability equations to size the empennage such that a static stability margin of 0.05 is enforced.
- Methods
    - Empennage()
- Output: Real numbers.
    - getSht() - Returns Sht, representing the planform area of the horizontal tail in m^2.
    - getSvt() - Returns Svt, representing the planform area of the vertical tail.
    - getSv() - Returns Sv, representing the planform area of the empennage for a v-tail configuration in m^2.
    - getWettedArea() - Returns $Swet$, representing the wetted area of the empennage in m^2.
- Usage
    - getSht() - Used for output and reference.

  - getSvt() - Used for output and reference.
  - getSv() - Used for output and reference.
  - getWettedArea() - Used by the drag module for parasite drag estimation.

Aircraft

- Source: Developed by the author.
- Input: A wing, fuselage, engine, empennage, design variables, and constraints object.
  - *Wing*, representing the wing part.
  - *Fuselage*, representing the fuselage part.
  - *Engine*, representing the engine part.
  - *Empennage*, representing the empennage part
  - *Constraints*, representing the constraints object.
  - *DesignVars*, representing the design variable object.
- Procedure
  - Contains and stores the aircraft parts for retrieval by the performance modules. Calls the drag and weight analysis modules and stores the parasite drag and weight estimates.
- Methods
  - Empennage()
- Output: Real numbers.
  - getCd0() - Returns $Cd0$, representing the parasite drag coefficient.
  - getEmptyWeight() - Returns $Wempty$, representing the aircraft empty weight in N.
  - getGrossWeight() - Returns $Wgross$, representing the aircraft gross weight in N.
  - getFuelWeight() - Returns $Wfuel$, representing the aircraft fuel weight in N.

  - The aircraft parts are declared as public, making any "get" method contained in the other parts accessible.

- Usage
  - getCd0() - Used in the drag model for all performance calculations.
  - getEmptyWeight() - Used in all performance modules and in the evaluation module for fitness rating.
  - getGrossWeight() - Used in all performance modules.
  - getFuelWeight() - Used in all performance modules.

A.4.1 UAV-Interface

The UAV-Interface library handles user input and output forms.

Display

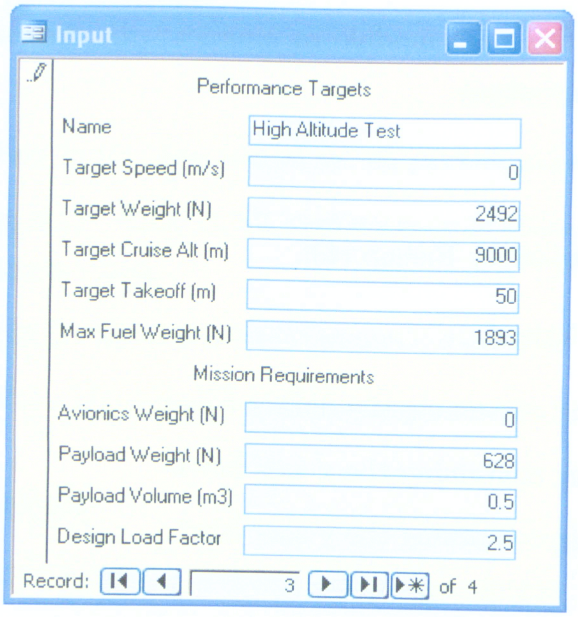
- Source: Developed by Bishop [43]. Note: Only the modules used in UAVOpt are discussed.
- Input: None directly.
- Procedure
  - Displays a graphical window containing input forms and algorithm output.
- Methods
  - println(String) - Displays a string to the user.
  - prompt(String) - displays a string followed by a prompt to the user.
  - ready() - Displays a "ready" button that executes the program when clicked.
- Output: Real numbers, integers, or strings.
  - getString() - Returns *String*, containing text user input.
  - getInt() - Returns *Int*, containing integer user input.
  - getDouble() - Returns *Double*, containing real number use input.

- Usage
  - Used by the main module to interact with the user by gathering input and displaying output.

Graph

- Source: Developed by Bishop [43].
- Input: String values.
  - g, a string containing the data set name.
  - x, a string containing the x axis label.
  - y, a string containing the y axis label.
- Procedure
  - Displays a graphical window containing input forms and algorithm output.
- Methods
  - add(x, y) - Adds a new set of co-ordinates.
  - nextGraph() - Adds another data set.
  - setTitle(s) - Sets the graph title.
  - showGraph() - Displays the graph to the user.
- Output: On display.
- Usage
  - Used by the GA to track convergence and display convergence history.

A.5 Sample Input and Output


Input

Performance Targets

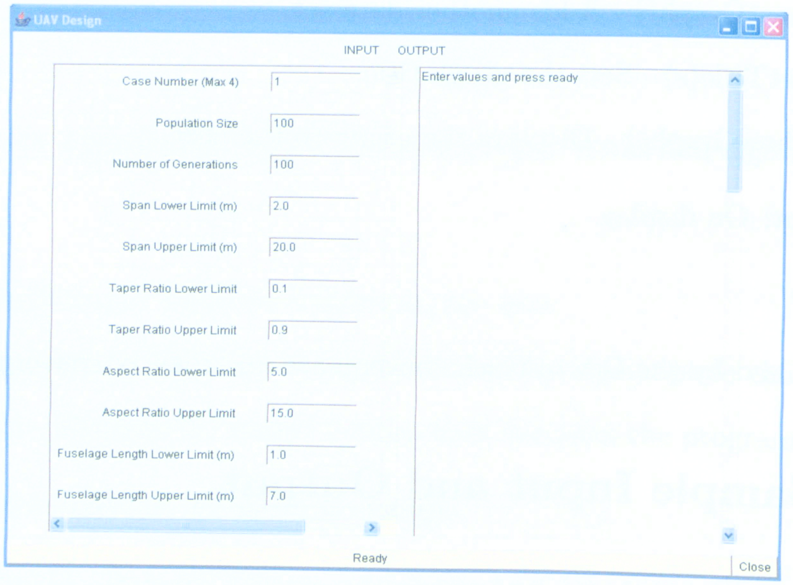
Name	High Altitude Test
Target Speed (m/s)	0
Target Weight (N)	2492
Target Cruise Alt (m)	9000
Target Takeoff (m)	50
Max Fuel Weight (N)	1893

Mission Requirements

Avionics Weight (N)	0
Payload Weight (N)	628
Payload Volume (m3)	0.5
Design Load Factor	2.5

Record: 3 of 4

Figure A.4: Sample Input - The input database form. The design goals and performance requirements are entered here.


UAV Design

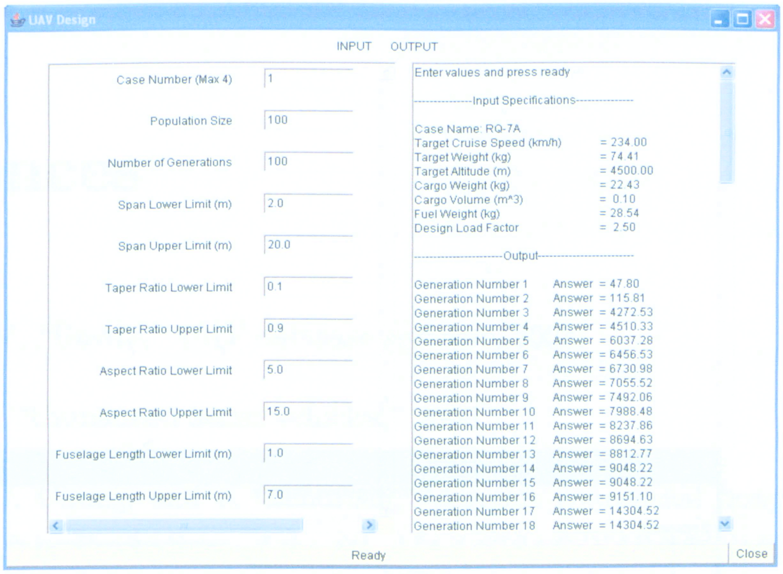
INPUT OUTPUT

Case Number (Max 4)	1
Population Size	100
Number of Generations	100
Span Lower Limit (m)	2.0
Span Upper Limit (m)	20.0
Taper Ratio Lower Limit	0.1
Taper Ratio Upper Limit	0.9
Aspect Ratio Lower Limit	5.0
Aspect Ratio Upper Limit	15.0
Fuselage Length Lower Limit (m)	1.0
Fuselage Length Upper Limit (m)	7.0

```
Enter values and press ready
```

Ready Close

Figure A.5: Sample Input - The UAVOpt main menu screen. The GA parameters and design variable limits are entered here.


UAV Design

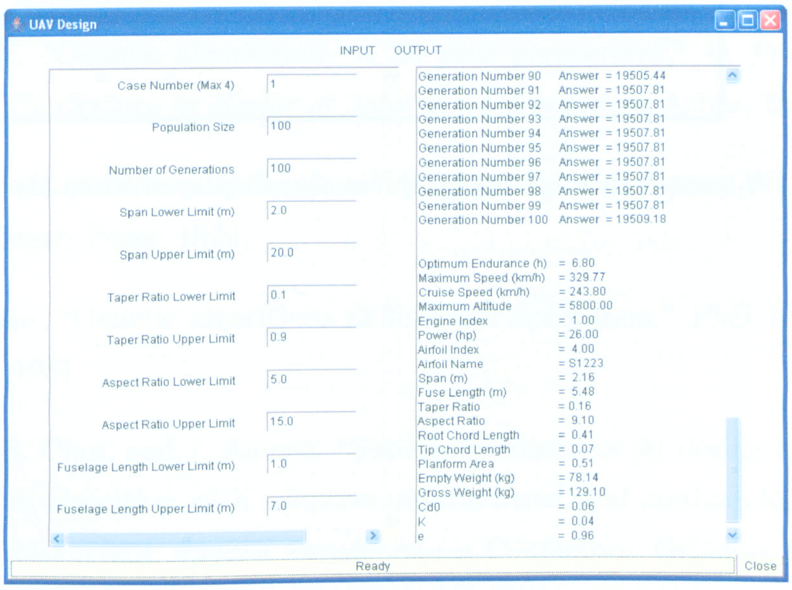
INPUT OUTPUT

Case Number (Max 4)	1
Population Size	100
Number of Generations	100
Span Lower Limit (m)	2.0
Span Upper Limit (m)	20.0
Taper Ratio Lower Limit	0.1
Taper Ratio Upper Limit	0.9
Aspect Ratio Lower Limit	5.0
Aspect Ratio Upper Limit	15.0
Fuselage Length Lower Limit (m)	1.0
Fuselage Length Upper Limit (m)	7.0

```
Enter values and press ready

--------------Input Specifications--------------

Case Name: RQ-7A
Target Cruise Speed (km/h)        = 234.00
Target Weight (kg)                = 74.41
Target Altitude (m)               = 4500.00
Cargo Weight (kg)                 = 22.43
Cargo Volume (m^3)                =  0.10
Fuel Weight (kg)                  = 28.54
Design Load Factor                =  2.50

--------------------Output----------------------

Generation Number 1      Answer  = 47.80
Generation Number 2      Answer  = 115.81
Generation Number 3      Answer  = 4272.53
Generation Number 4      Answer  = 4510.33
Generation Number 5      Answer  = 6037.28
Generation Number 6      Answer  = 6456.53
Generation Number 7      Answer  = 6730.98
Generation Number 8      Answer  = 7055.52
Generation Number 9      Answer  = 7492.06
Generation Number 10     Answer  = 7988.48
Generation Number 11     Answer  = 8237.86
Generation Number 12     Answer  = 8694.63
Generation Number 13     Answer  = 8812.77
Generation Number 14     Answer  = 9048.22
Generation Number 15     Answer  = 9048.22
Generation Number 16     Answer  = 9151.10
Generation Number 17     Answer  = 14304.52
Generation Number 18     Answer  = 14304.52
```

Ready Close

Figure A.6: Sample Output - Immediately, the specifications defined in the input form are displayed for reference and the fitness level of the best design in the GA population is displayed as it is calculated for each generation.


UAV Design

INPUT OUTPUT

Case Number (Max 4)	1
Population Size	100
Number of Generations	100
Span Lower Limit (m)	2.0
Span Upper Limit (m)	20.0
Taper Ratio Lower Limit	0.1
Taper Ratio Upper Limit	0.9
Aspect Ratio Lower Limit	5.0
Aspect Ratio Upper Limit	15.0
Fuselage Length Lower Limit (m)	1.0
Fuselage Length Upper Limit (m)	7.0

```
Generation Number 90     Answer  = 19505.44
Generation Number 91     Answer  = 19507.81
Generation Number 92     Answer  = 19507.81
Generation Number 93     Answer  = 19507.81
Generation Number 94     Answer  = 19507.81
Generation Number 95     Answer  = 19507.81
Generation Number 96     Answer  = 19507.81
Generation Number 97     Answer  = 19507.81
Generation Number 98     Answer  = 19507.81
Generation Number 99     Answer  = 19507.81
Generation Number 100    Answer  = 19509.18


Optimum Endurance (h)    =  6.80
Maximum Speed (km/h)     = 329.77
Cruise Speed (km/h)      = 243.80
Maximum Altitude         = 5800.00
Engine Index             =  1.00
Power (hp)               = 26.00
Airfoil Index            =  4.00
Airfoil Name             = S1223
Span (m)                 =  2.16
Fuse Length (m)          =  5.48
Taper Ratio              = 0.16
Aspect Ratio             =  9.10
Root Chord Length        =  0.41
Tip Chord Length         =  0.07
Planform Area            =  0.51
Empty Weight (kg)        = 78.14
Gross Weight (kg)        = 129.10
Cd0                      =  0.06
K                        =  0.04
e                        =  0.96
```

Ready Close

Figure A.7: When the algorithm finishes, a table displays the specifications of the optimal design produced by the algorithm.

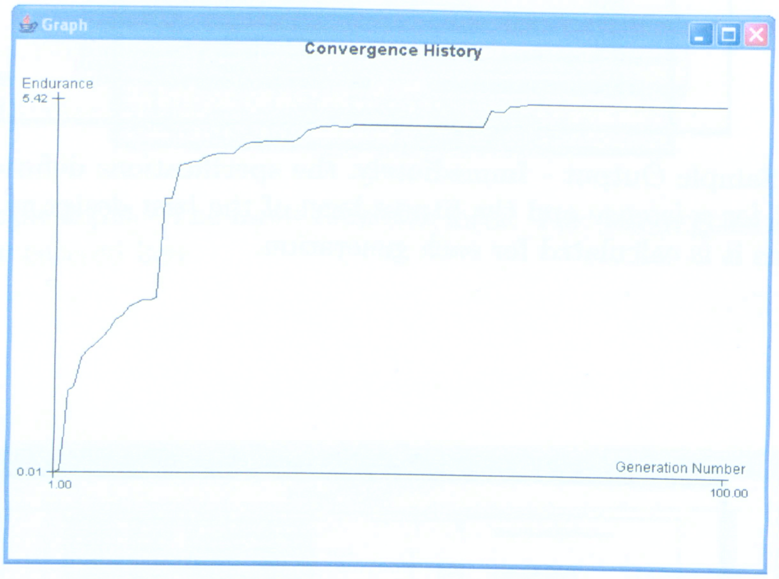


Figure A.8: A convergence history graph is also displayed when the algorithm finishes.

References

[1] Smart UAV, "Uavdb," *UAV database software*, 2003.

[2] G. Goebel, "Unmanned aerial vehicles," 2005.

[3] R. Perez, J. Chung, and K. Behdinan, "Aircraft conceptual design using genetic algorithms," in *Proceedings of the 8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Long Beach, CA, Sept. 1995.

[4] D. Whitley, "Genetic algorithms and evolutionary computing," *Van Nostrand's Scientific Encyclopedia 2002*, 2002.

[5] D. Whitley, "An overview of evolutionary algorithms," *Journal of Information and Software Technology*, 2001.

[6] K. D. Jong, "Genetic algorithms: A 30 year perspective," in *Proceedings of the Festschrift Conference in Honor of John H. Holland*, Ann Arbor, MI, May 1999.

[7] J. Holland, *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: University of Michigan Press, 1975.

[8] A. D. Bethke, "Genetic algorithms as function optimizers," *PhD Thesis, Michigan University*, 1981.

[9] H. Chung, S. Choi, and J. Alonso, "Supersonic business jet design using knowledge based genetic algorithm with adaptive unstructured grid methodology," in *Proceedings of the 21st AIAA Applied Aerodynamics Conference*, Orlando, FL, June 2003.

[10] S. Letournou, F. Famili, and S. Matwin, "Data mining for prediction of aircraft component replacement," *IEEE Intelligent Systems Special Issue on Data Mining*, pp. 59–66, Dec. 1999.

[11] J. Crawford and F. Crawford, "Data mining in a scientific environment," in *Proceedings of the AUUG 96 and Aisia Pacific World Wide Web 2nd Joint Conference*, Melbourne, Australia, Sept. 1996.

[12] S. Obayashi, "Multidisciplinary design optimization of aircraft wing planform based on evolutionary algorithms," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, San Diego, CA, 1998.

[13] D. Raymer, "Enhancing aircraft design using multidisciplinary optimization," *PhD Thesis, Royal Institute of Technology*, 2002.

[14] I. Witten and F. Eibe, *Data Mining.* San Francisco, CA: Morgan Kaufmann Publishers, 1999.

[15] M. DeGarmo and G. Nelson, "Prospective unmanned aerial vehicle operations in the future national aispace system," *The Mitre Corporation for Advanced Aviation Development*, 2004.

[16] C. Jones, "Unmanned aerial vehicls (uavs): An assessment of historical opertions and future possibilities," *Research Department, Air Command and Staff College, Air University*, Mar. 1997.

[17] J. Garamone, "From the u.s. civil war to afghanistan: A short history of uavs," *Defend America*, Apr. 2002.

[18] H. Smith, "U-99 uninhabited tactical aircraft preliminary systems design," *Aircraft Engineering and Aerospace Technology*, 2001.

[19] R. M.-V. adn Carlos Hernandez, "Preliminary design of a low speed, long endurance remote piloted vehicles (rpv) for civil applications," *Elsevier Science, Aircraft Design 2*, 1999.

[20] J. Bauer, "Nasa: Continuing uav development," *UAVs: A Vision of the Future*, pp. 7–8, 2002.

[21] UVS Canada, "UAVs in Canada - 2004 Update," 2004.

[22] T. R. of MDO Within Aerospace Design and P. T. an MDO Capability Through European Collaboration, "Multidisciplinary techniques for commercial aircraft systems design," in *Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, St. Louis, MO, Sept. 1998.

[23] J. P. Giesign and J.-F. M. Barthelemy, "A Summary of Industry MDO Applications and Needs," in *Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, St. Louis, MO, Sept. 1998.

[24] J. Markish and K. Willcox, "Multidisciplinary techniques for commercial aircraft systems design," in *Proceedings of the 9th Symposium on Multidisciplinary Analysis and Optimization*, Atlanta, GA, Sept. 2002.

[25] T. A. Zang and L. L. Green, "Multidisciplinary design optimization techniques: Implications and opportunities for fluid dynamic research," in *Proceedings of the 30th Fluid Dynamics Conference*, Orlando, FL, June 1999.

[26] S.-H. Nah, "Desaid - the development of an expert system for aircraft initial design," *PhD Thesis, Cranfield Institute of Technology*, 2002.

[27] D. W. E. Rentema, F. W. Jansen, and E. W. Torenbeek, "The application of ai and geometric modelling techniques in conceptual aircraft design," in *AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Louis, MO, Sept. 1998, pp. 10–31.

[28] N. Ramakrishnan and A. Y. Grama, "Mining scientific data," 2000.

[29] Z. Liu, W. W. Chu, A. Huang, C. Folk, and C.-M. Ho, "Mining sequence patterns form wind tunnel experimental data for flight control," in *Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Hong Kong, China, Apr. 2001.

[30] M. H. Dunham, *Data Mining: Introductory and Advanced Topics.* New York, NY: Prentice Hall, 2003.

[31] J. Quinlan, "A case study for machine learning," in *16th Australian Computer Science Conference*, Brisbane, Australia, 1993.

[32] P. Jackson, *Introduction to Expert Systems.* Essex, UK: Addison Wesley, 1998.

[33] P. Butterworth-Hayes, *UAVs: A Vision of the Future.* London, UK: Newsdesk Communications Ltd, 2003.

[34] L. Breiman, J. H. Freidman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees.* Wadsworth, CA: Chapman and Hall, 1984.

[35] R. Timofeev, "Classification and regression trees (cart) theory and applications," *Master Thesis, Humboldt University*, 2004.

[36] J. Quinlan, *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann, 1992.

[37] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Learning.* Boston, MA: Addison Wesley Longman Inc., 1989.

[38] L. Y. an dLeung Kwong-Sak and M. S. K. Tony, "Evolutionary drug scheduling model for cancer chemothrapy," in *Proceedings of the 2004 Genetic and Evolutionary Computation Conference*, Seattle, WA, June 2004.

[39] E. Pires, M. Jos, and P. Oliveira, "Robot trajectory planner using multi-objective genetic algorithms," in *Proceedings of the 2004 Genetic and Evolutionary Computation Conference*, Seattle, WA, June 2004.

[40] J. Anderson, *Fundamentals of Aerodynamics.* New York, NY: McGraw-Hill Higher Education, 2001.

[41] S. A. B. et al., *Introduction to Aeronautics: A Design Perspective.* Reston, VA: AIAA, 2004.

[42] Laminar Research, "X plane version 8.15," *Flight Simulator Software*, 2005.

[43] J. Bishop and N. Bishop, *Java Gently for Scientists and Engineers.* Essex, UK: Addison Wesley, 2000.