

Investigation of Methods for Increasing the Energy Efficiency on Unmanned Aerial Vehicles (UAVs)

A THESIS SUBMITTED TO
THE FACULTY OF BUILT ENVIRONMENT AND ENGINEERING
OF QUEENSLAND UNIVERSITY OF TECHNOLOGY
IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING



Jane Yu-Chun Hung

Faculty of Built Environment and Engineering
Queensland University of Technology

12 August, 2011

Copyright in Relation to This Thesis

© Copyright 2011 by Jane Yu-Chun Hung. All rights reserved.

Statement of Original Authorship

The work contained in this thesis has not been previously submitted to meet requirements for an award at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signature:

Date:

Abstract

In recent years, development of Unmanned Aerial Vehicles (UAV) has become a significant growing segment of the global aviation industry. These vehicles are developed with the intention of operating in regions where the presence of onboard human pilots is either too risky or unnecessary. Their popularity with both the military and civilian sectors have seen the use of UAVs in a diverse range of applications, from reconnaissance and surveillance tasks for the military, to civilian uses such as aid relief and monitoring tasks.

Efficient energy utilisation on an UAV is essential to its functioning, often to achieve the operational goals of range, endurance and other specific mission requirements. Due to the limitations of the space available and the mass budget on the UAV, it is often a delicate balance between the onboard energy available (i.e. fuel) and achieving the operational goals.

This thesis presents an investigation of methods for increasing the energy efficiency on UAVs. One method is via the development of a Mission Waypoint Optimisation (MWO) procedure for a small fixed-wing UAV, focusing on improving the onboard fuel economy. MWO deals with a pre-specified set of waypoints by modifying the given waypoints within certain limits to achieve its optimisation objectives of minimising/maximising specific parameters. A simulation model of a UAV was developed in the MATLAB Simulink environment, utilising the AeroSim Blockset and the in-built Aerosonde UAV block and its parameters. This simulation model was separately integrated with a multi-objective Evolutionary Algorithm (MOEA) optimiser and a Sequential Quadratic Programming (SQP) solver to perform single-objective and multi-objective optimisation procedures of a set of real-world waypoints in order to minimise the onboard fuel consumption. The results of both procedures show potential in reducing fuel consumption on a UAV in a flight mission.

Additionally, a parallel Hybrid-Electric Propulsion System (HEPS) on a small fixed-wing UAV incorporating an Ideal Operating Line (IOL) control strategy was developed. An IOL analysis of an Aerosonde engine was performed, and the most efficient (i.e. provides greatest torque output at the least fuel consumption) points of operation for this engine was determined. Simulation models of the components in a HEPS were designed and constructed in the MATLAB Simulink environment. It was demonstrated through simulation that an UAV with the current HEPS configuration was capable of achieving a fuel saving of 6.5%, compared to the ICE-only configuration. These components form the basis for the development of a complete simulation model of a Hybrid-Electric UAV (HEUAV).

Acknowledgments

I would like to express my sincerest gratitude to my Principal Supervisor, Dr. Felipe Gonzalez, for the patient help and constant support throughout the entire duration of my candidature, and for showing me a light at the end of the tunnel, even when I could not see a way out. A special thanks go to Richard Glassock for his assistance in understand the inner workings of an aircraft propulsion system. Also, I would like to express my thanks to the fellow staff and researchers at QUT and at ARCAA for their help and camaraderie in the last few years of my studies.

No thanks would ever be enough for my parents, Su and Meng, and my brothers, Kevin and Harry, for their unwavering support throughout my life. They always knew what I needed to hear - whether it was words of encouragement, a shoulder to cry on, or a stern talk-to - when the light at the end of the tunnel seemed dim. They have been the rock in my life.

Lastly, I would like to thank my close circle of friends for just being there for me while I completed my studies.

Table of Contents

Abstract	iii
Acknowledgments	v
Nomenclature	xiii
List of Figures	xxi
List of Tables	xxv
1 Introduction	1
1.1 Background and Motivation	1
1.1.1 Mission Waypoint Optimisation	3
1.1.2 Hybrid-Electric Propulsion System	3
1.2 Research Objectives	6
1.3 Research Contribution	7
1.4 Research Methodology	7
1.5 Outline of Thesis	8
2 Review of Simulation Optimisation Techniques	11
2.1 Introduction to Optimisation	11
2.2 Simulation Optimisation	13
2.3 Sequential Quadratic Programming	16
2.3.1 MATLAB SQP Solver	19

2.4	Stochastic Approximation Methods	20
2.5	Response Surface Methodology	26
2.6	Simulated Annealing	30
2.7	Evolutionary Algorithms	32
2.7.1	HAPMOEA Optimiser	35
2.8	Summary	38
3	UAV Simulation Environment	41
3.1	Introduction	41
3.2	Simulation Software	41
3.3	UAV Simulation Model	43
3.3.1	Flight Planner Module	43
3.3.2	Aircraft Control Module	48
3.3.3	Test Scenario and Results	51
3.4	Mission Scenario	53
3.5	Conclusions	55
4	Single-Objective Mission Waypoint Optimisation	57
4.1	Introduction	57
4.2	Optimisation Problem Definition	58
4.3	Design Variables	59
4.4	Fitness Function	59
4.5	Upper and Lower Bounds of a Waypoint	60
4.5.1	Waypoint Bounds for Mission Scenario 1	60
4.5.2	Waypoint Bounds for Mission Scenario 2	64
4.6	Physical Constraints	65
4.7	Mathematical Formulation of SOMWO Problem	65
4.8	Optimiser Setup	66

4.8.1	HAPMOEA Optimiser Setup	66
4.8.2	MATLAB SQP Solver Setup	68
4.9	Optimisation Results and Analysis	69
4.9.1	Mission Scenario 1	70
4.9.2	Mission Scenario 2	80
4.10	Conclusions	87
5	Multi-Objective Mission Waypoint Optimisation	89
5.1	Introduction	89
5.2	Optimisation Problem Definition	90
5.3	Design Variables	90
5.4	Fitness Functions	90
5.4.1	Fitness Functions for Mission Scenario 1	90
5.4.2	Fitness Functions for Mission Scenario 2	91
5.5	Upper and Lower Bounds of a Waypoint	92
5.6	Physical Constraints	93
5.6.1	Mathematical Formulation of MOMWO Problem	93
5.6.2	Optimiser Setup	94
5.7	Optimisation Results and Analysis	95
5.7.1	Mission Scenario 1	96
5.7.2	Mission Scenario 2	98
5.8	Conclusions	101
6	Ideal Operating Line Analysis of Aerosonde ICE	103
6.1	Introduction	103
6.2	Overview of IOL Analysis	104
6.3	Ideal Operating Line Analysis of Aerosonde ICE	105
6.3.1	ICE Torque Calculations	105

6.3.2	ICE BSFC Calculations	107
6.3.3	Engine Map	107
6.3.4	Determining the IOL	108
6.4	Summary	113
7	Modelling of Hybrid-Electric Propulsion Subsystem Components	115
7.1	Introduction	115
7.2	HEPS Powertrain	117
7.2.1	Piston Engine Model	118
7.2.2	Electric Motor Model	118
7.2.3	Generator Model	121
7.2.4	Battery Model	122
7.2.5	CVT Dynamics Model	124
7.2.6	Charge Battery Now (CBN) Module	127
7.2.7	Powertrain Output Allocation (POA) Module	130
7.3	HEPS IOL Controller	131
7.3.1	Operating Mode (OM) Module	133
7.3.2	Power Demand (PD) Module	136
7.3.3	Engine Operation (EO) Module	138
7.3.4	Engine Throttle Command (ETC) Module	140
7.3.5	Rate of Change of Ratio Command (RCRC) Module	141
7.3.6	Torque Difference Calculation (TDC) Module	143
7.3.7	Integration of IOL Controller Components	146
7.4	Integrated HEPS Model	148
7.5	Summary	155
8	Conclusions	157
8.1	Summary of the Research	157

8.2	Future Work	159
	References	178
A	Aircraft Configuration for the Aerosonde UAV	179
B	UAV Simulation Model: Flight Planner Module MATLAB Code	185
C	HAPMOEA Code for One-Objective MWO: Excerpts	189
	C.1 optimisation.parameters: An Example	190
	C.2 HierarEA.cpp	190
	C.3 Analyser.cpp	197
D	MATLAB SQP Solver Code for Single-Objective MWO	201
	D.1 <i>fminconOpti</i> Function for Mission Scenario 1	202
	D.2 <i>fminconOpti</i> Function for Mission Scenario 2	205
E	HAPMOEA Code for Two-Objective MWO: Excerpts	211
	E.1 optimisation.parameters: An Example	212
	E.2 HierarEA.cpp	212
	E.3 Analyser.cpp	219
	E.3.1 Mission Scenario 1	219
	E.3.2 Mission Scenario 2	222
F	MATLAB SQP Solver Code for Two-Objective MWO	227
	F.1 Mission Scenario 1	228
	F.1.1 <i>twoOptiPareto</i> Function	228
	F.2 Mission Scenario 2	232
	F.2.1 <i>twoOptiPareto</i> Function	232
	F.2.2 Updated <i>NavigateWaypoints</i> Function	237
	F.2.3 <i>distToHazard</i> Function	238

G	Aerosonde ICE - Open Throttle Calculations	241
H	Plettenberg HP220/25 Motor Data	247
I	MATLAB Code for HEPS Powertrain Components	249
I.1	<i>Motor</i> Function	250
I.2	<i>Generator</i> Function	250
I.3	<i>Battery</i> Function	251
I.4	<i>Powertrain Output Allocation</i> Function	252
I.5	<i>ChargeBattNow</i> Function	253
J	MATLAB Code for HEPS IOL Controller Components	257
J.1	<i>OpMode</i> Function	258
J.2	<i>PowerDemandAlloc</i> Function	258
J.3	<i>EngOp</i> Function	259
J.4	<i>EngThrCommand</i> Function	260
J.5	<i>RCRCommand</i> Function	260
J.6	<i>TorqueDiffCalc</i> Function	261

Nomenclature

Abbreviations

ACM	Aircraft Control Module
ANOVA	Analysis of variance
ARCAA	Australian Research Centre for Aerospace Automation
BFGS	Broyden-Fletcher-Goldfarb-Shanno; a quasi-Newton approximation method
BOT	Bearings-only tracking
BRS	Blind random search
BSFC	Brake specific fuel consumption
CBN	Charge Battery Now; a HEPS Powertrain module
CFD	Computational fluid dynamics
CG	Centre of gravity
COTS	Commercially off-the-shelf
CPP	Cooperative path planning
CPU	Central processing unit
CVT	Continuously variable transmission
DCNLP	Direct collocation with nonlinear programming
DEA	Distributed evolutionary algorithm
DOF	Degrees of freedom
DV	Decision variable
EA	Evolutionary algorithm
EM	Electric motor

EP	Evolutionary programming
ES	Evolution strategies
ETC	Engine Throttle Command; a HEPS IOL Controller module
FC	Fuel consumption
FDSA	Finite difference stochastic approximation
FPM	Flight Planner Module
G&C	Guidance and control
GA	Genetic algorithm
GP	Genetic programming
HAPMOEA	Hierarchical Asynchronous Parallel Multi-Objective Evolutionary Algorithm
HEPS	Hybrid-Electric Propulsion System
HEUAV	Hybrid-Electric Unmanned Aerial Vehicle
ICE	Internal Combustion Engine
IOL	Ideal operating line
KW	Kiefer-Wolfowitz; a stochastic approximation algorithm
LB	Lower bound
LCS	Learning classifier system
Li-Po	Lithium-polymer; battery
LUT	Look-up table
MAP	Manifold pressure
MFP	Mission flight planning
MOEA	Multi-objective evolutionary algorithm
MOMWO	Multi-objective mission waypoint optimisation
MOO	Multi-objective optimisation
MPC	Model prediction control
MWO	Mission waypoint optimisation
NAS	National Airspace System

NLP	Nonlinear programming
NTG	Nonlinear Trajectory Generation
OM	Operating Mode; a HEPS IOL Controller module
OTP	Observer trajectory planning
PEM	Prediction error method
PD	Power Demand; a HEPS IOL Controller module
PI	Proportional-Integral; controller
PID	Proportional-Integral-Derivative; controller
POMDP	Partially observed Markov decision process
PVM	Parallel Virtual Machine
QP	Quadratic programming
RCR	Rate of change of ratio; CVT
RCRC	RCR Command; a HEPS IOL Controller module
RDSA	Random direction stochastic approximation
RM	Robbins-Monro; a stochastic approximation algorithm
RPM	Revolutions per minute
RSM	Response surface methodology
SA	Stochastic approximation
SD	Steepest descent
SOC	State of charge; battery
SOMWO	Single-objective mission waypoint optimisation
SPSA	Simultaneous perturbation stochastic approximation
SQP	Sequential quadratic programming
TDC	Torque Difference Calculation; a HEPS IOL Controller module
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
UAVSM	UAV Simulation Model

UB	Upper bound
WPT	Waypoint
WPTTable	Waypoint table
XTE	Cross track error

List of Figures

1.1	Series configuration of an HEPS.	5
1.2	Parallel configuration of an HEPS.	5
2.1	A general simulation model.	14
2.2	The simulation optimisation process.	14
2.3	A summary of the RSM process when applied to simulation optimisation.	27
2.4	A generic Simulated Annealing algorithm [1].	30
2.5	A basic cycle of EAs.	33
2.6	A canonical evolution strategy.	36
2.7	Pareto optimality.	37
2.8	Hierarchical topology.	37
2.9	Parallel computing and asynchronous evaluation.	38
3.1	The Aerosonde UAV block, modified from the AeroSim 6DOF aircraft model.	42
3.2	Inside the Aerosonde UAV block.	44
3.3	The UAV Simulation Model.	45
3.4	Flight Planner Module.	46
3.5	Illustrative definition of a great circle track.	47
3.6	Geometric relationship between DTR, θ_{TE} and XTE.	48
3.7	Aircraft Control Module.	49
3.8	Waypoint Table in MATLAB code.	52
3.9	Test scenario - top view of flight path.	52

3.10	Test scenario - side view of flight path.	53
3.11	Flight profile of Mission Scenario 1 (not to scale).	54
3.12	Mission Scenario 1 with GPS Waypoints in central QLD, Australia (image generated using Google Earth).	54
3.13	Mission Scenario 2 with GPS Waypoints in central QLD, Australia (image generated using Google Earth).	55
4.1	Example comparison of an optimised set of waypoints (green) to the baseline flight mission (red) (not to scale).	58
4.2	Waypoints ($i - 1$), i and ($i + 1$) are not in a straight line.	62
4.3	Waypoints ($i - 1$), i and ($i + 1$) are in a straight line.	62
4.4	Illustration of the waypoint bounds for Mission Scenario 1.	63
4.5	Enlarged view of the waypoint bounds for Waypoints 3 & 7 in Mission Scenario 1.	63
4.6	Illustration of the waypoint bounds for Mission Scenario 2.	64
4.7	Updated UAVSM for SOMWO.	67
4.8	The optimisation rationale flow diagram for the HAPMOEA optimiser. . .	68
4.9	MATLAB function call for the SQP-solver.	69
4.10	Fuel consumption vs function evaluations for MS1 (1 waypoint - HAPMOEA). . .	70
4.11	Fuel consumption vs function evaluations for MS1 (1 waypoint - SQP). . .	71
4.12	Fuel consumption vs function evaluations for MS1 (2 waypoints - HAPMOEA).	71
4.13	Fuel consumption vs function evaluations for MS1 (2 waypoints - SQP). . .	72
4.14	Fuel consumption vs function evaluations for MS1 (3 waypoints - HAPMOEA).	72
4.15	Fuel consumption vs function evaluations for MS1 (3 waypoints - SQP). . .	73
4.16	Fuel consumption vs function evaluations for MS1 (4 waypoints - HAPMOEA).	73
4.17	Fuel consumption vs function evaluations for MS1 (4 waypoints - SQP). . .	74

4.18	Comparison of optimised waypoints for MS1 (1 waypoint optimised). . . .	78
4.19	Comparison of optimised waypoints for MS1 (2 waypoints optimised). . . .	78
4.20	Comparison of optimised waypoints for MS1 (3 waypoints optimised). . . .	79
4.21	Comparison of optimised waypoints for MS1 (4 waypoints optimised). . . .	79
4.22	Fuel consumption vs function evaluations for MS2 (1 waypoint - HAPMOEA). . . .	80
4.23	Fuel consumption vs function evaluations for MS2 (1 waypoint - SQP). . . .	80
4.24	Fuel consumption vs function evaluations for MS2 (2 waypoints - HAP- MOEA).	81
4.25	Fuel consumption vs function evaluations for MS2 (2 waypoints - SQP). . . .	81
4.26	Fuel consumption vs function evaluations for MS2 (3 waypoints - HAP- MOEA).	82
4.27	Fuel consumption vs function evaluations for MS2 (3 waypoints - SQP). . . .	82
4.28	Fuel consumption vs function evaluations for MS2 (4 waypoints - HAP- MOEA).	83
4.29	Fuel consumption vs function evaluations for MS2 (4 waypoints - SQP). . . .	83
4.30	Comparison of optimised waypoints for MS2 (1 waypoint optimised). . . .	85
4.31	Comparison of optimised waypoints for MS2 (2 waypoints optimised). . . .	86
4.32	Comparison of optimised waypoints for MS2 (3 waypoints optimised). . . .	86
4.33	Comparison of optimised waypoints for MS2 (4 waypoints optimised). . . .	87
5.1	The Hazard Areas (in red) near the MS2 waypoints.	91
5.2	Sectioning of the region around a Hazard Area.	92
5.3	HAPMOEA Optimiser results	96
5.4	SQP solver results	97
5.5	Pareto plot of HAPMOEA Optimiser and SQP Solver results (MS2). . . .	99
6.1	A typical engine map with the Ideal Operating Line (IOL)	104
6.2	Engine fuel map with BSFC contours using Aerosonde ICE data only	108
6.3	FuelMapCalc.mdl, the Simulink model based on the Piston Engine block . . .	109

6.4	Engine fuel map with BSFC contours using Aerosonde ICE data and open throttle calculations, with the “new island” boxed in red	109
6.5	Constant power contours for the Aerosonde ICE, obtained using Aerosonde ICE data and open throttle calculations	110
6.6	Engine map of the Aerosonde ICE, with power contours (magenta)	110
6.7	Engine map of the Aerosonde ICE, with the IOL (magenta)	111
6.8	Power (red) and torque(blue) values for the IOL	112
6.9	BSFC (red) and fuel flow (blue) values for the IOL	112
6.10	MAP values on the IOL (red - MAP from power, blue - MAP from fuel flow)	113
6.11	MAP values on the IOL	113
7.1	Parallel configuration of an HEPS.	116
7.2	The <i>Piston Engine</i> block.	119
7.3	Inside the <i>Piston Engine</i> block.	119
7.4	The <i>Motor Subsystem</i> block.	121
7.5	Inside the <i>Motor Subsystem</i> block.	121
7.6	The <i>Generator Subsystem</i> block.	122
7.7	Inside the <i>Generator Subsystem</i> block.	122
7.8	The <i>Battery Subsystem</i> block.	123
7.9	Inside the <i>Battery Subsystem</i> block.	124
7.10	Battery discharge curves.	125
7.11	The <i>CVT Dynamics Subsystem</i> block.	127
7.12	Inside the <i>CVT Dynamics Subsystem</i> block.	128
7.13	The <i>Charge Battery Now</i> block.	129
7.14	Inside the <i>Charge Battery Now</i> block.	129
7.15	The <i>Powertrain Output Allocation</i> block.	130
7.16	Inside the <i>Powertrain Output Allocation</i> block.	132
7.17	The basic control loop of the IOL Controller.	132

7.18	The <i>Operating Mode</i> block.	135
7.19	Inside the <i>Operating Mode</i> block.	135
7.20	The <i>Power Demand</i> block.	137
7.21	Inside the <i>Power Demand</i> block.	138
7.22	The <i>Engine Operation</i> block.	139
7.23	Inside the <i>Engine Operation</i> block.	140
7.24	The <i>Engine Throttle Command</i> block.	141
7.25	Inside the <i>Engine Throttle Command</i> block.	141
7.26	The <i>RCR Command</i> block.	143
7.27	Inside the <i>RCR Command</i> block.	143
7.28	The <i>Torque Difference Calc</i> block.	145
7.29	Inside the <i>Torque Difference Calc</i> block.	146
7.30	The <i>IOL Controller</i> block.	146
7.31	Inside the <i>IOL Controller</i> block.	147
7.32	The integrated HEPS block.	148
7.33	Inside the integrated HEPS block.	149
7.34	Integrating HEPS model (in orange) inside Aerosonde UAV block	150
7.35	Mission simulation using UAVSM with integrated HEPS (Plot 1).	151
7.36	Mission simulation using UAVSM with integrated HEPS (Plot 2)	152
7.37	Flight path comparison of UAVSM with integrated HEPS and original configuration (side view).	152
7.38	Flight path comparison of UAVSM with integrated HEPS and original configuration (top view).	153

List of Tables

3.1	Gain values for the PI and PID controllers in ACM.	50
3.2	Physical constraints on ACM outputs.	50
3.3	Waypoints in the test scenario.	51
3.4	Waypoints in the Mission Scenario 2 loop.	55
4.1	Waypoint bounds for Mission Scenario 1.	62
4.2	Waypoint bounds for Mission Scenario 2.	64
4.3	Physical constraints on ACM outputs.	65
4.4	Table of statistics for SOMWO (MS1).	74
4.5	Table of optimised waypoints for SOMWO (MS1).	77
4.6	Table of statistics for SOMWO (MS2).	84
4.7	Table of optimised waypoints for SOMWO (MS2).	85
5.1	MOMWO waypoint bounds for MS1.	93
5.2	MOMWO waypoint bounds for MS2.	93
5.3	Table of MOMWO statistics for HAPMOEA optimiser (MS1).	98
5.4	Table of MOMWO statistics for SQP solver (MS1).	99
5.5	Table of MOMWO statistics for HAPMOEA optimiser (MS2).	100
5.6	Table of MOMWO statistics for SQP solver (MS2).	100
6.1	Table of power output (W) at given values of RPM and MAP for the Aerosonde ICE	105

6.2	Table of fuel flow (g/hr) at given values of RPM and MAP for the Aerosonde ICE	106
6.3	Table of torque output (Nm) at given values of RPM and MAP for the Aerosonde ICE	106
6.4	Table of BSFC (g/W-hr) values at given values of RPM and MAP for the Aerosonde ICE	107
7.1	I/O configuration of <i>Piston Engine</i>	118
7.2	I/O configuration of <i>Motor Subsystem</i>	120
7.3	I/O configuration of <i>Generator Subsystem</i>	122
7.4	I/O configuration of <i>Battery Subsystem</i>	124
7.5	I/O configuration of <i>CVT Dynamics Subsystem</i>	126
7.6	I/O configuration of <i>Charge Battery Now</i> module.	129
7.7	I/O configuration of <i>Powertrain Output Allocation</i> module.	131
7.8	Switching conditions of operating modes.	134
7.9	I/O configuration of <i>Operating Mode</i> module.	135
7.10	Functionality of the <i>Power Demand</i> module.	136
7.11	I/O configuration of <i>Power Demand</i> module.	137
7.12	I/O configuration of <i>Engine Operation</i> module.	139
7.13	I/O configuration of <i>Engine Throttle Command</i> module.	140
7.14	I/O configuration of <i>RCR Command</i> module.	142
7.15	I/O configuration of <i>Torque Difference Calc</i>	145
7.16	Waypoints in MS1.	151
A.1	Aircraft configuration for an Aerosonde UAV	180
A.2	Look-up table for propeller coefficient of thrust and coefficient of power	183
A.3	Look-up table for Engine fuel flow	183
A.4	Look-up table for Engine power	184

G.1	Look-up table for Engine power (open throttle calculations)	242
G.2	Look-up table for Engine fuel flow (open throttle calculations)	243
G.3	Look-up table for Engine torque (open throttle calculations)	244
G.4	Look-up table for Engine BSFC (open throttle calculations)	245
H.1	Look-up table for Plettenberg HP220/25 Motor Data	248

Chapter 1

Introduction

1.1 Background and Motivation

An Unmanned Aerial Vehicle (UAV) is a “remotely piloted or self-piloted aircraft that can carry cameras, sensors, communications equipment or other payloads” [2] and is often used synonymously with the term *Unmanned Aircraft System* (UAS), which in fact refers to the entire system of unmanned aircraft, payload, and all direct support equipment (ground-based operators and support crews, etc.) [3]. UAVs have emerged as a viable platform to operate in regions where the presence of onboard human pilots is either too risky or unnecessary, in a diverse range of applications, from reconnaissance and surveillance tasks for the military, to civilian uses such as aid relief and monitoring tasks [4]. Their lower operation costs (as compared to manned aircraft and satellites) and availability in a great variety of sizes and capabilities have contributed towards the surging interest in UAVs from both the military and civilian sectors. In recent years, the development of UAVs has become a significant growing segment of the global aviation industry, with a total worldwide UAV expenditures forecast of USD\$80 billion from year 2010 to 2019 [5].

The focus of UAV applications has hitherto been predominantly in the military domain, but there has been increasing global interest in civilian and commercial UAV applications over the last decade, especially the use of small UAVs. Examples of these smaller and less sophisticated cousins of the military UAVs, e.g. MQ-1 Predator or RQ-4 Global Hawk, include the Shadow, the Aerosonde, the Global Observer, the Bell Eagle Eye, the SkySeer, the ARCAA Flamingo UAV and the KillerBee 4 UAVs. The lower-cost

factor of these small UAVs, much of which is due to the increase in the availability and quality of commercially off-the-shelf (COTS) components, and rapid increases in their capabilities as a result of technological advances, together present attractive incentives [6] for non-military and amateur UAV operators and developers, who are generally under more strict cost budgets when compared to their military counterparts.

Trade-offs between payload capability and aircraft endurance is always a problem that needs to be solved in the development and construction of UAVs, regardless of their size. As with all aircraft, there are mass and space limitations onboard UAVs for all onboard systems, including electronics, powerplant(s), fuel storage and payloads. However, these limitations are stricter for a small UAV due to their already smaller sizes.

The miniaturisation of avionics equipment and an increase in their capabilities have seen significant reductions in size, weight and power usage for small UAV components [7]. On the other hand, the use of COTS aeromodelling powerplants onboard a large proportion of emergent propeller-driven UAV platforms can be significantly disadvantageous in operational utility and energy efficiency compared to traditional aircraft powerplants [8]. While some UAV developers have successfully modified COTS aeromodel powerplants to achieve excellent efficiency, the utilisation of COTS components that are not sized ideally for the UAVs is still a common practice amongst civilian UAV developers, mostly due to cost issues. Consequently, the associated weight and space penalties contribute to limits on onboard fuel and/or energy resources (fuel, battery, etc.). This in turn brings about the problem of how to efficiently utilise the available energy resources onboard a small UAV.

Research in the area of efficient energy usage onboard a small UAV has mainly been conducted in two directions. One approach is by developing and implementing alternative energy technology for use onboard the UAV. Examples of these include fuel cells, solar cells and hybrid propulsion systems. The other approach focuses on adapting the UAV's flight path in order to optimise the energy efficiency of the aircraft, either by minimising energy usage or taking advantage of atmospheric energy resources such as wind (energy harvesting), as well as other mission goals. This thesis explores two possible methods of achieving energy efficiency onboard the UAV, and these are:

- optimisation of mission waypoints (§1.1.1); and

- utilisation of a Hybrid-Electric Propulsion System onboard the UAV (§1.1.2).

and they are described in the respective sections.

1.1.1 Mission Waypoint Optimisation

An UAV is deployed to carry out specific mission objectives, which can be time- and space-related (i.e. to arrive at a certain location within a specified time duration), or task-related (i.e. to obtain an image of a specified phenomenon). An integral part of achieving this is a flight path (or mission plan) [9], which consists of a series of waypoints which the UAV is required to fly through in a safe and efficient manner. This is to be executed while conforming to the rules of the air as with all civilian applications (as an essential requirement to integrate UAVs within the National Airspace System (NAS) [10]) and without exceeding the performance limitations of the aircraft.

With the necessity to take into account the flight time, distance, energy efficiency and conformance with aviation regulations [11], the optimisation of a specified mission plan is a multi-objective problem. Contrasting with the global path planning of Mission Flight Planning (MFP) [12] (pre- or in-flight design of a flight path to direct the UAV from a specified starting point to a specified destination point) and the local path planning method of trajectory optimisation [13] (the generation of smooth trajectory through a set of mission waypoints, typically generated by a global planner), waypoint optimisation in this case is the evaluation and modification of a **given** series of waypoints which the UAV has to fly through. However, these waypoints are not definite in the sense that they may be adjusted within set bounds in order to be optimised. This waypoint optimisation technique has the potential be integrated with MFP and trajectory optimisation, both of which are performed online (in-flight) onboard the UAV, by generating an off-line flight plan which can be further refined by these two techniques to assist in the operation of UAVs in uncertain and dynamic environments.

1.1.2 Hybrid-Electric Propulsion System

Traditionally, small civilian UAVs are mostly powered by Internal Combustion Engines (ICEs), but as they have a thermal efficiency of at most 40% [14] and, despite the high

energy density of the liquid hydrocarbon fuels used by ICEs, with energy preservation issues on the rise, more efficient powerplant configurations have been sought.

The most popular alternative powerplant is the Electric Motor (EM), which are capable of operating with an efficiency of close to 100% [15]. However, EM's high efficiency is negated by the necessary use of a power storage system which drives the EM in order to power the UAV. This power storage system, in most cases a battery, is often the largest component by weight in an UAV, representing a large weight penalty, as well as having a limited operating duration and relatively long period of time required to replenish its charge, resulting in a relatively short operating range and the need to charge frequently. Despite recent advances in power storage technology which have reduced the impact these drawbacks have had on the use of EM in civilian UAVs in the past, the sizes and relative inefficiencies of power storage systems still hold back the development of purely electrically powered UAV.

A way of overcoming the shortcomings of both powerplants is to integrate an ICE with an EM to form a Hybrid-Electric Propulsion System (HEPS), which can be designed and implemented in two main powertrain configurations - series and parallel - with numerous control strategies for each [16].

In the series powertrain configuration, shown in Figure 1.1, the EM is the only means of providing power to the mechanical drive train. This means the ICE is able to operate in an optimum torque and speed range, regardless of the driving conditions, in the execution of its role as an auxiliary power unit to drive the EM to propel the aircraft, or the Generator to provide power to the energy storage system, or the Battery. The series configuration performs best for low-speed, high-torque applications such as in buses and other urban work vehicles.

However, because the mechanical energy from the ICE is firstly converted to electrical energy in the Battery, then passed on to the EM, and lastly converted once again into mechanical energy to power the propeller, there exist large energy conversion losses between the mechanical and electrical systems. Thus the overall system efficiency is reduced [17]. Also, in the series configuration, although the ICE is typically smaller because it only has to meet average power demands, the EM and the Battery generally need to be sized larger to accommodate the peak power demands. This, combined with

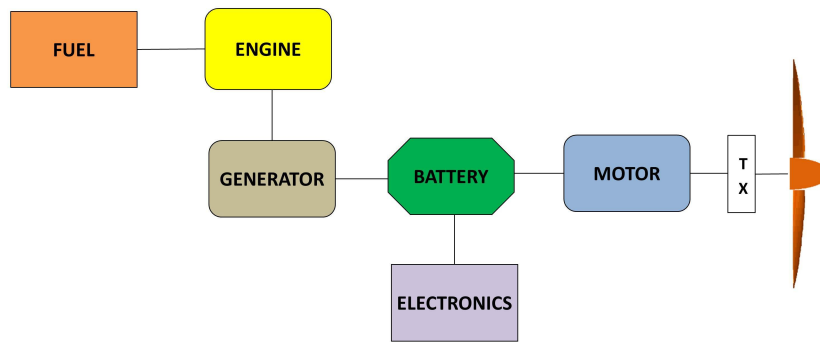


Figure 1.1: Series configuration of an HEPS.

the Generator required in this configuration, results in a significant weight penalty, which is expensive onboard a UAV. Harmon [18] estimated that a series configuration for a small UAV can result in a 8%, or 2.5 lb (1.13kg), weight penalty for a 30-lb (13.61kg) UAV.

The parallel configuration, shown in Figure 7.1, enables the powering of the UAV using the ICE alone, the EM alone, or both depending on the operating conditions, as well as benefitting from redundancy, which is important in both civilian and military applications. Realisation of this configuration can be seen in various commercially available ground vehicles such as the Honda Insight, Civic and Accord hybrids [19].

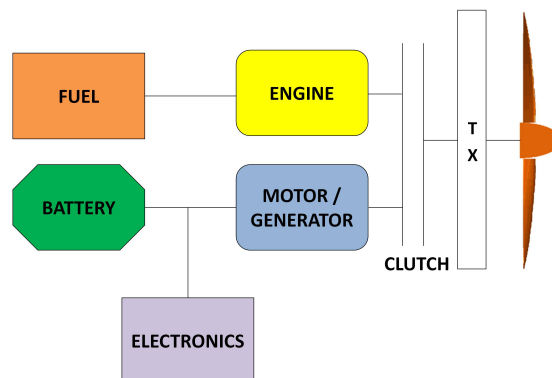


Figure 1.2: Parallel configuration of an HEPS.

Although this configuration can potentially suffer from an inability to operate the ICE in its most efficient region (it is directly coupled to the wheels through a transmission, thus limiting the energy efficiency), integrating a Continuously Variable Transmission (CVT) between the ICE and the wheels will help to mitigate this potential problem. However, this introduces additional difficulty into the control strategy to schedule the torque from the individual or combined power sources for maximum efficiency [16, 20].

The control strategy of HEPS is an integral component for a Hybrid-Electric UAV (HEUAV). Cho [21] reviewed the five categories of algorithms for a supervisory HEPS controller and opted for the use of predictive control methods, while Harmon [18] developed a neural network controller. Francisco [16] and Frank [22], on the other hand, both implemented an Ideal Operating Line (IOL) control strategy, which was made possible with the use of a CVT in their systems. Due to the proven nature of this strategy, this approach was adopted in the course of this research.

1.2 Research Objectives

The aim of this research is to explore and develop two strategies to increase the energy efficiency of a small fixed-wing UAV. Firstly, a Mission Waypoint Optimisation (MWO) strategy was developed for a small fixed-wing UAV, focusing on the fuel economy benefit. Secondly, a simulation model of a parallel HEPS onboard a small fixed-wing UAV incorporating an IOL control strategy was developed and implemented.

The main objectives are as follows:

- Objective 1** To investigate the use of optimisation techniques and algorithms that are currently available for the purpose of increasing energy efficiency on a small fixed-wing UAV;
- Objective 2** To construct a simulation environment, including a model of an ICE-powered UAV and a baseline mission scenario;
- Objective 3** To implement a MWO procedure with the main focus on maximising fuel economy;
- Objective 4** To demonstrate the fuel economy benefit of a MWO strategy through coupling with the UAV simulation model and performing computer simulations;
- Objective 5** To investigate and perform an IOL analysis of an ICE;
- Objective 6** To develop and implement simulation models of HEPS components in a HEUAV; and
- Objective 7** To demonstrate the fuel economy benefit of the HEPS through coupling the HEUAV simulation model and performing computer simulations.

1.3 Research Contribution

The main outcome, and one of the novel contributions, of this research is a framework for the implementation and simulation of an UAV – ICE or hybrid-electric powered – in the Simulink environment. Due to the modular nature of the simulation model and the incorporation of aircraft-specific configuration and parameters, simulation of any small fixed-wing UAV can be achieved using this framework.

This research also sees the application of the HAPMOEA optimiser in the area of MWO, which is a novel area of application. Previous employment of HAPMOEA had been mainly in the design and optimisation of aircraft and airfoils. The coupling of the HAPMOEA optimiser with MATLAB and Simulink also had not been attempted previously. Due to the popularity of the MATLAB and Simulink environment among academia and industry alike, this may very well enable the HAPMOEA optimiser to be used in a wider range of applications.

1.4 Research Methodology

The investigation of methods for increasing the energy efficiency on UAVs was conducted using the following methodology:

1. Conducting a literature review on simulation optimisation to determine the essential requirements and elements in carrying out a simulation optimisation, during which a list of optimisation methods that are commonly associated with simulation optimisation were also investigated – see §2;
2. Designing, implementing and improving a 6-DOF simulation model of an UAV by adding modules to enable unmanned operations – see §3;
3. Formulating the optimisation problem – for both single- and multi-objective optimisation – in a manner that can be solved using the coupling of the simulation model with the respective optimisers – see §4 for single-objective optimisation and §5 for multi-objective optimisation; and
4. Designing and implementing the HEPS components, as well as the integration of these components into the existing UAV simulation model to set up hybrid-electric

operations – see §7; this includes the IOL analysis of the Aerosonde ICE – see §6.

This research methodology was then followed and the results and outcomes are presented as outlined in §1.5.

1.5 Outline of Thesis

This thesis consists of 8 chapters.

Chapter 2 presents a literature review on Simulation Optimisation and the optimisation methods that have been coupled with computer simulations in Simulation Optimisation, which are essential components in the MWO strategy. An overview of the theoretical aspects of the Multi-Objective Evolutionary Algorithm (MOEA) based optimiser and the Sequential Quadratic Programming based solver used in Chapter 4 are also given.

Chapter 3 introduces the simulation environment, including the baseline UAV model and two mission scenarios - Mission Scenario 1 and Mission Scenario 2.

Chapter 4 describes the Single-Objective MWO strategy and its components. The MWO procedure was performed for each of the mission scenarios, using the EA-based optimiser and the SQP-based optimiser respectively for comparison purposes. The results and analysis of both procedures are then presented.

Chapter 5 presents the Multi-Objective MWO process. A similar approach to the Single-Objective MWO as described in Chapter 4 was used, in which the optimisation was performed for the two mission scenarios, using both the EA-based and SQP-based optimisers. The resulting Pareto plots and optimised waypoints for each mission scenario are then presented and analysed.

Chapter 6 describes the process of an IOL analysis using data from an Aerosonde ICE as an example.

Chapter 7 presents the modelling of the HEPS components, namely an Electric Motor, a Generator, a Battery, a Controller and its components based on the IOL determined in Chapter 6, and the dynamics of a Continuously Variable Transmission. The resulting models were then integrated to form a HEPS block that is compatible with the AeroSim Aerosonde simulation model. In conjunction with Chapter 6, the hybridisation of the

UAV is presented.

Chapter 8 presents conclusions of the work conducted and areas for future research.

Chapter 2

Review of Simulation Optimisation Techniques

2.1 Introduction to Optimisation

The Oxford English Dictionary defines the term *optimisation* as the act of “mak[ing] the best or most effective use of a situation or resource” [23]. In the case of MWO, the optimisation process attempts to find the set of mission waypoints that, when the entire mission is executed, results in the minimisation/maximisation of the specified objective function(s). If fuel efficiency/consumption is the only parameter to be optimised (maximised if efficiency, minimised if consumption), a single-objective MWO problem is formed. On the other hand, if other parameters - flight time, endurance, distance travelled, for example - also need to be optimised, a multi-objective MWO will need to be performed.

Mathematically, the basic optimisation problem can be represented as:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{y} = f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{g}_i(\mathbf{x}) = 0, \quad i = 1, \dots, n_e \\ & \mathbf{h}_j(\mathbf{x}) \geq 0, \quad j = 1, \dots, n_i \end{aligned} \tag{2.1}$$

where $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$ is the set of *decision variables* and $\mathbf{y} : \mathbb{R}^n \rightarrow \mathbb{R}$ is the set of *objective functions* given a particular set of decision variable values. There are two

types of *constraint functions* associated with optimisation: *equality constraint functions*, $\mathbf{g}(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$, and *inequality constraint functions*, $\mathbf{h}(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$, which are evaluated at \mathbf{x} . The solution of the optimisation problem is a member of the *feasible set* (or *solution space*), represented by $D = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{g}_i(\mathbf{x}) = 0, \mathbf{h}_j(\mathbf{x}) \geq 0\}$. \mathbf{y} , $\mathbf{g}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$ are scalar functions of \mathbf{x} , i.e. each element of these vectors is of a scalar value.

An important component of any optimisation process is a model of the system that is to be optimised. Since the optimisation process works solely with the model's responses to any given inputs, the results of the optimisation will be the results expected from the system **only if** the model is a valid representation of the system performance. Traditionally, the system is portrayed as a mathematical or analytical model, consisting of an objective function and a set of constraints in the form of Equation 2.1. These can then be used in conjunction with an appropriate optimisation method to determine the values of the decision variables which will optimise the desired parameters. However, the complexities of most real-world problems make the evaluation of performance measures and finding optimal decision variables analytically very difficult, if not impossible.

With the advances in computer technology, simulation is becoming widely accepted as a decision-making tool [24]. Instead of representing a system's performance as a mathematical model, using computer simulations of real-world events enables a complex problem to be examined and analysed in an efficient, safe and cost-effective manner. They are also useful in the optimisation of a situation where experiments on the real-world system are difficult or not possible, to be conducted. Consequently, simulation optimisation has received considerable attention from both simulation researchers and practitioners [25] and has undoubtedly been stimulated by the availability of optimisation tools in commercial software packages [26].

MWO is one application which could greatly benefit from the utilisation of simulation optimisation. The system to be modelled in the MWO problem is a small fixed-wing UAV and given a set of candidate waypoints, the MWO needs to determine the values of the desired output parameter(s) (e.g. fuel consumption). However, its multidisciplinary nature, e.g. aerodynamics, aircraft inertia, atmosphere, etc., renders a purely analytical representation of the relationship between a set of waypoints and the desired output parameter(s) impossible. Since commercially-available simulation software are widely

available today (e.g. MATLAB Simulink), constructing a simulation model of the UAV system and then using it in MWO is an attractive and viable method.

This chapter will firstly give an overview of the concept of simulation optimisation (§2.2). The four major classes of methods for simulation optimisation will be presented and some popular techniques from each class will be discussed (§2.3–§2.7), although this does not claim to be an exhaustive literature survey on simulation optimisation techniques. Then §2.8 presents a summary of findings to conclude the chapter.

2.2 Simulation Optimisation

Discrete-event computer simulation is a powerful and useful tool in evaluating real-world systems from a wide range of areas such as manufacturing, supply chain management, financial management and aircraft systems, that are often too complex to be modelled analytically [27, 28]. Given a set of values for the decision variables of the system, certain performance measures are obtained in order for the system to be evaluated. However, a more practical procedure is to seek optimum values for these decision variables so that one or more given responses are maximised or minimised. This calls for the process of optimisation to be applied to a system modelled using computer simulations, and gives rise to the method of simulation optimisation.

Simulation optimisation is an optimisation method where the objective function (or objective functions, in the case of a multi-objective problem) and/or some constraints are responses evaluated by a simulation model [29]. A general simulation model, shown in Figure 2.1, consists of n input variables (x_1, x_2, \dots, x_n) and m output variables (y_1, y_2, \dots, y_m) . Due to the complexity of the systems, the simulation models are often highly nonlinear in nature [30]. Simulation optimisation is therefore a *nonlinear constrained optimisation* problem with the general form as described in Equation 2.1.

In the context of simulation optimisation, a simulation model can be considered as a “mechanism that turns input parameters into output performance measures” [31], or a function, whose explicit form is unknown, that evaluates the merit of a set of specifications. This is essentially viewing the simulation model as a black-box function evaluator [32]. In the simulation optimisation process, the output of the simulation model is used in

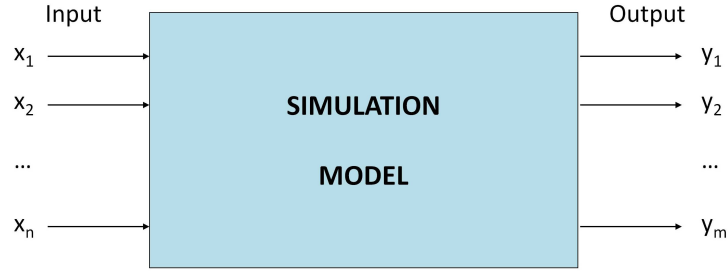


Figure 2.1: A general simulation model.

an optimisation strategy, which generates new candidate solutions that are fed into the simulation model for another evaluation. This iterative process is continued until a specific stopping rule is reached. Figure 2.2 illustrates this process.

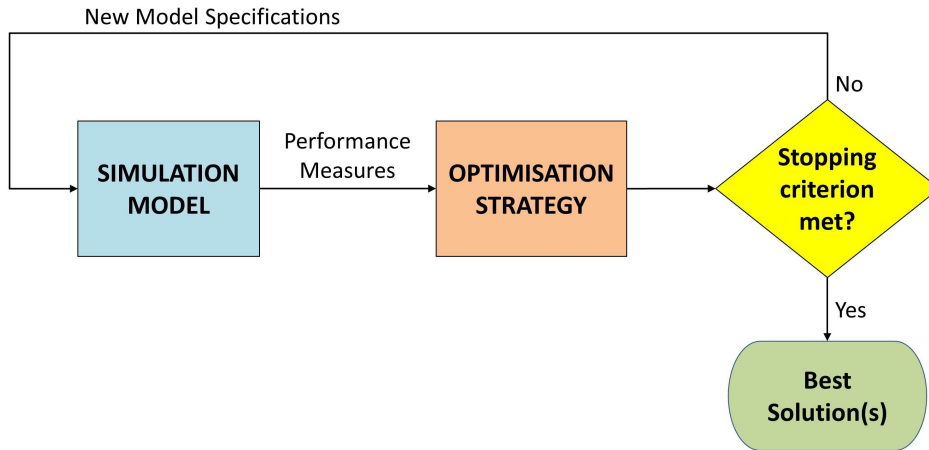


Figure 2.2: The simulation optimisation process.

Various simulation optimisation techniques have been developed [33], and four major classes of approaches can be distinguished [29, 34, 35]:

- Gradient-based search methods;
- Stochastic approximation methods;
- Response surface methodology; and
- Heuristic methods.

each proposing a strategy to explore the solution space D with a limited number of simulations. These strategies can be divided into two types. The first type begins with collecting a sample of interesting points and then utilising these points in a second step (e.g. using a response surface). The other type requires a connection between the optimisation algorithm and the simulation model in order to search iteratively in D .

Due to the black-box nature of the simulation model, and thus its objective function [36], optimisation of the decision variables is carried out without the knowledge of how the value of the objective function is determined within the simulation model itself [37]. Consequently, the gradient-based search methods which use a direct gradient estimator to estimate the true gradient in each iteration and therefore requires some additional analysis of the underlying mechanics of the simulation model [38], will be unsuitable for this purpose. On the other hand, such knowledge is not required when implementing gradient-based methods which use an indirect gradient estimator (e.g. finite differences and simultaneous perturbations) [38], response surface methods, and metaheuristic¹ methods (e.g. Simulated Annealing and Evolutionary Algorithms) since they use only function evaluations to make decisions regarding the selection of the next candidate solution [41].

In simulation optimisation, obtaining the objective function requires the execution of simulation runs that are often computationally expensive and time-consuming. Therefore it is imperative that the chosen optimisation algorithm is capable of finding optimal or near-optimal solutions in the early stages of the optimisation process. The algorithm should also be able to effectively balance exploration and exploitation of the solution space in search of the global optimal solution [42].

The main difficulty in using simulation optimisation is the trade-off between allocating computational resources for **searching** the solution space for candidate solutions and conducting additional simulation replication for better **estimating** the performance of current promising solutions, both of which can involve algorithmic and simulation computations [43]. If too much computational effort is spent on long simulation runs, the algorithm may evaluate only a few solutions in the time available and fail to identify an optimal, or even a good, solution. On the other hand, if variability cannot be controlled in the simulation model, the search can be misled or fail to recognise good solutions [28]. As computer simulation programs are relatively expensive to run, efficiency of the optimisation algorithms used is crucial.

¹**Metaheuristics** represent a family of “solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space” [39]. These approximate optimisation techniques “provide *acceptable* solutions in a reasonable time for solving hard and complex problems. Unlike exact optimisation algorithms, metaheuristics do not guarantee the optimality of the obtained solutions” [40].

Another complexity in simulation optimisation is the simulation/programming languages that are used to code each, which is often different. This results in challenges to be overcome when interfacing simulation models with optimisation algorithms, especially when newer higher-level user-friendly simulation languages are used [27].

In the following sections, a brief review of several popular simulation optimisation techniques and their applications, with special focus on those that are related to UAVs and/or mission/path/trajectory optimisation, will be presented. These techniques are as follows:

- Sequential Quadratic Programming (SQP) (gradient-based search method) (§2.3);
- Stochastic Approximation (SA) Methods (§2.4);
- Response Surface Methodology (RSM) (§2.5);
- Simulated Annealing (heuristic method) (§2.6); and
- Evolutionary Algorithms (heuristic method) (§2.7).

2.3 Sequential Quadratic Programming

Sequential Quadratic Programming (SQP) methods are very powerful techniques for solving small-, medium-sized and certain classes of large-scale nonlinear programming (NLP), or nonlinear constrained optimisation, problems [44, 45]. Based on the work of Biggs [46], Han [47] and Powell [48, 49], and popularised in the 1970s, SQP methods are a family of specific algorithms sharing the same conceptual method [50]. These methods are commonly included in commercially-available software packages, such as in the MATLAB Optimization Toolbox [51], and are used to solve a range of important practical problems in science, engineering, industry and management.

SQP is an iterative method which finds the solution to a given NLP problem via successive quadratic approximations of the objective function [52]. Essentially, SQP models an NLP problem at a given approximate solution, \mathbf{x}_k , by a Quadratic Programming (QP) subproblem based on the quadratic approximation of the Lagrangian function:

$$L(\mathbf{x}, \gamma, \beta) = f(\mathbf{x}) + \sum_{i=1}^{n_e} \gamma \mathbf{g}_i(\mathbf{x}) + \sum_{j=1}^{n_i} \beta \mathbf{h}_j(\mathbf{x}) \quad (2.2)$$

where γ and β are the Lagrangian multipliers associated with the equality and inequality constraint functions, $\mathbf{g}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$ respectively. The QP subproblem is formulated as follows:

$$\begin{aligned} \min_{\mathbf{d} \in \mathbb{R}^n} \quad & \frac{1}{2} \mathbf{d}^T \mathbf{H}_k \mathbf{d} + \nabla f(\mathbf{x}_k)^T \mathbf{d} \\ \text{subjected to} \quad & \nabla \mathbf{g}_i(\mathbf{x}_k)^T \mathbf{d} + \mathbf{g}_i(\mathbf{x}_k) = 0, \quad i = 1, \dots, n_e \\ & \nabla \mathbf{h}_j(\mathbf{x}_k)^T \mathbf{d} + \mathbf{h}_j(\mathbf{x}_k) \leq 0, \quad j = 1, \dots, n_i \end{aligned} \quad (2.3)$$

where \mathbf{d} is the solution of the QP subproblem, and \mathbf{H}_k is a positive definite approximation of the Hessian matrix of the Lagrangian function in Equation 2.2.

The solution to the QP subproblem, \mathbf{d} , is then used in the computation of a new iterate, \mathbf{x}_{k+1} , of the overall NLP problem:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k \quad (2.4)$$

where α_k is the step length of a value in the interval $[0, 1]$ and is determined by an appropriate line search procedure so that a sufficient decrease in a merit function, $\phi(\mathbf{x})$, is obtained. After each iteration, the matrix \mathbf{H}_k can be updated using any of the quasi-Newton methods, with the Broyden-Fletcher-Goldfarb-Shanno (BFGS) approximation a popular algorithm for this purpose [45, 52].

This process is iterated to create a sequence of approximations that, hopefully, will converge to a solution of the NLP problem, \mathbf{x}^* . Because SQP can be viewed as an extension of the Newton and quasi-Newton methods to the problem of nonlinear constrained optimisation, with an appropriate choice of the QP subproblem it provides rapid convergence to a local solution when a good estimate is available [44]. However, when the iterates are far from a solution, SQP may exhibit possible erratic behaviour which will need to be carefully controlled [50] and a substantial number of QP iterations may be required, which may result in poor reliability of the optimisation process [53]. Furthermore, if the problem behaves poorly, the Hessian matrix \mathbf{H}_k will require updating, which in turn requires an extra $n + 1$ function evaluations per iteration [52].

A great strength of SQP is its ability to solve problems with nonlinear constraints. However, like most gradient-based optimisation algorithms, SQP requires the determination of the first- and/or second-order gradient information for the objective and constraint functions with respect to the design variables [52]. In most simulation optimisation problems, this gradient information is not analytically available and in order to obtain an approximation to this information, expensive simulations need to be run to determine the values of objective and constraint functions.

As mentioned earlier, SQP is used extensively in commercial optimisation software packages. Examples of these include NPSOL [54], SNOPT [55, 56] and, also mentioned previously, MATLAB Optimization Toolbox [51].

The Fortran-based NPSOL is utilised in the Nonlinear Trajectory Generation (NTG) software package developed at Caltech by Milan *et al.* [57]. NTG solves optimal control problems in real-time by parameterising system dynamics into B-Splines, coefficients for which are solved using NPSOL to minimise the desired cost function subject to the linear and nonlinear constraints of the problem. Applications of NTG include trajectory generation to minimise the probability of detection of UAVs by opponent radar detection systems by Inanc *et al.* [58], for low observability flight path planning of UAVs in the presence of radar detection systems by Misovec *et al.* [59], for Cooperative Path Planning (CPP) in a multi-vehicle system by Lian and Murray [60], and the reconfiguration manoeuvring of a spacecraft fleet by Garcia [61]. In each case, the performance of the optimiser was satisfactory with fast convergence to the solution, however an initial estimate far from the solution can still lead to a large number of iterations to be computed [62].

On the other hand, the *fmincon* function in the MATLAB Optimization Toolbox, mentioned previously, is commonly used in applications where nonlinear constrained optimisation needs to be solved. A brief overview of this function, referred to as *MATLAB SQP Solver*, is described in §2.3.1. Bradley *et al.* [63] used this solver in their investigation of the effects of flight path optimisation on fuel cell powered and ICE-powered aircraft. Although *fmincon* is generally very robust against not reaching a global optimum, a design space exploration was performed to ensure local optima were found. *fmincon* was also used by Vanek *et al.* [64] to implement a nonlinear Model Prediction Control (MPC) approach to track the trajectory of UAV formations in a strategy similar to the NTG

software package described above. It was found that the *fmincon* solver was able to handle the nonlinear constraints of the problem at hand and, because the initial estimate was well placed, convergence to the reference trajectory was rapid. Geiger *et al.* [65] also utilised the *fmincon* function in their development of a trajectory generation algorithm using direct collocation with nonlinear programming (DCNLP) in order to maximise the time a target is in the view of multiple cameras on different UAVs. Simulation results in MATLAB showed that suitable trajectories were generated which satisfied the problem constraints. However, in the latter two applications, it was found that the computational costs associated with executing the *fmincon* solver in each optimisation cycle are a concern and this makes real-time trajectory generation difficult.

SQP methods, like Newton's method to which they closely mimic, are only guaranteed to find a local solution to the NLP problem [50]. This could be problematic if the initial estimate of the solution is located close to a local optimum. One approach to maximise the ability of the optimisation technique to find a global minimum is by combining global and local optimisation techniques [66] and a prominent combination is to use genetic algorithms (GA) with SQP methods. This hybrid strategy was adopted by Okuda *et al.* [67] to optimise the trajectory of a winged rocket after reaching the apogee toward the landing point, and also in the analysis of missile evasion using aircraft trajectory optimisation methods by Ranta [68]. Initially, GA searches through the feasible set for a solution, which is then used as an initial estimate by the SQP method. This strategy utilises GA's capability to search through a wide range of candidate solutions to avoid being trapped in a local optimum, while exploiting SQP's properties of computational efficiency and rapid convergence to a solution given a good initial estimate.

2.3.1 MATLAB SQP Solver

The SQP solver is an in-built medium-scale constrained solver in MATLAB the utilises a sequential quadratic programming (SQP) method [46–49].

This solver finds a minimum of a constrained nonlinear multivariable function as defined by:

$$\begin{aligned}
\text{Find } \min_x f(x) \text{ subject to: } & c(x) \leq 0 \\
& c_{eq}(x) = 0 \\
& A \cdot x \leq b \\
& A_{eq} \cdot x = b_{eq} \\
& lb \leq x \leq ub
\end{aligned}$$

where x is the vector input to the function $f(x)$, b and b_{eq} are the vector bounds for linear equalities, lb and ub are upper and lower bounds for the design variables x , A and A_{eq} are coefficient matrices, $c(x)$ and $c_{eq}(x)$ are functions that return vectors, and $f(x)$ is a function that returns a scalar. $f(x)$, $c(x)$ and $c_{eq}(x)$ can be nonlinear functions.

The optimisation process requires the specification of an initial estimate vector, x_0 , and a set of constraints for the variables to be optimised. The constraints can be any or a combination of the following:

1. Linear equalities ($A \cdot x \leq b$ and $A_{eq} \cdot x = b_{eq}$)
2. Lower and upper bounds ($lb \leq x \leq ub$)
3. Nonlinear inequalities ($c(x) \leq 0$ or $c_{eq}(x) = 0$)

2.4 Stochastic Approximation Methods

Consider a nonlinear black-box system which, for inputs \mathbf{x} , gives as its output “noisy” measurements of the objective function, $f(\mathbf{x}) + \varepsilon$, where ε is a zero mean random variable representing noise. Such a system cannot be optimised using deterministic methods such as an SQP method (see §2.3) since they do not take into consideration the stochastic nature of the noise and its effects on the system outputs. Rather, stochastic optimisation methods that provide a means of coping with inherent system noise are required, and one such method is stochastic approximation (SA).

Introduced by Robbins and Monro in 1951 [69], the SA algorithm is a general optimisation method when only noisy measurements of the underlying objective function

is available. The algorithm is comparable to the steepest descent (SD) gradient search method in deterministic nonlinear optimisation, but assumes no direct knowledge of the gradient of the objective function, $\mathbf{g}(\mathbf{x})$ [70]. This makes SA a good candidate for the optimisation of a simulation model that, due to its complexity, has no analytical expression for the objective function, and therefore has no analytical gradient [25]. Below is a brief overview of the SA method. For detailed descriptions, see [38, 70–73].

The basic SA algorithm, as proposed by Robbins and Monro [69], is of the following iterative form:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \tilde{\mathbf{g}}(\mathbf{x}_k) \quad (2.5)$$

where \mathbf{x}_k is the current solution for the decision variables, $\tilde{\mathbf{g}}(\mathbf{x}_k)$ a noisy estimate of the gradient function, and α_k a sequence that satisfies

$$\alpha_k > 0 \quad (2.6)$$

$$\sum_{k \geq 1} \alpha_k = \infty \quad (2.7)$$

$$\sum_{k \geq 1} \alpha_k^2 < \infty \quad (2.8)$$

If the direct measurement for $\tilde{\mathbf{g}}(\mathbf{x}_k)$ is available, Equation 2.5 is referred to as the Robbins-Monro (RM) algorithm. However, these direct measurements are not always obtainable, therefore gradient approximation, or the computation of $\tilde{\mathbf{g}}(\mathbf{x}_k)$, is an essential part of SA.

The traditional means of forming the gradient approximation is the *finite difference stochastic approximation* (FDSA) method. FDSA computes an estimator for the gradient by looking at some small perturbation c_k in each of the decision variables in \mathbf{x}_k by either an one-sided formulation, in which the i th component of $\tilde{\mathbf{g}}(\mathbf{x}_k)$ is given by:

$$\tilde{\mathbf{g}}_i(\mathbf{x}_k) = \frac{\tilde{f}(\mathbf{x}_k + c_k \mathbf{e}_i) - \tilde{f}(\mathbf{x}_k)}{c_k}, \quad i = 1, \dots, n \quad (2.9)$$

or a two-sided formulation, also known as the Kiefer-Wolfowitz (KW) algorithm [74]:

$$\tilde{\mathbf{g}}_i(\mathbf{x}_k) = \frac{\tilde{f}(\mathbf{x}_k + c_k \mathbf{e}_i) - \tilde{f}(\mathbf{x}_k - c_k \mathbf{e}_i)}{2c_k}, \quad i = 1, \dots, n \quad (2.10)$$

where $\tilde{f}(\mathbf{x}_k)$ is a noisy measurement of the objective function f at the current iteration of \mathbf{x} , \mathbf{e}_i the unit vector along the i th axis and n the dimensionality of the system, or the number of decision variables. The one-sided estimate requires $n + 1$ measurements of the objective function for each iteration of \mathbf{x}_k computations, while the KW algorithm requires $2n$ such measurements [25, 71]. As the dimension n increases, as is the case with simulation models of complex systems, the number of measurement required, and thus the computational effort, may become prohibitive [70].

Extended approaches have been developed which dramatically improve the computational efficiency of FDSA, including the *random direction stochastic approximation* (RDSA) and the *simultaneous perturbation stochastic approximation* (SPSA) methods.

The RDSA algorithm, first introduced by Kushner and Clark [75], uses the following estimator:

$$\tilde{\mathbf{g}}(\mathbf{x}_k) = \left[\frac{\tilde{f}(\mathbf{x}_k + c_k \xi_k) - \tilde{f}(\mathbf{x}_k - c_k \xi_k)}{2c_k} \right] \xi_k \quad (2.11)$$

where $\xi \in \mathbb{R}^n$ is a direction vector normalised so that $|\xi|^2 = n$. ξ can be chosen from several choices of random distributions, including the *axis* distribution ($\xi = \pm\sqrt{p} \mathbf{e}_i$ with coordinate i chosen at random from $1, \dots, n$), the *normal* distribution (takes each component ξ_i to be distributed as $N(0, 1)$), and the *Bernoulli* distribution (takes each component ξ_i at random from $\{-1, 1\}$) [76]. Note that both the perturbations to \mathbf{x}_k and the estimated gradient are in the same direction, ξ .

On the other hand, the SPSA algorithm [77] is a special case of RDSA that employs two different directions, ξ and ζ , in its general form:

$$\tilde{\mathbf{g}}(\mathbf{x}_k) = \left[\frac{\tilde{f}(\mathbf{x}_k + c_k \xi_k) - \tilde{f}(\mathbf{x}_k - c_k \xi_k)}{2c_k} \right] \zeta_k \quad (2.12)$$

where ξ is chosen from a distribution that has to satisfy some particular constraints, and

the components of ζ are given by

$$\zeta_i = \frac{1}{\xi_i} \quad (2.13)$$

Both the RDSA and SPSA methods require only two measurements of the objective function per iteration, regardless of the dimension of the system, to form a gradient approximation, thus providing the potential for large savings in the overall cost of the optimisation process [42].

The SA algorithm has been a cornerstone in scientific computation since its inception in 1951, mainly because of the following advantages [73]:

- Its ability to handle noisy situations – In practice, the noise may not originate only from measurement errors or approximations, but may also be added deliberately as a probing device or a randomised action;
- It is incremental – The algorithm typically exhibit more graceful behaviour, although this is at the expense of speed; and
- Its computational efficiency – The number of computations per iterate is generally low for typical applications, especially when SPSA is utilised. This can result in easy implementation of the algorithm.

However, the SA algorithm and its different forms – including FDSA, RDSA and SPSA algorithms – are prone to several drawbacks. One of these drawbacks is the algorithm’s slow speed of convergence to an optimal solution [71], which mainly results from the use of small increments in the computation of the algorithm, as mentioned previously. It is for this reason that many efforts to accelerate the SA algorithm, such as the Kesten algorithm [78, 79], have been focused on the choice of the step size, α_k . Another disadvantage is that, in general, the SA algorithm converges only to a local optimum, although this can be counteracted by the appropriate injection of “noise” into the algorithm [38].

SA has developed into an important area in the control and optimisation of stochastic systems [72]. Its strengths make it ideal for applications where the keyword is ‘adaptive’, such as adaptive signal processing, adaptive control, and certain subdisciplines of soft computing/artificial intelligence (e.g. neural networks, reinforcement learning; see

Bertsekas and Tsitsiklis [80] and Haykin [81, 82]).

In recent years, much attention has been focused on the application of SPSA to the optimisation of, to name a few, queuing systems, aircraft design, process control, fault detection, vehicle traffic management, human-machine interaction and simulation optimisation [70].

Hutchison and Spall investigated the use of SPSA in a simulation optimisation procedure to determine the control measures required for air traffic delay control [83], which was shown to be feasible in earlier studies [84, 85]. The SIMMOD simulation [86] was used to model the flow of 84 flights (departures and arrivals) on a network of four airports. A number of Monte Carlo trials were run, and the results indicated that the SPSA-based optimisation process reduced the total weighted delay by 13.3%, with a significant reduction of 31.8% in the expensive component of air delay.

Burnett [87] conducted a comparison of three optimisation methods – Blind Random Search (BRS), SPSA and simulated annealing – for the optimisation of vessel traffic management in a high vessel density environment. It was found that BRS and SPSA performed relatively well on this problem with SPSA producing the lowest values of the objective function (distance of the main vessel to surrounding vessels) and its standard deviation. However, in order for convergence of the SPSA algorithm to occur, the objective function was required to be three-times continuously differentiable, which could very likely be problematic for most simulation models.

The SPSA algorithm is also employed in the work of Xing and Damodaran [88] in optimising the design of aerodynamic shapes using computational fluid dynamics (CFD), which involves time-consuming numerical computation of the flowfield characteristics [89]. Comparisons were made against a simulated annealing method and it was shown that the SPSA method required fewer function evaluations to reach the target design than that required by the simulated annealing method. Xing and Damodaran [90] also investigated the possibility of using SPSA as a global optimisation method and combining it with a local method, in this case the gradient-based BFGS method was used. This hybrid method was shown to improve the design accuracy, as well as reducing the computational cost significantly, provided an appropriate switchover point (from SPSA to BFGS) was chosen. They also explored the feasibility of parallelising an SPSA method in the inverse design

of a transonic airfoil shape [91], and the comparisons to the parallel simulated annealing method developed by Wang and Damodaran [92] showed that the parallel SPSA method is more computationally efficient when eight or more processors were used.

Another application of SPSA is parameter tuning for models and algorithms. Kothandaraman and Rotea [93] utilised SPSA to determine the set of parameters for a nonlinear dynamic parachute model that would minimise the prediction error by comparing the model output with experimental test data, or otherwise known as the prediction error method (PEM). The algorithm was implemented using MATLAB commands and results showed that this SPSA-based method of parameter tuning has considerably reduced both the mean and the standard deviation values of the objective function, which indicated that the method was successful.

Optimisation of Guidance and Control (G&C) algorithms is another example of parameter tuning using the SPSA algorithm. Work by Palumbo *et al.* [94] and Reardon *et al.* [95] both focused on the integration of G&C functions in a missile system, the model of which is highly nonlinear with complex, noisy outputs, and both teams deemed the SPSA algorithm to be suitable for this task. Simulation results attested to the capability of the SPSA algorithm to substantially reduce the objective function – approximately a 20% reduction with a 10% improvement in the performance measure of mean distance for the work by Reardon *et al.*, and an improvement of 40-50% in mean distance and 40-70% in miss standard deviation for the work by Palumbo *et al.* It was suggested by Reardon *et al.* [95] that a hybrid algorithm, consisting of a global optimisation method (i.e. GA or simulated annealing) and SPSA for local convergence, maybe be able to explore the search space more efficiently and thoroughly.

In an application that is relevant to aircraft path/mission planning, SA methods has been considered by many as useful in solving general *adaptive optimal tracking problems*, also known as *Partially Observed Markov Decision Processes* (POMDP) [96]. Singh *et al.* [97] extended previous work on this application and the optimisation of an observer trajectory planning (OTP) problem for a bearings-only tracking (BOT) application using the SPSA algorithm was investigated. This problem involves tracking a manoeuvring target based on noise-corrupted measurements of the target's state that are received by a

moving observer, and the optimisation goal is to improve the quality of the target state observations by appropriately positioning the observer relative to the target during tracking. An SA algorithm using an *adaptive control variates method* for gradient estimation was introduced and implementation results showed that convergence of the observer trajectory to the optimised solution was rapid and that the variance of the objective function was reduced by two orders of magnitude.

It can be seen that SA methods, in particularly the SPSA algorithm, are suitable for the optimisation of noisy complex nonlinear systems in, but not limited to, the above-mentioned applications and the significant improvements can be observed in the objective functions. However, the suggestion by many of the possibility of combining SA methods with another optimisation method in order to explore the search space more efficiently and thoroughly indicated that the underlying weaknesses of SA methods, namely its slow rate of convergence and possible convergence to a local optimum, are still of concern.

2.5 Response Surface Methodology

Introduced by Box and Wilson [98] in the early 1950s, Response Surface Methodology (RSM) is a collection of mathematical and statistical techniques that is widely used for the approximation and optimisation of stochastic functions [99]. These techniques constitute a well-known approach for constructing approximation models – known as *empirical models*, *metamodels* or *response surfaces* – based on either physical experiments, computer simulations [100, 101] and experimented observations. The objective function is sampled at designated points close to the current iterate, and regression analysis is performed to fit a polynomial of appropriate degree to the response of the system of interest [102].

The application of RSM to simulation optimisation falls into two main categories - metamodels and sequential procedures, with the latter being the commonly adopted approach in the context of optimisation and provide a general methodology for simulation optimisation[103]. This approach treats the simulation model as a black box, where no gradient information is available [104]. Instead of exploring the entire solution space, RSM explores a sequence of small *sub-regions* which are selected for their potential

improvement to the optimisation process. Through a two-phase process, RSM uses first-order polynomial response surface to find an optimal region (**Phase 1**), where a “final” (usually second-order) polynomial is fitted and the optimum determined using deterministic means (**Phase 2**). Each of these phases are briefly described below and summarised in Figure 2.3; for a detailed overview, see Gonda Neddermeijer *et al.* [105].

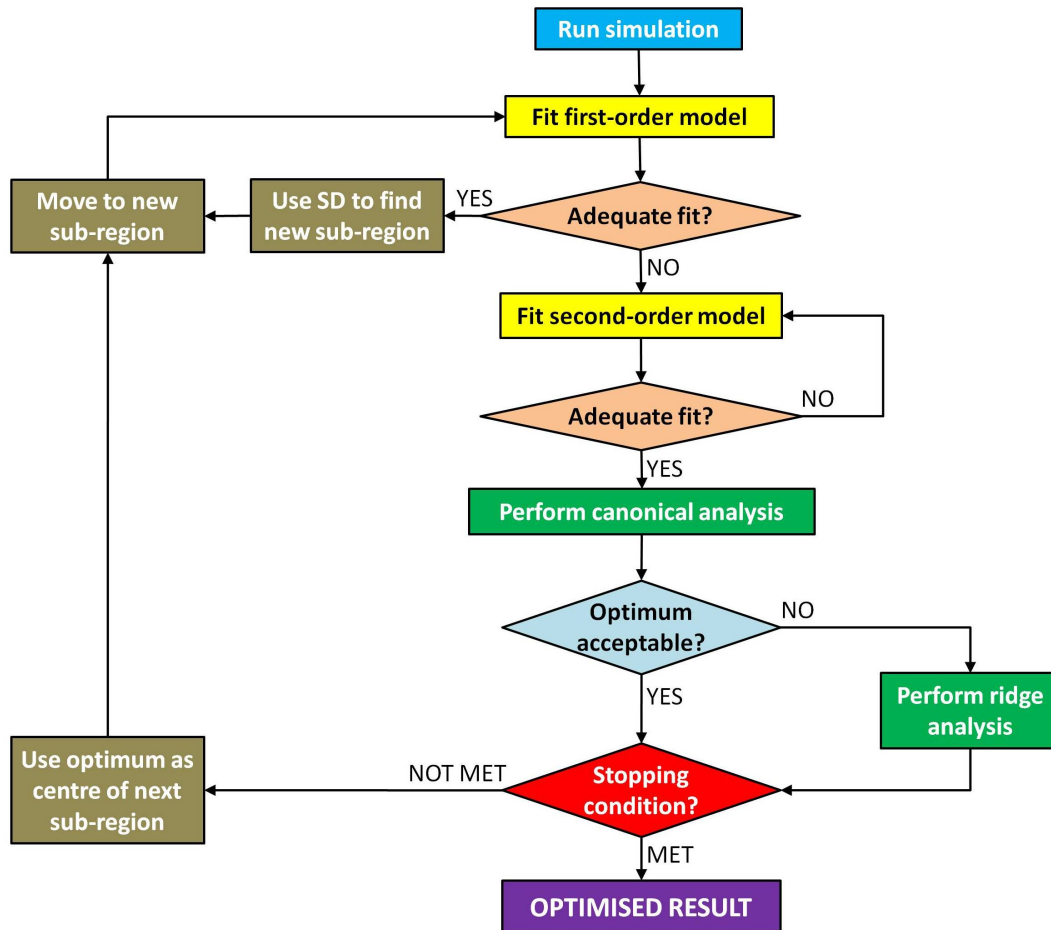


Figure 2.3: A summary of the RSM process when applied to simulation optimisation.

In Phase 1, the simulation model is evaluated a number of times at a specific arrangement of points within a sub-region and using these responses, the response surface, y , of this sub-region is approximated using linear regression as a first-order polynomial of the form

$$y = \beta_0 + \sum_{i=1}^n \beta_i x_i + \varepsilon \quad (2.14)$$

where x_1, \dots, x_n denote the set of decision variables at each of these evaluation points

and β_0, \dots, β_n denote the regression coefficients. The residual error, ε , is assumed to be normally, identically and independently distributed (NIID) with zero mean ($\mu_\varepsilon = 0$) and constant variance σ_ε^2 .

This linear model is put through an analysis of variance (ANOVA) to test if it adequately describes the behaviour of the response in the current sub-region. If the model is deemed adequate, then a steepest descent (SD) direction is estimated from this model, and a new sub-region is chosen to be explored via

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla J(\mathbf{x}_k) \quad (2.15)$$

where \mathbf{x}_k is the representative point (usually the centre point) of the k th explored sub-region, α_k is a step size determined by a line search or some other means, and $\nabla J(\mathbf{x}_k)$ is the estimated (from the fitted linear model) gradient direction at \mathbf{x}_k . This process is repeated until the first-order polynomial approximation becomes inadequate, which is indicated when the gradient of the approximated response surface is “approximately” zero, implying that the sub-region encompasses the optimal point and a polynomial of a higher order is required for appropriate fitness [42]. Then RSM moves to Phase 2.

Phase 2 involves approximating the response surface at the most recent iterate from Phase 1 using a second-order polynomial in the form of

$$y = \beta_0 + \sum_{i=1}^n \beta_i x_i + \sum_{i=1}^n \sum_{j=1}^n \beta_{i,j} x_i x_j + \varepsilon \quad (2.16)$$

Once fitted, this approximate model is again tested for adequacy. Unlike in Phase 1, where a polynomial of a higher order is used if the linear model is found to be inadequate in describing the response surface, it is not customary to use a polynomial that is of a higher order than a second-order [106]. Therefore regression is to be carried out again, but using a smaller sub-region [107] or a larger simulation size. If the quadratic model is found to be adequate, canonical analysis is performed to determine the location and the nature of the stationary point of this model. In the case that the stationary point is a maximum or a saddle point, a new nearby stationary point that is a minimum is determined via ridge analysis.

Once a minimum is found, it is first compared to some stopping criterion (i.e. no significant improvement in the estimated optimal simulation response value, the sub-region becomes too small, or a fixed maximum number of evaluations reached) for the entire optimisation process. If the stopping criterion has not been reached, this minimum acts as the representative point of this current sub-region, and Equation 2.15 is employed to find the next sub-region, and the process returns to Phase 1.

RSM is well-established and simple to implement. It is broadly applicable in the sense that its integration into both stochastic and deterministic simulations can be easily accomplished, since RSM does not necessarily exploit the stochastic structure of the simulated model. For this reason, its generality and flexibility are its biggest advantages [103, 108]. It is expected that RSM would perform better when optimising low-order nonlinear objective functions, therefore it is best suited for small scale applications, i.e. with less than 10 decision variables [109].

However, RSM suffers from the disadvantage of being computationally expensive, particularly when the number of decision variables is large [42, 102]. In order to obtain a better approximation of the objective function, replications with a smaller sub-region is required to provide more accurate information, or by using higher order polynomials, both of which lead to significant increases in computational time [42, 109]. Therefore, RSM is often coupled with other techniques or analyses, such as efficient gradient estimation techniques, based on the nature of simulation model, which may improve the efficiency of RSM [103]. Additionally, because RSM is similar to gradient-based approaches, it may not necessarily find a global optimum [108]. Also, the use of a SD line search technique in the optimisation process introduces two well-known problems – that SD is scale dependent and that the step size α_k along the SD path is selected intuitively [110], therefore application of RSM requires careful planning and design.

Applications of RSM in the area of path planning are, to the best of the author's knowledge, limited. Kewlani *et al.* employed a stochastic RSM (SRSM) method in conjunction with a rapidly exploring random tree (RRT) algorithm in the generation of paths on uncertain and uneven terrains. Their simulation results of this method showed significantly better computational efficiency than a Monte Carlo implementation of the same problem.

2.6 Simulated Annealing

Simulated Annealing is a gradient search method that attempts to find a global optimum by emulating the process of annealing [111]. During the annealing process, a crystalline solid is firstly heated to melting point and then cooled, enabling the material to form into the preferable structure by controlling the temperature change as it cooled [112].

As discussed by Kirkpatrick *et al.* [113], the simulated annealing algorithm starts with a randomly-chosen initial solution and moves from one solution to the next, attempting to converge on the global optimum. simulated annealing avoids being trapped in a locally optimum region by accepting inferior solutions with a probability that corresponds to the degradation of the performance measure. A generic simulated annealing algorithm is shown in Figure 2.4 [1]:

```

Initialise  $x$  to  $x_0$  and  $T$  to  $T_0$ 

loop – Cooling
  loop – Local search
    Derive a neighbour,  $x'$ , of  $x$ 
     $\Delta E := E(x') - E(x)$ 
    if  $\Delta E < 0$ 
      then  $x := x'$ 
    else derive random number  $r \in [0,1]$ 
      if  $r < e^{-\frac{\Delta E}{T}}$ 
        then  $x := x'$ 
      end if
    end if
  end loop – Local search
  exit when the goal is reached or a pre-defined stopping condition is satisfied
   $T := C(T)$ 
end loop – Cooling

```

Figure 2.4: A generic Simulated Annealing algorithm [1].

When the simulated annealing algorithm is executed, the temperature is decreased in stages, and at each stage the temperature is kept constant until thermal quasi-equilibrium is reached. This is controlled by the cooling schedule, or the set of parameters that determine the temperature reduction (i.e. initial temperature, stop criterion, temperature decrement between successive stages, number of transitions for each temperature value), which is critical to the success of the optimisation [1] and must be determined a priori [24].

Discussions of simulated annealing by Van Laarhoven and Arts [114] and Arts and

Korst [115] present experimental evidence that the algorithm is capable of finding near optimal results for both linear and non-linear problems. Alrefaei and Andradóttir [116] developed a method for simulation optimisation which approximates the objective function of a simulation model and uses simulated annealing to solve the objective function. However, their underlining assumption is that the system can be modelled as a stochastic function or a Markov chain, whereas one of the major reasons for developing simulation models is the inability to model the system by other techniques.

Simulated annealing methods have the capability to optimise multi-objective problems. Alrefaei and Diabat [117] developed an simulated annealing-based method with constant temperature for multi-objective optimisation (MOO) and applied it on a multi-objective (s, S) inventory model. Their results show significant speed in converging to the optimum than the variable-temperature SA algorithms that were used as comparison.

Computational costs are a major concern when it comes to implementing simulated annealing algorithms [118]. Haddock and Mitenthal [119] demonstrated with their approach that simulated annealing can solve simulation problems with integer variables, using a flexible manufacturing system design problem as a case study. But the CPU time for large problems can be extremely long. A modified simulated annealing algorithm that can converge to the global optimum with a finite number of iterations was developed by Alkhamis *et al.* [120]. However, this number can be extremely large and is not feasible for many simulation problems. For more detailed simulation models, the required computational time to determine an improved solution will be a major problem.

Application of simulated annealing algorithms in the area of path planning has been extensively investigated. Miao and Tian [121] employed simulated annealing as the core algorithm in their research in path planning for robots in a dynamic environment. This was extended by Miao [122] into a multi-operator based simulated annealing technique, which was demonstrated through case studies to be effective and efficient in both off-line and on-line processing of robot dynamic path planning. A similar situation for an aircraft was tackled by Kastella [123] using an adaptive simulated annealing method. The adaptive cooling technique based on equilibration and relaxation times was found to reduce the sensitivity of the algorithm and increase the computational speed by a factor of approximately 10 with no loss in quality. Meng and Xin [124] developed a

genetic simulated annealing algorithm for UAV route planning and, through simulations, showed that the genetic simulated annealing algorithm was more computationally efficient compared to the normal GA.

2.7 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a family of population-based metaheuristic optimisation algorithms that use biology-inspired mechanisms (e.g. mutation, crossover, natural selection and survival of the fittest) in order to refine a set of solution candidates iteratively [125, 126]. As opposed to a single solution used in traditional methods, EAs work on a population of solutions in such a way that poor solutions become extinct, and good solutions evolve to reach for the optimum.

The five members of the EA family and their differences, mainly in the representation of a *candidate solution*, are as follows [118, 127]:

1. **Genetic Algorithms (GAs):** The most popular member of the EA family, GAs were developed by John Holland [128] and works with candidate solutions in the form of bit (binary, 0 or 1) strings; focuses on optimizing general combinatorial problems;
2. **Evolution Strategies (ES):** Approaches that use vectors of real numbers as candidate solutions and focuses on optimising continuous functions with recombination;
3. **Genetic Programming (GP):** An approach that includes all EAs that grow programs, algorithms, etc., and those that evolve tree-shaped individuals;
4. **Learning Classifier Systems (LCS):** Online learning approaches that use a GA to find new rules of mapping to assign output values to given input values; and
5. **Evolutionary Programming (EP):** An approach that treats instances of the *genome*, or the search space, as different species instead of as individuals; focuses on optimising continuous functions without recombination; one that has more or less merged into GP and other EAs over the decades.

An EA-based optimisation process begins with the generation of a set of *individuals* (potential solutions) called the *population*. This population is then put through the

reproduction-evaluation cycle which evolves the initial population towards a population that is expected to contain the optimal solution. For each *generation* (iteration) of the cycle, individuals from the current population are selected to form the mating pool. This selection of individuals is based on their *fitness* (computed from the objective function that is to be optimised) with respect to the current population such that the stronger individuals will have a higher probability of being in the mating pool. Individuals in the mating pool are submitted into the process of *reproduction*, during which these individuals are subjected to *mutation* (introduction of innovation into the population) and *recombination* (mixing of parental information to pass to their descendants, or *offspring*). These operations allow different regions of the search space to be explored, preventing the process from getting trapped at a local optimum. This cycle is iterated, evolving the population toward better and better regions of the search space, until a stopping condition has been met. A summary of the EA cycle is shown in Figure 2.5.

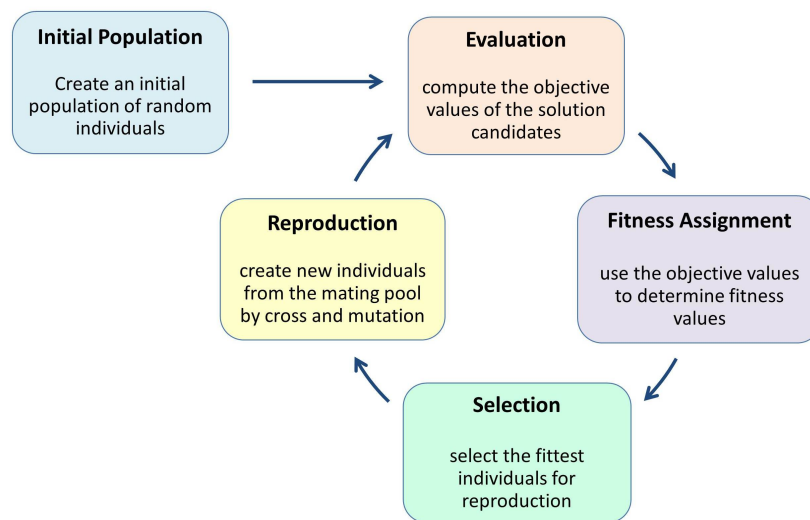


Figure 2.5: A basic cycle of EAs.

Because EAs are stochastic and often there are no guarantees that an optimum will be reached, stopping conditions are often required to stop the EA from continuing for a long time. Common stopping conditions are [129]:

- Maximum allowed computational time has elapsed;
- The total number of fitness evaluations has reached a given limit;
- The fitness improvement has remained under a threshold value for a given period of time; and

- The population diversity has dropped under a given threshold.

In recent years, there has been an increasing interest in using EAs in simulation optimisation because they require no restrictive assumptions or prior knowledge about the shape of the response surface [24]. Most of this attention has been focused on GAs, especially in optimisation problems arising from complex manufacturing and distribution systems [112]. A GA-based simulation optimisation method was developed by Kochel and Nielander [130] to optimise problems related to Kanban manufacturing systems. However, this uses steady-state simulation to compute measurements and such a state is hard to reach for most manufacturing environments. McWilliams *et al.* developed a GA-based simulation approach to solve a distribution centre scheduling problem which showed improvements to the solutions [131]. The main concern is that the population needs to be evaluated by simulation every single generation, which is computationally very expensive for large-scale simulation models.

An example of using an ES-based simulation optimisation method was developed by Hall *et al.* [132] to solve the kanban sizing problem for a manufacturing system with 39 decision variables. Insights were gathered on the effects of the number of decision variables on the population size and the fitness of the solutions.

The use of EAs in simulation optimisation will require the linking of an EA to the simulation model and this can be challenging since the two components are rarely coded in the same language. Once they are connected, the role of the EA is to generate individuals for the simulation model to evaluate, and the model will return the fitness of each individual to the EA to continue on with the reproduction-evaluation cycle. Since multiple individuals are selected in each generation, several simulation runs will be required to evaluate these individuals. If the simulation model is complex and requires a long runtime for each evaluation, or if many replications are necessary, the computational costs of carrying out this process can be expensive. Combining this with the notoriety of EAs being very slow, parallelising approaches such as distributed EAs (DEAs) have been developed [29], in which several processors are used to carry out several simulations simultaneously. This can drastically reduce the computing time, as shown by the ES-based simulation optimisation method Periaux *et al.* have developed for airfoil design [133] and the GP-based simulation optimisation method developed by Núñez *et al.* [134].

EAs are well-known for their ability to tackle multi-objective optimisation. Unlike traditional optimisation methods that generate only one solution at the end of the optimisation process, EA-based optimisers generate many *Pareto-optimal* solutions, each with a different trade-off among the multiple objective functions [135]. Eskandari *et al.* [136] presented a GA-based multi-objective simulation optimisation method, and the previously-mentioned ES-based optimiser by Periaux *et al.* tackled the multi-objective optimisation problem of airfoil design successfully [133] with the design and implementation of the HAPMOEA optimiser, which is described in §2.7.1.

There has been extensive research in the application of EA-based algorithms in the area of path planning. Previously, because of the indeterministic nature of EAs, applications in path or trajectory planning have been focused on off-line optimisation; see Nikolos *et al.* [137], Nikolos *et al.* [138], Sanders and Ray [139], Pohl and Lamont [140], Mittal and Deb [135] and Rathbun *et al.* [141]. Lately, EAs are gradually being employed in on-line path planning of UAVs, usually to refine a flight path that had been generated off-line; see Kostaras *et al.* [142], Jia and Vagners [143], Pongpunwattana and Rysdyk [144] and Basada-Portas *et al.* [145].

2.7.1 HAPMOEA Optimiser

The Hierarchical Asynchronous Parallel Multi-Objective Evolutionary Algorithm (HAPMOEA) optimiser is an optimisation tool developed by Gonzalez *et al.* [146] which uses the following concepts for the solutions:

- A modified canonical evolution strategy;
- MOO using a Pareto tournament selection;
- A hierarchical/multi-fidelity approach to the solution; and
- Parallel computation using a modified asynchronous approach.

Canonical Evolution Strategy

In a canonical evolution strategy, a pseudocode of which is illustrated in Figure 2.6, a population μ_0 is firstly initialised and evaluated. Then while a stopping condition – maximum number of function evaluations, target fitness value, maximum amount of

computational time, for example – is not met, the algorithm executed for a number of generations g , during which offsprings λ^{g+1} go recursively through the process of recombination, mutation, evaluation and selection.

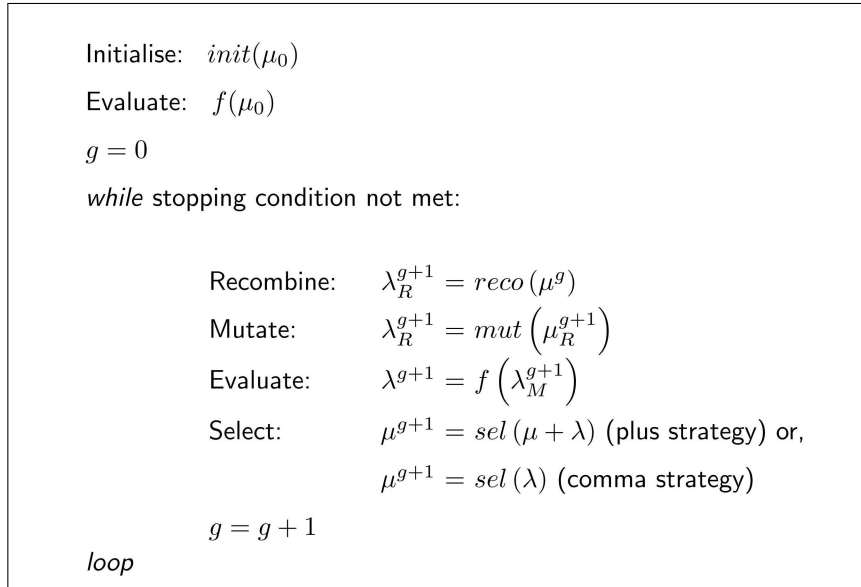


Figure 2.6: A canonical evolution strategy.

Multi-Objective EAs (MOEAs)

Most real world problems involve conflicting objectives, to which there exists no unique optimum, but a set of compromised individuals known as Pareto optimal solutions, or non-dominated individuals. A solution to a multi-objective problem is considered Pareto optimal if there is no other solutions that satisfy better all the objectives simultaneously. The Pareto optimality of a problem with two conflicting objectives is shown in Figure 2.7. The Pareto set is the set of Pareto optimal solutions that trade off the information among the conflicting objectives.

EAs are capable of finding a number of solutions in a Pareto set, since they evaluate multiple populations of points. Pareto selection ranks the population and selects the non-dominated individuals for the Pareto front. The HAPMOEA algorithm has been proven, through various multi-objective test cases, to be robust and efficient to find the optimal Pareto front [147].

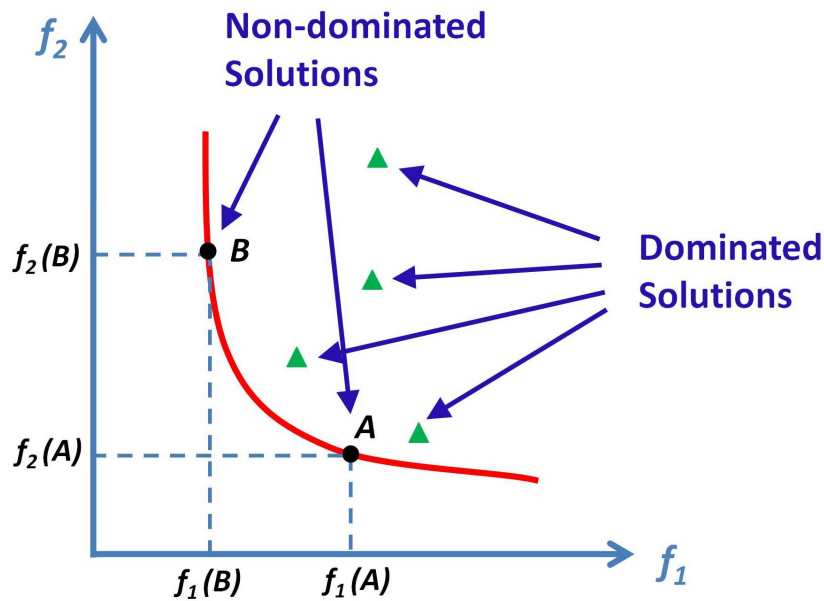


Figure 2.7: Pareto optimality.

Hierarchical Topology

The HAPMOEA algorithm is designed to handle multiple fidelity models for the solution [148], a representation of this formulation is shown in Figure 2.8. The bottom layer can be entirely devoted to exploration of the search space, the intermediate layer is a compromise between exploration and exploitation, while the top layer focuses on refining the solutions. This is achieved through the use of a very precise model in the top layer, resulting in a time-consuming solution. And since the purpose of the sub-populations of the bottom layer is to explore the search space, simple models with fast numerical solvers that do not yield precise results can be utilised.

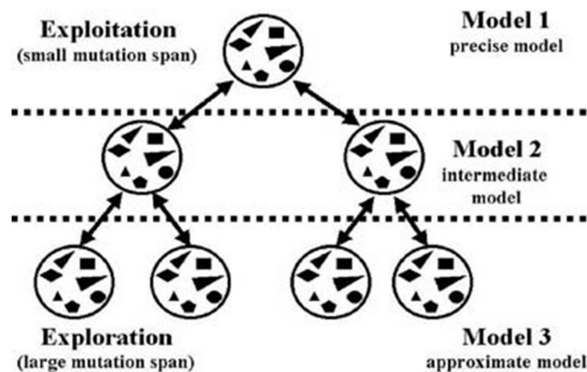


Figure 2.8: Hierarchical topology.

Parallel Computing and Asynchronous Evaluation

Parallelised computing of EAs can be achieved by sending individuals to remote machines to be evaluated, and then incorporated back into the optimisation process [149–151]. A network of computers can be used to parallelise the optimisation, using the Parallel Virtual Machine (PVM) [152] as the message-passing model within the network. As the canonical ES usually evaluates entire populations simultaneously, this parallel implementation requires modifications to the canonical ES [153, 154].

An asynchronous method distinctively differs from traditional EAs by generating only one candidate solution at any one time and only re-incorporates one individual at any one time [155], therefore solutions can be generated and returned out of order. This is solved with the implementation of an asynchronous fitness evaluation, which is illustrated in Figure 2.9.

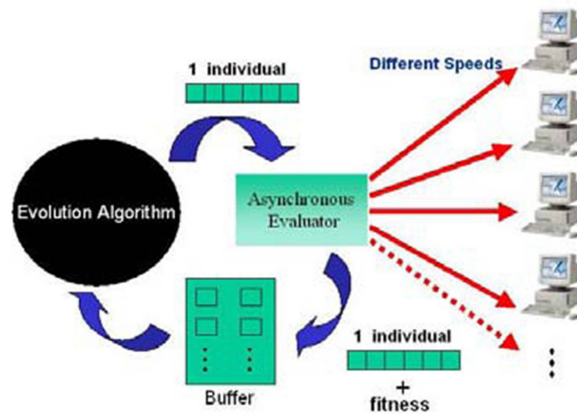


Figure 2.9: Parallel computing and asynchronous evaluation.

The HAPMOEA Optimiser is used in the MWO, details of which are in §4 and §5.

2.8 Summary

The aim of this literature review was to determine the most appropriate method of implementing the task of Mission Waypoint Optimisation for a small fixed-wing UAV.

In order for any optimisation process to be carried out, a good representation of the system to be optimised, in this case a UAV, is vital. Therefore constructing a valid representation of a small fixed-wing UAV is required. However, because the performance

of an aircraft, regardless of its size, is dependent and affected by a myriad of factors, most of which are interrelated, this multidisciplinary nature makes it impossible for an analytical model of an UAV to be constructed. As a result, the UAV will be represented by a simulation model. Additionally, the optimisation of a set of mission waypoints is a multi-objective problem. Consequently, the use of a MOO method that is able to work with a simulation model is required. One suitable method is the method of simulation optimisation.

The investigation of simulation optimisation and the five common optimisation methods - SQP (§2.3), SA (§2.4), RSM (§2.5), simulated annealing (§2.6) and EA (§2.7) - showed that SA is not suitable for MWO, because the MWO problem is associated with many constraints, but incorporating constraints into the SA process adds much difficulty. As a gradient-based method, SQP is potentially less computationally expensive than the other methods, which is definitely an advantage. Even though RSM, simulated annealing and EA have demonstrated that they work well with a simulation model and all can be used for MOO problems, their common drawback of high computational costs can be problematic. However for EAs, this can be commonly overcome with the use of distributed EAs.

In completing this review, Research Objective 1, specified in §1.2, was fulfilled.

To summarise, the development of a MWO procedure for a small fixed-wing UAV, with the focus on increasing the onboard energy efficiency, would require the completion of the following tasks:

1. Construction of a simulation environment of a small fixed-wing UAV (§3); and
2. Coupling of the constructed UAV simulation model with a SQP solver (§2.3.1) and a Multi-Objective EA (MOEA) optimiser (§2.7.1) to form a MOEA simulation optimisation method for MWO.

The details of each of these tasks are described in the chapters and sections as indicated.

Chapter 3

UAV Simulation Environment

3.1 Introduction

As mentioned in Chapter 1, the purpose of this research was to investigate methods of increasing energy efficiency onboard a small fixed-wing UAV. One way of achieving this is to develop a MWO procedure for a small fixed-wing UAV, with the focus on increasing the onboard energy efficiency. In Chapter 2, it was established that two tasks are required:

1. Construction of a simulation environment of a small fixed-wing UAV; and
2. Coupling of the constructed UAV simulation model with a SQP solver and a Multi-Objective EA (MOEA) optimiser to form a MOEA simulation optimisation method for MWO.

This chapter presents the development and implementation of a UAV simulation model. The simulation environment is presented in §3.2 and detailed descriptions of essential components inside the simulation model are given in §3.3. Two mission scenarios have also been designed in §3.4, so that the simulation of an UAV in flight can be conducted.

3.2 Simulation Software

Computer simulations are commonly used nowadays as actual flight tests are often extremely time-consuming as well as cost-prohibitive. A UAV Simulation Model (UAVSM)

was developed to simulate the operation of a fixed-wing UAV in flight.

The MATLAB Simulink environment was chosen for the UAVSM to be implemented in, due to previous MATLAB experience and the ease of simulating using Simulink's graphical block approach.

The AeroSim Blockset [156] is used to develop the UAVSM. A commonly used aircraft simulation and analysis package in the aerospace industry [157], the AeroSim Blockset provides a comprehensive set of tools for development of non-linear 6-degrees-of-freedom (6DOF) aircraft models. The AeroSim Blockset also offers a detailed set of parameters on the Aerosonde (see Appendix A), a small real-world fixed-wing UAV, which can be incorporated into the aircraft model to simulate an Aerosonde in flight. The Aerosonde UAV block is shown in Figure 3.1.

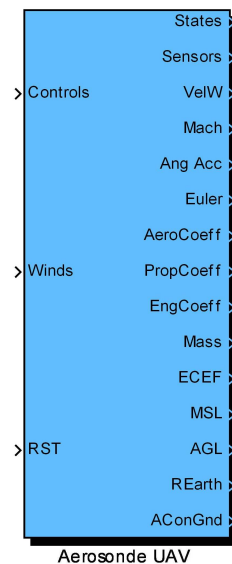


Figure 3.1: The Aerosonde UAV block, modified from the AeroSim 6DOF aircraft model.

Inside the Aerosonde UAV block is a detailed model of the inner workings of an Aerosonde UAV, which consists of Aerodynamic, Propulsion, Atmosphere, Aircraft Inertia, Acceleration and Moments, Equations of Motion and Earth models, as shown in Figure 3.2. The inputs into the Aerosonde UAV block, on the far left in Figure 3.2, are the aircraft control inputs (control surfaces, throttle, mixture, ignition), and wind velocities. The outputs of the block, on the far right, are the aircraft states in various coordinate systems and aircraft coefficients. Many of these output values are used in the calculations of other blocks in the UAVSM (see §3.3), while some are displayed for reference purposes, passed to FlightGear for visualisation through the FlightGear 0.9.8 Interface (this option

was not utilised in this research), or passed to MATLAB for further computations.

3.3 UAV Simulation Model

The complete UAVSM, as shown in Figure 3.3, is constructed around the Aerosonde UAV block as described in §3.2. The Aircraft Control Module (ACM) (§3.3.2) and Flight Planner Module (FPM) (§3.3.1) were developed and added to the Aerosonde UAV block to simulate unmanned operations. These two blocks are circled in red in Figure 3.3.

3.3.1 Flight Planner Module

The Simulink model of the FPM is presented in Figure 3.4.

The main component inside the FPM is a MATLAB function, *NavigateWaypoints* (Appendix B), which carries out the task of waypoint navigation. Given a set of waypoints in their GPS coordinates (latitudes, longitudes and altitudes), *NavigateWaypoints* calculates the necessary bearing/yaw adjustment the aircraft is required to make in order for it to fly from its current position to the next waypoint.

The *NavigateWaypoints* function uses a basic navigational algorithm based on the great-circle navigation method [158], which calculates the great circle track, or the shortest distance following the curvature of the Earth, and the bearing between the two points, as shown in Figure 3.5. Thus, the aircraft flies through the series of waypoints by flying a sequence of direct, curved paths from one waypoint to the next. Using spherical trigonometry, the range D and bearing B_T to the target are given by [158]:

$$D = R_G \cos^{-1} (\sin \phi \sin \phi_t + \cos \phi \cos \phi_t \cos(\lambda - \lambda_t)) \quad (3.1)$$

$$B_T = \sin^{-1} \left\{ \frac{\cos \phi_t}{\sin \left(\frac{D}{R_G} \right)} \sin(\lambda - \lambda_t) \right\} \quad (3.2)$$

$$R_G = \sqrt{R_M \cdot R_P} \quad (3.3)$$

where R_G is the Gaussian radius of curvature at the current location, R_M and R_P are

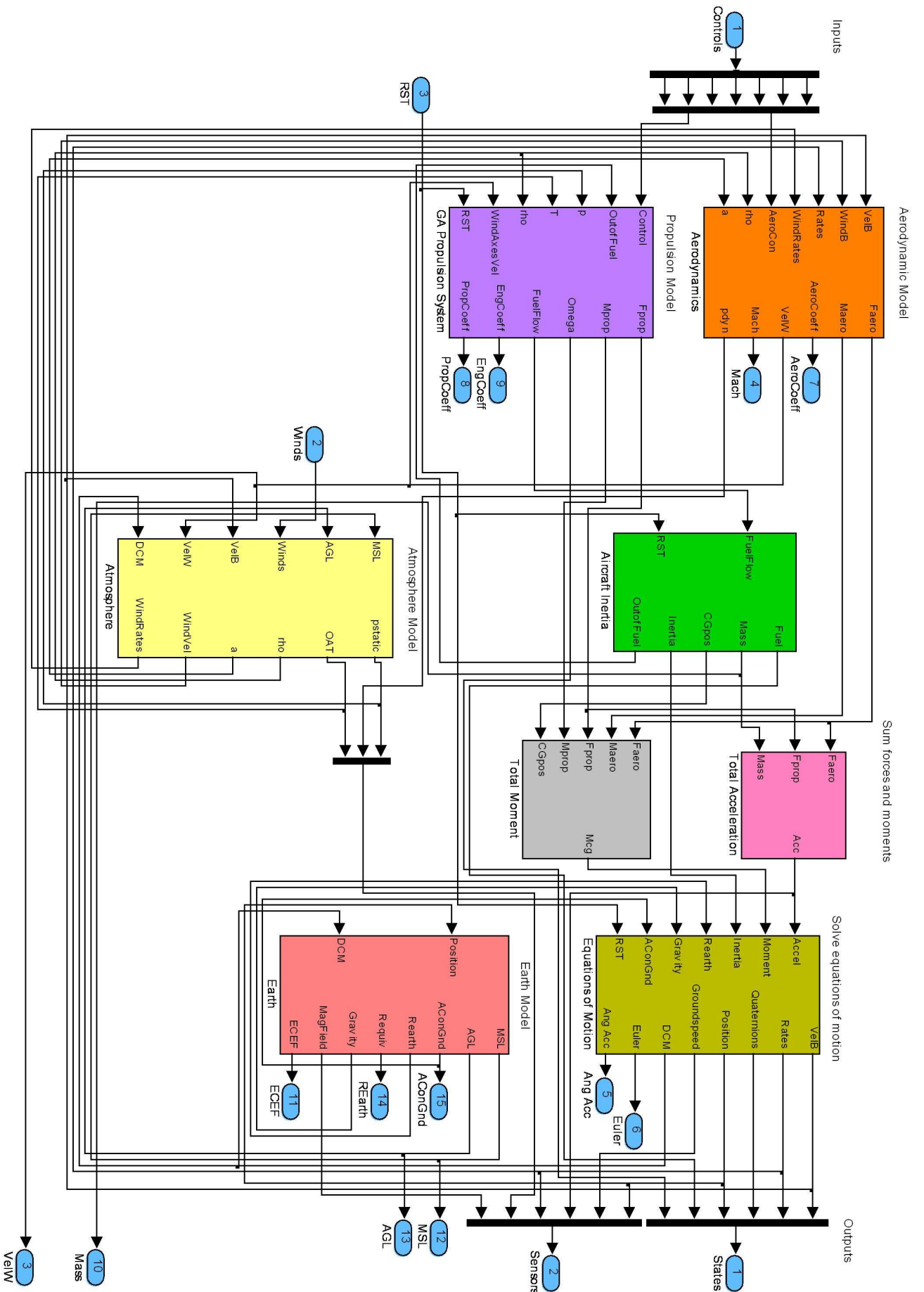


Figure 3.2: Inside the Aerosonde UAV block.

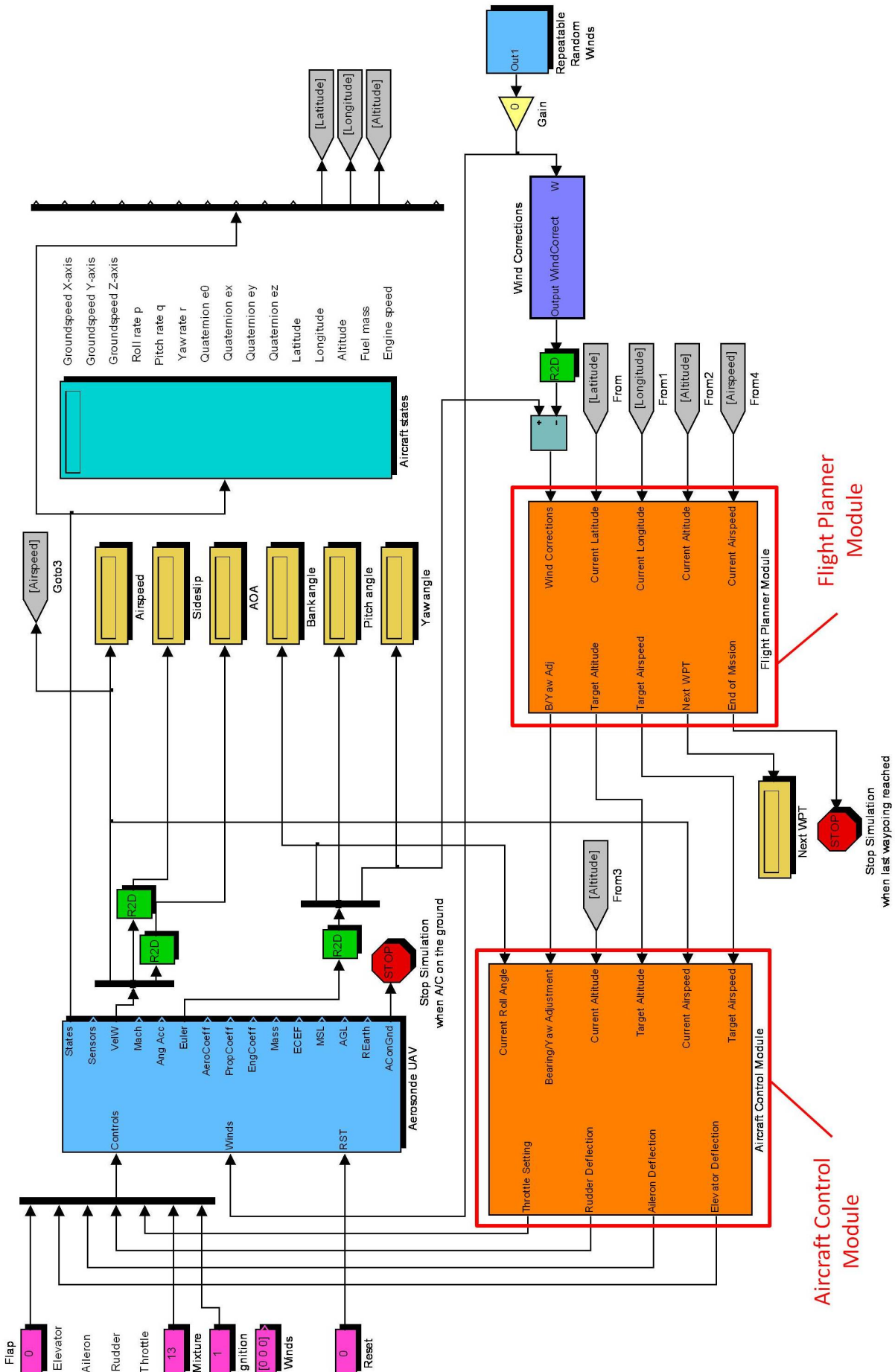


Figure 3.3: The UAV Simulation Model.

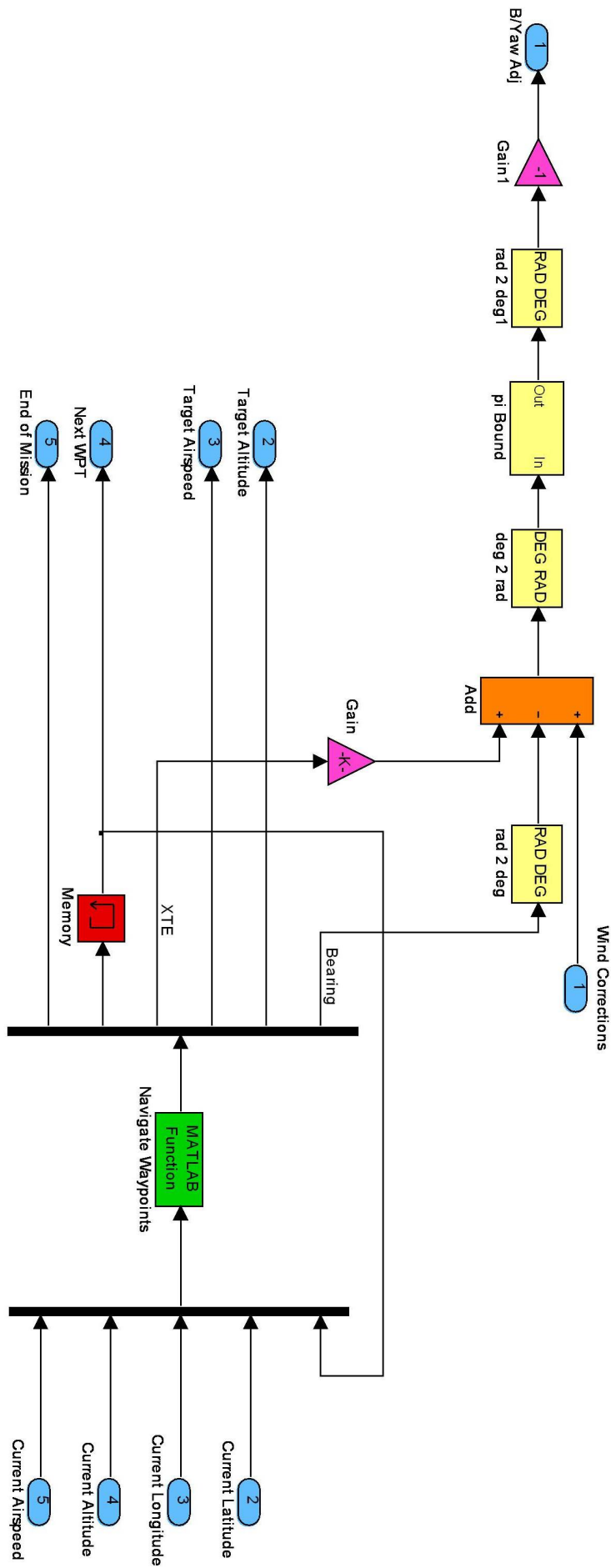


Figure 3.4: Flight Planner Module.

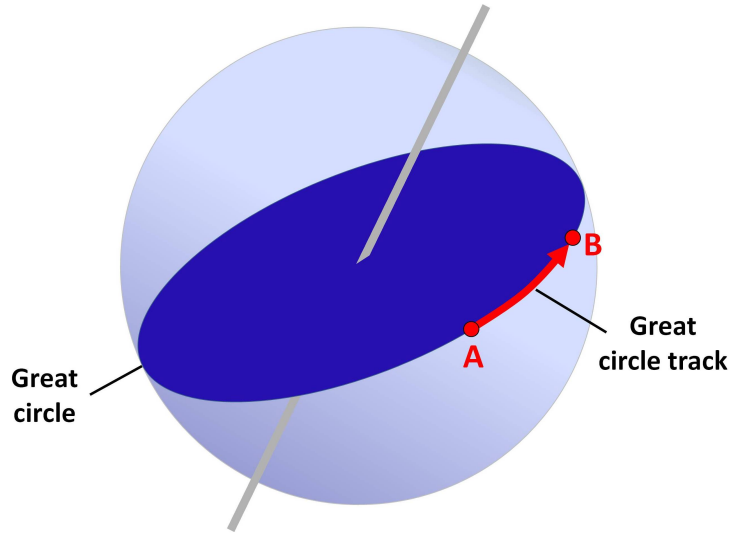


Figure 3.5: Illustrative definition of a great circle track.

the meridian and prime radii, respectively, at the current location, (ϕ, λ) are the latitude and longitude at the currently location, and (ϕ_t, λ_t) are the latitude and longitude at the target point. This calculated distance to target is in fact the *distance-to-go* to the target, since the aircraft position is constantly changing.

Another factor that needs to be considered for the aircraft to follow the flight path set by the series of waypoints is the cross track error, or *XTE*, which is calculated by [159]:

$$XTE = DTR \cdot \sin(\theta_{TE}) \quad (3.4)$$

where *DTR* is the *track distance*, or the distance from the current location to the next waypoint, and θ_{TE} is the angle of track error. This relationship is illustrated in Figure 3.6. Incorporating *XTE* in the waypoint navigation algorithm enables the aircraft position to be constantly adjusted.

The UAV heading command, ϕ_{cmd} , is calculated as follows:

$$\phi_{cmd} = \phi_T - \theta_{Wc} - B_T + k\theta_{TE} \quad (3.5)$$

where ϕ_T is the aircraft's true heading, θ_{Wc} is the wind correction angle, B_T is the true bearing to the target, and θ_{TE} is the angle of track error, multiplied by a constant k . In the case of *NavigateWaypoints* function, the value of k is 1.

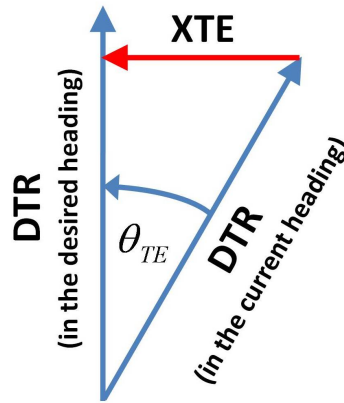


Figure 3.6: Geometric relationship between DTR, θ_{TE} and XTE.

The bearing/yaw adjustment, along with other parameters calculated by *Navigate-Waypoints* are passed to the ACM.

3.3.2 Aircraft Control Module

The ACM, whose details are shown in Figure 3.7, is essentially a collection of displacement autopilots which are used to control the angular orientation of the aircraft [160]. This allows the UAV to “fly”, i.e. adjust the flight controls (control surface deflections and throttle) so that the desired conditions are met, without human intervention, thus enabling unmanned operations onboard the UAV.

A *roll attitude autopilot* is implemented in the ACM with two Proportional-Integral (PI) controllers. This autopilot takes as inputs the *current roll angle* and the *desired bearing/yaw adjustment* as calculated by FPM, and the PI controllers use the difference between these measurements to determine the amount of deflection required for the rudder and the ailerons. The gain values for the PI controllers were obtained through empirical means and are presented in Table 3.1. The outputs of these PI controllers are the *rudder deflection* and the *aileron deflection* in radians, which are passed to the Aerosonde UAV Block as two of its control inputs.

A Proportional-Integral-Derivative (PID) controller is used in an *airspeed hold controller* to maintain the UAV at a constant airspeed of 20m/s (72km/h) when the aircraft is climbing or in cruise, or 30m/s (108km/h) when the aircraft is descending. This change in airspeed command is to ensure that the aircraft would descend at a fast enough rate. The difference between the inputs, *current airspeed* and *target airspeed*, are used by the

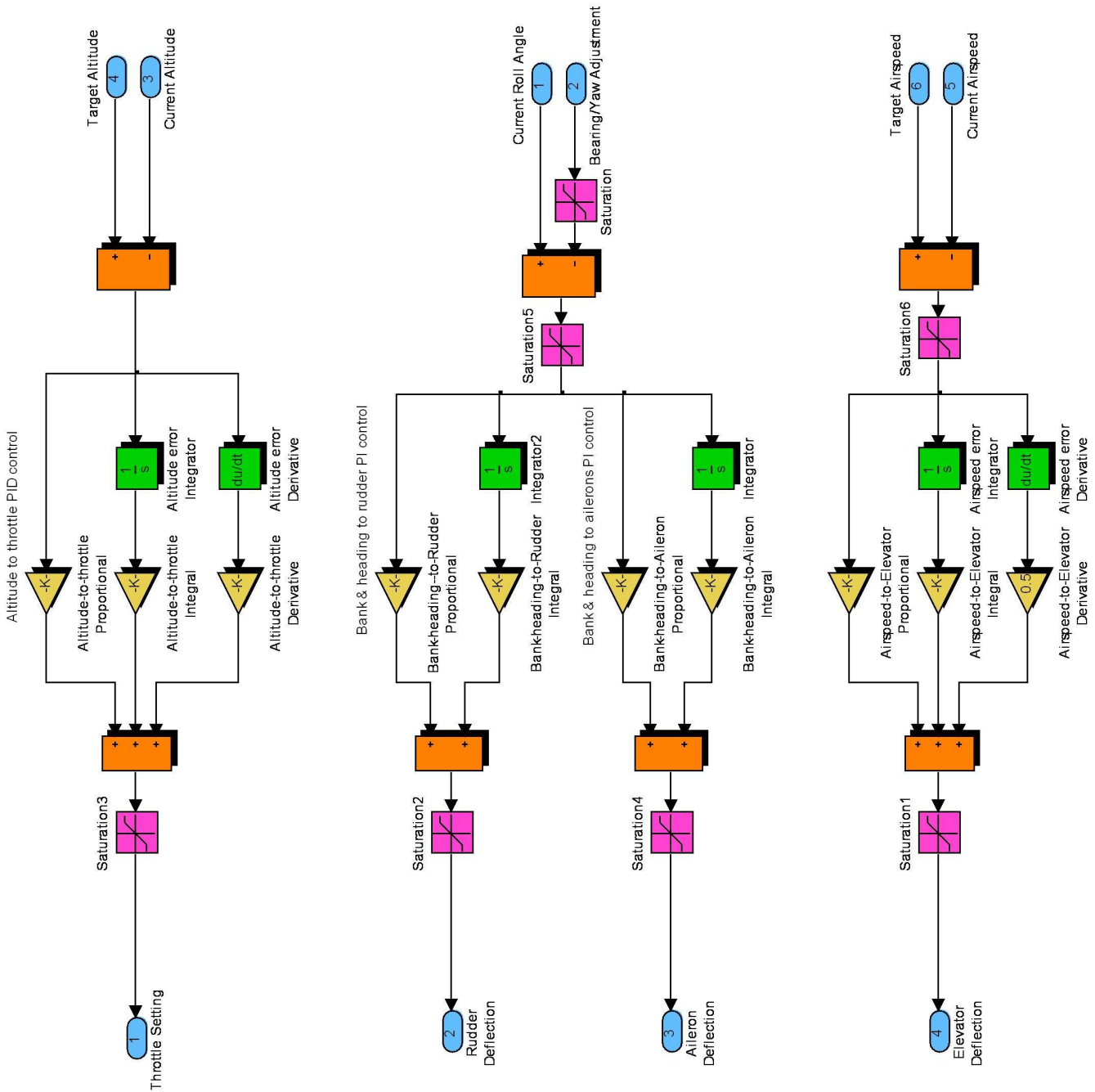


Figure 3.7: Aircraft Control Module.

Control Surface	Controller Type	Gain Value
Bank-to-Ailerons	Proportional	$\frac{0.5\pi}{180}$
	Integral	$\frac{0.05\pi}{180}$
Bank-to-Rudder	Proportional	$\frac{0.3\pi}{180}$
	Integral	$\frac{0.03\pi}{180}$
Airspeed-to-elevator	Proportional	0.08
	Integral	0.01
	Derivative	0.3
Altitude-to-throttle	Proportional	0.15
	Integral	0.000001
	Derivative	0.01

Table 3.1: Gain values for the PI and PID controllers in ACM.

PID controller to determine the *elevator deflection* in radians, which is the output of the controller. Also obtained using empirical means, the gain values for the PID controller are listed in Table 3.1.

The *altitude hold control system* also uses a PID controller to maintain the UAV at the desired altitude according to the pre-specified list of waypoints. The inputs to this controller are the *current altitude* and *target altitude*. The difference between the inputs are passed to the PID controller to determine a corresponding value for the *engine throttle* output, ranged between 0.01 (1%) and 1 (100%). The PID controller gains are obtained empirically and listed in Table 3.1.

Some physical constraints are imposed on the outputs of ACM in the form of *saturation blocks* near the output blocks of each PID or PI controller and these are listed in Table 3.2.

Control	Constraints
Engine throttle	$0.01 \leq TC \leq 1$
Rudder deflection	$-20^\circ \leq \theta_r \leq 20^\circ$
Aileron deflection	$-10^\circ \leq \theta_a \leq 10^\circ$
Elevator deflection	$-20^\circ \leq \theta_e \leq 20^\circ$

Table 3.2: Physical constraints on ACM outputs.

3.3.3 Test Scenario and Results

To validate the UAVSM, a test scenario was constructed with the series of waypoints as shown in Table 3.3. In this flight path, Waypoint 1 is the starting point and Waypoints 2 to 9 constitute the “mission”.

Waypoint	Waypoint Coordinates		
	Latitude (DD:MM:SS)	Longitude (DD:MM:SS)	Altitude (m)
WP1 (Kingaroy Airport)	26°34'51" S	151°50'28" S	800
WP2	26°33'58" S	151°51'10" E	900
WP3	26°34'08" S	151°51'25" E	900
WP4	26°34'16" S	151°53'13" E	750
WP5	26°34'08" S	151°53'18" E	750
WP6	26°33'59" S	151°53'13" E	750
WP7	26°34'08" S	151°51'25" E	900
WP8	26°34'14" S	151°51'17" E	900
WP9 (Kingaroy Airport)	26°34'51" S	151°40'28" S	800

Table 3.3: Waypoints in the test scenario.

This series of waypoints is entered into the UAVSM in the form of a Waypoint Table as shown in Figure 3.8. Each row of this *WPTTable* denotes a waypoint, comprising of its latitude, longitude, target altitude and target airspeed. Note that Waypoint 1 is not part of the *WPTTable*, because it is set as an initial condition of the UAVSM. Each of the latitudes and longitudes, excluding Waypoints 1 and 9 (which are the same point), were kept in the format of DD:MM.MMMM in order to retain as much precision as possible and the conversion into radians was computed when MATLAB inputs the values of this *WPTTable* variable.

```

% Waypoint Table: Mission

% Starting point: KINGAROY [26.5808*pi/180 151.8411*pi/180 800]
WPTTable = [...
(26+33.9722/60)*pi/180 (151+51.1746/60)*pi/180 900 20; ...
(26+34.1315/60)*pi/180 (151+51.4181/60)*pi/180 900 20; ...
(26+34.2729/60)*pi/180 (151+53.2179/60)*pi/180 750 30; ...
(26+34.1322/60)*pi/180 (151+53.3082/60)*pi/180 750 20; ...
(26+33.9915/60)*pi/180 (151+53.2179/60)*pi/180 750 20; ...
(26+34.1315/60)*pi/180 (151+51.4181/60)*pi/180 900 20; ...
(26+34.2358/60)*pi/180 (151+51.2796/60)*pi/180 900 20; ...
26.5808*pi/180 151.8411*pi/180 800 30];

```

Figure 3.8: Waypoint Table in MATLAB code.

Two views of the resulting flight are shown in Figures 3.9 and 3.10. Note that the starting point, Waypoint 1, is located at (0,0).

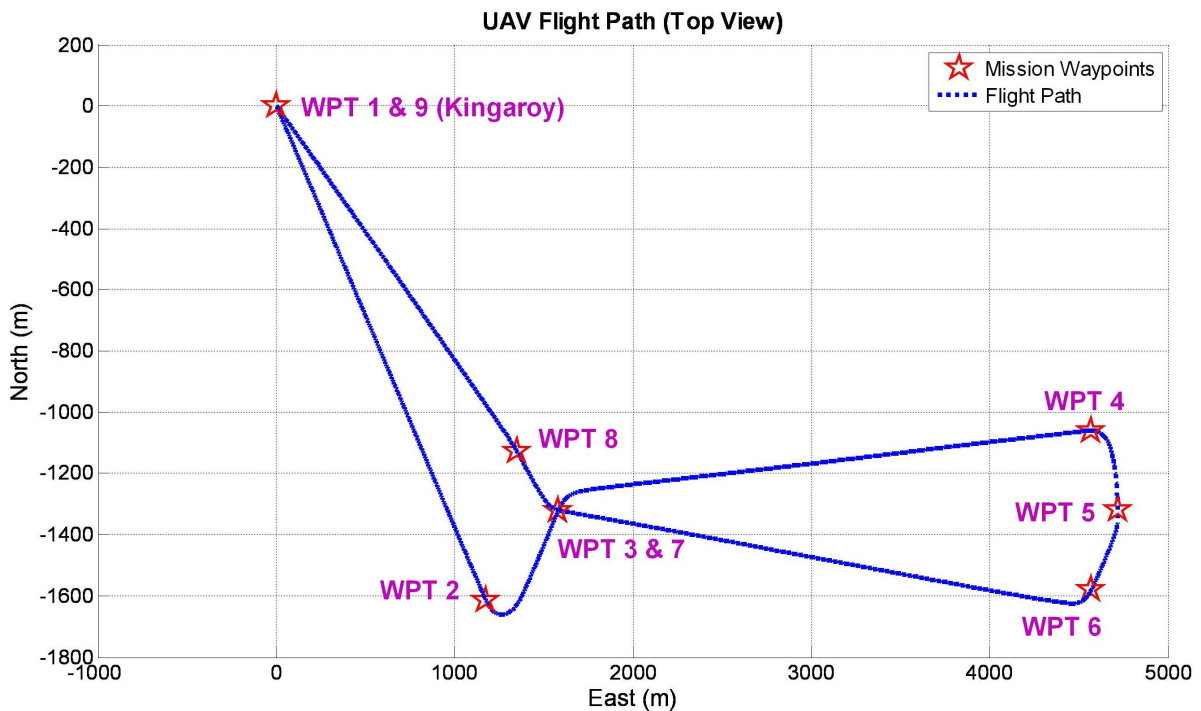


Figure 3.9: Test scenario - top view of flight path.

From both views of the flight path, it can be seen that the UAVSM is able to follow the specified sequence of waypoints in the flight mission quite well. The aircraft was able to “capture” a waypoint, or reach the coordinates (latitude and longitude) of the waypoint, before adjusting its heading towards the next waypoint, thus creating a smooth turn. This is good evidence that the model has no great problems. It can also be observed from the

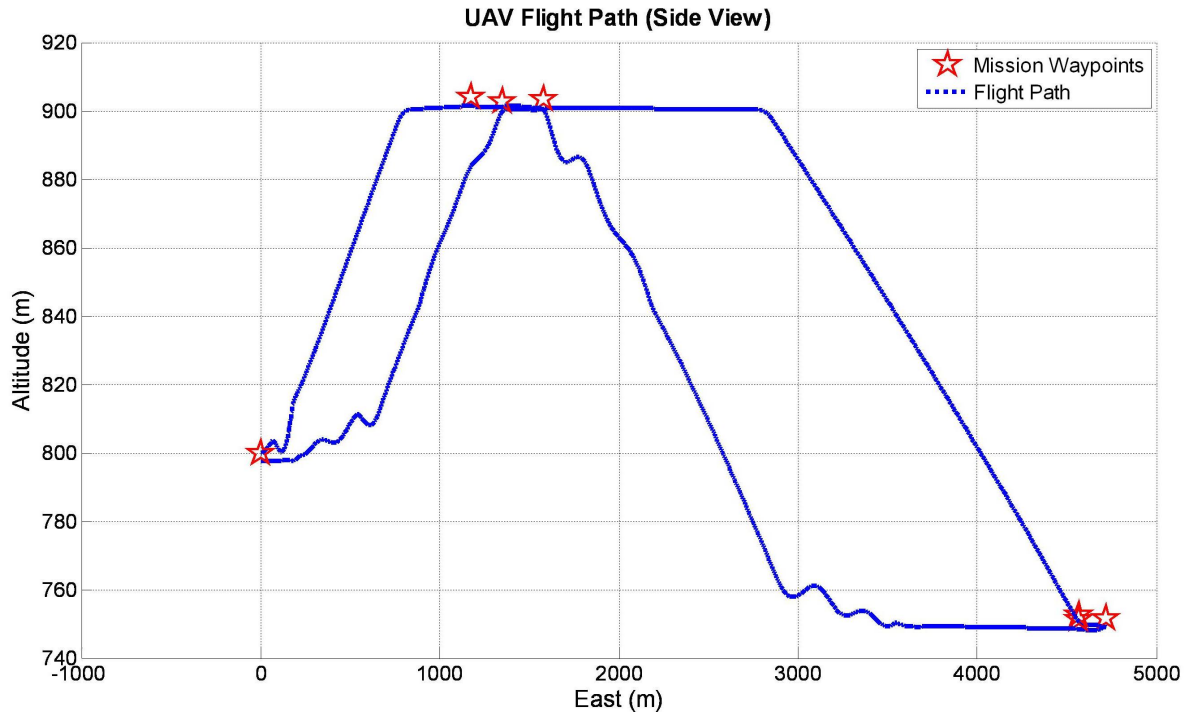


Figure 3.10: Test scenario - side view of flight path.

plots that the UAVSM can track waypoints both horizontally and vertically throughout the mission

3.4 Mission Scenario

As an integral part of the UAV simulation process, two baseline mission scenarios were constructed, one of which is based on the mission used in the test scenario in §3.3.3, which the other on a similar mission to the test scenario. Each of these mission scenarios not only provides a “flight plan”(or a series of waypoints) to run the UAV simulations, it also acts as a *reference* scenario to which the optimised mission can be compared. Basic UAV operations such as Climb, Cruise, Descent and Loiter were included in the mission scenario and its flight profile is shown in Figure 3.11.

The first baseline mission scenario, Mission Scenario 1 (MS1), is a realistic mission scenario comprising of 100 loops of the test mission, which has a distance of approximately 15km, and for each loop, Waypoint 9 becomes Waypoint 1 of the next loop. It uses GPS waypoints located in central Queensland, Australia, each loop is as listed in Table 3.3 and

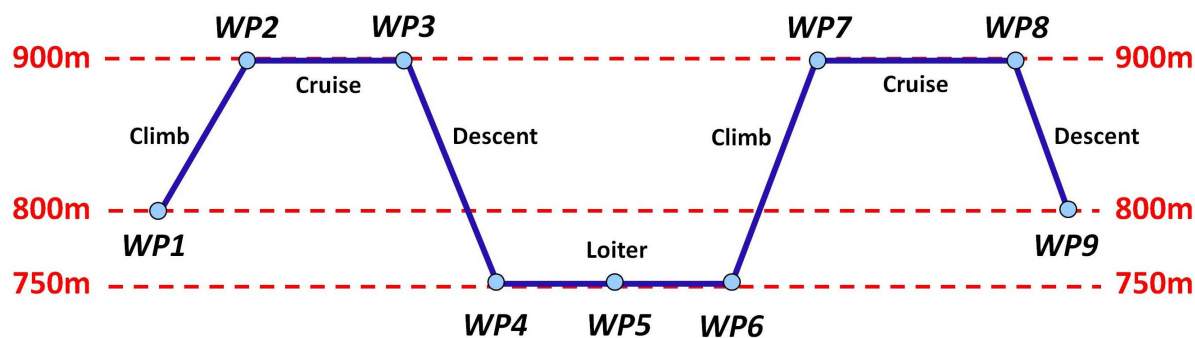


Figure 3.11: Flight profile of Mission Scenario 1 (not to scale).

illustrated in sequence in Figure 3.12.

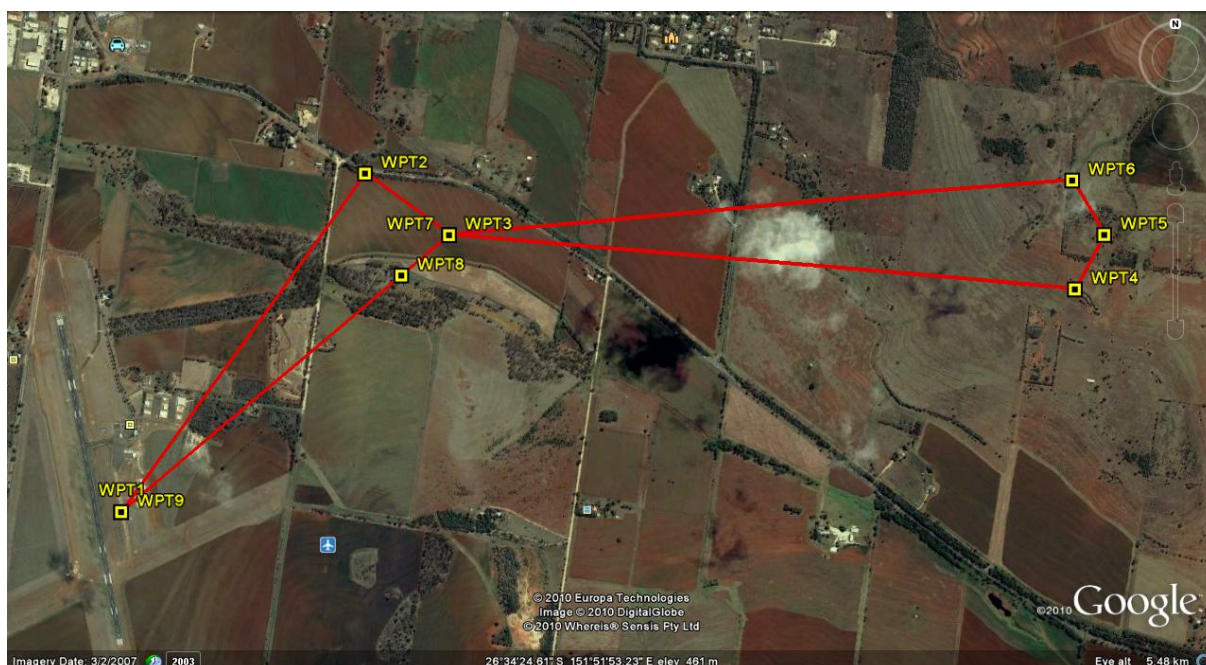


Figure 3.12: Mission Scenario 1 with GPS Waypoints in central QLD, Australia (image generated using Google Earth).

It needs to be noted that in each loop, the leg from Waypoints 4 to 6 in the mission scenario is the *loiter* phase, as shown in Figure 3.11. During this leg, some special mission requirement is carried out – i.e. electric-only flight, video recording, etc. – therefore these waypoints need to remain as specified and not be involved in the optimisation process.

Mission Scenario 2 (MS2) also consists of 100 loops of a base circuit, which is similar to the test mission that forms the basis of MS1. The waypoints of MS2 are listed in Table 3.4 and one loop of MS2 is shown in Figure 3.13.

Waypoint	Waypoint Coordinates		
	Latitude (DD:MM:SS)	Longitude (DD:MM:SS)	Altitude (m)
WP1 (Kingaroy Airport)	26°34'51" S	151°50'28" E	800
WP2	26°33'46" S	151°51'10" E	900
WP3	26°33'58" S	151°51'31" E	900
WP4	26°35'02" S	151°53'37" E	750
WP5	26°34'48" S	151°53'46" E	750
WP6	26°34'34" S	151°53'37" E	750
WP7	26°33'58" S	151°53'31" E	900
WP8	26°34'13" S	151°51'25" E	900
WP9 (Kingaroy Airport)	26°34'51" S	151°50'28" 3	800

Table 3.4: Waypoints in the Mission Scenario 2 loop.



Figure 3.13: Mission Scenario 2 with GPS Waypoints in central QLD, Australia (image generated using Google Earth).

3.5 Conclusions

The aim of this study was to develop a simulation model to provide a good representation of a small fixed-wing UAV, as well as to construct a mission scenario so that the simulation

of an UAV in flight can be carried out. This is to enable the implementation of a MOEA simulation optimisation method for MWO in Chapters 4 and 5.

The UAVSM was developed in the MATLAB Simulink environment, utilising the Aerosonde UAV block from the AeroSim Blockset, both were relatively easy to use. Both MATLAB Simulink and AeroSim Blockset are software packages that are commonly used in the aerospace industry, therefore unexpected problems inherent in the software are unlikely to arise.

The validity of the UAVSM was examined by the test scenario carried out in this study. The results of this showed that in normal mission operations, the UAVSM was able to function as an UAV would in reality, producing a smooth flight profile. The implementation of the two additional modules enabled a flight path to be navigated without human assistance, thus enabling unmanned operations. The current formulation of the UAVSM does not take into consideration the effects of wind on the UAV when in flight. This will be considered and included in future research.

The work achieved in this chapter satisfied Research Objective 2 as specified in §1.2.

The UAVSM and the constructed mission scenario will be used in the Single- and Multi-Objective MWO processes, which will be described in Chapters 4 and 5 respectively.

Chapter 4

Single-Objective Mission Waypoint Optimisation

4.1 Introduction

It was established in Chapter 2 that the following two tasks are required in order to develop a MWO procedure for a small fixed-wing UAV, focusing on improving the fuel economy on the UAV:

1. Construction of a simulation environment of a small fixed-wing UAV; and
2. Coupling of the constructed UAV simulation model with a SQP solver and a Multi-Objective EA (MOEA) optimiser to form a MOEA simulation optimisation method for MWO.

Chapter 3 saw the implementation of UAVSM, a simulation model to represent an Aerosonde UAV that is capable of navigating through a given flight mission. Additionally, a mission scenario was constructed to be utilised in the MWO process.

This chapter begins with a detailed description of the SOMWO problem, presented in §4.2 to §4.8. The aim of optimising onboard fuel efficiency was performed using two optimisation methods - one a MOEA optimiser as stated in §2.7.1, and the other a SQP solver as described in §2.3.1, the setup of which are described in §4.8. The SOMWO results are shown and analysed in §4.9 and concluding remarks are given in §4.10.

4.2 Optimisation Problem Definition

As stated in §1.1.1, the SOMWO problem considers the optimisation of a given series of waypoints that forms a flight mission, with the objective of minimising the onboard fuel consumption. Both mission scenarios MS1 and MS2, as defined in Tables 3.3 and 3.4, are used as baseline flight missions in the SOMWO process. However, the waypoints which the UAV has to fly through are not definite and may be adapted within a set of bounds in order to achieve the optimisation objective, namely the minimising of onboard fuel consumption. This is mainly because in most UAV flight missions, the main responsibility of the UAV is to achieve some operation goals such as taking images and/or videos of a certain region, or taking measurements of some specific elements over a particular area, but the path via which the UAV arrives and departs from this region is often not specified. Figure 4.1 shows an example of a possible flight mission after the MWO process has been carried out.

Note that Waypoints 1 and 9 (which are the start and end points, and are the same location), and 4 to 6 are not involved in the SOMWO process, as explained in §3.4.

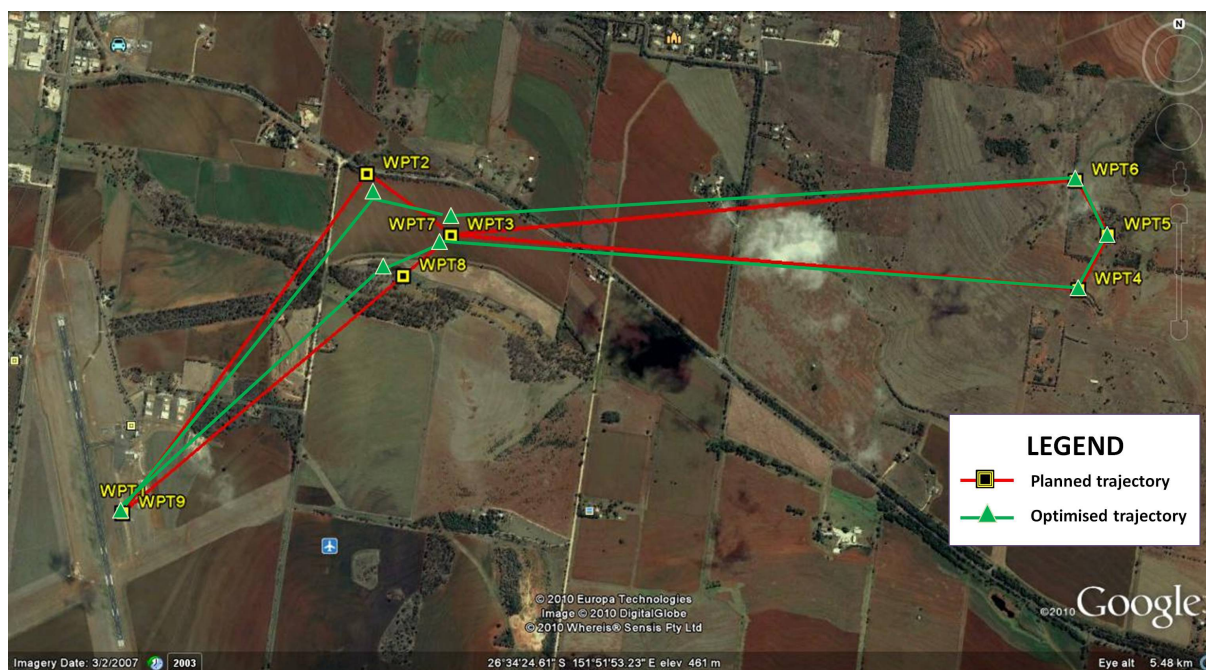


Figure 4.1: Example comparison of an optimised set of waypoints (green) to the baseline flight mission (red) (not to scale).

4.3 Design Variables

The design variables for SOMWO are the coordinates - latitude and longitude - of the waypoints. In the SOMWO process, the optimisation method would generate the candidate coordinates, which are passed to the UAVSM to be evaluated.

As per usual convention, the latitudes and longitudes are measured in radians. Also, North latitudes and East longitudes are taken as positive values, while South and West are negative. Since the location in which the mission scenarios are based in is in Australia, which is positioned south of the Equator and east of the prime meridian, this means the waypoints as listed in Tables 3.3 and 3.4 have negative latitudes and positive longitudes.

4.4 Fitness Function

A fitness function is a function or procedure that assigns a quality measure to a candidate [129]. A candidate solution that optimises the fitness function is called an optimal solution.

For SOMWO, in order to optimise the energy efficiency on a UAV, the fitness function, f , is defined as the UAV's fuel consumption, FC , over 100 laps of the flight mission of candidate waypoints and the SOMWO is then defined as:

$$\min(f) : f = FC \quad (4.1)$$

In each function evaluation, i.e. a simulation run in the case of this research, the UAV executes **five laps** of the flight mission and the fuel consumption over these two laps, FC_{5laps} , is recorded. The total fuel consumption over the 100 laps is estimated by:

$$FC = 20FC_{5laps} \quad (4.2)$$

The reason for conducting the optimisation in this manner is to reduce the computational efforts required for executing the simulation model. As described in §2.2, simulations runs are generally computationally expensive, therefore any savings in this area is desired.

For the optimisation of MS1, an initial fuel mass of 2kg was used in the UAVSM. On the other hand, taking into account the longer distances in MS2, an initial fuel mass of 5kg was used for the optimisation of MS2.

4.5 Upper and Lower Bounds of a Waypoint

The upper and lower bounds of a waypoint define a set of values from which a candidate waypoint is selected. The generation of a set of candidate waypoints is a key component of the optimisation process, and is performed by the optimiser used. These upper and lower bounds of each coordinate (latitude and longitude) are defined taking into consideration the mission requirements as well as airspace and class restrictions. The chosen set of waypoints is then passed to the UAVSM to determine the fitness function of this particular set of waypoints.

The waypoint bounds used in the two mission scenarios are different, and they are presented in §4.5.1 and §4.5.2 respectively.

4.5.1 Waypoint Bounds for Mission Scenario 1

Defining the upper and lower bounds for latitude, ϕ , and longitude, λ , of a waypoint requires the calculation of the great circle track (instead of a straight-line path) because the UAV is travelling over the surface of the Earth. The computation of the upper and lower bounds for the latitude and longitude of a waypoint is based on 10% of the distance between a waypoint and the preceding waypoint. A rearranged form of the *Haversine Formula* [161] is used to calculate the distance between two waypoints (ϕ_1, λ_1) and (ϕ_2, λ_2) by the following:

$$d = 2 \arcsin \left\{ \sqrt{\sin^2 \left(\frac{\phi_1 - \phi_2}{2} \right) + \cos(\phi_1) \cos(\phi_2) \sin^2 \left(\frac{\lambda_1 - \lambda_2}{2} \right)} \right\} \quad (4.3)$$

where d is the great circle distance in radians. Then the distance margin, Δd , is calculated by:

$$\Delta d = 0.1d \quad (4.4)$$

Also, the course from waypoint 1 to waypoint 2 is calculated by:

$$course = \text{mod} \left[\arctan \left\{ \frac{\sin(\lambda_1 - \lambda_2) \cos(\phi_2)}{\cos(\phi_1) \sin(\phi_2) - \sin(\phi_1) \cos(\phi_2) \cos(\lambda_1 - \lambda_2)} \right\}, 2\pi \right] \quad (4.5)$$

And lastly, given a course $course$ and a distance d from waypoint 1, a point (ϕ, λ) can be determined by:

$$\phi = \arcsin [\sin(\phi_1) \cos(d) + \cos(\phi_1) \sin(d) \cos(course)] \quad (4.6)$$

$$\Delta lon = \arctan \left[\frac{\sin(course) \sin(d) \cos(\phi_1)}{\cos(d) - \sin^2(\phi_1)} \right] \quad (4.7)$$

$$\lambda = \text{mod} (\lambda_1 - \Delta lon + \pi, 2\pi) - \pi \quad (4.8)$$

Using Equations 4.3 to 4.8, the following steps were used to compute the upper and lower bounds for the latitude and longitude of waypoint i :

1. Compute the distance between waypoint i and waypoint $(i - 1)$ using Equation 4.3.
2. Find the point that lies Δd_1 distance away from waypoint i in the direction of waypoint $(i - 1)$. Equations 4.5 to 4.8 are used for this computation, and the resulting latitude and longitude values, lat_{B1} and lon_{B1} respectively, constitute two of the bounds. However, whether these are upper and/or lower bounds will be determined in later steps.
3. Compute the distance between waypoint i and waypoint $(i + 1)$ as in Step 1.
4. Find the point that lies Δd_2 distance away from waypoint i in the direction of waypoint $i + 1$ as in Step 2. The resulting coordinates is used as follows:
 - **If the waypoints $(i - 1)$, i and $(i + 1)$ are not in a straight line** (see Figure 4.2), then the resulting longitude, lon_{B2} , is an upper/lower bound. Proceed to Step 5 to determine lat_{B2} .
 - **If these three waypoints are in a straight line** (see Figure 4.3), then the resulting latitude and longitude values, lat_{B2} and lon_{B2} respectively, are the remaining two upper/lower bounds. Proceed to Step 7.

5. Compute the distance from (lat_{B1}, lon_i) to (lat_i, lon_i) , denoted by d_{lat} . lat_{B2} is obtained by finding the point d_{lat} distance away from Waypoint i in the direction from (lat_{B1}, lon_i) to (lat_i, lon_i) .
6. Compare lat_{B1} to lat_{B2} to determine lat_{UB} and lat_{LB} , also compare lon_{B1} to lon_{B2} for lon_{UB} and lon_{LB} .

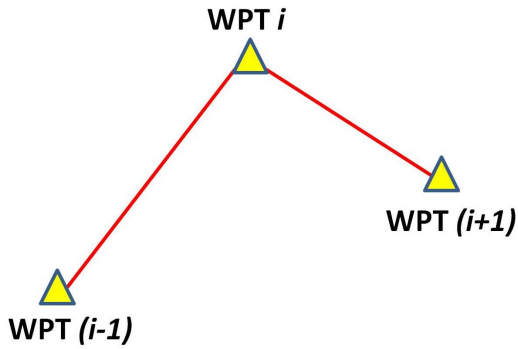


Figure 4.2: waypoints $(i - 1)$, i and $(i + 1)$ are not in a straight line.



Figure 4.3: Waypoints $(i - 1)$, i and $(i + 1)$ are in a straight line.

Using the above steps, the upper and lower bounds of the waypoint coordinates are calculated, adjusted to meet optimisation requirements, and presented in Table 4.1. Figure 4.4 illustrates these bounds in relation to the entire mission. In the case of Waypoints 3 and 7, where the bounds are quite similar, an enlarged plot is given in Figure 4.5.

Waypoint #	Coordinate	Lower Bound	Upper Bound
2	Latitude	0.463642241	0.463693195
	Longitude	2.650312648	2.650340341
3	Latitude	0.463700000	0.463720000
	Longitude	2.650397006	2.650456443
7	Latitude	0.463700000	0.463720000
	Longitude	2.650400061	2.650456445
8	Latitude	0.463741362	0.463762205
	Longitude	2.650340137	2.650367830

Table 4.1: Waypoint bounds for Mission Scenario 1.

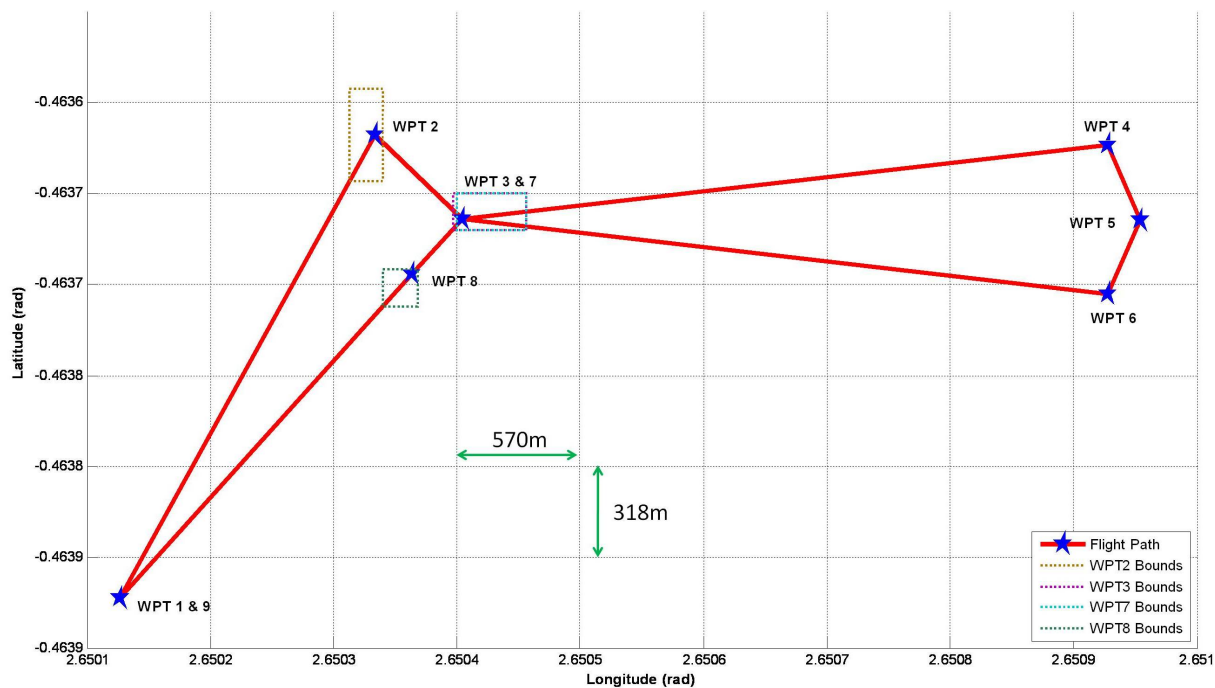


Figure 4.4: Illustration of the waypoint bounds for Mission Scenario 1.

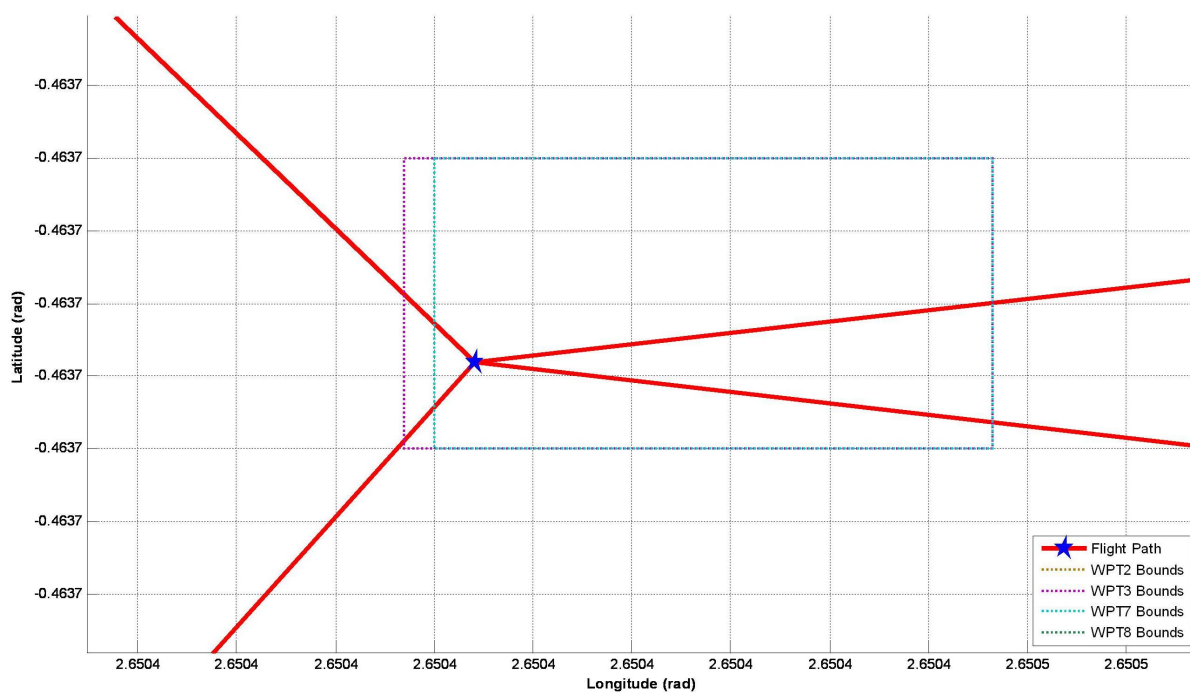


Figure 4.5: Enlarged view of the waypoint bounds for Waypoints 3 & 7 in Mission Scenario 1.

4.5.2 Waypoint Bounds for Mission Scenario 2

For MS2, a set of waypoint bounds encompassing a greater search space was designed. In contrast to the waypoint bounds for MS1, described in §4.5.1, which are determined according to the respective distances between the waypoints and are different for each waypoint to be optimised, the waypoint bounds for MS2 form a “common area” of approximately $1.5\text{km} \times 2.5\text{km}$, from which the candidate waypoints for all the waypoints to be optimised are chosen. This area is illustrated in Figure 5.1, with the bounds listed in Table 4.2.

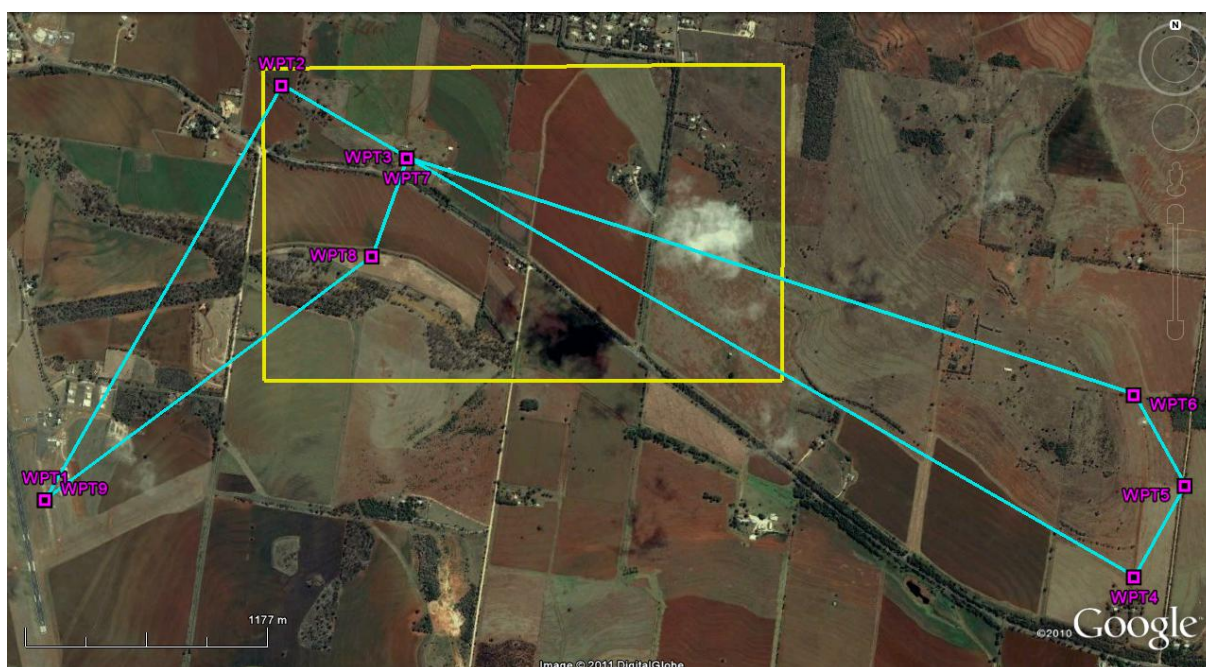


Figure 4.6: Illustration of the waypoint bounds for Mission Scenario 2.

Coordinate	Lower Bound	Upper Bound
Latitude	0.46359639	0.46383268
Longitude	2.65031243	2.65075036

Table 4.2: Waypoint bounds for Mission Scenario 2.

4.6 Physical Constraints

Several physical constraints – limits for engine throttle, rudder deflection, aileron deflection and elevator deflection – were incorporated into the UAVSM with the intention of making the model more realistic. These constraints were implemented as part of the ACM of the simulation model using saturation blocks and they are reiterated in Table 4.3.

Control	Constraints
Engine throttle	$0.01 \leq TC \leq 1$
Rudder deflection	$-20^\circ \leq \theta_r \leq 20^\circ$
Aileron deflection	$-10^\circ \leq \theta_a \leq 10^\circ$
Elevator deflection	$-20^\circ \leq \theta_e \leq 20^\circ$

Table 4.3: Physical constraints on ACM outputs.

An additional constraint is:

- The airspeed is controlled at 20m/s when the aircraft is at level flight or climbing, and increased to 30m/s when descending.

and this was incorporated into the simulation process as part of the Waypoint Table that is called upon by the *NavigateWaypoints* function (see Appendix B) inside the FPM.

4.7 Mathematical Formulation of SOMWO Problem

The mathematical formulation of the SOMWO is as follows:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{y} = f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{lb} < \mathbf{x} < \mathbf{ub} \end{aligned} \tag{4.9}$$

where $\mathbf{x} = (lat_1, lon_1, lat_2, lon_2, \dots, lat_n, lon_n)^T$ is the set of coordinates (latitude and longitude) of n waypoints that is to be optimised, $y = FC$ is the objective function of fuel consumption, $\mathbf{lb} = (latLB_1, lonLB_1, latLB_2, lonLB_2, \dots, latLB_n, lonLB_n)^T$ and $\mathbf{ub} = (latUB_1, lonUB_1, latUB_2, lonUB_2, \dots, latUB_n, lonUB_n)^T$ are the lower and upper bound vectors, respectively, for the optimised waypoints.

4.8 Optimiser Setup

In the implementation of SOMWO, two optimisers were used:

1. The HAPMOEA optimiser (EA-based) (§2.7.1); and
2. The MATLAB SQP solver (SQP-based) (§2.3.1).

The MATLAB-based UAVSM, configured to output the *fuel mass* and shown in Figure 4.7, was used with both optimisers in the SOMWO process and the fundamental sample time, Δt , set at 0.1 seconds.

4.8.1 HAPMOEA Optimiser Setup

The HAPMOEA optimisation rationale is displayed in Figure 4.8.

For the SOMWO problem, the HAPMOEA optimiser was set up with only one layer and contains the following settings, entailed in the *optimisation.parameters* file (see Appendix C) as part of the output files generated by the HAPMOEA optimiser.

- Population size = 10
- Parents in recombination = 2
- Buffer length = 12
- Tournament-in-buffer ratio = 2.0

The HAPMOEA optimisation process begins with setting up the problem to be optimised. This includes defining the decision variables (see §4.3), parameters and constraints (see §4.5 to §4.6), then continue to define the number of subpopulations (nodes), hierarchical levels and integrated analysis. Once these are defined, the optimiser obtains enough individuals to fill the population size.

After the population has been initialised, the optimiser begins the actual optimisation. This is achieved by the optimiser updating the solver with a candidate waypoint and the solver evaluating this candidate waypoint using MATLAB and UAVSM to obtain the fitness value(s), which are then sent back to the optimiser. This process is continued until

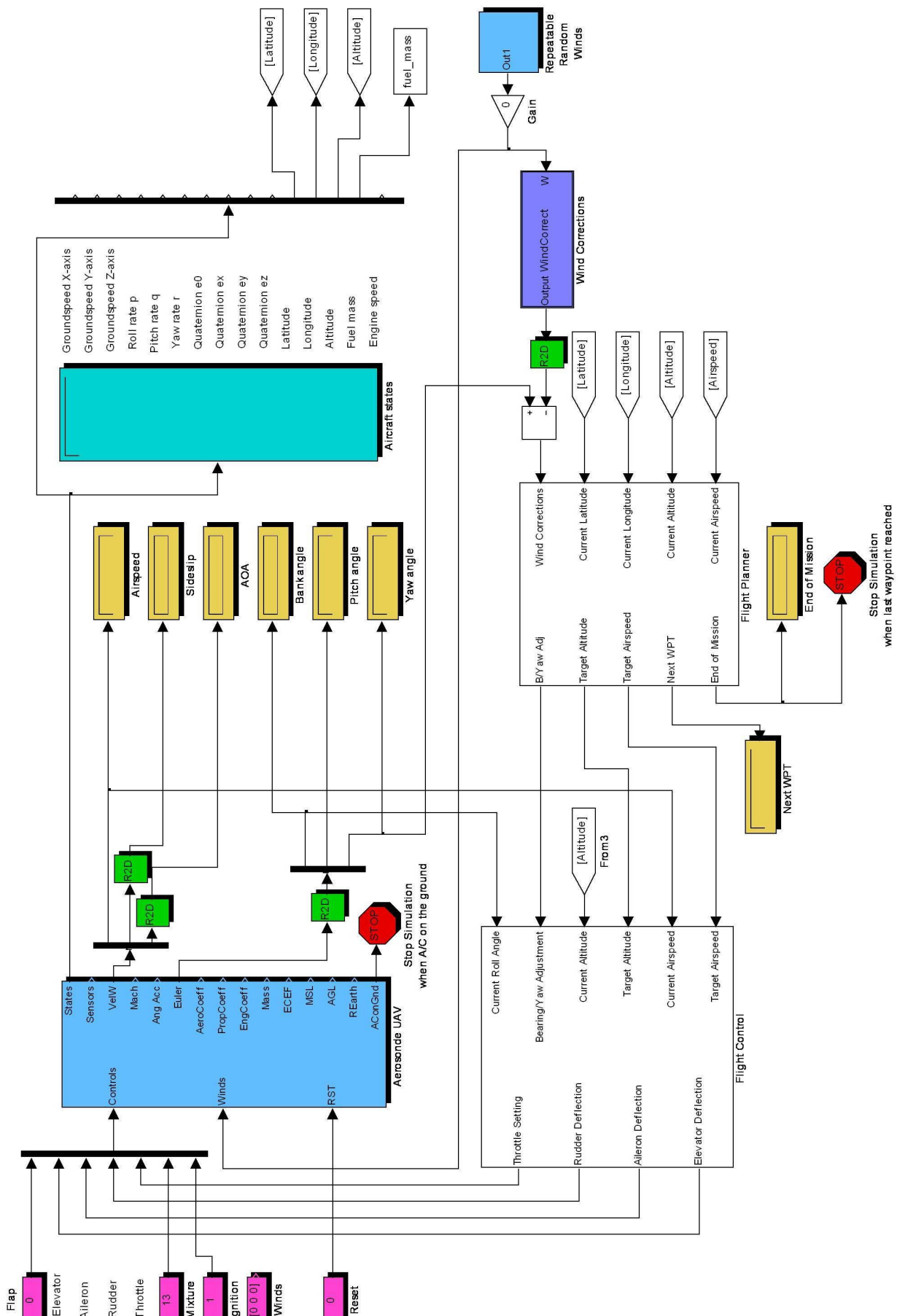


Figure 4.7: Updated UAVSM for SOMWO.

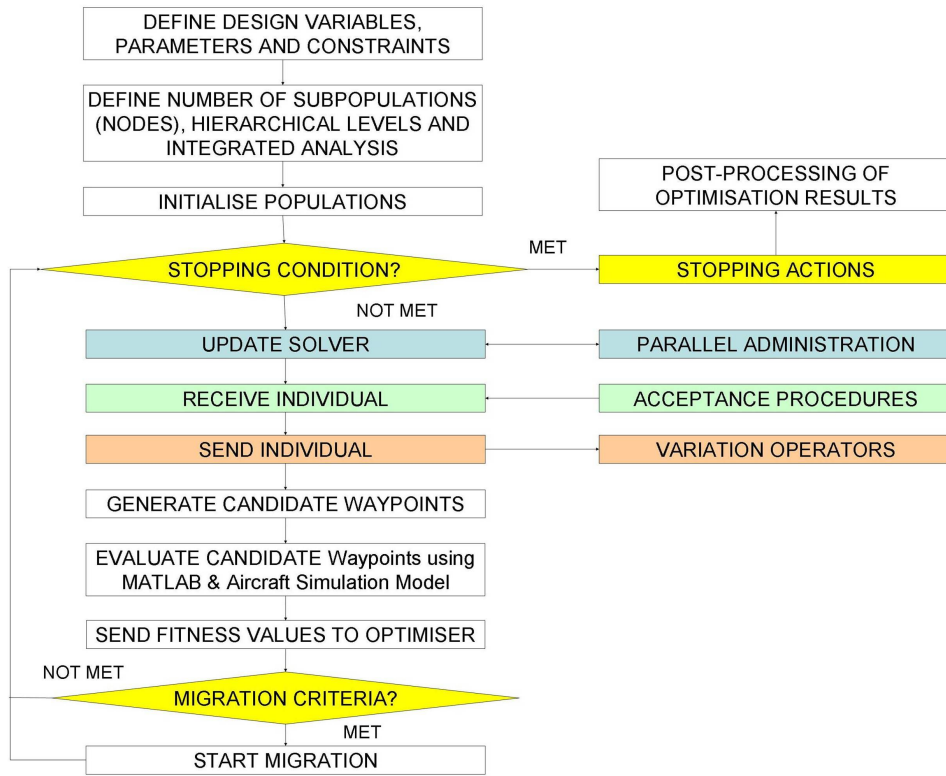


Figure 4.8: The optimisation rationale flow diagram for the HAPMOEA optimiser.

some migration criteria have been met, after which migration starts. Or the process may be stopped by the stopping condition specified, usually a maximum number of function evaluations, maximum run time, or a certain precision has been met.

HAPMOEA takes into account the lower and upper bound constraints (see §4.5) and generates candidate waypoints with their *latitude* and *longitude* coordinates as the design variables. The waypoint table constructed using these candidate waypoints are passed to UAVSM to be evaluated. The output of the UAVSM is the *fuel consumption* over the mission and this is the objective which is to be minimised by HAPMOEA.

Note that an initial estimates vector is not required for HAPMOEA.

4.8.2 MATLAB SQP Solver Setup

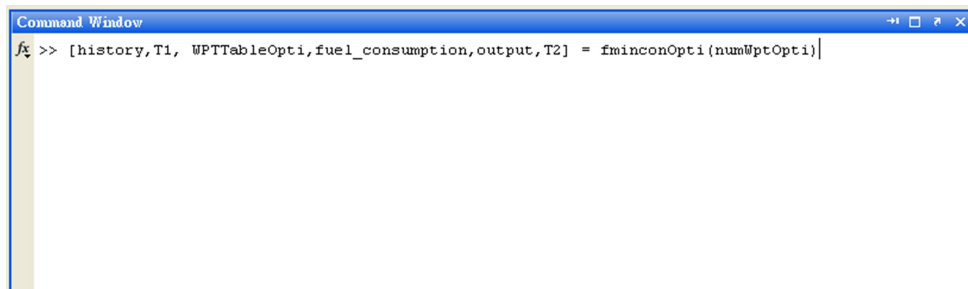
The setup of the SQP solver for the SOMWO problem consists of the definition of the initial estimates vector, x_0 , and the lower and upper bounds vectors, lb and ub respectively.

The initial estimates vector, x_0 , is defined as a column vector as follows:

$$x_0 = \begin{pmatrix} lat_1 \\ lon_1 \\ \vdots \\ lat_n \\ lon_n \end{pmatrix} \quad (4.10)$$

where lat_i and lon_i are the latitude and longitude coordinates of the i -th waypoint respectively, in a mission with n waypoints. The values of the waypoint coordinates are obtained from the baseline mission waypoints as defined in Table 3.3.

The lower and upper bounds vectors, lb and ub respectively, are calculated using equations described in §4.5. These are used by the SQP solver to generate candidate waypoint coordinates as a column vector which are then passed to the *fminconOpti* function (refer to Appendix D) to be minimised. This function constructs a waypoint table, an $n \times 3$ matrix, from the candidate waypoint coordinates. The function then calls the UAVSM, which accepts this waypoint table of candidate waypoints and executes the candidate mission. The output of the simulation is the *fuel consumption* during the mission, which is the objective to be minimised by the SQP solver. Figure 4.9 shows MATLAB at work performing the SQP-based optimisation.



```
Command Window
fx >> [history,T1, WPTTableOpti,fuel_consumption,output,T2] = fminconOpti(numWptOpti)|
```

Figure 4.9: MATLAB function call for the SQP-solver.

4.9 Optimisation Results and Analysis

The optimisation procedure for each optimiser used different stopping conditions:

- HAPMOEA optimiser: a time limit of 3 hours on one machine only
- SQP solver: Default stopping condition (terminates if the magnitude of the directional derivative in the search direction is less than 2×10^{-6} and the maximum constraint violation is less than 1×10^{-6}) in addition to the maximum iteration count of 20 iterations.

4.9.1 Mission Scenario 1

Figures 4.10 to 4.17 illustrate the fitness-vs-function-evaluations graphs for both optimisers, with the number of waypoints being optimised ranging from one waypoint (Waypoint 2) to four waypoints (Waypoints 2, 3, 7 and 8).

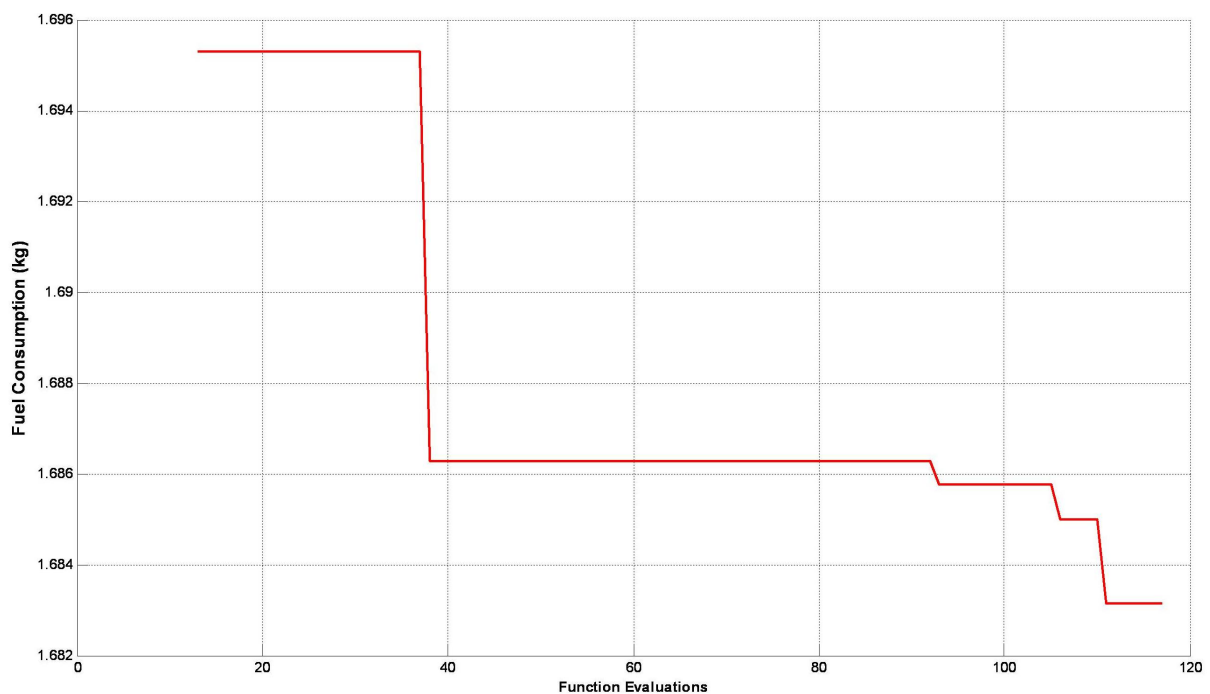


Figure 4.10: Fuel consumption vs function evaluations for MS1 (1 waypoint - HAPMOEA).

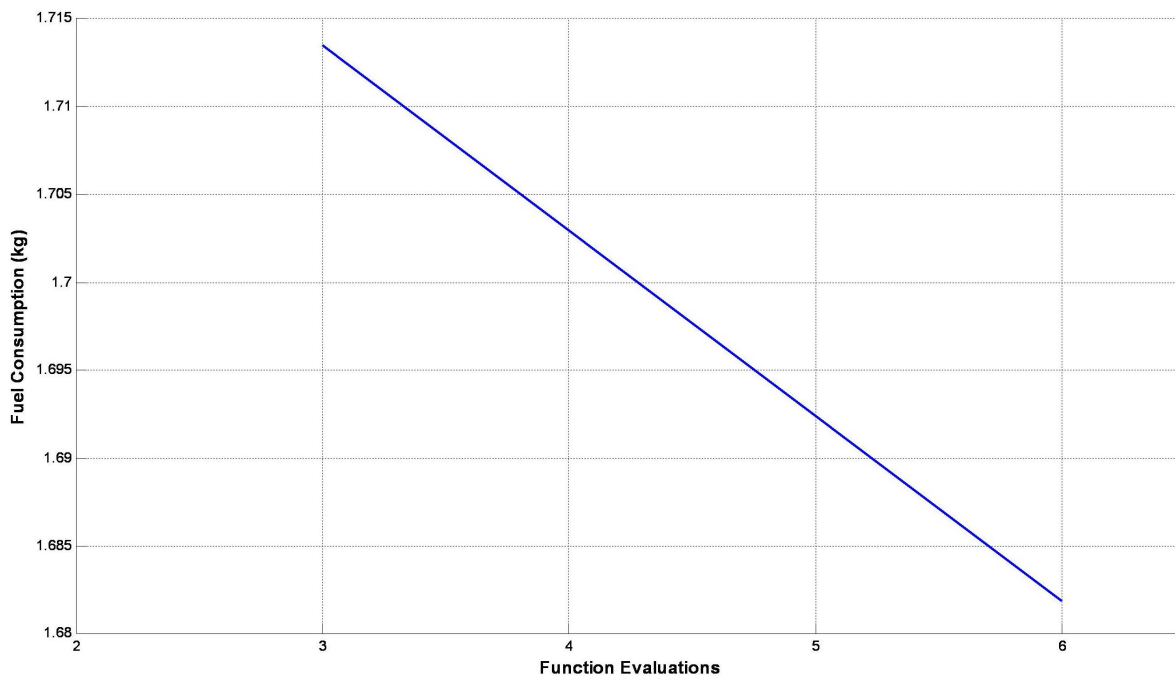


Figure 4.11: Fuel consumption vs function evaluations for MS1 (1 waypoint - SQP).

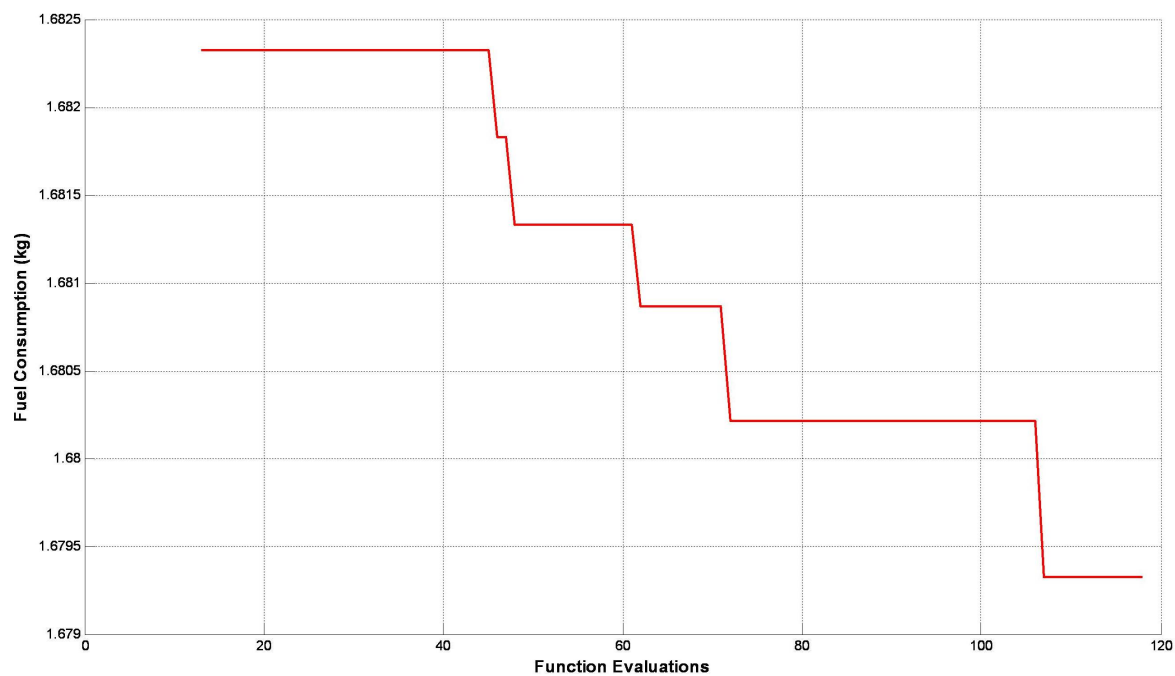


Figure 4.12: Fuel consumption vs function evaluations for MS1 (2 waypoints - HAPMOEA).

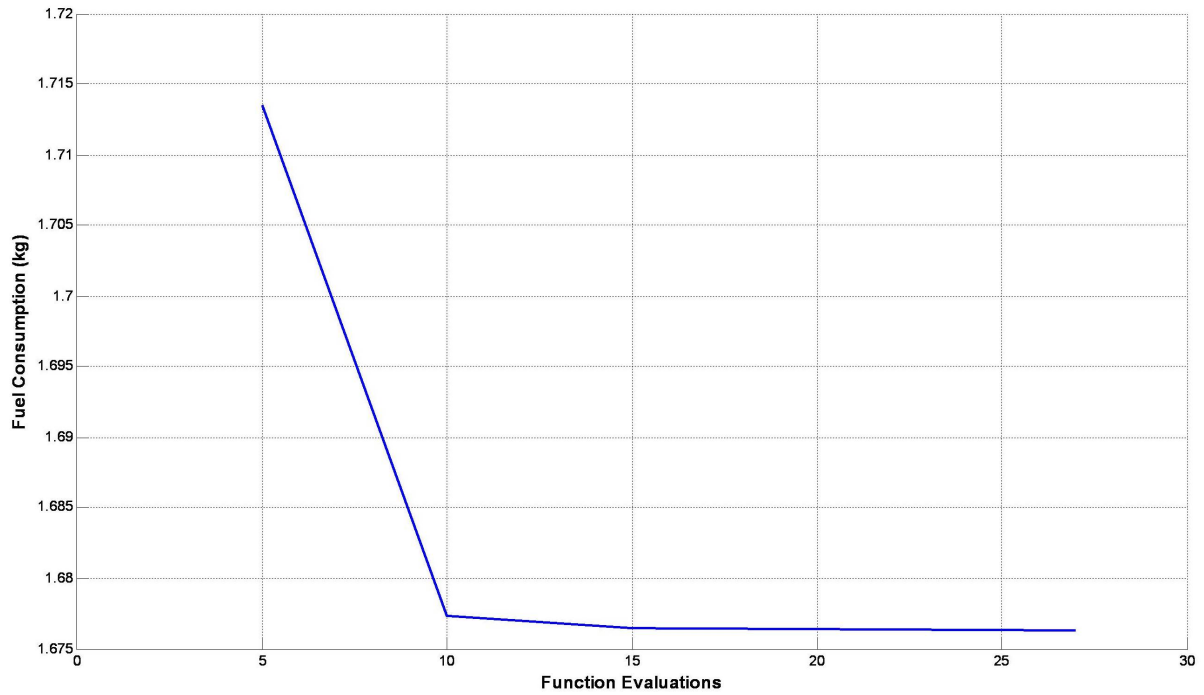


Figure 4.13: Fuel consumption vs function evaluations for MS1 (2 waypoints - SQP).

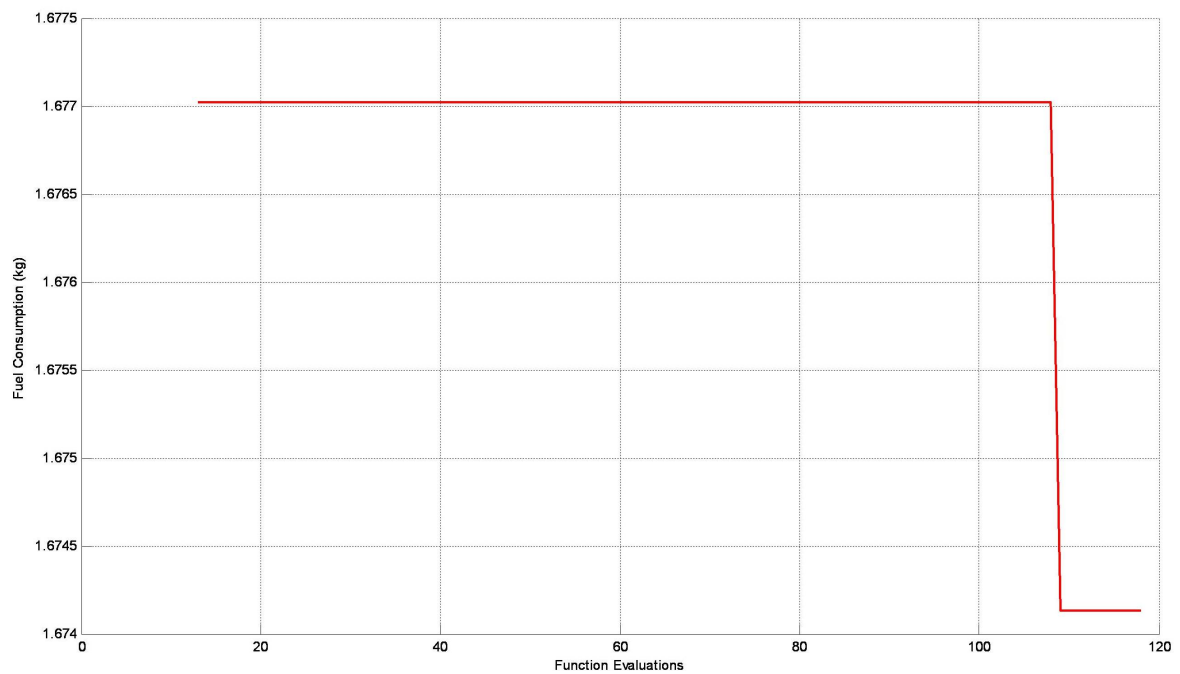


Figure 4.14: Fuel consumption vs function evaluations for MS1 (3 waypoints - HAPMOEA).

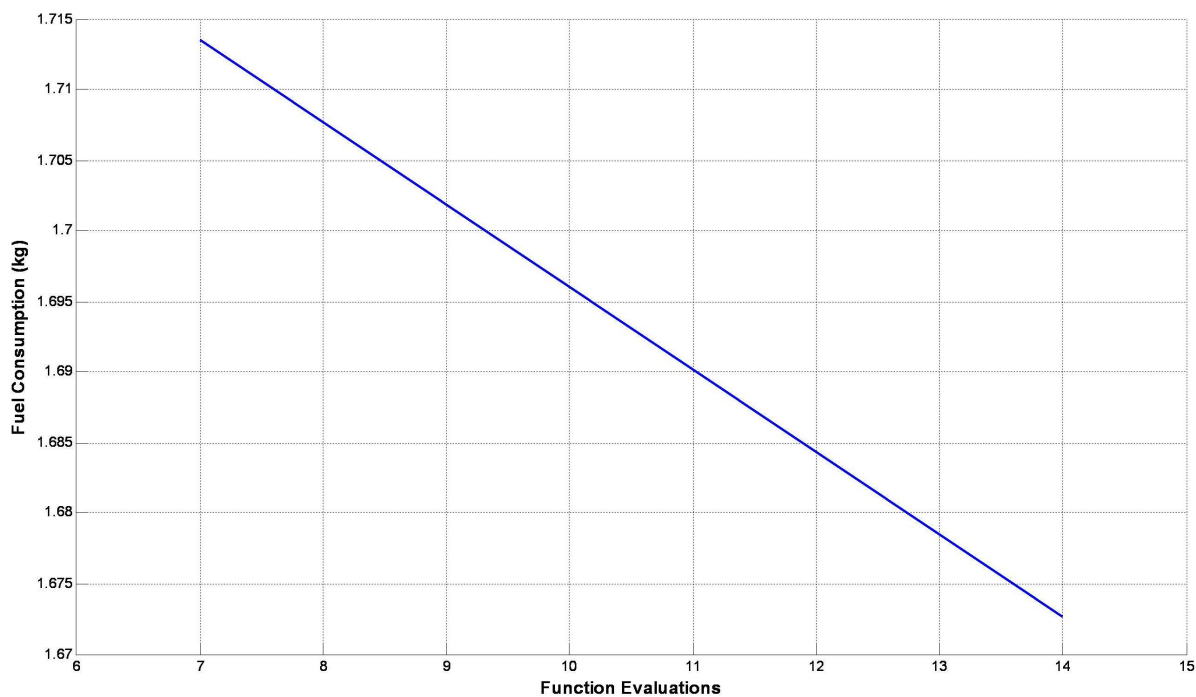


Figure 4.15: Fuel consumption vs function evaluations for MS1 (3 waypoints - SQP).

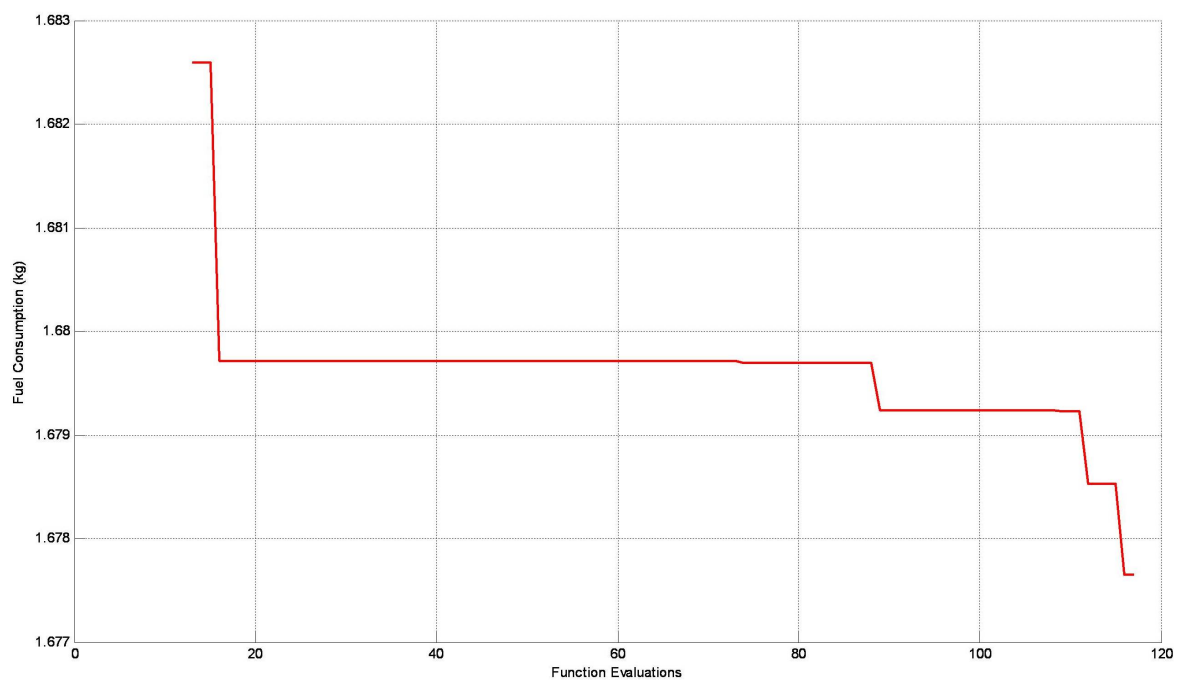


Figure 4.16: Fuel consumption vs function evaluations for MS1 (4 waypoints - HAPMOEA).

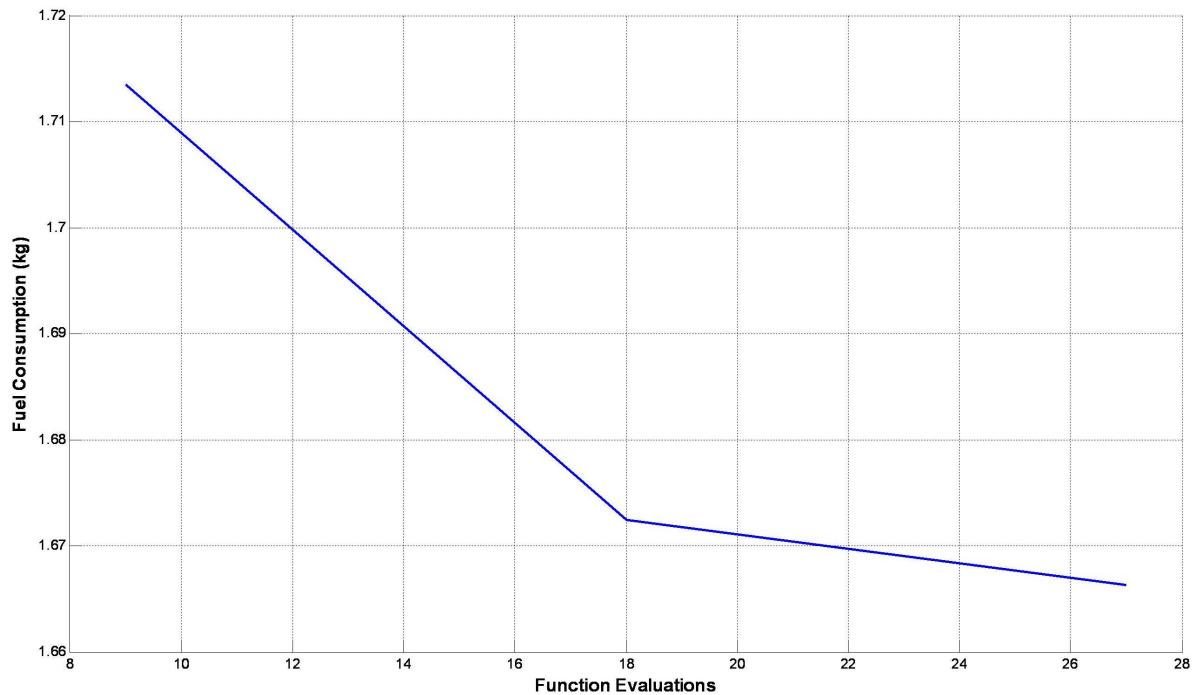


Figure 4.17: Fuel consumption vs function evaluations for MS1 (4 waypoints - SQP).

Table 4.4 compares the optimisation parameters, the run time statistics and the fitness values when using the two optimisers. The fitness values were also compared to that of the baseline mission, which is listed in the first entry.

# Waypoints / Design Variables Optimised	Optimisation Method	Function Evaluations Performed	Time Taken	Fuel Consumption (kg)
-	Baseline	-	11min 48s	1.713496854
1 (2 DV)	HAPMOEA	115	3hrs	1.68314748
	SQP	6	51min 46s	1.68186140
2 (4 DV)	HAPMOEA	116	3hrs	1.67932596
	SQP	27	5hrs 12min	1.67627787
3 (6 DV)	HAPMOEA	116	3hrs	1.67413965
	SQP	14	2hrs 4min	1.67263123
4 (8 DV)	HAPMOEA	115	3hrs	1.6776534
	SQP	27	3hrs 49min	1.6663055

Table 4.4: Table of statistics for SOMWO (MS1).

It can be seen from Table 4.4 that both the HAPMOEA and SQP optimisers have

improved the fuel consumption, with little differences between the outcomes of the two methods. It can be noted that as the number of waypoints to be optimised increases, the bigger the improvements resulted from the optimisation processes.

However, the SQP optimiser was able to produce the set of optimised waypoints that “uses” less fuel than that from the HAPMOEA optimiser. A possible reason for this large difference between the effectiveness of the two methods is that the search region, i.e. the region within the upper/lower bounds for the latitude and longitude coordinates, for each optimised waypoint is relatively small. In this case, a global optimisation method such as an EA would be expected to be less effective than a gradient-based method, which works very well in a localised search region.

It can also be observed that a plateau in the fuel consumption value appears in each of Figures 4.10, 4.14 and 4.16, the optimised results from the HAPMOEA optimiser. In the EA/GA process, the introduction of randomness into a population to form the next generation enables the exploration of the search region in directions not necessarily corresponding to the current minimum fitness, therefore avoiding convergence to a local optimum and increase the possibility of finding the global optimum. However, depending on the initial population that was generated, early convergence to a local optimum may result, and this is most often the cause of the plateaus in the fitness value, or fuel consumption value, as seen in Figures 4.10, 4.14 and 4.16. In each of these cases, it can be seen that after a number of function evaluations, during which many generations of populations were formed, new optima were found. Ideally, the set of optimised waypoints can be determined if time and computational resources are in abundance. However, this is not possible in real-world applications. Therefore, the EA/GA optimisation process is terminated when a termination condition has been reached. Common termination conditions are listed in §2.7.

The improvements in fuel consumption ranges from 0.03034937kg for HAPMOEA 1-waypoint optimisation to 0.04719135 for SQP 4-waypoint optimisation. Given that the simulation execution time for two laps of the Baseline mission consumes 0.034031kg, it means with these savings from the optimisation, the UAV could potentially execute an extra two laps of the flight mission. This may result in two more opportunities at accomplishing the mission goal as extra insurance in case one or more previous attempts

failed, or to collect more data.

It can be seen in the comparison that the run time for the SQP solver varied quite significantly across the number of decision variables being optimised. This can be problematic when a large number of function evaluations is required for one SQP optimisation run. On the other hand, the HAPMOEA optimiser is able to be stopped when a maximum amount of time has been reached.

It must be noted that the SQP solver requires multiple simulation runs, $n + 1$ runs was observed in this case (n denotes the number of waypoints to be optimised), in order to complete one iteration, before moving to the next potential set of candidate waypoints. This is due to the fact that the gradient at one iterate needs to be determined before the process can determine the “location” of the next iterate. This can be a problem when the number of variables becomes large. On the other hand, the HAPMOEA optimiser finds a new set of candidate waypoints after each evaluation, therefore it is much more capable at tackling optimisation problems with a large number of variables.

The optimised waypoints - in their latitude and longitude coordinates - are displayed in Table 4.5 and illustrated in Figures 4.18 to 4.21.

# Waypoints / Design Variables	Optimisation Method	WPT #	Optimised Latitude (rad)	Optimised Longitude (rad)
1 (2 DV)	HAPMOEA	2	0.463692003	2.65032962
	SQP	2	0.463693195	2.650340341
2 (4 DV)	HAPMOEA	2	0.463692361	2.65033336
		3	0.46370348	2.65042352
	SQP	2	0.463693195	2.650340341
		3	0.463700020	2.650445568
3 (6 DV)	HAPMOEA	2	0.463691097	2.65033806
		3	0.463709128	2.65045171
		7	0.463713306	2.65045368
	SQP	2	0.463693195	2.650340341
		3	0.463700000	2.650456443
		7	0.463720000	2.650456445
4 (8 DV)	HAPMOEA	2	0.463686392	2.65033528
		3	0.463706622	2.6504452
		7	0.463716449	2.65042714
		8	0.463757052	2.650366
	SQP	2	0.463693195	2.650340341
		3	0.463700000	2.650456443
		7	0.463720000	2.650456445
		8	0.463762205	2.650367830

Table 4.5: Table of optimised waypoints for SOMWO (MS1).

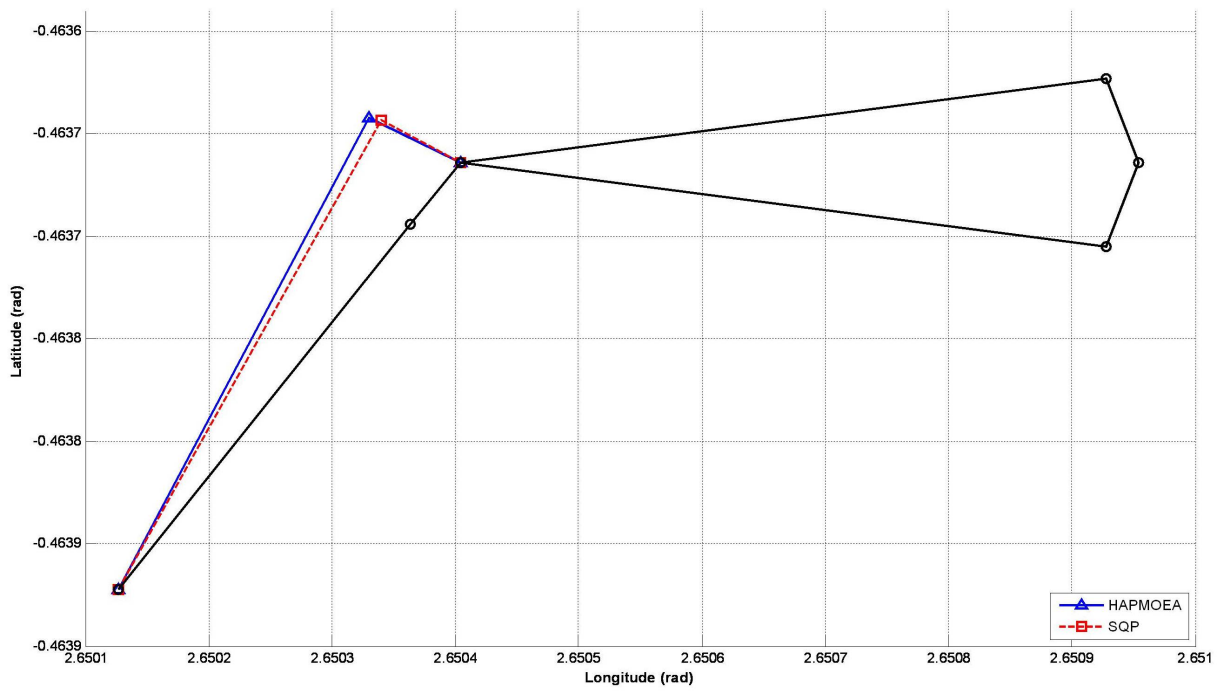


Figure 4.18: Comparison of optimised waypoints for MS1 (1 waypoint optimised).

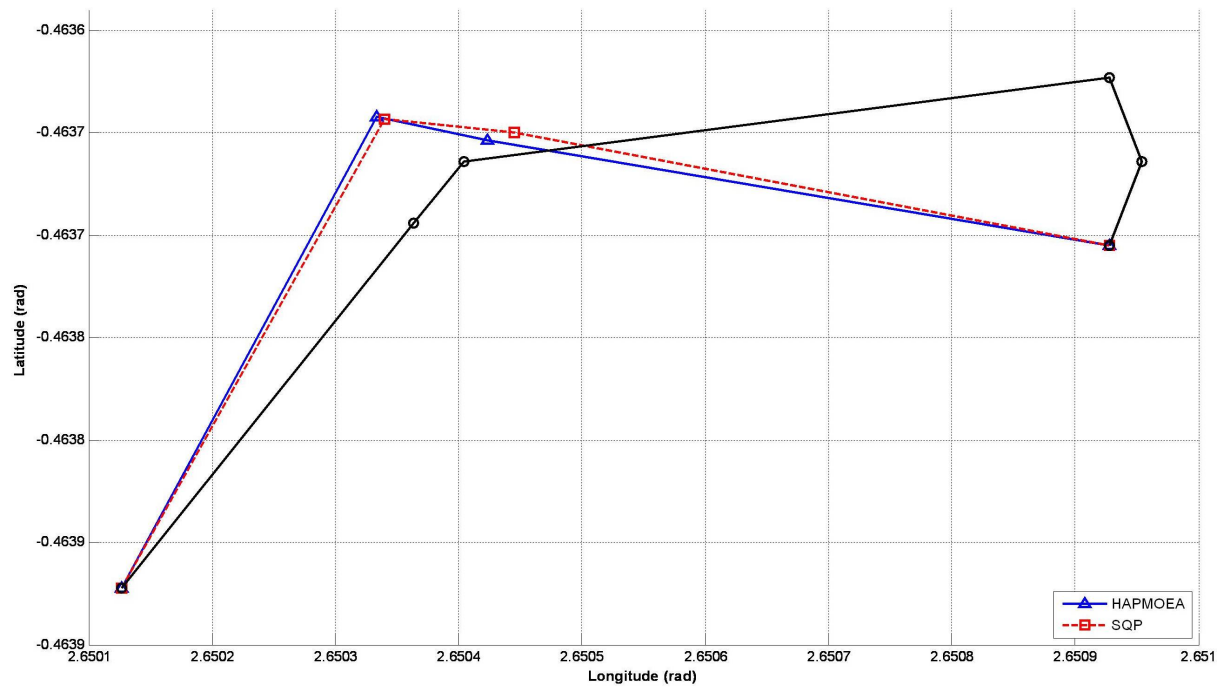


Figure 4.19: Comparison of optimised waypoints for MS1 (2 waypoints optimised).

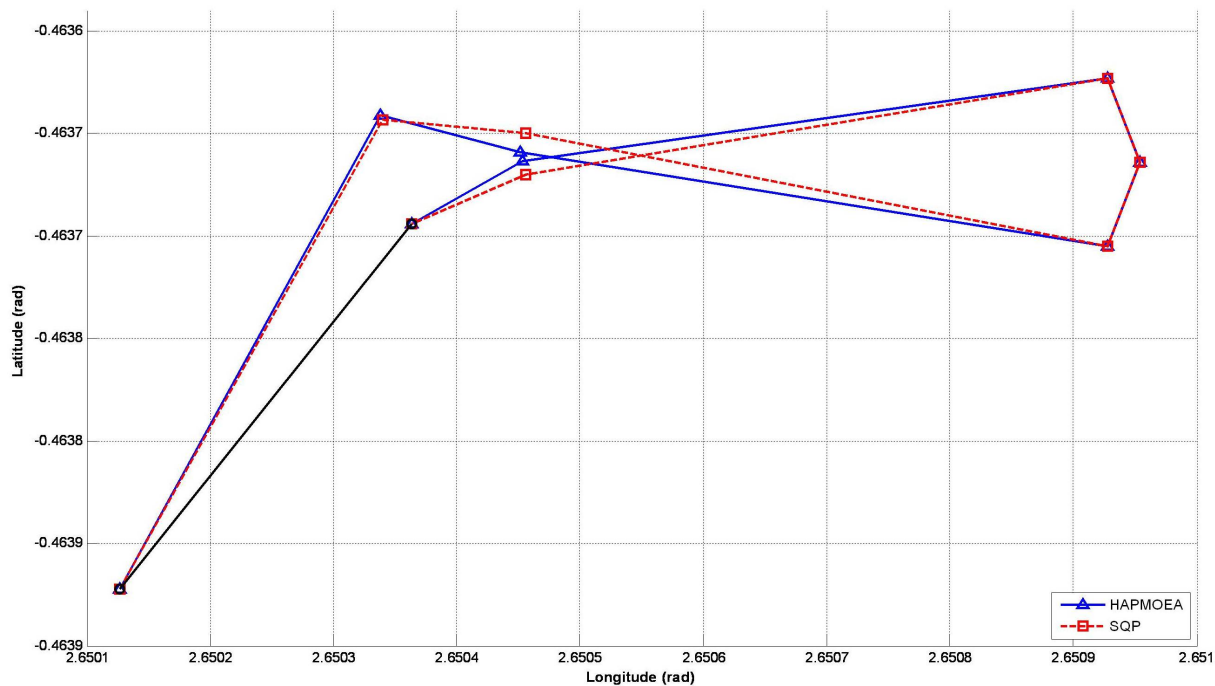


Figure 4.20: Comparison of optimised waypoints for MS1 (3 waypoints optimised).

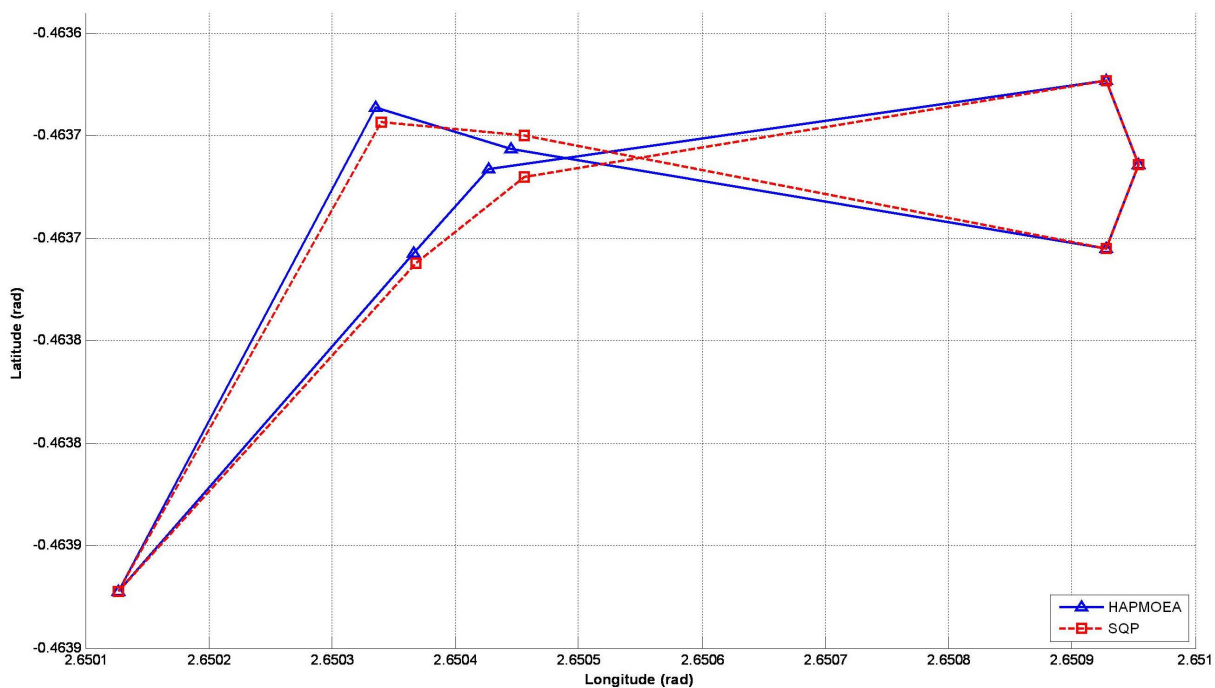


Figure 4.21: Comparison of optimised waypoints for MS1 (4 waypoints optimised).

4.9.2 Mission Scenario 2

Figures 4.22 to 4.29 illustrate the fitness-vs-function-evaluations graphs for both optimisers, with the number of waypoints being optimised ranging from one waypoint (Waypoint 2) to four waypoints (Waypoints 2, 3, 7 and 8).

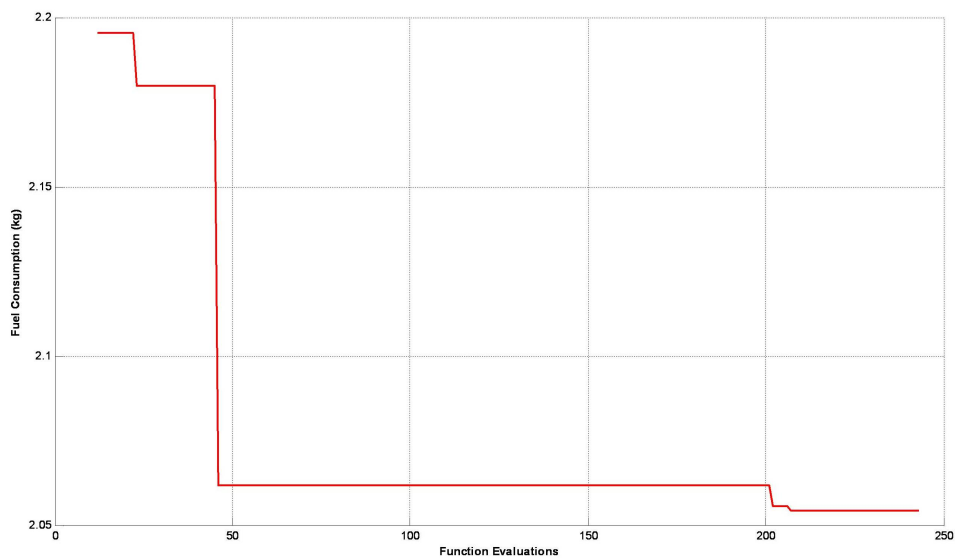


Figure 4.22: Fuel consumption vs function evaluations for MS2 (1 waypoint - HAPMOEA).

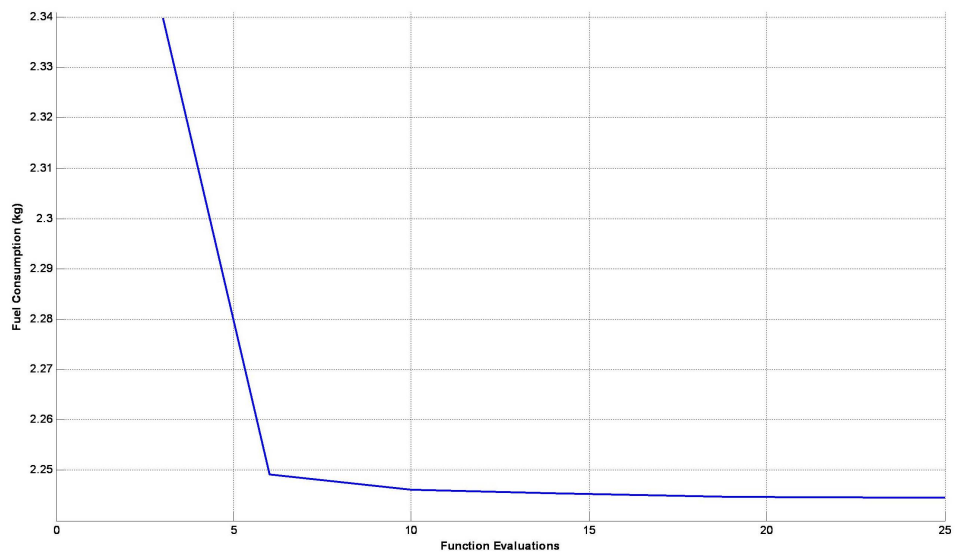


Figure 4.23: Fuel consumption vs function evaluations for MS2 (1 waypoint - SQP).

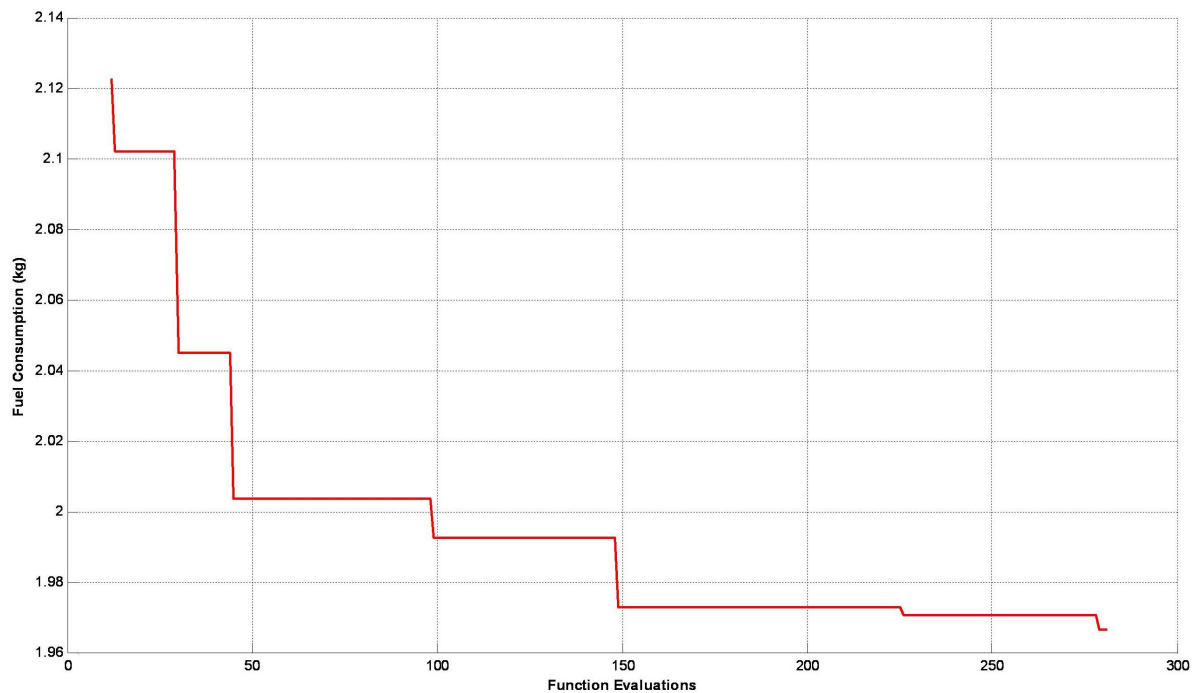


Figure 4.24: Fuel consumption vs function evaluations for MS2 (2 waypoints - HAPMOEA).

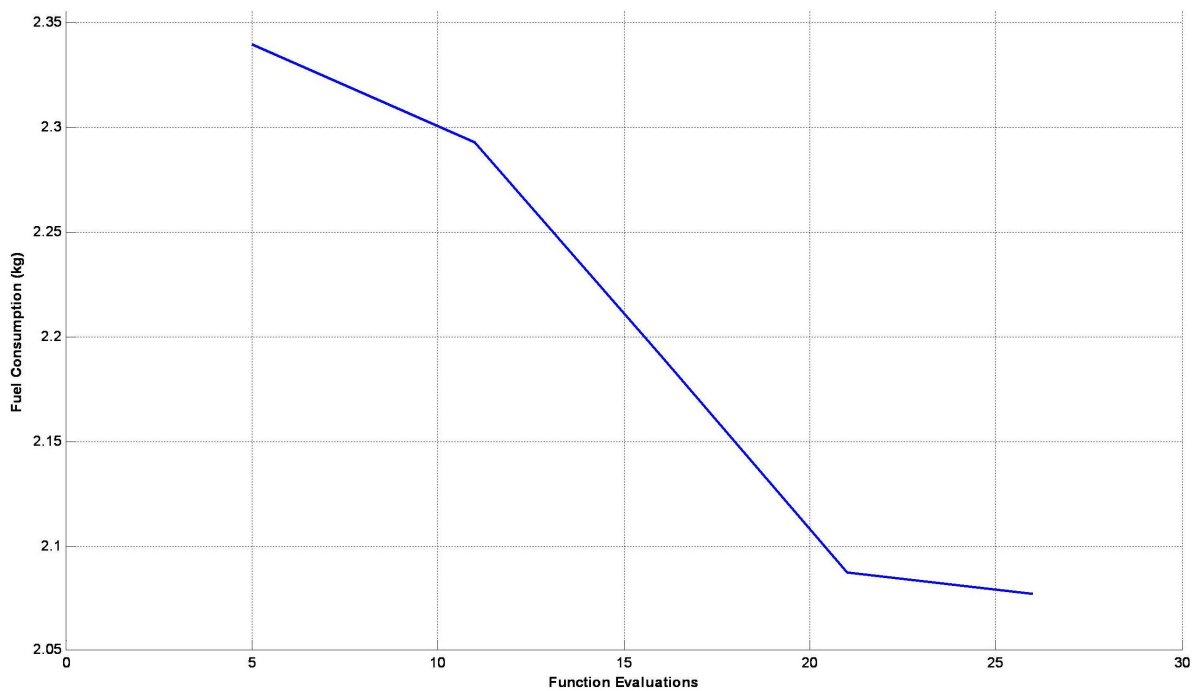


Figure 4.25: Fuel consumption vs function evaluations for MS2 (2 waypoints - SQP).

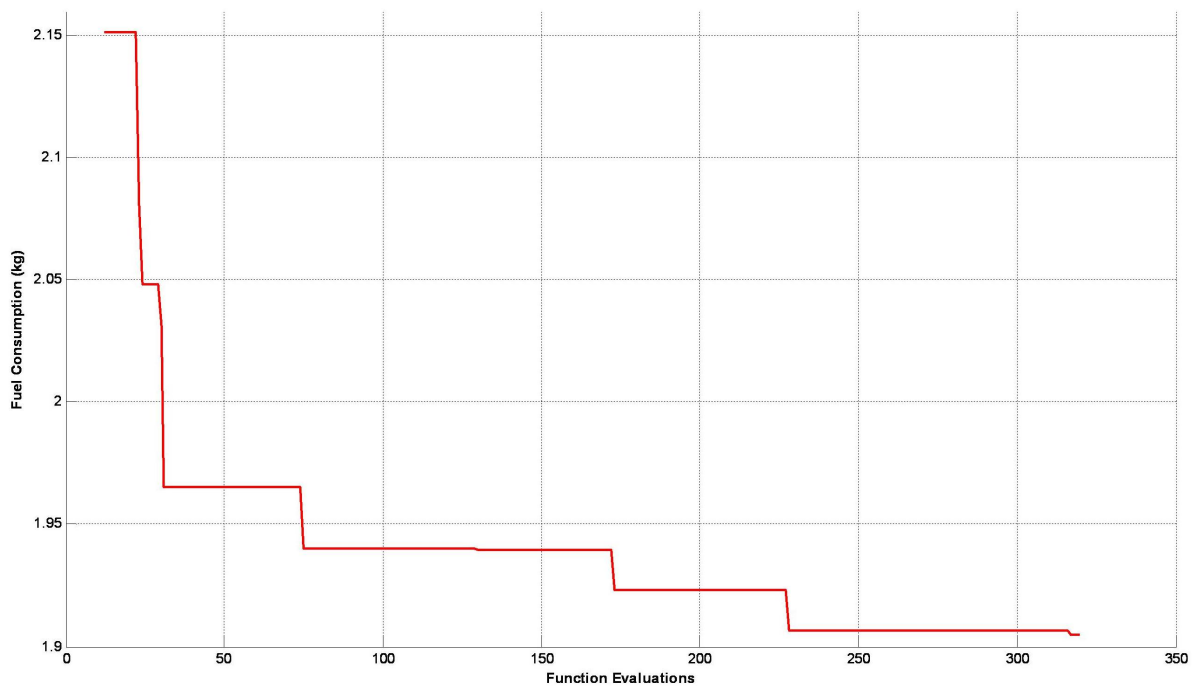


Figure 4.26: Fuel consumption vs function evaluations for MS2 (3 waypoints - HAPMOEA).

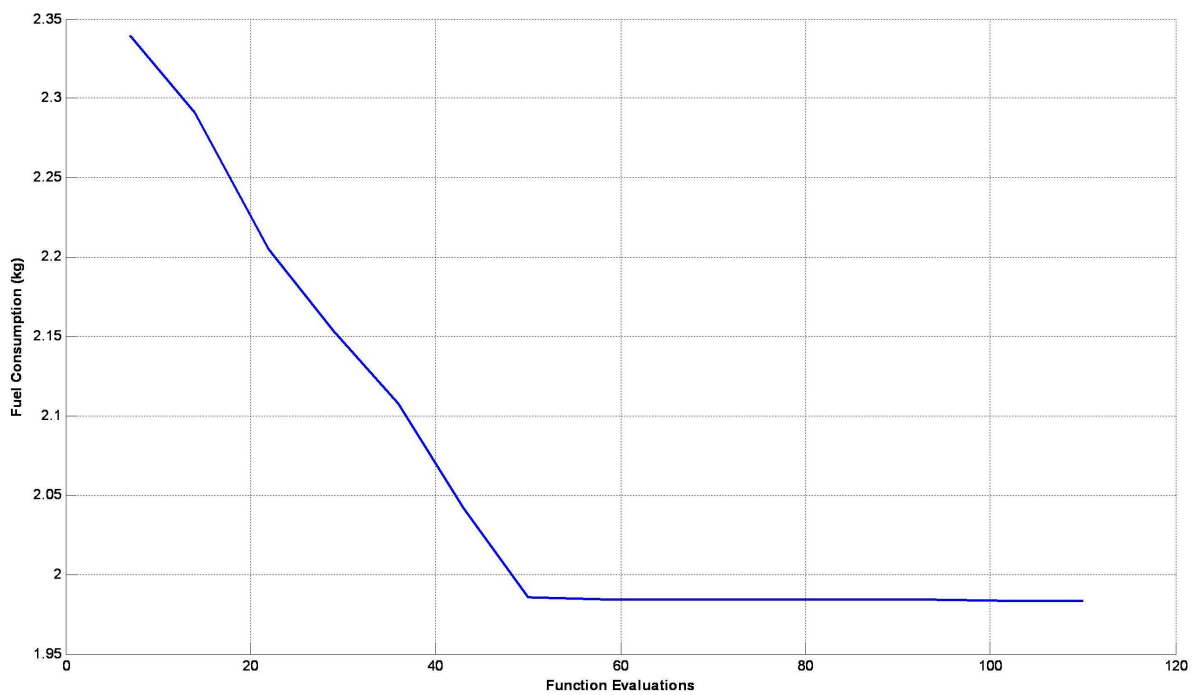


Figure 4.27: Fuel consumption vs function evaluations for MS2 (3 waypoints - SQP).

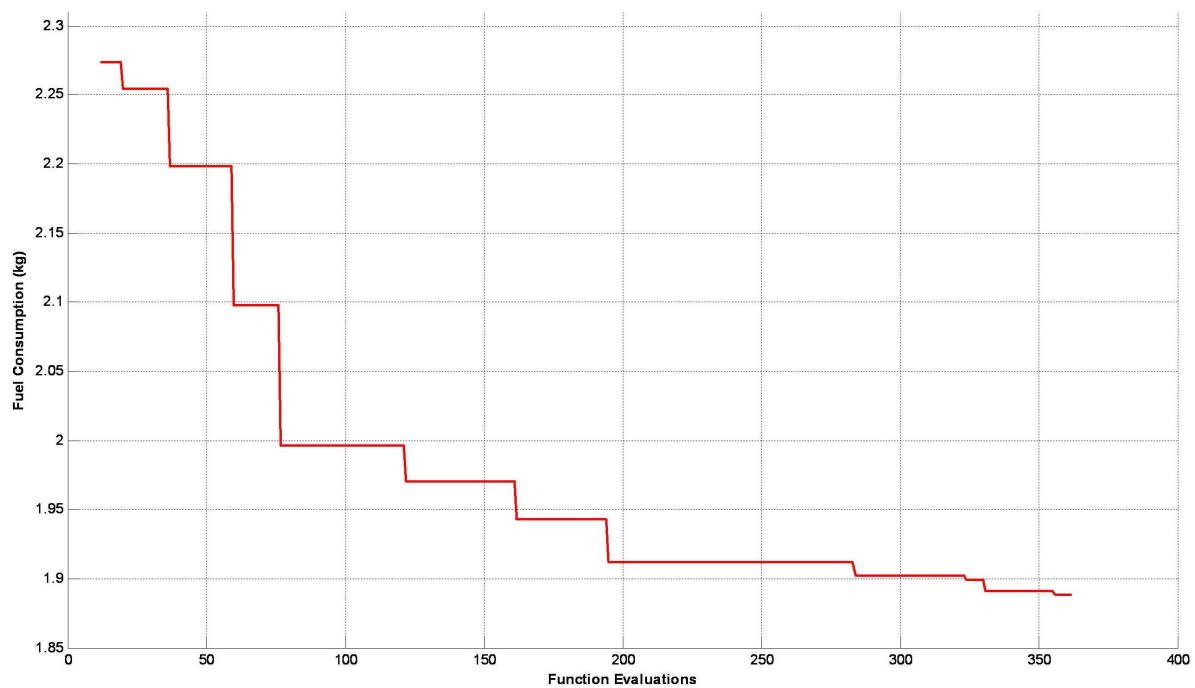


Figure 4.28: Fuel consumption vs function evaluations for MS2 (4 waypoints - HAPMOEA).

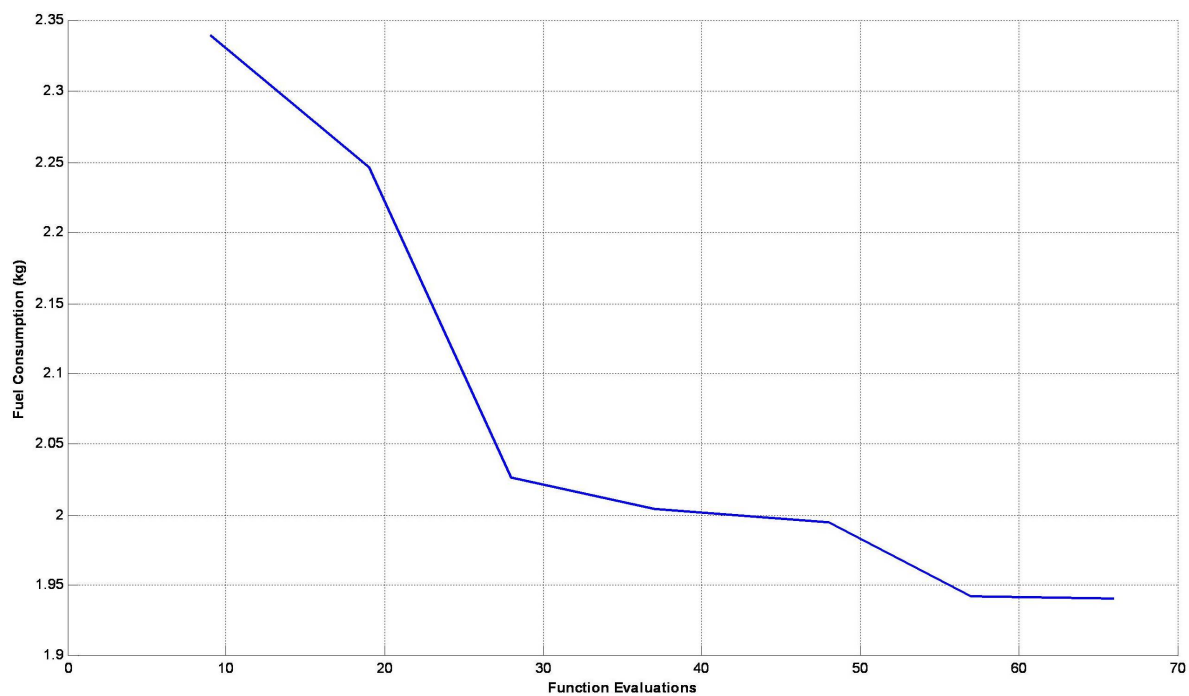


Figure 4.29: Fuel consumption vs function evaluations for MS2 (4 waypoints - SQP).

Table 4.6 compares the optimisation parameters, the run time statistics and the fitness values when using the two optimisers. The fitness values were also compared to that of the baseline mission, which is listed in the first entry.

# Waypoints / Design Variables Optimised	Optimisation Method	Function Evaluations Performed	Time Taken	Fuel Consumption (kg)
-	Baseline	-	8min 4sec	2.33982
1 (2 DV)	HAPMOEA	243	3hrs	2.05463
	SQP	25	1hr 9min	2.24454
2 (4 DV)	HAPMOEA	281	3hrs	1.96680
	SQP	26	57min	2.07742
3 (6 DV)	HAPMOEA	320	3hrs	1.90489
	SQP	110	3hrs 21min	1.98364
4 (8 DV)	HAPMOEA	362	3hrs	1.88818
	SQP	66	2hrs 10min	1.94054

Table 4.6: Table of statistics for SOMWO (MS2).

It can be seen from Table 4.6 that both the HAPMOEA and SQP optimisers have improved the fuel consumption, with the HAPMOEA results showing greater improvements over those of SQP in this case. A likely explanation for this is the larger search region encompassed by the waypoint bounds in MS2 as compared to MS1, which would favour the HAPMOEA optimiser.

The optimised waypoints - in their latitude and longitude coordinates - are displayed in Table 4.7 and illustrated in Figures 4.30 to 4.33.

# Waypoints / Design Variables	Optimisation Method	WPT #	Optimised Latitude (rad)	Optimised Longitude (rad)
1 (2 DV)	HAPMOEA	2	0.463665641	2.650366495
	SQP	2	0.463686297	2.650368469
2 (4 DV)	HAPMOEA	2	0.463762844	2.650438574
		3	0.463751545	2.650503195
	SQP	2	0.463832681	2.650477333
		3	0.463832681	2.650544524
3 (6 DV)	HAPMOEA	2	0.463750116	2.650448115
		3	0.463747647	2.650556192
		7	0.463751847	2.650472011
	SQP	2	0.463832681	2.650464771
		3	0.463832681	2.650517603
		7	0.463744975	2.650502581
4 (8 DV)	HAPMOEA	2	0.463767637	2.650455658
		3	0.463764437	2.650554251
		7	0.463727209	2.650518072
		8	0.463755385	2.650404785
	SQP	2	0.463832681	2.650474837
		3	0.463832681	2.650527535
		7	0.463832681	2.650518644
		8	0.463832681	2.650465382

Table 4.7: Table of optimised waypoints for SOMWO (MS2).

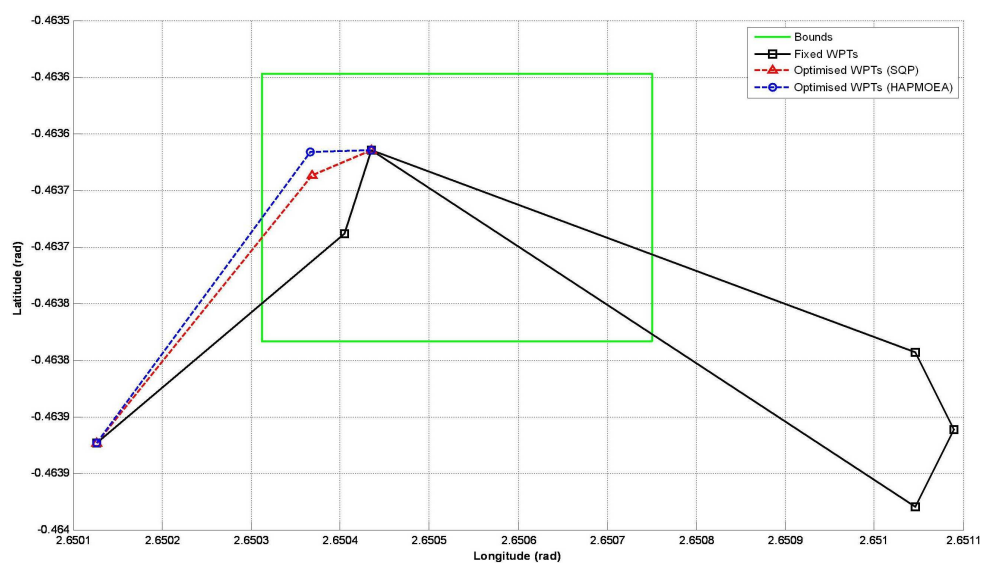


Figure 4.30: Comparison of optimised waypoints for MS2 (1 waypoint optimised).

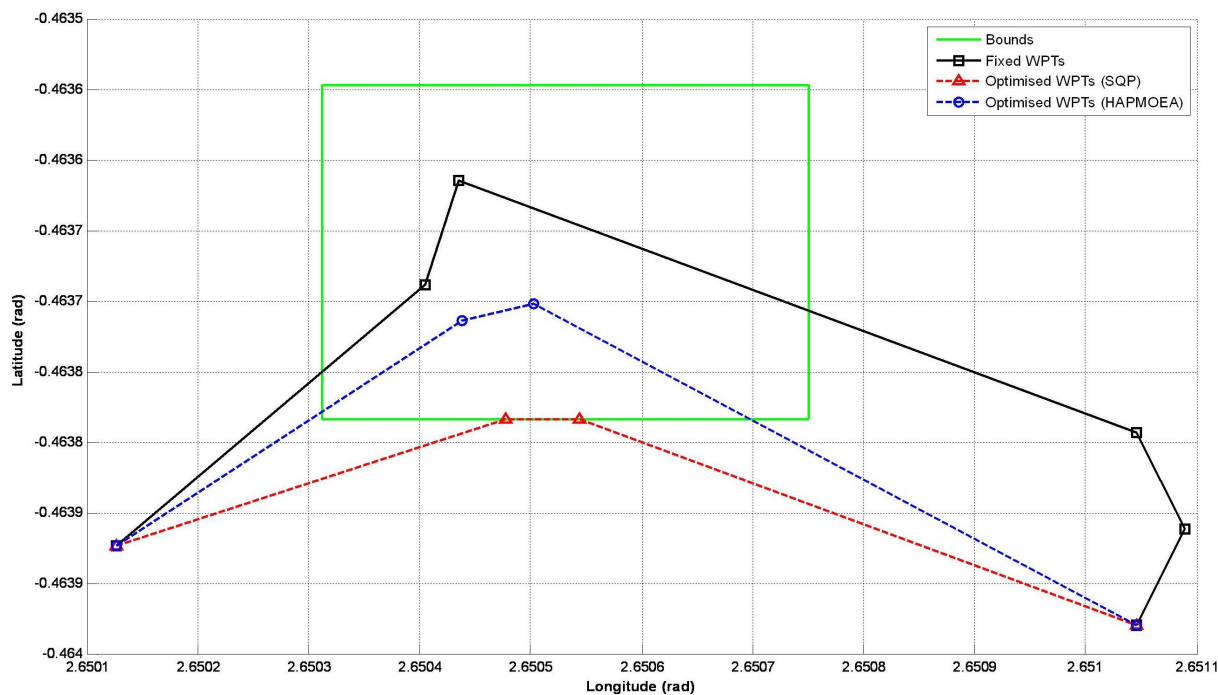


Figure 4.31: Comparison of optimised waypoints for MS2 (2 waypoints optimised).

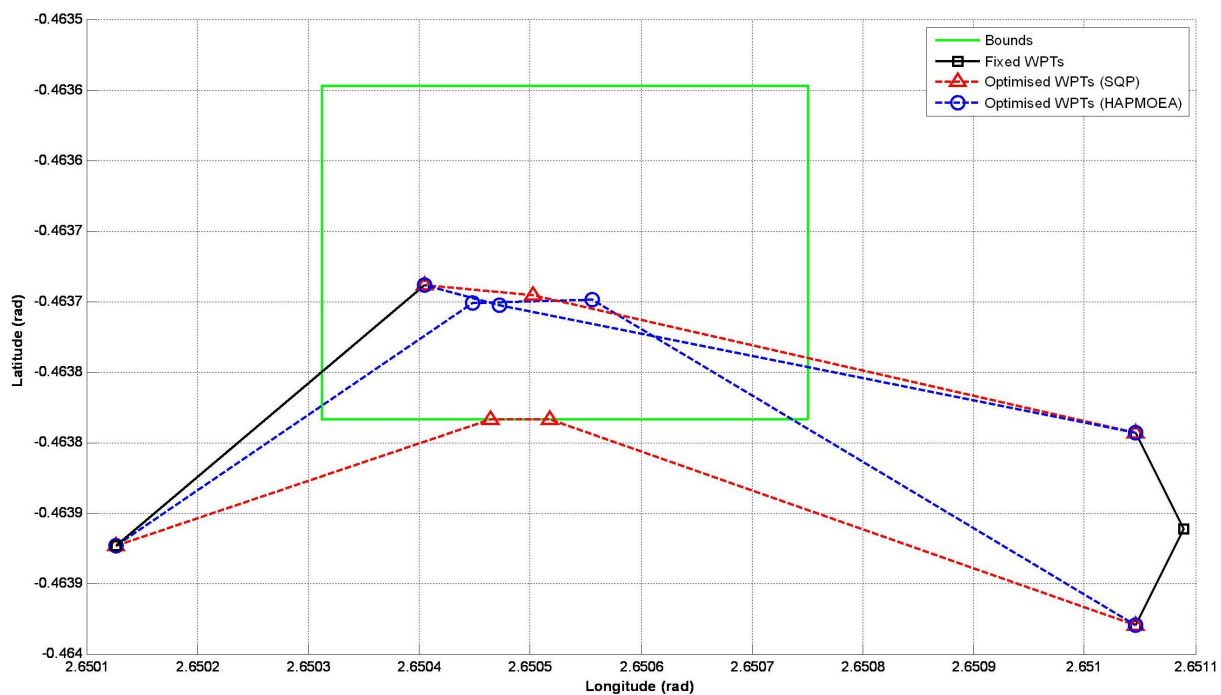


Figure 4.32: Comparison of optimised waypoints for MS2 (3 waypoints optimised).

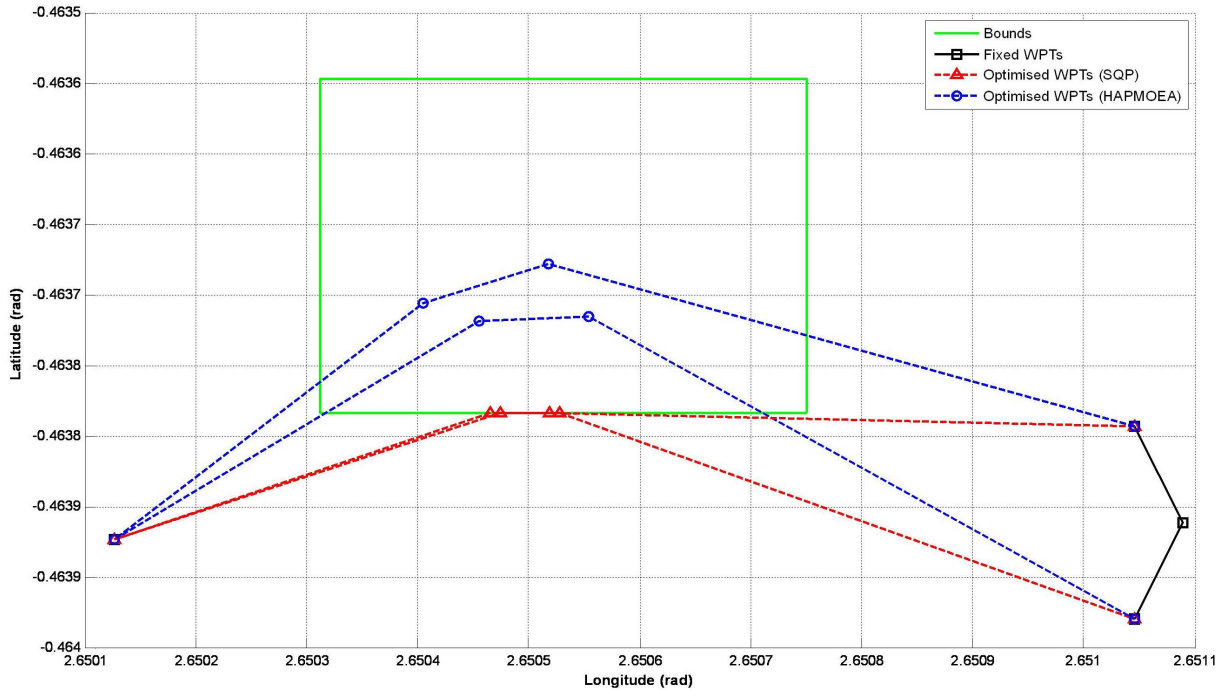


Figure 4.33: Comparison of optimised waypoints for MS2 (4 waypoints optimised).

4.10 Conclusions

The aim of this chapter was to couple the UAV simulation constructed in Chapter 3 into a MOEA optimiser to form a MOEA simulation optimisation method for solving a MWO problem.

In this chapter, a detailed description of the SOMWO problem was given, including the definitions of the optimisation parameters.

Two optimisation methods were described. The UAVSM was coupled with two optimisers, the HAPMOEA optimiser and the SQP solver, but the experience was not without problems. As mentioned in §2.2, merging the MATLAB-based UAVSM and the C++-based HAPMOEA optimiser was made more challenging because the HAPMOEA optimiser only worked under the LINUX environment, whereas the AeroSim Blockset was developed for use in the Windows environment. In the end, the problematic executables which some AeroSim blocks were compiled in a format that can be used under Linux and the subsequent development process was comparatively smooth.

Each optimiser was able to optimise from one to four waypoints, or 2 to 8 decision variables, the results of which are shown in Tables 4.6 and 4.7 for MS1 and MS2 respectively. It is evident that the SQP solver is significantly more efficient than the HAPMOEA optimiser in the case of MS1, which is very likely due to the fact that the search region for each waypoint is relatively small. As gradient-based methods, such as SQP, excel at optimising problems in a localised scale, the SQP solver is more appropriate than the HAPMOEA optimiser in this example of SOMWO. On the other hand, for MS2, the HAPMOEA optimiser consistently outperformed the SQP solver, since a larger search space is used in MS2.

It can be observed from the SOMWO results that the computational cost required to obtain these optimised values can still be a concern, especially when the SQP solver is used, since a time limit cannot be used as a stopping condition. For HAPMOEA, it must be noted that the distributed or parallel computing capability of the HAPMOEA optimiser was not explored in the course of this research and when implemented, the computational costs for the HAPMOEA optimiser will be expected to improve significantly.

Overall, the capabilities of two simulation optimisation methods integrating the HAPMOEA optimiser and the SQP solver respectively with the MATLAB-based UAVSM were demonstrated in this chapter, meeting Research Objectives 1 to 4 as listed in §1.2.

The next chapter, Chapter 5, will present the MOMWO process.

Chapter 5

Multi-Objective Mission Waypoint Optimisation

5.1 Introduction

As established in Chapter 2, the following two tasks are required in order to develop a MWO procedure for a small fixed-wing UAV, focusing on improving the fuel economy on the UAV:

1. Construction of a simulation environment of a small fixed-wing UAV; and
2. Coupling of the constructed UAV simulation model with a SQP solver and a Multi-Objective EA (MOEA) optimiser to form a MOEA simulation optimisation method for MWO.

Chapter 3 saw the implementation of UAVSM, a simulation model to represent an Aerosonde UAV that is capable of navigating through a given flight mission. Additionally, two mission scenarios - MS1 and MS2 - were constructed to be utilised in the MWO process, and the single-objective MWO problem was presented in Chapter 4.

This chapter expands on the SOMWO problem and presents the optimisation of a multi-objective MWO (MOMWO) problem, as described in §5.2. The results of the MOMWO process are presented and analysed in §5.7, and concluding remarks are given in §5.8.

5.2 Optimisation Problem Definition

The MOMWO problem extends upon the existing SOMWO problem described in §4.2, which considers the optimisation of a given series of waypoints that forms a flight mission, by adding a second objective in addition to the original objective of minimising the onboard fuel consumption. Two cases of this second objective was considered and are described in §5.4. The baseline flight mission is defined in Table 3.3, of which Waypoints 1 and 9 (which are the start and end points, and are the same location), and 4 to 6 are not involved in the MOMWO process, as was the case in the SOMWO.

5.3 Design Variables

The design variables for MOMWO are the same ones used in the SOMWO process, as described in §4.3, and are the coordinates – latitude and longitude – of the waypoints. These coordinates are measured in radians, and South latitudes and East longitudes are taken as positive values.

5.4 Fitness Functions

5.4.1 Fitness Functions for Mission Scenario 1

As stated in §5.2, the MOMWO problem considers two objectives. In the case of MS1, these objectives are to minimise the total onboard fuel consumption, FC and maximise the mission time, T_m , over 100 laps of the flight mission formed by the candidate waypoints. These objectives are defined as fitness functions, f_1 and f_2 respectively as follows:

$$\min(f_1) : f_1 = FC \quad (5.1)$$

$$\max(f_2) : f_2 = T_m \quad (5.2)$$

As stated in §4.4 for SOMWO, the UAV executes **five laps** of the flight mission during each function evaluation, and the fuel consumption and mission time over the five laps,

FC_{5laps} and $T_{m,5laps}$ respectively, are recorded. The total fuel consumption and mission time over the 100 laps are estimated by:

$$FC = 20FC_{5laps} \quad (5.3)$$

$$T_m = 20T_{m,5laps} \quad (5.4)$$

5.4.2 Fitness Functions for Mission Scenario 2

For MS2, a different second objective, essentially a risk factor, was considered in addition to the objective of minimising fuel consumption. Two Hazard Areas, A1 and A2, were set up in nearby areas of the MS2 waypoints as listed in Table 3.4 and shown in Figure 5.1. These are high risk areas which should be avoided by the UAV during its mission, therefore it is desired to maximise the distance between the UAV and each of these Hazard Areas.

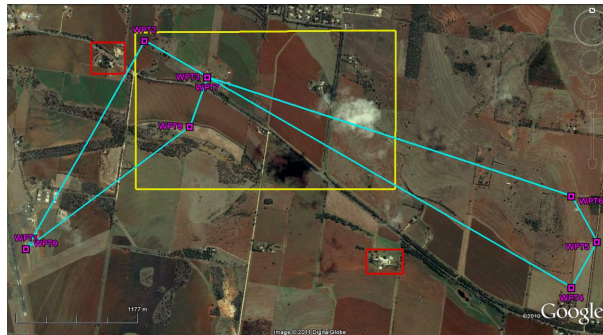


Figure 5.1: The Hazard Areas (in red) near the MS2 waypoints.

This risk factor is defined as:

$$\min(f_2) : f_2 = \frac{1}{dist_1} + \frac{1}{dist_2} \quad (5.5)$$

where $dist_1$ and $dist_2$ are the shortest distances from A1 and A2, respectively, that the UAV reaches in the duration of the mission. This distance is calculated by firstly sectioning the regions surrounding a Hazard Area as in Figure 5.2. If the UAV is in the corner sections, i.e. S1, S3, S6 and S8, the nearest point in the Hazard Area from the UAV is taken to be the corner point (e.g. (lon_1, lat_1) in S1) and the distance from the UAV to this point

is calculated using the great circle formula; see Equation 3.2. If the UAV is in the other sections, then the nearest point is the point that the perpendicular path from the UAV to the Hazard Area border intersects with the border, and the distance is calculated.

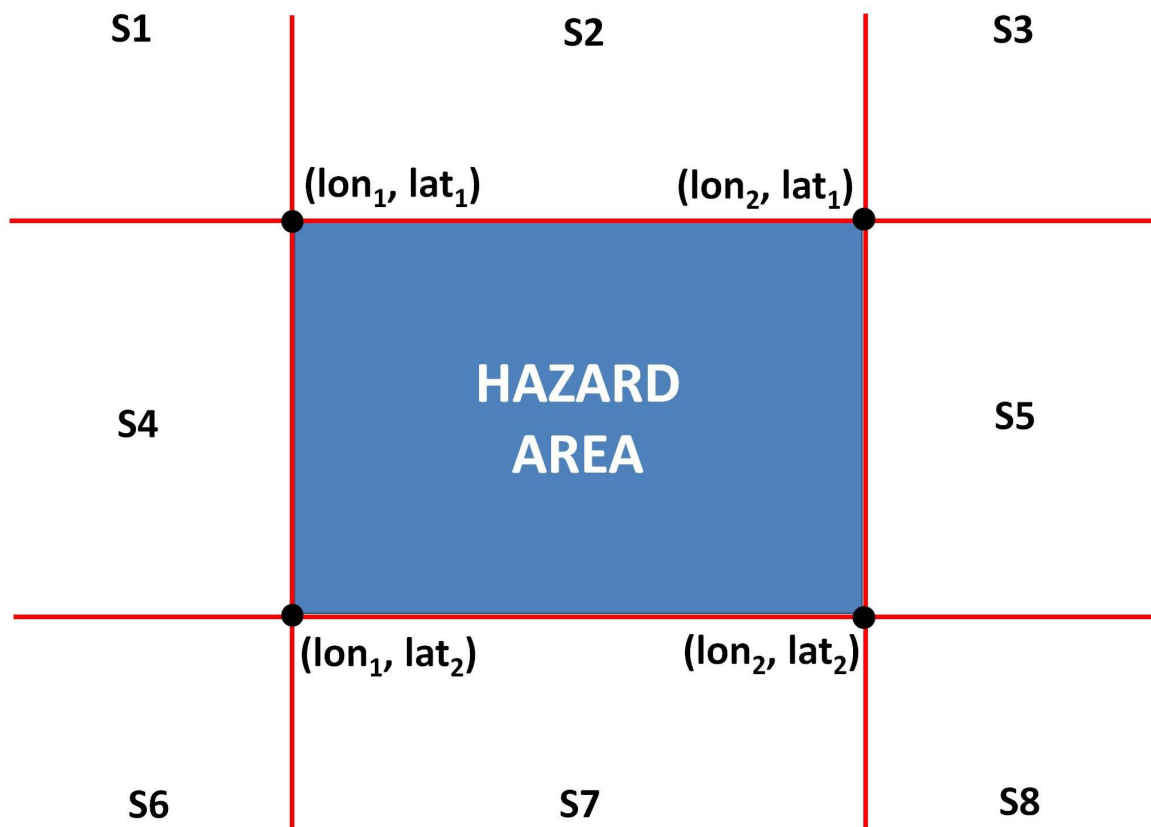


Figure 5.2: Sectioning of the region around a Hazard Area.

At any one point in time during the flight mission, the distances from the UAV to each of these Hazard Areas are calculated and compared to previous shortest distances so that the overall shortest distances are determined.

5.5 Upper and Lower Bounds of a Waypoint

The waypoint bounds used in the MOMWO process are identical to those used in the SOMWO in §4.5. These are re-iterated in Tables 5.1 and 5.2 and are illustrated in Figures 4.4 and 5.1.

Waypoint #	Coordinate	Lower Bound	Upper Bound
2	Latitude	0.463642241	0.463693195
	Longitude	2.650312648	2.650340341
3	Latitude	0.463700000	0.463720000
	Longitude	2.650397006	2.650456443
7	Latitude	0.463700000	0.463720000
	Longitude	2.650400061	2.650456445
8	Latitude	0.463741362	0.463762205
	Longitude	2.650340137	2.650367830

Table 5.1: MOMWO waypoint bounds for MS1.

Coordinate	Lower Bound	Upper Bound
Latitude	0.46359639	0.46383268
Longitude	2.65031243	2.65075036

Table 5.2: MOMWO waypoint bounds for MS2.

5.6 Physical Constraints

The constraints considered in the MOMWO process are of two types – the upper and lower bounds of waypoint coordinates, and physical constraints. These are the same constraints used in the SOMWO process and are described in §4.6 and are listed in Tables 4.1, 4.2 and 4.3 respectively. Also, the UAV’s airspeed is maintained at 20m/s throughout the mission, except when in descent, during which the airspeed increases to 30m/s.

5.6.1 Mathematical Formulation of MOMWO Problem

The mathematical formulation of the MOMWO is as follows:

$$\begin{aligned}
 \min_{\mathbf{x}} \quad & \mathbf{y} = f(\mathbf{x}) \\
 \text{subject to} \quad & \mathbf{lb} < \mathbf{x} < \mathbf{ub}
 \end{aligned} \tag{5.6}$$

where $\mathbf{x} = (lat_1, lon_1, lat_2, lon_2, \dots, lat_n, lon_n)^T$ is the set of coordinates (latitude and

longitude) of n waypoints that is to be optimised, $y = (FC, 1/T_m)$ are the two objective functions of fuel consumption and mission time¹ for MS1 and $y = (FC, \frac{1}{dist_1} + \frac{1}{dist_2})$ are the objective functions of fuel consumption and risk factor for MS2, $\mathbf{lb} = (latLB_1, lonLB_1, latLB_2, lonLB_2, \dots, latLB_n, lonLB_n)^T$ and $\mathbf{ub} = (latUB_1, lonUB_1, latUB_2, lonUB_2, \dots, latUB_n, lonUB_n)^T$ are the lower and upper bound vectors, respectively, for the optimised waypoints.

5.6.2 Optimiser Setup

In the implementation of MOMWO, the same two optimisers as in SOMWO – HAPMOEA optimiser and SQP solver – were used. The MATLAB-based UAVSM was used with both optimisers in the SOMWO process and the fundamental sample time, Δt , set at 0.1 seconds.

HAPMOEA Optimiser Setup

For the MOMWO problem, the HAPMOEA optimiser was set up with only one layer and contains the following:

- Population size = 40
- Parents in recombination = 2
- Buffer length = 42
- Tournament-in-buffer ratio = 2.0

The population size and buffer length were increased from the settings for SOMWO (see §4.8.1) so a better observation of the Pareto members and non-members that were generated during the optimisation process.

HAPMOEA takes into account the lower and upper bound constraints (see §5.5) and generates candidate waypoints with their *latitude* and *longitude* coordinates as the design variables. The waypoint table constructed using these candidate waypoints are passed to UAVSM to be evaluated. The outputs of the UAVSM is the *fuel consumption* and *mission*

¹Because mission time is to be maximised, but the optimisation process in fact minimises the fitness functions, therefore the fitness for mission time was converted to its inverse, or $1/T_m$.

time over the mission for MS1, and the *fuel consumption* and *risk factor* as determined in §5.4.2 for MS2. These are passed to HAPMOEA to be optimised. Refer to Appendix E for relevant excerpts of the HAPMOEA code for performing MOMWO.

Note that an initial estimates vector is not required for HAPMOEA.

MATLAB SQP Solver Setup

Unlike HAPMOEA, the SQP solve does not have the capability of performing multi-objective optimisations because the *fmincon* function can only process scalar objective functions, i.e. one objective only. This was overcome by combining the two objective functions, *FC* and $1/T_m$, using a weighted sum approach [162]. In this approach, the two objective functions are added together using different weighting coefficients for each one of them, which transforms the MOMWO problem into a scalar optimisation problem of the form:

$$\min w_1 f_1 + w_2 f_2 \quad (5.7)$$

where $w_i \geq 0$ are the weighting coefficients representing the relative importance of the objectives. It is usually assumed that:

$$\sum w_i = 1 \quad (5.8)$$

For the MOMWO process, the weighting coefficients are varied from 0 to 1 in increments of 0.05, and one optimisation run was performed for each set of weighting coefficients. This will result in 19 “optimised” points on the Pareto plot.

The relevant MATLAB code can be found in Appendix F for reference.

5.7 Optimisation Results and Analysis

The optimisation procedure for each optimiser used different stopping conditions:

- HAPMOEA optimiser: a time limit of 18 hours on one machine only

- SQP solver: Default stopping condition (terminates if the magnitude of the directional derivative in the search direction is less than 2×10^{-6} and the maximum constraint violation is less than 1×10^{-6}) in addition to the maximum iteration count of 20 iterations.

5.7.1 Mission Scenario 1

Figures 5.3 and 5.4 illustrate the solutions from the HAPMOEA optimiser and SQP solver respectively, with the number of waypoints being optimised as four waypoints (Waypoints 2, 3, 7 and 8).

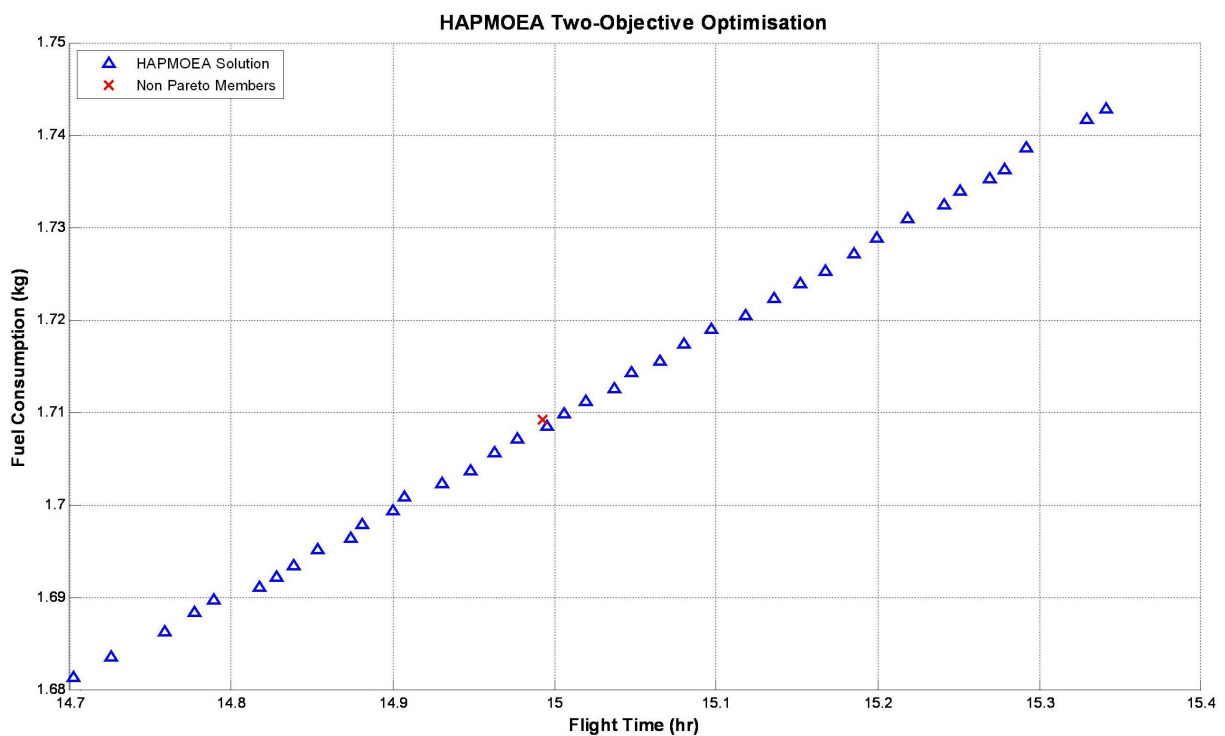


Figure 5.3: HAPMOEA Optimiser results

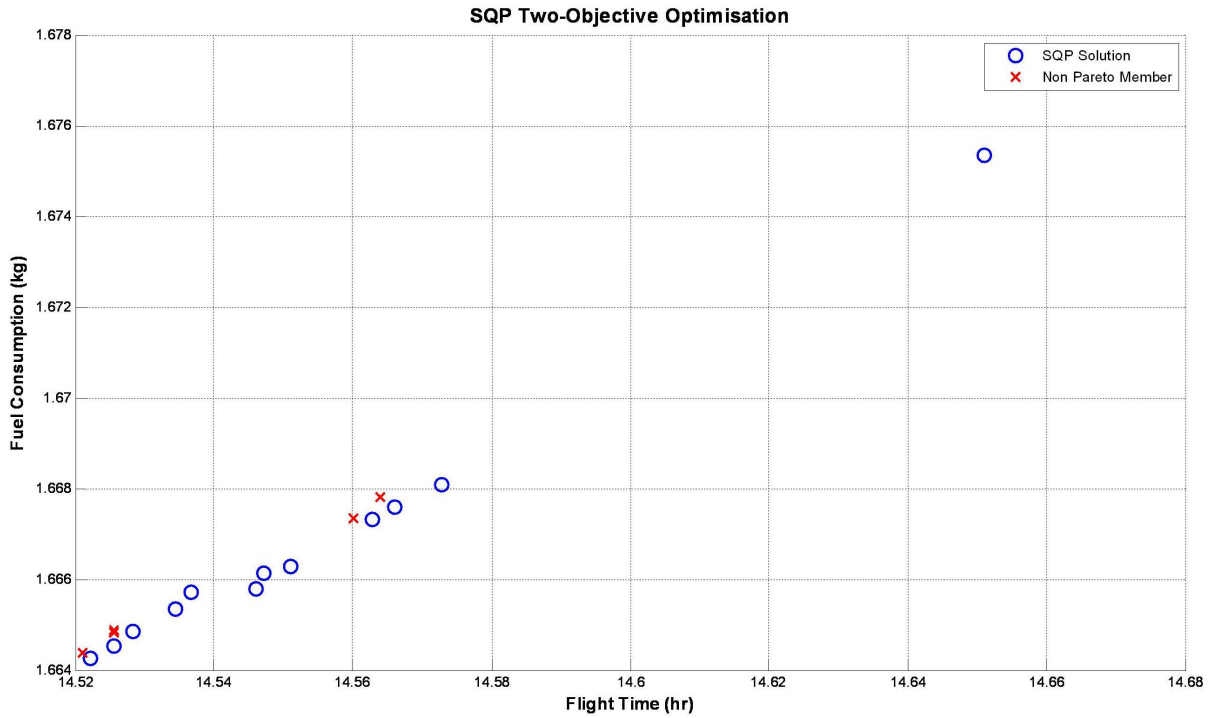


Figure 5.4: SQP solver results

Tables 5.3 and 5.4 display the solutions and their run time statistics and the fitness values for each optimiser.

It can be seen that while the solutions found by the SQP solver have lower fuel consumption values, they also have significantly lower mission time values when compared to the solutions found by the HAPMOEA optimiser. Since the MOMWO problem aims to minimise the fuel consumption and maximise the mission time, it seems that the SQP solver, using the weighted sum approach, does not provide a balanced optimisation between the two objective functions and there is much variation between the fitness values obtained using this approach. On the other hand, the fitness values from the HAPMOEA optimiser provide a better balance between minimising the fuel consumption and maximising the mission time.

With regards to the run time, HAPMOEA performed the optimisation in 18 hours, which was the maximum run time used as the stopping condition for the optimiser, while the SQP solver required a total of 38.45 hours to perform the optimisation using 19 sets of weight coefficients.

Run Time (hr)	HAPMOEA Solution #	Fuel Consumption (kg)	Mission Time (hr)	HAPMOEA Solution #	Fuel Consumption (kg)	Mission Time (hr)
18	1	1.68142562	14.702	21	1.71131979	15.037
	2	1.68365991	14.726	22	1.71267527	15.048
	3	1.68636541	14.759	23	1.71437602	15.065
	4	1.68840183	14.777	24	1.71562739	15.080
	5	1.68984161	14.789	25	1.71741699	15.097
	6	1.69117107	14.828	26	1.71905349	15.118
	7	1.69233406	14.838	27	1.72050302	15.136
	8	1.69354510	14.838	28	1.72239056	15.152
	9	1.69524439	14.853	29	1.72406186	15.168
	10	1.69653692	14.874	30	1.72532036	15.185
	11	1.69799614	14.881	31	1.72726686	15.199
	12	1.69940997	14.900	32	1.72895751	15.218
	13	1.70092973	14.907	33	1.73100991	15.241
	14	1.70240555	14.931	34	1.73255030	15.251
	15	1.70378363	14.948	35	1.73404760	15.269
	16	1.70575365	14.963	36	1.73539729	15.278
	17	1.70719733	14.977	37	1.73629716	15.292
	18	1.70852489	14.995	38	1.73871166	15.329
	19	1.70930684	15.006	39	1.74293000	15.341
	20	1.70994184	15.019			

Table 5.3: Table of MOMWO statistics for HAPMOEA optimiser (MS1).

Therefore, for the purpose of MOMWO, it seems that the HAPMOEA optimiser is significantly more efficient, while generating a more balanced set of results.

Note that the plots generated by both the HAPMOEA and SQP optimisers both display a fairly linear relationship between the Pareto members. A possible reason for this is that the two fitness functions, fuel consumption and mission time, are correlated linearly. Further investigation will be required on this aspect.

5.7.2 Mission Scenario 2

Figure 5.5 illustrates the Pareto members for both optimisers, with the number of waypoints being optimised as four waypoints (Waypoints 2, 3, 7 and 8).

SQP Solution #	Run Time (hr)	Fuel Consumption (kg)	Mission Time (hr)
1	2.631	1.664264	14.5222
2	2.644	1.664264	14.5222
3	2.611	1.664526	14.5256
4	3.280	1.664852	14.5283
5	3.225	1.665344	14.5344
6	2.358	1.665719	14.5256
7	1.116	1.665793	14.5461
8	1.096	1.666134	14.5472
9	1.227	1.666278	14.5511
10	1.799	1.667333	14.5628
11	2.227	1.667587	14.5661
12	1.319	1.668095	14.5728
13	0.949	1.675356	14.6511
Total Run Time:	38.4503 hrs		

Table 5.4: Table of MOMWO statistics for SQP solver (MS1).

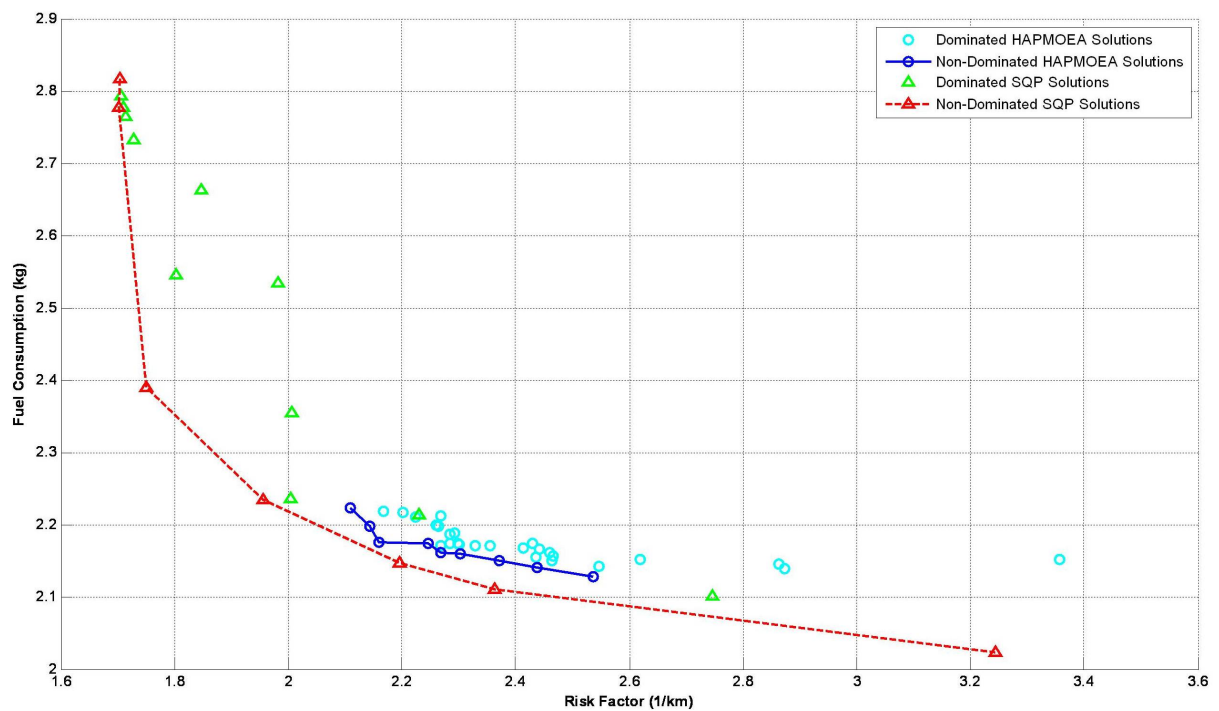


Figure 5.5: Pareto plot of HAPMOEA Optimiser and SQP Solver results (MS2).

Tables 5.5 and 5.6 display the Pareto members and their run time statistics and the fitness values for each optimiser.

Run Time (hr)	Pareto Member #	Fuel Consumption (kg)	Risk Factor (1/km)
18	1	2.12847973	2.53608465
	2	2.13999824	2.43775235
	3	2.15023037	2.37180592
	4	2.1594097	2.3018945
	5	2.16060612	2.26741787
	6	2.17326282	2.24682338
	7	2.17609764	2.15958842
	8	2.19772815	2.14428743
	9	2.223122	2.10868906

Table 5.5: Table of MOMWO statistics for HAPMOEA optimiser (MS2).

Pareto Member #	Run Time (hr)	Fuel Consumption (kg)	Risk Factor (1/km)
1	20.126	2.816810	1.704373
2	16.631	2.777218	1.701709
3	21.868	2.390286	1.749873
4	29.79	2.234099	1.956561
5	27.873	2.146928	2.196180
6	31.122	2.111094	2.362687
7	17.629	2.023689	3.245058
Total Run Time:		444.0811 hrs	

Table 5.6: Table of MOMWO statistics for SQP solver (MS2).

The most significant difference between the MOMWO results from the HAPMOEA optimiser and SQP solver is the computational time required to perform the MWO procedure. On one hand, the HAPMOEA optimiser was capable of performing the MOMWO efficiently in both time and ease of use, as well as the capability of ranking the Pareto members automatically. In comparison, the SQP solver required more than 20 times the computational time that the HAPMOEA optimiser used. Therefore, it can be concluded that the HAPMOEA optimiser is significantly the more appropriate optimiser

to use for MOMWO.

5.8 Conclusions

The aim of this chapter was to extend the work performed for the SOMWO problem in Chapter 4 into solving a MOMWO problem.

In this chapter, a detailed description of the MOMWO problem was given, including the definitions of the optimisation parameters.

The MOMWO process were performed using two different sets of fitness functions. In the case of MS1, the objectives were to minimise the fuel consumption and maximise the flight time. However, the optimisation results showed that these two objectives are somewhat linearly correlated and therefore these two objectives are perhaps not the most appropriate combination of objectives to be evaluated in this case. On the other hand, the objectives for MS2 were to minimise the fuel consumption and maximise the shortest distances from the UAV to the two Hazard Areas, A1 and A2. No obvious correlations were observed in this case.

One significant observation is the computational time required for the SQP solver to perform the MOMWO, which is more than 20 times that of the HAPMOEA optimiser. It should also be noted that in order to use the SQP solver to perform multi-objective optimisation, a weighted sum approach was used. This required the appropriate weights to be known beforehand, or extra evaluations would be required in order to determine the appropriate weights, thus introducing extra computational expense.

Overall, the capabilities of two simulation optimisation methods integrating the HAPMOEA optimiser and the SQP solver respectively with the MATLAB-based UAVSM to perform MOMWO were demonstrated in this chapter, meeting Research Objectives 1 to 4 as listed in §1.2.

The next chapters, Chapters 6 and 7, will describe the development of a Hybrid-Electric Propulsion System (HEPS) on an UAV. This was accomplished in two stages:

1. Performing of an Ideal Operating Line (IOL) analysis on an Aerosonde ICE (§6);
and

2. Development and implementation of a HEPS simulation model and its integration into the UAVSM (§7).

The details of each of these tasks are described in the chapters as indicated.

Chapter 6

Ideal Operating Line Analysis of Aerosonde ICE

6.1 Introduction

As mentioned in Chapter 5, the development of a Hybrid-Electric Propulsion System (HEPS) on an UAV was accomplished in two stages:

1. Performing of an Ideal Operating Line (IOL) analysis on an Aerosonde ICE; and
2. Development and implementation of a HEPS simulation model and its integration into the UAVSM.

As presented in §1.1.2, the literature review has shown that a Hybrid-Electric Propulsion System (HEPS) with a Continuously Variable Transmission (CVT) and utilising an Ideal Operating Line (IOL) control strategy has the potential to economise the energy efficiency on a small fixed-wing UAV.

In order for these components to work in an organised manner, a controller is required. Its primary function will be to determine when and how each of the components of the HEPS will work while the UAV is in operation. For this reason, it will need to take into account the status and states of each of the HEPS components, in order to generate appropriate signals to operate the components accordingly. In this research, the IOL control strategy is used in the controller.

This chapter presents the IOL analysis of the Aerosonde ICE, which is the basis of the IOL control strategy for the HEPS on an UAV. The resulting IOL data forms the basis for the modelling of HEPS.

6.2 Overview of IOL Analysis

The *Ideal Operating Line* (IOL), also known as the e-line, is a smooth line made up of all the points which represent the torque and speed combinations at which the fuel consumption is minimal on different power lines for steady-state conditions [163]. A powerplant operated on the IOL will, theoretically, enable the best performance while consuming the least amount of fuel possible.

Typically, finding the IOL for an engine requires the engine map, which is a plot of the engine's performance in terms of its RPM and torque output values, with the values represented as level contours of the corresponding brake specific fuel consumption (BSFC) values. A typical engine map with the IOL is shown in Figure 6.1 [163].

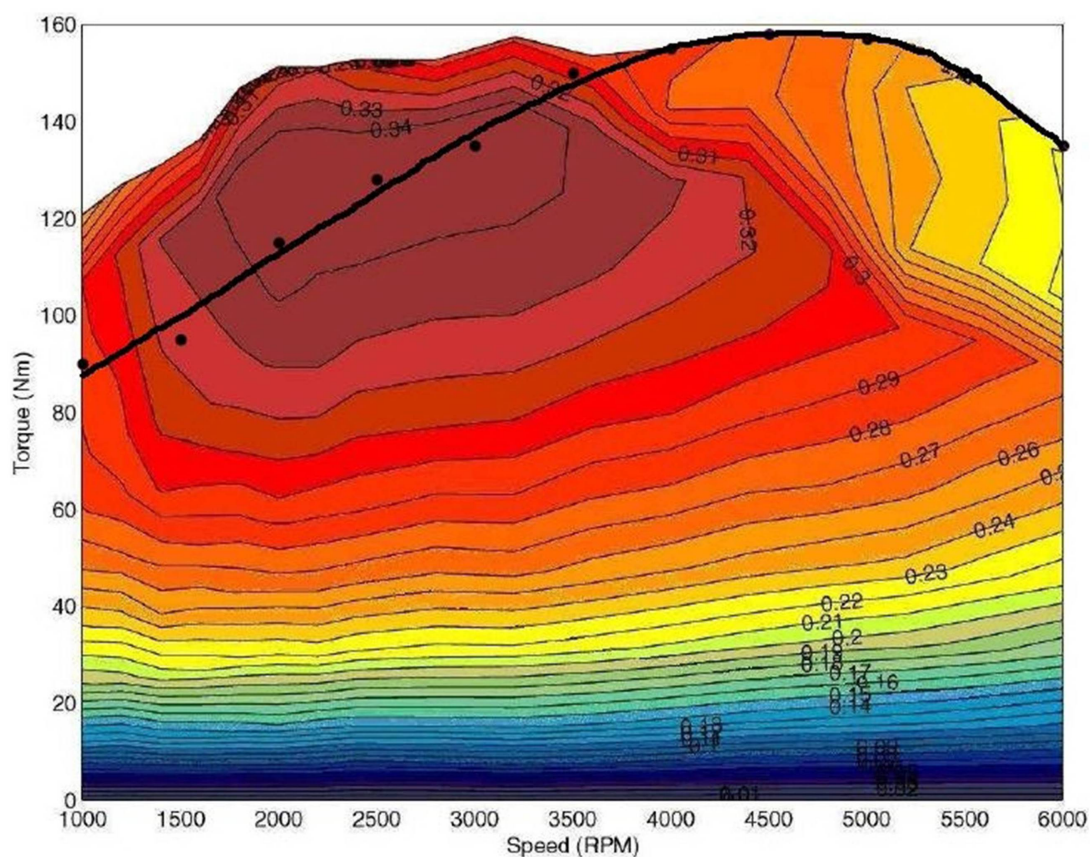


Figure 6.1: A typical engine map with the Ideal Operating Line (IOL)

To find the IOL, firstly lines of constant power output are plotted on the engine map. On each of these power lines, there is a point with the smallest fuel consumption. When all these points on all the power lines are connected together, the IOL is generated. Often these points do not form a smooth line, mainly due to the limited number of data points available in the engine map, as well as the very small fluctuations of fuel consumption in a rather large area. In order to obtain a workable IOL, a smooth line is fitted through these points and a polynomial function is created for this line.

6.3 Ideal Operating Line Analysis of Aerosonde ICE

The objective of this IOL analysis is to obtain the IOL for the ICE used on an Aerosonde UAV. The data for this ICE - the power output and fuel flow at given values of engine RPM and manifold pressure (MAP) - are available as part of the AeroSim Blockset [156] and are shown in Tables 6.1 and 6.2.

RPM	MAP								
	60	70	80	90	92	94	96	98	100
1500	18.85	47.12	65.97	67.54	69.12	67.54	67.54	69.12	86.39
2100	59.38	98.96	127.55	149.54	151.74	160.54	178.13	200.12	224.31
2800	93.83	149.54	187.66	237.50	249.23	255.10	307.88	366.52	398.77
3500	109.96	161.27	245.57	307.88	326.20	351.86	421.50	591.14	531.45
4500	164.93	245.04	339.29	438.25	447.68	494.80	565.49	673.87	772.83
5100	181.58	245.67	389.87	496.69	528.73	571.46	662.25	822.47	993.37
5500	184.31	293.74	403.17	535.64	570.20	622.04	748.75	956.09	1059.80
6000	163.36	276.46	420.97	565.49	609.47	691.15	860.80	1131.00	1193.80
7000	124.62	249.23	417.83	586.43	645.07	762.36	996.93	1246.20	1429.40

Table 6.1: Table of power output (W) at given values of RPM and MAP for the Aerosonde ICE

6.3.1 ICE Torque Calculations

Using the power output values for the Aerosonde ICE, the torque output can be calculated by:

RPM	MAP								
	60	70	80	90	92	94	96	98	100
1500	31	32	46	53	55	57	65	73	82
2100	40	44	54	69	74	80	92	103	111
2800	50	63	69	92	95	98	126	145	153
3500	66	75	87	110	117	127	150	175	190
4500	83	98	115	143	148	162	191	232	246
5100	93	102	130	159	167	182	208	260	310
5500	100	118	137	169	178	190	232	287	313
6000	104	126	151	184	191	206	253	326	337
7000	123	144	174	210	217	244	321	400	408

Table 6.2: Table of fuel flow (g/hr) at given values of RPM and MAP for the Aerosonde ICE

$$Torque = \frac{Power \cdot 60}{2\pi \cdot RPM} \quad (6.1)$$

where *Power* is in Watts (W) and the resulting *Torque* is in Newton-metres (Nm). The resulting torque output values are shown in Table 6.3.

RPM	MAP								
	60	70	80	90	92	94	96	98	100
1500	0.12	0.30	0.42	0.43	0.44	0.43	0.43	0.44	0.55
2100	0.27	0.45	0.58	0.68	0.69	0.73	0.81	0.91	1.02
2800	0.32	0.51	0.64	0.81	0.85	0.87	1.05	1.25	1.36
3500	0.30	0.44	0.67	0.84	0.89	0.96	1.15	1.34	1.45
4500	0.35	0.52	0.72	0.93	0.95	1.05	1.20	1.43	1.64
5100	0.34	0.46	0.73	0.93	0.99	1.07	1.24	1.54	1.86
5500	0.32	0.51	0.70	0.93	0.99	1.08	1.30	1.66	1.84
6000	0.26	0.44	0.67	0.90	0.97	1.10	1.37	1.80	1.90
7000	0.17	0.34	0.57	0.80	0.88	1.04	1.36	1.70	1.95

Table 6.3: Table of torque output (Nm) at given values of RPM and MAP for the Aerosonde ICE

6.3.2 ICE BSFC Calculations

On the other hand, the brake specific fuel consumption (BSFC) values can be calculated using the power output and fuel flow values as follows:

$$BSFC = \frac{FuelFlow}{Power} \quad (6.2)$$

where *FuelFlow* is in grams per hour (g/hr), *Power* is in Watts (W) and the resulting *BSFC* is in grams per Watt-hour (g/W-hr). The BSFC values are shown in Table 6.4.

RPM	MAP								
	60	70	80	90	92	94	96	98	100
1500	1.6446	0.6791	0.6973	0.7847	0.7957	0.8439	0.9624	1.0561	0.9492
2100	0.6736	0.4446	0.4234	0.4614	0.4877	0.5983	0.5165	0.5147	0.4949
2800	0.5329	0.4213	0.3677	0.3874	0.3812	0.3842	0.4093	0.3956	0.3837
3500	0.6002	0.4651	0.3543	0.3573	0.3587	0.3609	0.3559	0.3563	0.3575
4500	0.5032	0.3999	0.3389	0.3263	0.3306	0.3274	0.3378	0.3443	0.3183
5100	0.5122	0.4152	0.3334	0.3201	0.3159	0.3185	0.3141	0.3161	0.3121
5500	0.5426	0.4017	0.3398	0.3155	0.3122	0.3054	0.3098	0.3002	0.2953
6000	0.6366	0.4558	0.3587	0.3254	0.3134	0.2981	0.2939	0.2882	0.2823
7000	0.9870	0.5778	0.4164	0.3581	0.3364	0.3201	0.3220	0.3210	0.2854

Table 6.4: Table of BSFC (g/W-hr) values at given values of RPM and MAP for the Aerosonde ICE

6.3.3 Engine Map

In order to generate the engine map from these data, the in-built MATLAB function *contour* was utilised and the resulting contour plot is shown below in Figure 6.2. Each of the lines, called contours, are formed by joining the operating points that have the same BSFC value. Because there is equal spacing of BSFC values from one contour to the next, a region of dense contours represents the presence of a “steep” slope, whereas a region of widely spaced contours illustrates a “gentle” slope. In other words, an engine map is similar to a relief map, but instead of showing contours of equal altitude like a relief map does, an engine map shows contours of equal BSFC. In this case, the computation of these contours involved much estimation by inter- and extrapolation, since the available data

are fairly limited in their descriptions of these engine characteristics.

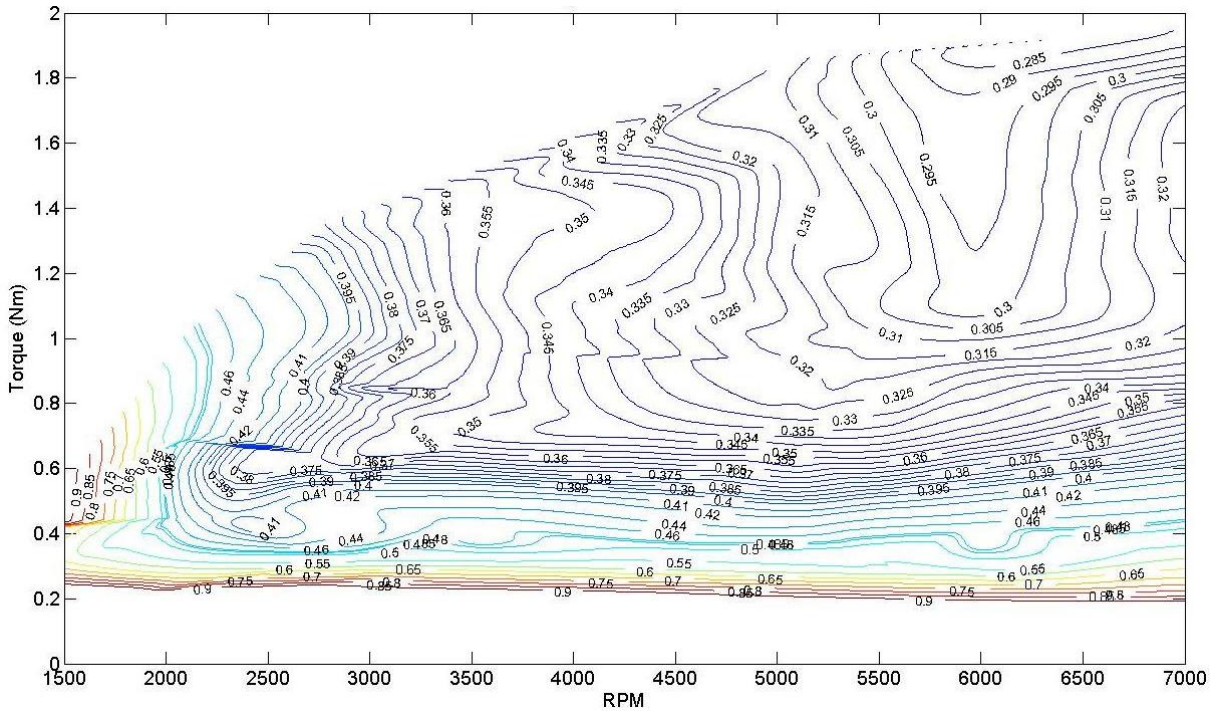


Figure 6.2: Engine fuel map with BSFC contours using Aerosonde ICE data only

However, it seemed as though this fuel map should have more data in the upper regions of the plot, therefore it was decided to obtain more ICE data values. This was conducted using the inner components of the *Piston Engine* block from the AeroSim Blockset in the form of *FuelMapCalc.mdl* as shown in Figure 6.3, and with the default Aerosonde data. By setting the throttle value to 1 for full throttle and varying the static pressures to simulate the changing in altitude (values obtained from the standard atmosphere data were used in the Aerosonde model) and the RPM, new values of power and fuel flow were obtained. Using Equations 6.1 and 6.2, additional values for torque and BSFC that correspond to each were calculated. For a detailed listing of these values, see Appendix G. The new fuel map was generated using the *contour* function as described previously, and is shown in Figure 6.4. It can be seen from this plot that there is more detail in the higher torque regions, including an “island” at approximately 4500 RPM.

6.3.4 Determining the IOL

The next step was determining the IOL. In order to do this, the power contours are required. These were obtained using the *contour* function in MATLAB, as per the BSFC

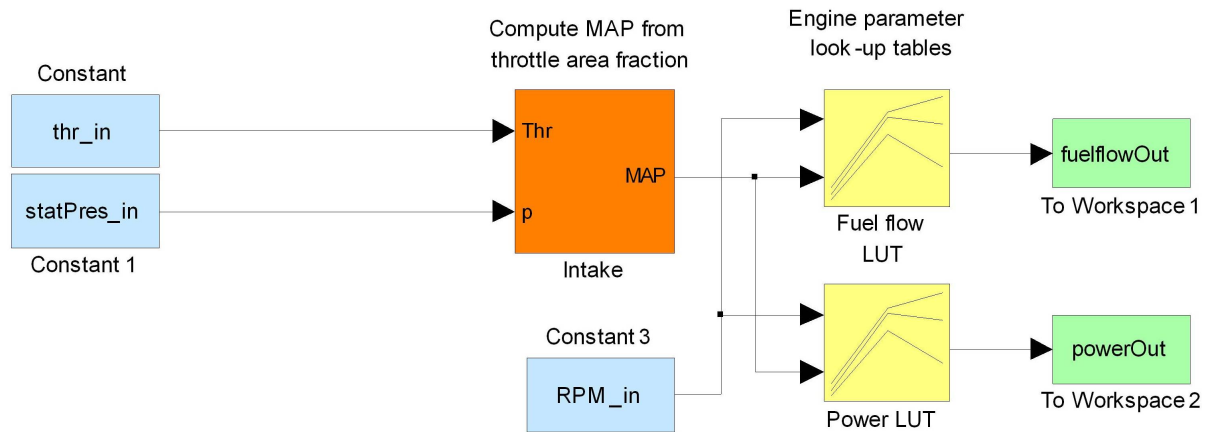


Figure 6.3: FuelMapCalc.mdl, the Simulink model based on the Piston Engine block

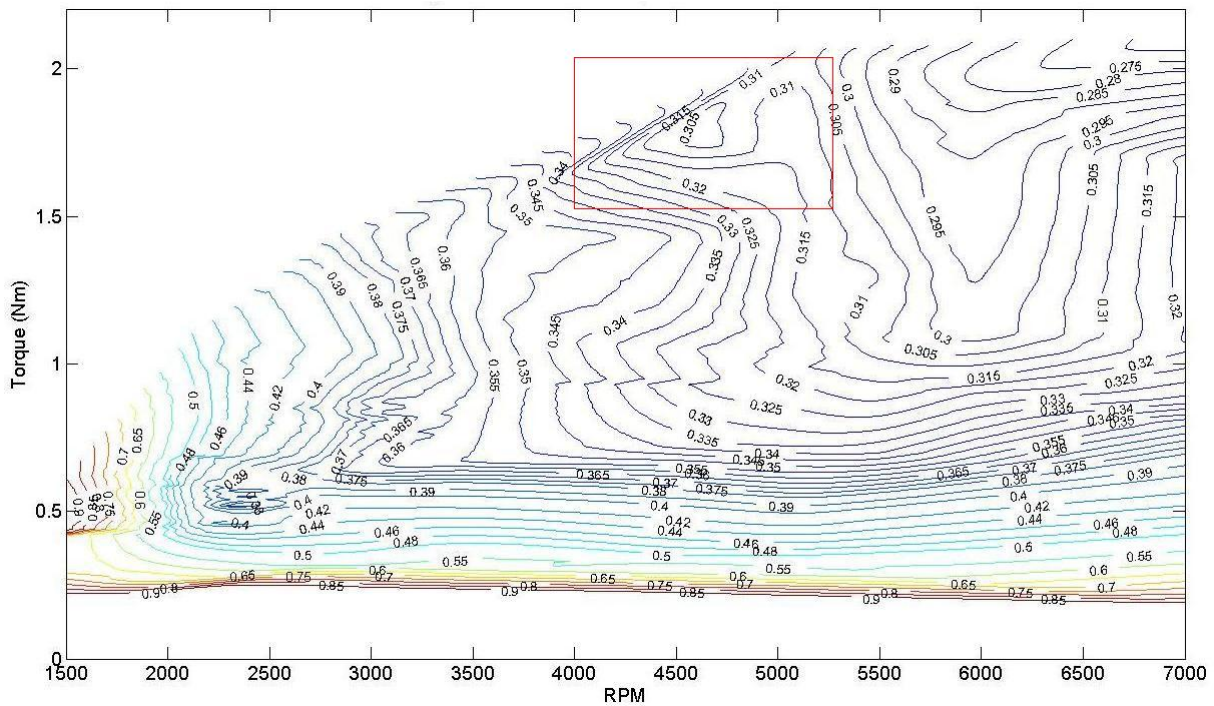


Figure 6.4: Engine fuel map with BSFC contours using Aerosonde ICE data and open throttle calculations, with the “new island” boxed in red

contours, and are shown in Figure 6.5. These power contours were then superimposed onto the engine map, shown in Figure 6.6.

On each of the power contours, the point with the smallest BSFC value was identified manually¹, and connecting the point for all the power contours on the plot gives the IOL,

¹The IOL was identified manually because the power and BSFC contours were computed using the *contour* function in MATLAB, therefore determining the gradient of each of those curves was very difficult. For the purpose of this research, it was deemed that manual identification of the IOL was sufficient.

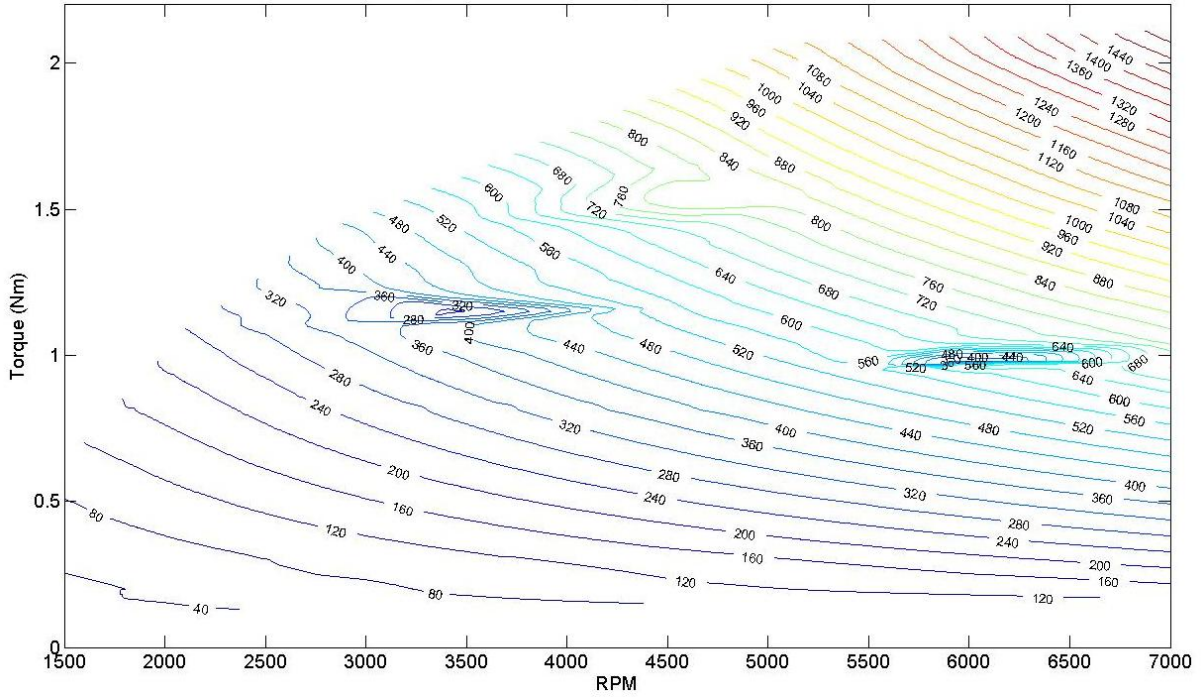


Figure 6.5: Constant power contours for the Aerosonde ICE, obtained using Aerosonde ICE data and open throttle calculations

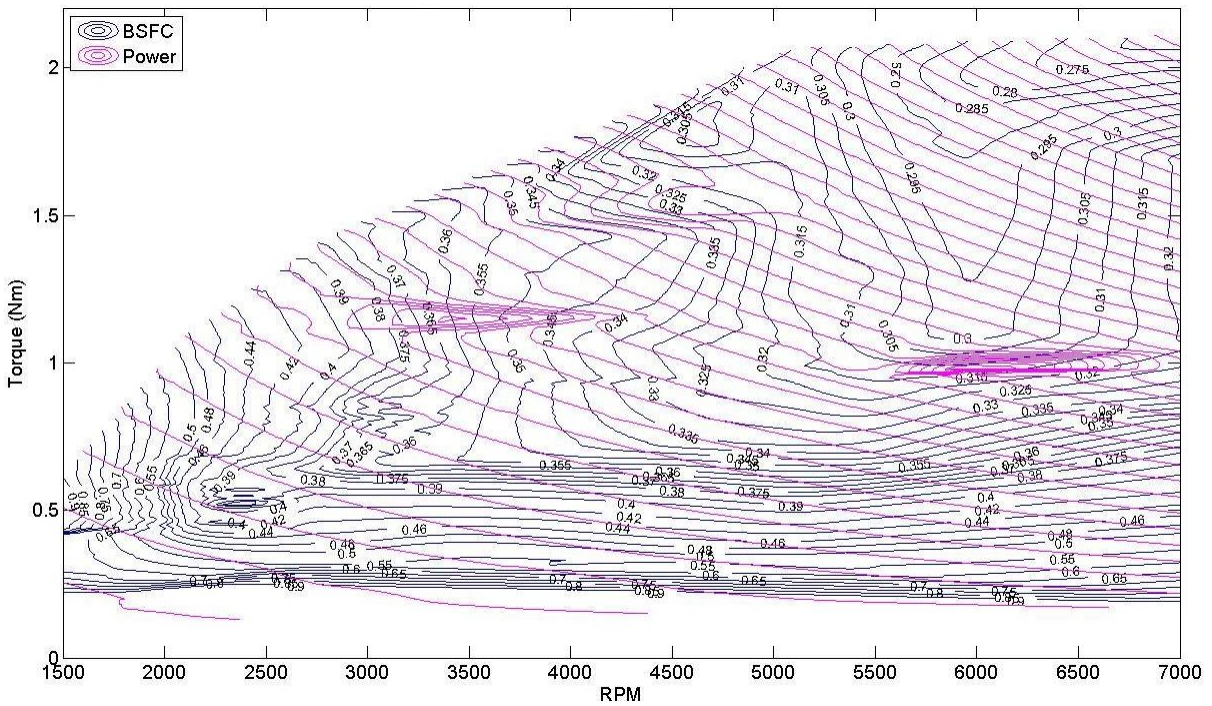


Figure 6.6: Engine map of the Aerosonde ICE, with power contours (magenta)

illustrated in Figure 6.7. For each of these points on the IOL, the corresponding BSFC values were obtained from this graph. These, along with the torque values, form the basis

of operating the ICE on the IOL.

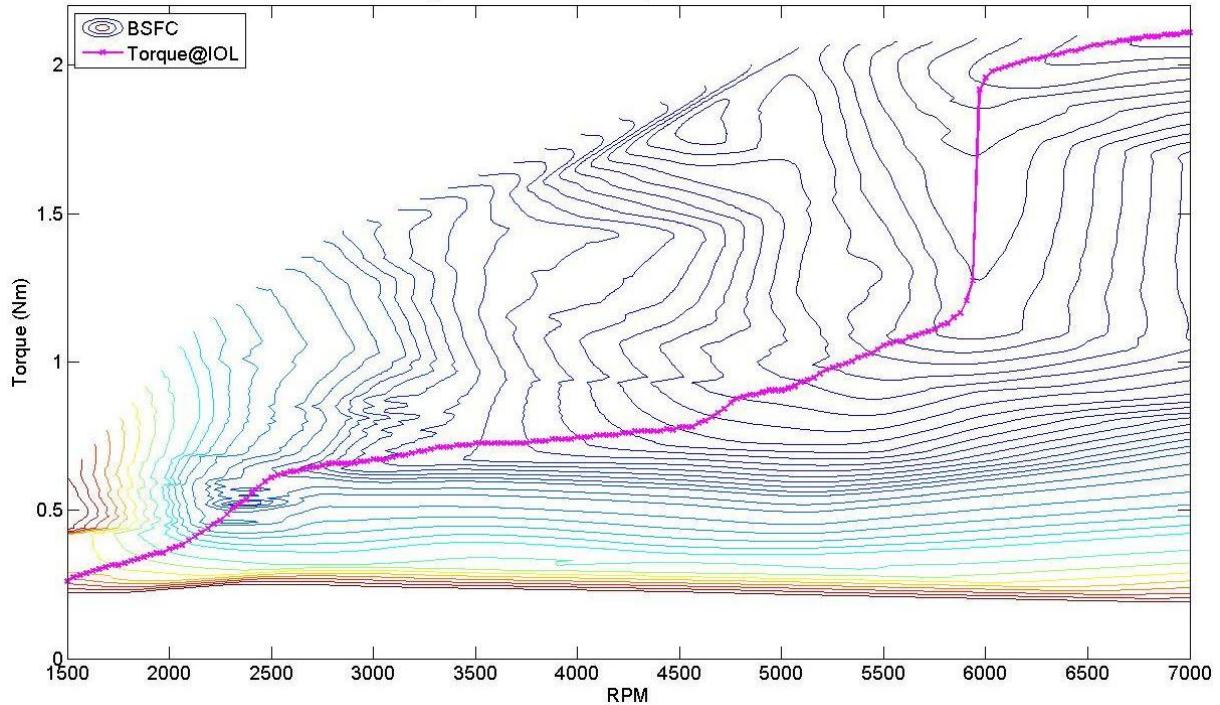


Figure 6.7: Engine map of the Aerosonde ICE, with the IOL (magenta)

Figure 6.8 shows the power and torque values for each point on the IOL, while Figure 6.9 illustrates the BSFC and fuel flow values. The power and fuel flow values were obtained from the torque and BSFC values using Equations 6.1 and 6.2.

The next and final step of the IOL analysis was to determine the corresponding MAP values to these torque and BSFC values. The power and fuel flow look-up tables (Figures 6.1 and 6.2) for the Aerosonde model were used to do this. The results are shown in Figure 6.10.

It can be seen that in Figure 6.10 that there are noticeable differences in the lower RPM region between the MAP values obtained from power and from fuel flow values, but the two curves approximate each other from an RPM value of about 2700 upwards. Due to the fact that minimising energy consumption is the main goal, where the MAP values did differ, the MAP values that give better (or smaller) fuel flow values were used, forming the plot as shown in Figure 6.11.

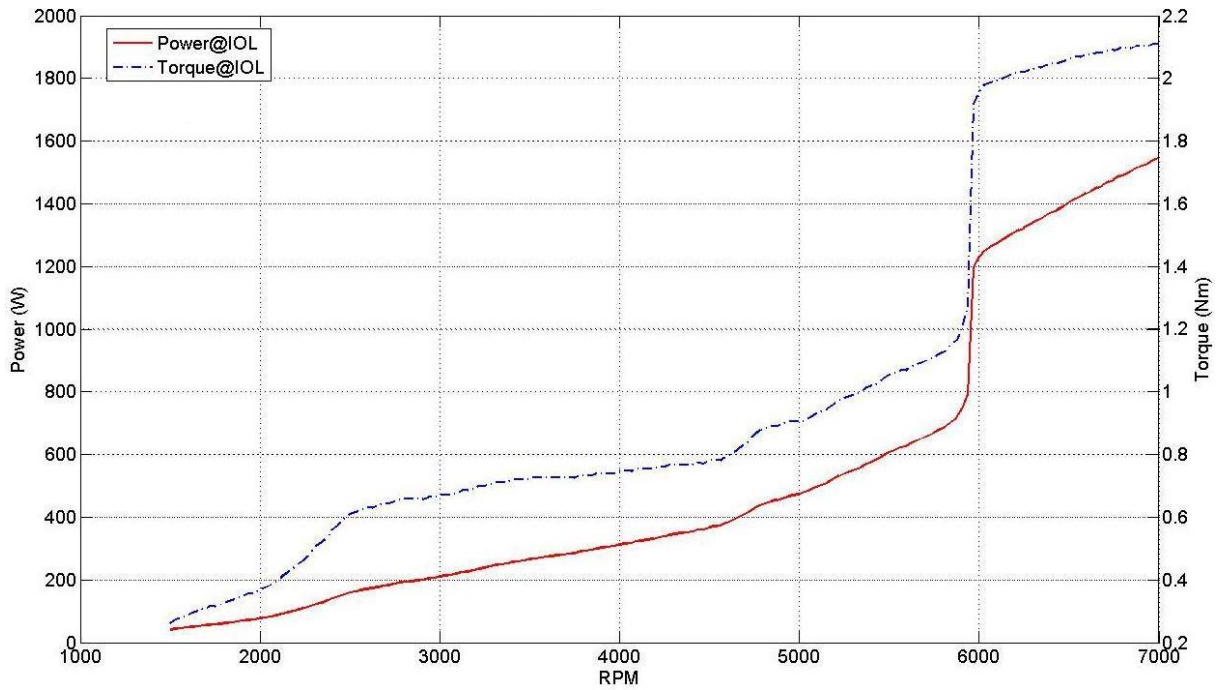


Figure 6.8: Power (red) and torque(blue) values for the IOL

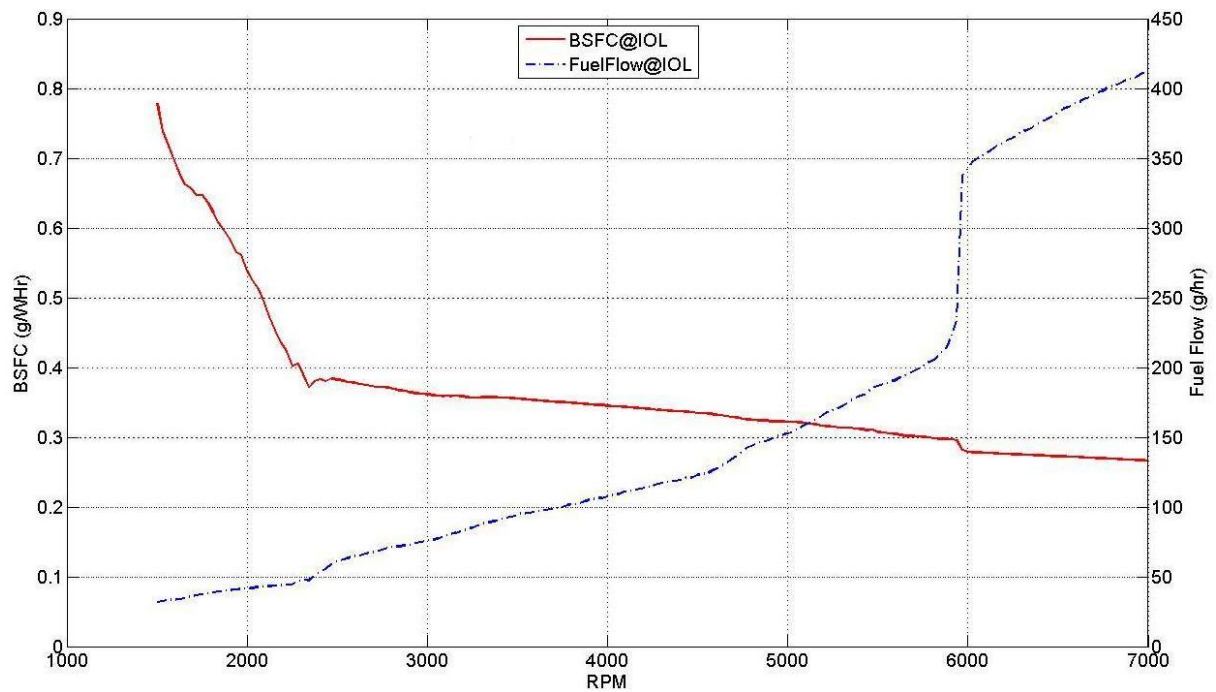


Figure 6.9: BSFC (red) and fuel flow (blue) values for the IOL

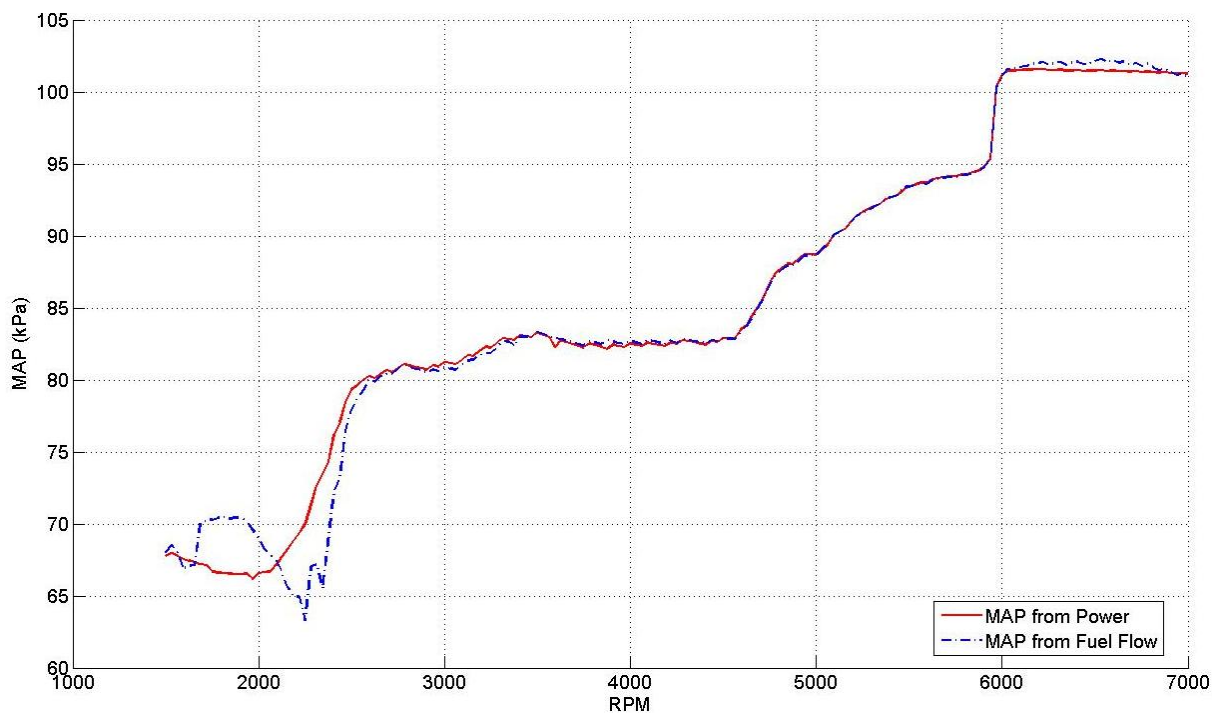


Figure 6.10: MAP values on the IOL (red - MAP from power, blue - MAP from fuel flow)

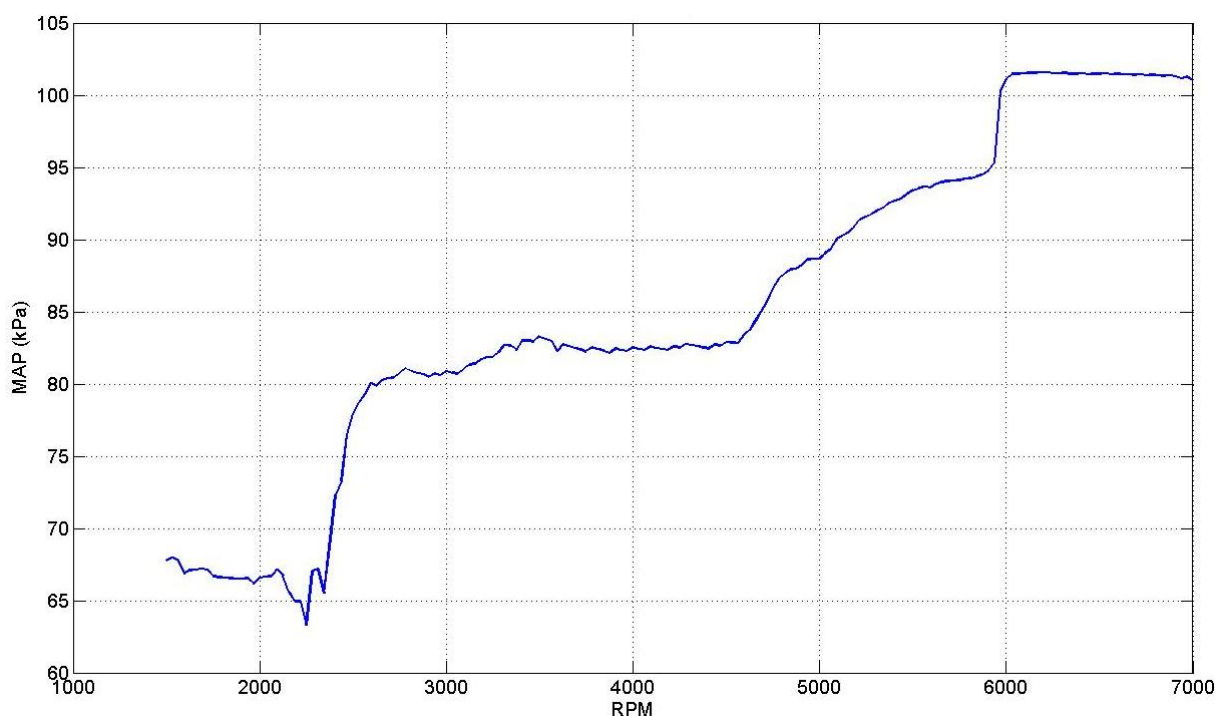


Figure 6.11: MAP values on the IOL

6.4 Summary

In this chapter, an IOL analysis was carried out on the existing Aerosonde ICE. This allows the determining of the most efficient (i.e. best performance at least amount of fuel

possible) points of operation for this ICE. The resulting IOL was incorporated into the IOL Controller in Chapter 7, which is an important component in the Hybrid-Electric Propulsion System (HEPS).

In completing the work presented in this chapter, Research Objective 5 as listed in §1.2 was fulfilled

Chapter 7

Modelling of Hybrid-Electric Propulsion Subsystem Components

7.1 Introduction

As mentioned in Chapter 5, the development of a Hybrid-Electric Propulsion System (HEPS) on an UAV was accomplished in two stages:

1. Performing of an Ideal Operating Line (IOL) analysis on an Aerosonde ICE; and
2. Development and implementation of a HEPS simulation model and its integration into the UAVSM.

As presented in §1.1.2, the literature review has shown that a Hybrid-Electric Propulsion System (HEPS) with a Continuously Variable Transmission (CVT) and utilising an Ideal Operating Line (IOL) control strategy has the potential to economise the energy efficiency on a small fixed-wing UAV. Figure 7.1 is the parallel Hybrid-Electric configuration and this shows that a possible arrangement for the components that constitute a HEPS are:

- Engine
- Fuel
- Electric Motor (EM)
- Generator
- Battery

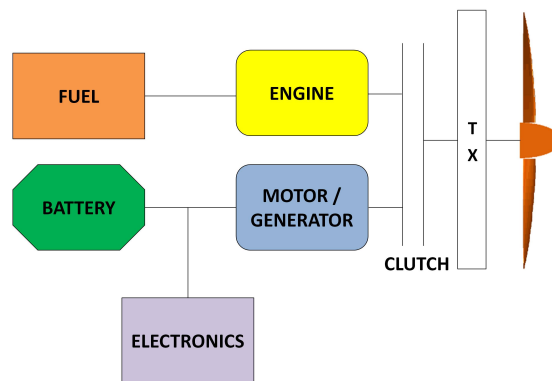


Figure 7.1: Parallel configuration of an HEPS.

- Transmission (CVT)

Of these components, the Engine and the Fuel models are in-built in the existing UAVSM. Even though the EM and the Generator are the same physical machine, they will require to be modelled separately because they have different functions. The dynamics of a Battery and the Transmission, which is a CVT, will be examined and a model for each created.

In order for these components to work in an organised manner, a controller is required. Its primary function will be to determine when and how each of the components of the HEPS will work in order to operate the ICE on the IOL, or a close approximation of this, while the UAV is in operation. For this reason, it will need to take into account the status and states of each of the HEPS components, in order to generate appropriate signals to operate the components accordingly. In this research, the IOL control strategy, based on the IOL determined in Chapter 6 is used in the controller.

With the inclusion of the IOL controller, the electro-mechanical components of the HEPS can be grouped as follows:

- Powertrain
 - Piston Engine (ICE)
 - Electric Motor (EM)
 - Generator
 - Battery

- Transmission (CVT)
- IOL Controller

The implementation of these HEPS components as MATLAB Simulink models, which complies with the Research Methodology as stated in §1.4. These models will form the basis for adapting the UAVSM as a full simulation model of a Hybrid-Electric UAV (HEUAV).

In this chapter, the modelling of HEPS and its components for the AeroSim-based Aerosonde simulation model in the MATLAB Simulink environment is presented. Firstly, the Powertrain components - namely the Electric Motor (EM), Generator, Battery, CVT dynamics and related control modules - are firstly presented in §7.2. §7.3 describes the implementation of the IOL Controller and its components. These two components are integrated in §7.4 and finally, §7.5 presents a brief conclusion to this chapter.

7.2 HEPS Powertrain

Of the HEPS Powertrain components, the Piston Engine (ICE) and Fuel components are already present in the UAVSM, and they remained as they were to maintain the integrity of already working components. The ICE model is presented in §7.2.1.

The components that required modelling are:

- Electric Motor (EM);
- Generator;
- Battery;
- Transmission (CVT);
- Powertrain Output Allocation (POA) module; and
- Charge Battery Now (CBN) module.

and their simulation models are presented in §7.2.2 to §7.2.7. Relevant MATLAB codes for these component functions can be found in Appendix I.

7.2.1 Piston Engine Model

The Piston Engine (ICE) is the primary source of propulsion in the HEPS. The goal of the HEPS is to operate the ICE on the Ideal Operating Line (IOL) as determined in §6.

The preset configuration of the Piston Engine block from the AeroSim Blockset was used in this research. The modelling of the simple ICE is achieved using two-dimensional Look-Up Tables (LUTs) of engine parameters. Given a set of RPM and manifold pressure (MAP) values, the corresponding values for fuel flow and engine power at sea level are determined and these are used in computations of other ICE outputs. In this research, the engine parameters of an Aerosonde engine were used. The input/output configuration of *Piston Engine* is listed in Table 7.1.

	Identifier	Unit	Description
Input	Thr	-	Engine throttle fraction (0.01 to 1)
	Mix	-	Air-to-fuel ratio, or mixture
	Eng Omega	rad/s	Engine speed
	Atm Pressure	Pa	Atmosphere pressure at current altitude
	Temp	K	Atmosphere temperature at current altitude
Output	MAP	kPa	Manifold air pressure for current throttle setting and altitude
	AirFlow	kg/s	Instantaneous mass air flow
	FuelFlow	kg/s	Instantaneous mass fuel flow
	BSFC	g/(W*hr)	Brake specific fuel consumption
	Power	W	Output power generated by the ICE
	Torque	Nm	Output torque generated at engine shaft

Table 7.1: I/O configuration of *Piston Engine*.

The MATLAB Simulink schematic for the ICE module is shown in Figure 7.2, with the inner schematics in Figure 7.3.

7.2.2 Electric Motor Model

The function of the Electric Motor (EM) within the HEPS is two-fold. It acts as a supplementary powerplant to the ICE when required, or as the sole powerplant when the UAV is operating in Motor-only mode. This particular mode is most likely for the performance of a particular task, i.e. data collection or aerial imaging.

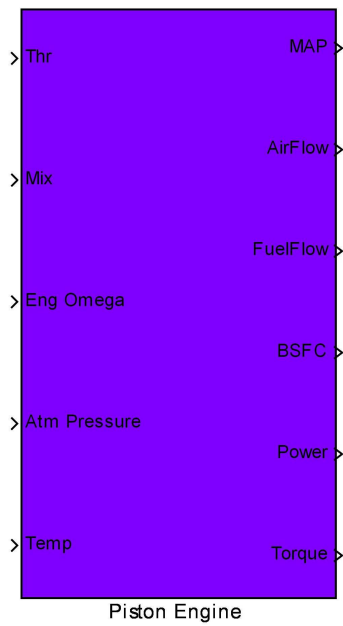


Figure 7.2: The *Piston Engine* block.

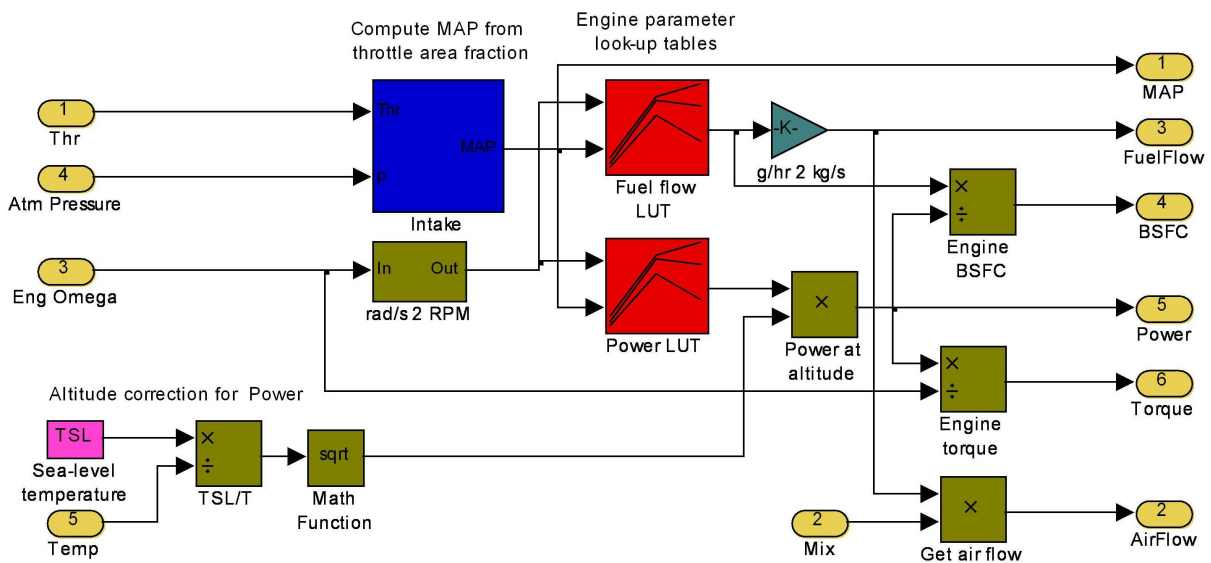


Figure 7.3: Inside the *Piston Engine* block.

The EM is simulated with the use of the MATLAB function *interp1*, which approximates a Look-Up Table (LUT) and uses a pre-existing set of input-output values and interpolation and/or extrapolation methods to associate a given input to the appropriate output. The *interp1* function is suitable in the modelling of EMs as the outputs (mainly torque and power) are usually controlled by and has a direct relationship with the input current.

To implement the *Motor Subsystem* module to be used in the UAVSM, a *MATLAB function* block was used in conjunction with a one-dimensional LUT. This allows a *MATLAB* function to be implemented in M-code, thus offering more flexibility programming-wise than a simulation block module. The input/output configuration of *Motor Subsystem* is listed in Table 7.2.

	Identifier	Unit	Description
Input	MotorTorqueReq	Nm	Torque required to be supplemented by the EM so the Engine can continue to operate on or near the IOL
	Engine Omega	rad/s	Engine speed
	MotorEnable	-	Signal to activate the EM - 0 = disabled - 1 = activated
	BattSOC	%	Current Battery state-of-charge (SOC)
	BattVoltage	V	Current Battery voltage
Output	M_Power	W	Output power produced by the EM
	M_Torque	Nm	Output torque produced by the EM
	M_Current	A	Current drawn by the EM from the Battery

Table 7.2: I/O configuration of *Motor Subsystem*.

The EM module will only output Motor Power and Torque values and draw current from the Battery if it is activated (i.e. when *MotorEnable* = 1), and when the Battery has adequate SOC and can provide enough voltage.

Here the EM is implemented using data from a Plettenberg HP220/25 EM with constant 18V input to construct the LUTs; see Appendix H for details. This particular model of EM was selected because it was readily available and it was capable of producing the power and torque required by the Aerosonde UAV. However, switching to another EM model will not be complicated, if the same data is available for this new EM.

The *MATLAB* Simulink schematic for the EM module is shown in Figure 7.4, with the inner schematics in Figure 7.5.

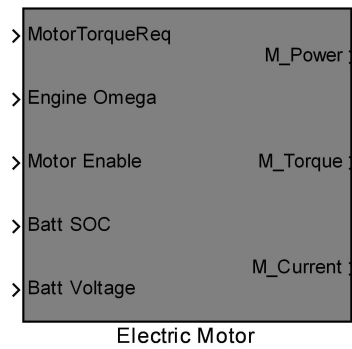


Figure 7.4: The *Motor Subsystem* block.

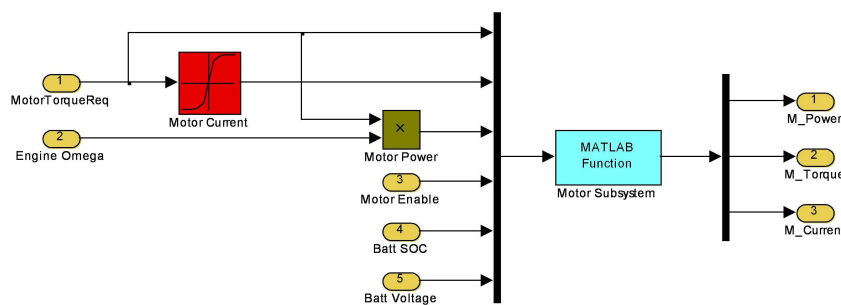


Figure 7.5: Inside the *Motor Subsystem* block.

7.2.3 Generator Model

Physically, the Generator and the EM are the same machine. But functionally, the Generator module is assumed to be the “reverse” of EM. Its only function is to provide the battery with charging current when there is extra torque available in the propulsion system, i.e. from the ICE.

As with the EM module, the Generator will not output any values unless it is activated, i.e. $GeneratorEnable = 1$. Note that it is physically impossible to activate both the EM and the Generator at the same time. If EM is activated, the Generator needs to be deactivated, and vice versa. This will need to be taken into account by the HEPS control strategy to ensure correct functioning of these two modules.

To implement the *Generator Subsystem* module to be used in the UAVSM, a *MATLAB function* block was used. The input/output configuration of *Generator Subsystem* is listed in Table 7.3.

The Simulink schematic of the *Generator Subsystem* module is shown in Figure 7.6 with the inner schematics shown in Figure 7.7.

	Identifier	Unit	Description
Input	GenEnable	-	Signal to activate the Generator - 0 = disabled - 1 = activated
	GenTorqueAvail	Nm	Torque available to power the Generator, so that current can be generated to charge the Battery
Output	G_Current	A	Current output to charge the Battery

Table 7.3: I/O configuration of *Generator Subsystem*.

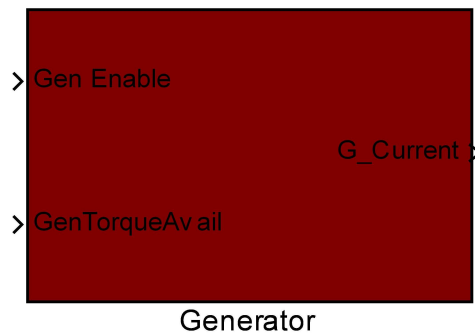


Figure 7.6: The *Generator Subsystem* block.

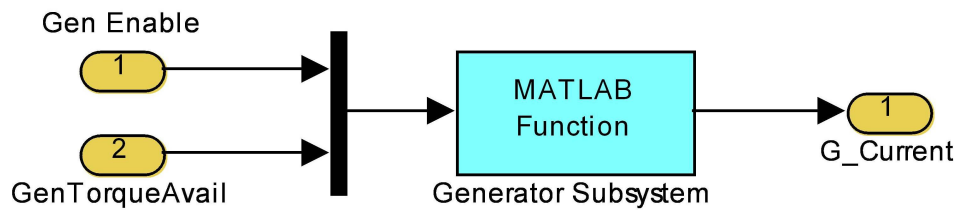


Figure 7.7: Inside the *Generator Subsystem* block.

7.2.4 Battery Model

The main function of the Battery is to provide the required current and voltage to the onboard avionics and also to EM when its activation is required. Additionally, the Battery needs to switch to charging when excess torque is available in the system, i.e. from the ICE. The discharging characteristics for a lithium-polymer (Li-Po) battery is modelled by the following equation [164]:

$$E = E_0 - K \cdot \frac{Q}{Q - i \cdot t} + Ae^{-B \cdot i \cdot t} \tag{7.1}$$

where

- E = battery output voltage (V)
 E_0 = battery constant voltage (V)
 K = polarity voltage (V)
 Q = maximum battery capacity
 A = exponential voltage coefficient
 B = exponential capacity coefficient (Ah^{-1})
 i = battery current (A)
 t = time (s)

The charging characteristics of the Battery modes are considered to follow Equation 7.1 as well, although with a negative current to indicate charging instead of discharging.

Another important characteristic of a Battery is its State-of-Charge (SOC), which is approximated by [165]:

$$SOC = 100 \left(1 - \frac{\int_0^t i \cdot t}{Q} \right) \quad (7.2)$$

Based on Equations 7.1 and 7.2, the Simulink schematic for the *Battery Subsystem* module is shown in Figure 7.8 with the inner schematics displayed in Figure 7.9.

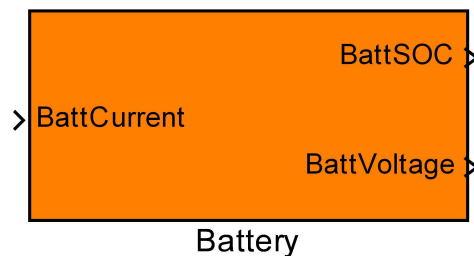


Figure 7.8: The *Battery Subsystem* block.

To implement the *Battery Subsystem* module to be used in the UAVSM, a *MATLAB function* block was also used. The input/output configuration of *Battery Subsystem* is listed in Table 7.4.

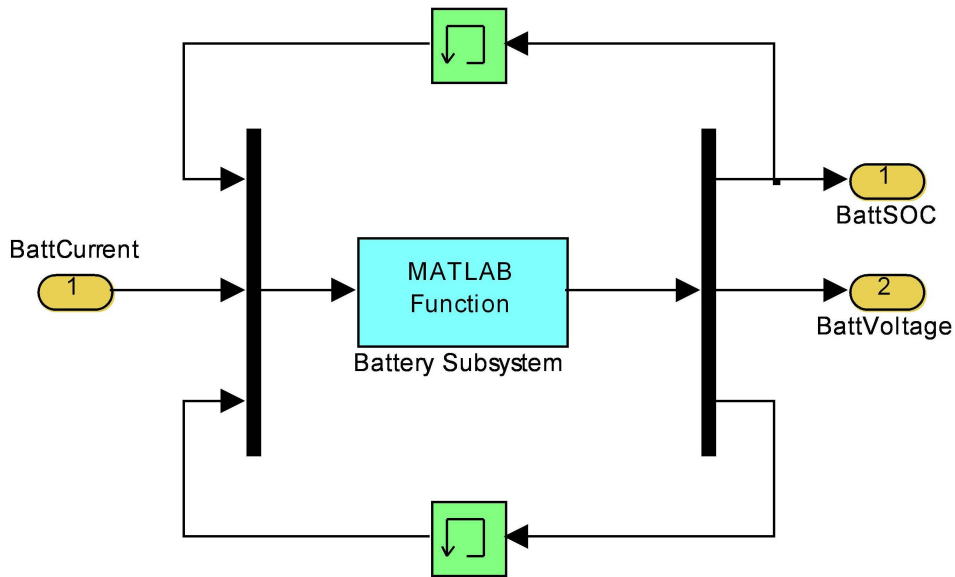


Figure 7.9: Inside the *Battery Subsystem* block.

	Identifier	Unit	Description
Input	BattCurrent	A	Input current into the Battery
Output	BattSOC	%	Battery SOC
	BattVoltage	V	Battery voltage

Table 7.4: I/O configuration of *Battery Subsystem*.

Here, the *Battery Subsystem* module is implemented using the data of two Air Thunder 5000mAh 6-cell Lithium-polymer (Li-Po) Battery Packs [166] in series to obtain the required output voltage for the EM. These particular batteries were selected because of their ability to meet the electrical requirements (voltage and current) to power the Plettenberg EM, as well as their availability as a COTS commodity. The discharge curves for different amount of current drawn from the Battery are shown in Figure 7.10. These discharge curves are fitted to (7.1) in order to establish the coefficients K , Q , A and B . These coefficients are determined using *interp1* functions in the Simulink schematic.

7.2.5 CVT Dynamics Model

The type of Transmission implemented in the HEPS is a Continuously Variable Transmission (CVT). In a conventional gearbox, output torque and speed of the engine are

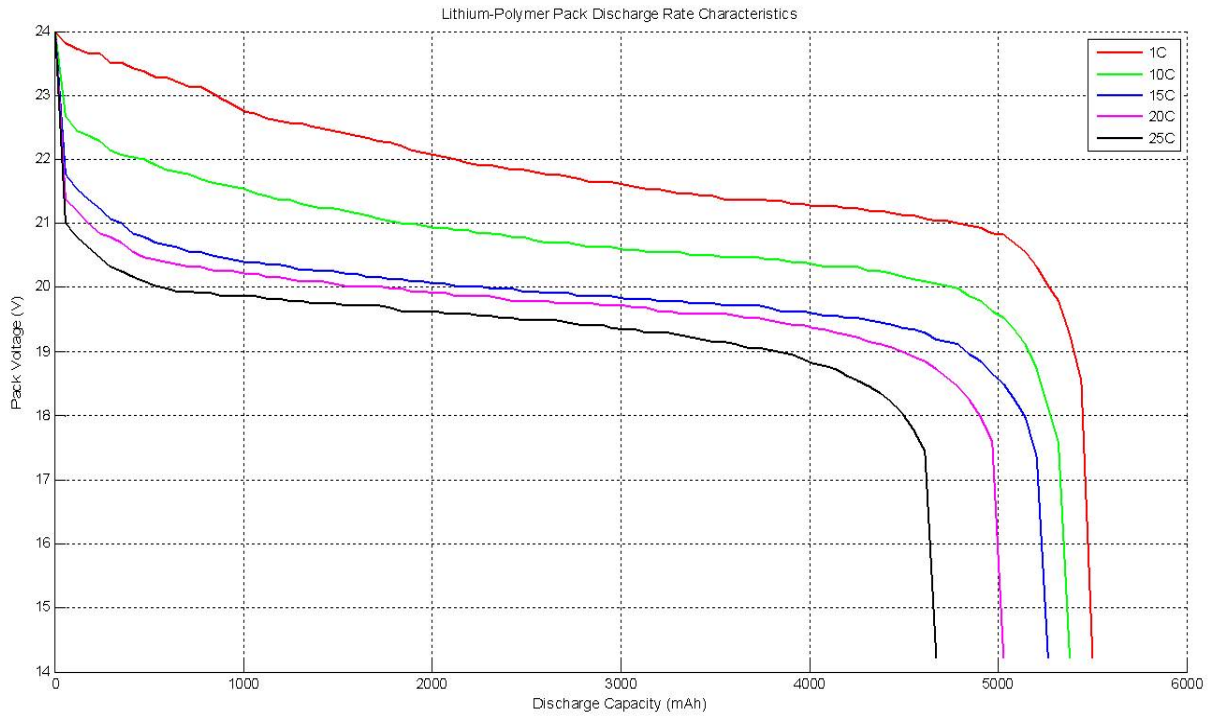


Figure 7.10: Battery discharge curves.

transmitted in discrete ratios, and the action of changing from one gear to another interrupts the power flow through the powertrain during the acceleration. One way of overcoming this intermittent behaviour is with the use of a CVT, which has the capability to transmit engine torque and speed in an undefined number of ratios [167]. This enables the output of a smooth, rapid and stepless response to the demand of the controller. Additionally, a CVT allows the engine speed to be operated independently of the vehicle speed (wheel speed for a ground vehicle or propeller speed for an aircraft, for example), therefore the engine can, in theory, be operated in its most fuel efficient operating point, with the help of a suitable controller strategy to control the CVT operations. The CVT's potential of reducing fuel consumption and lower the output of exhaust emissions has been confirmed various research projects [168, 169].

The dynamics of the CVT can be modelled by the following expression derived from a simplified powertrain with a CVT [16, 22]:

$$\dot{\omega}_{prop} = \frac{T_{prop} + rT_p - \dot{r}\omega_{eng}I_p}{I_{prop} + r^2I_p} \quad (7.3)$$

where

- ω_{eng} = rotational velocity of the engine (rad/s)
- ω_{prop} = rotational velocity of the propeller (rad/s)
- $\dot{\omega}_{prop}$ = rate of change of the propeller rotational velocity (rad/s²)
- T_{prop} = propeller torque (Nm)
- T_p = powertrain torque (Nm)
- I_p = total powertrain (ICE + EM) torque
- I_{prop} = total propeller inertia
- $r = \frac{\omega_{eng}}{\omega_{prop}}$; transmission ratio
- \dot{r} = rate of change of ratio (RCR)

The output of this equation is the rate of change of the propeller speed, $\dot{\omega}_{prop}$ using the other parameters, from which the propeller speed ω_{prop} can be determined with a simple integration process. To prevent overlarge values of $\dot{\omega}_{prop}$ from appearing, a saturation function was used to limit the $\dot{\omega}_{prop}$ output.

The CVT ratio range was taken as 0.47 to 2.455, which has been taken into account in the HEPS model in the form of saturation limits.

The input/output configuration of the *CVT Dynamics Subsystem* is listed in Table 7.5.

	Identifier	Unit	Description
Input	Eng Omega	rad/s	Engine speed
	Powertrain Torque	Nm	Powertrain torque output
	rdot	-	Rate of change of ratio (RCR)
	Prop Torque	Nm	Propeller torque (usually negative)
	Powertrain J	kg·m ²	Total powertrain inertia (ICE & EM)
	Prop J	kg·m ²	Propeller inertia
Output	OmegaPropDot	rad/s ²	Rate of change of propeller speed
	CVT ratio	-	CVT ratio

Table 7.5: I/O configuration of *CVT Dynamics Subsystem*.

Translating Equation 7.3 to a Simulink model gives the schematics as in Figure 7.11,

with the inner schematics in Figure 7.12.

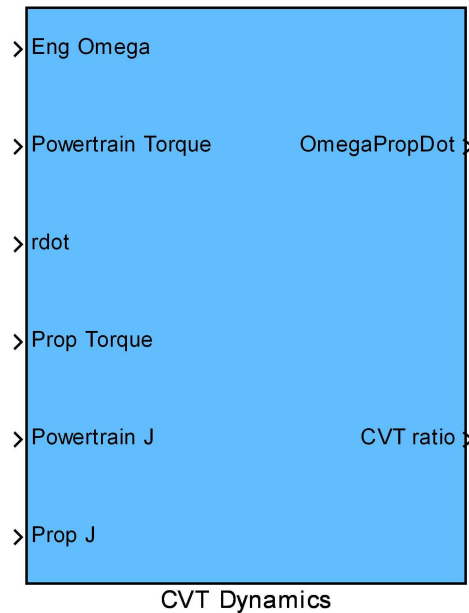


Figure 7.11: The *CVT Dynamics Subsystem* block.

7.2.6 Charge Battery Now (CBN) Module

A module that is included inside the *Powertrain* block is the Charge Battery Now (CBN) module. This module monitors the charging and discharging of the Battery as well as estimates the time remaining before the UAV is required to go into “Motor Only” mode. Because this mode is an important part of the mission, and since the EM requires adequate Battery SOC and voltage in order to operate, therefore it is important that the Battery is maintained at a very high level of SOC as the UAV approaches the “Motor Only” mode. This is achieved by estimating, at any one time, the amount of time it requires to charge the Battery back to 100%.

The input/output configuration of CBN is listed in Table 7.6.

The Simulink schematics of CBN is shown in Figure 7.13, with the inner schematics in Figure 7.14.

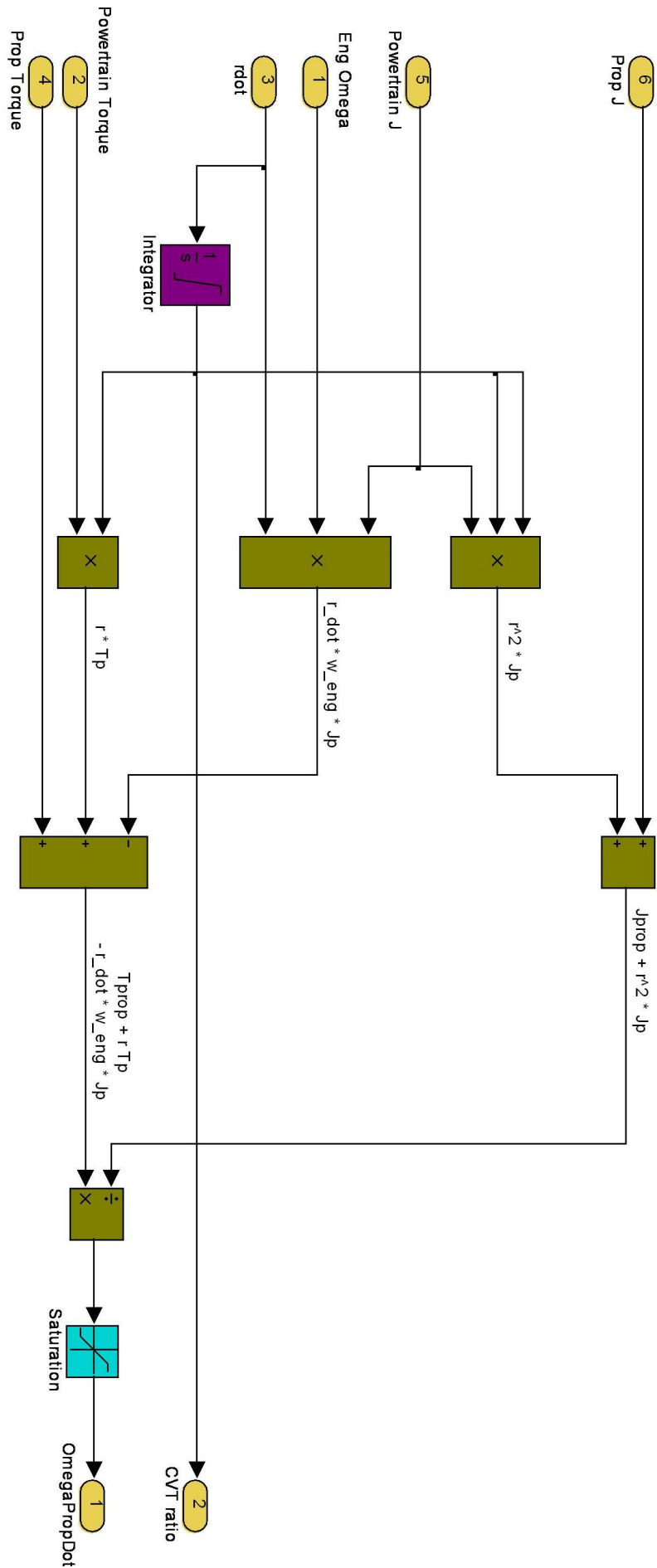


Figure 7.12: Inside the CVT Dynamics Subsystem block.

	Identifier	Unit	Description
Input	Batt Current	A	Current drawn from the Battery
	Current BattSOC	%	Current Battery SOC
	Current Airspeed	m/s	Current airspeed of UAV
	SimClock	s	Simulation time
Output	MustChargeBatt	-	Signal to start charging the Battery - 1 = charge now

Table 7.6: I/O configuration of *Charge Battery Now* module.

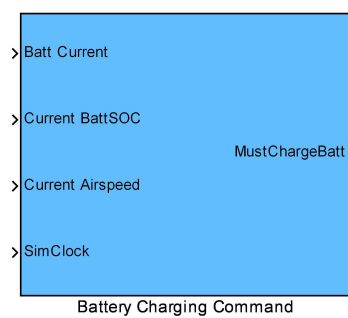


Figure 7.13: The *Charge Battery Now* block.

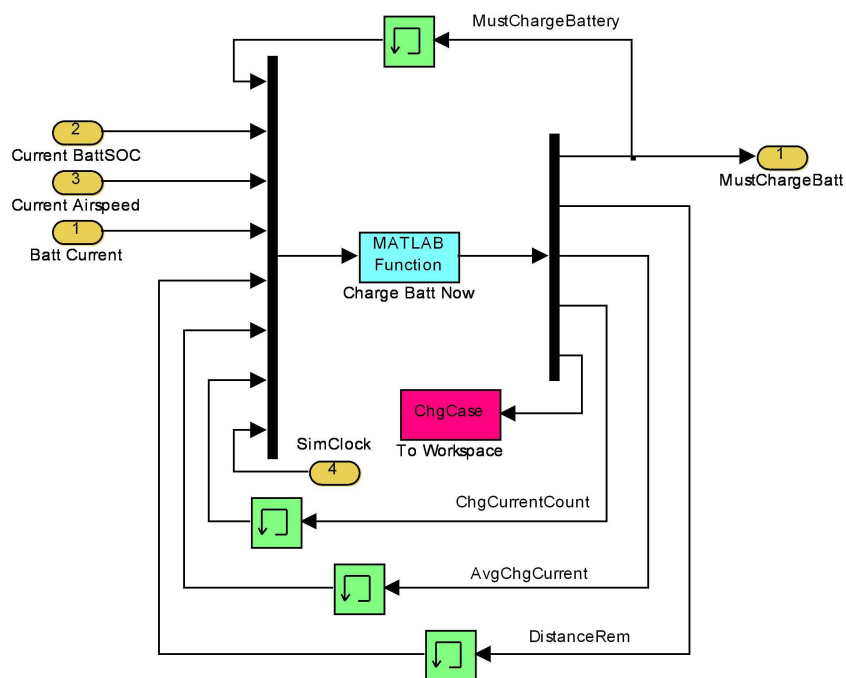


Figure 7.14: Inside the *Charge Battery Now* block.

7.2.7 Powertrain Output Allocation (POA) Module

The combined power and torque outputs of the Powertrain components - namely the ICE, EM and Generator - are passed to and used in the CVT Dynamics component as well as the subsequent components of the AeroSim aircraft block for further computations in the simulation of an HEUAV in flight. However, a slight difference between the values to be passed to each of these destination in the form of an RCR Torque (see §7.3.6 for further description) is required to be taken into consideration. This term is the result of the shifting of the CVT whilst in operation and needs to be compensated by the torque output of either EM or Generator. Therefore the combined Powertrain torque output into the CVT Dynamics component requires the corresponding compensation term to be included. But on the other hand, the compensation torque is not required in the output to the subsequent AeroSim components. The Powertrain Output Allocation (POA) module was implemented to allow for this distinction.

The input/output configuration of POA is listed in Table 7.6.

The Simulink schematics of POA is shown in Figure 7.15, with the inner schematics in Figure 7.16.



Figure 7.15: The *Powertrain Output Allocation* block.

	Identifier	Unit	Description
Input	Eng Power	W	Output power from the ICE
	Eng Torque	Nm	Output torque from the ICE
	Eng Omega	rad/s	Engine speed
	Engine Enable	-	Signal to activate the ICE - 0 = disable - 1 = enable
	Motor Enable	-	Signal to activate the EM - 0 = disable - 1 = enable
	Motor Power	W	Output power from the EM
	Motor Torque	Nm	Output torque from the EM
	Gen Torque	Nm	Torque available for the Generator
	Gen Enable	-	Signal to activate the Generator - 0 = disable - 1 = enable
	rdot	-	Rate of change of ratio (RCR)
	CVT Ratio	-	CVT ratio
Output	Powertrain Torque	Nm	Combined Powertrain torque output (passed to CVT Dynamics; includes compensation for RCR Torque)
	TorqueToPropShaft	Nm	Combined Powertrain torque output to subsequent AeroSim aircraft components (does not include compensation for RCR Torque)
	Powertrain Power	W	Combined Powertrain power output

Table 7.7: I/O configuration of *Powertrain Output Allocation* module.

7.3 HEPS IOL Controller

As stated in §6.2, the IOL is a smooth line comprising of all the points which represent the torque and speed combinations at which the fuel consumption is minimal on different power lines for steady state conditions [163]. A powerplant operated on the IOL will, theoretically, enable the best performance while consuming the least amount of fuel possible.

The implementation of a HEPS on an UAV requires a controller which manages the operation of each of the system components – ICE, EM, Generator and Battery – so that operation on the IOL, or a close approximation of this, can be achieved while the UAV is

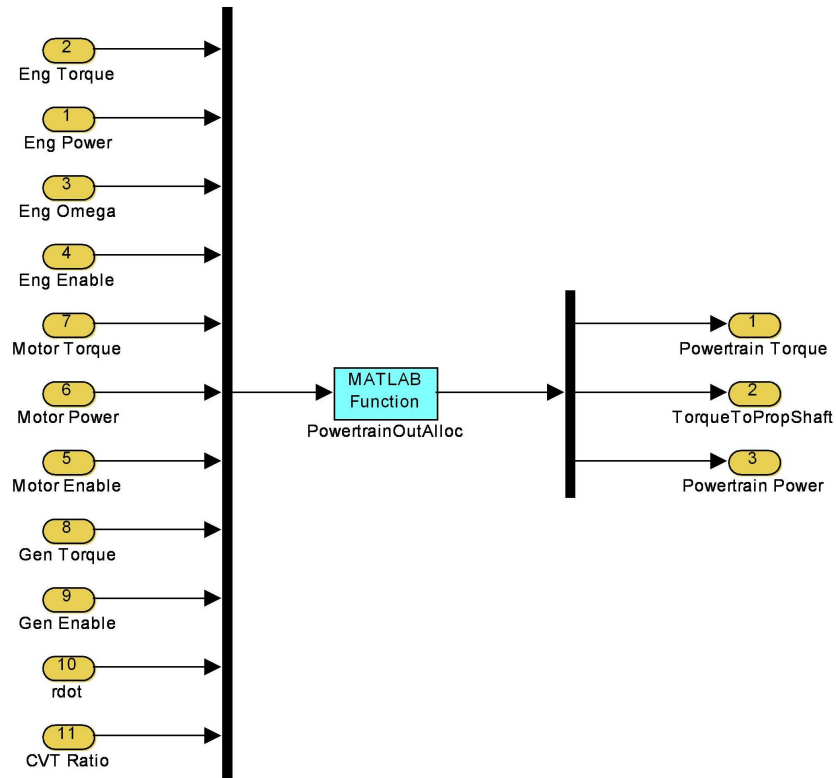


Figure 7.16: Inside the *Powertrain Output Allocation* block.

in operation.

The basic control loop of the IOL Controller when in Hybrid mode is shown in Figure 7.17 [16].

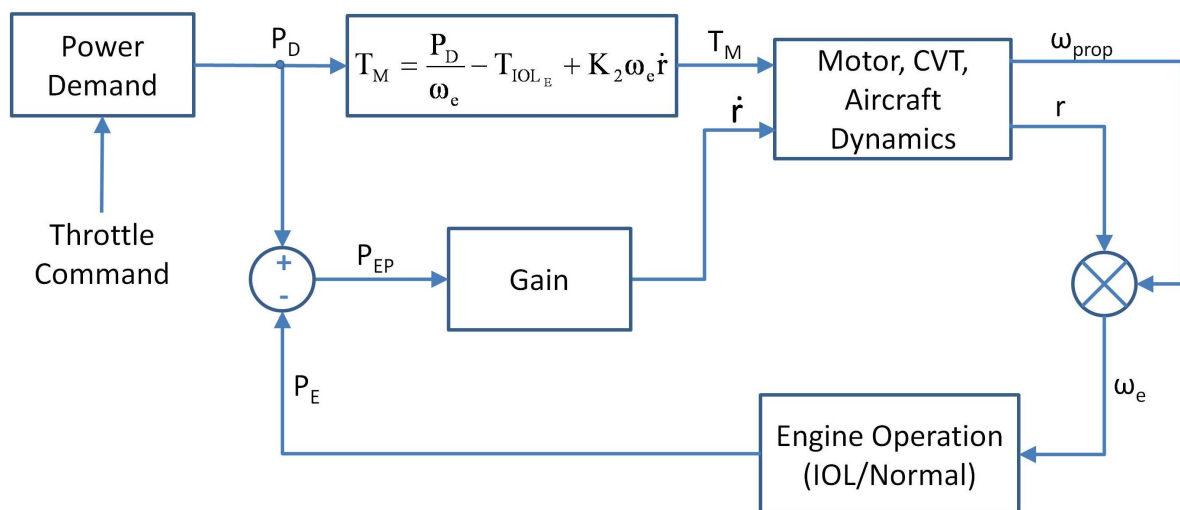


Figure 7.17: The basic control loop of the IOL Controller.

The concept behind this control loop is that the ACM demands a specific throttle

opening, which is converted to a power demand measurement and from this, the desired torque that needs to be matched or closely approximated by the HEPS is computed. However, the Engine should be operating on the IOL, thus producing a torque value which is often different to the desired torque. The difference between the torque values is provided by either the EM or the Generator - the EM provides extra torque required if IOL torque is less than desired torque, otherwise the Generator uses the excess torque to charge the Battery.

On the other hand, this torque difference is multiplied by the engine speed to determine the power error. This power error is used to determine the Rate of Change of Ratio (RCR) value within a preset limit, which determines the amount of shifting that is required of the CVT.

The components of the IOL controller are as follows:

- Operating Mode;
- Power Demand;
- Engine Operation;
- Engine Throttle Command;
- Rate of Change of Ratio Command; and
- Torque Difference Calculation.

The details of each of these components and its functions are described in the following sections. The MATLAB codes of each of these component functions can be found in Appendix J.

7.3.1 Operating Mode (OM) Module

The Operating Mode (OM) module determines the mode in which the HEPS is currently operating in. These modes can be one of the following:

- Hybrid Normal mode;
- Motor Only mode;
- Hybrid Charging mode;

- Engine Only mode; and
- Hybrid Climbing mode.

The conditions for the switching of the operating modes are summarised in Table 7.8. The default operating mode is the *Hybrid Normal* mode, during which the ICE is operating on the IOL, with the EM or Generator supplementing when a sudden increase or decrease in the power demand occurs. If the “Motor Only” signal has been detected, which can only occur at pre-planned, specified leg(s) of the mission, the operating mode switches over to *Motor Only* mode, and only the EM is powering the UAV to maintain its operation. If the “Must Charge Battery” signal, as generated by the CBN module in the Powertrain (see §7.2.6), is detected, the *Hybrid Charging* mode is triggered if the aircraft is not required to climb to a higher altitude (i.e. $InClimb = 0$), during which the ICE is operated at an Engine speed to provide excess torque to power the Generator and thus charge the Battery. Otherwise, the *Engine Only* mode is engaged in order to preserve fuel consumption since charging the Battery requires the ICE to run at regions with higher fuel consumptions. If charging of the Battery is not required and a higher altitude is desired (i.e. $InClimb = 1$), the *Hybrid Climbing* mode is engaged and the EM is activated to provide extra torque to assist the ICE in the climb. Lastly, if the “Engine Only” signal has been detected, which only occurs at pre-planned, specific leg(s) of the mission, the HEPS would operate in the *Engine Only* mode.

Operating Mode	Condition
Hybrid Normal	- Default operating mode - When Battery SOC is at least 15%
Motor Only	Only when the <i>Motor Only</i> signal is detected
Hybrid Charging	When <i>MustChgBatt</i> signal is detected <u>AND</u> no climbing is required
Engine Only	When the <i>Engine Only</i> signal is detected <u>OR</u> When <i>MustChgBatt</i> <u>AND</u> <i>InClimb</i> signals are detected simultaneously
Hybrid Climbing	When <i>InClimb</i> signal is detected <u>AND</u> no charging of Battery is required

Table 7.8: Switching conditions of operating modes.

The input/output configuration of the OM is presented in Table 7.9 and the Simulink and inner schematics are shown in Figures 7.18 and 7.19.

	Identifier	Unit	Description
Input	Motor/Engine Only	-	Signal for the <i>Motor Only</i> or <i>Engine Only</i> mode to occur - 0 = neither ' <i>Motor Only</i> nor <i>Engine Only</i> - 1 = <i>Motor Only</i> - 2 = <i>Engine Only</i>
	MustChgBatt	-	Signal to start charging the Battery; 1 = charge now
	InClimb	-	Signal to indicate climbing to a higher altitude is required - 0 = Climbing not required - 1 = Climbing is required
Output	Operating Mode	-	The current operating mode

Table 7.9: I/O configuration of *Operating Mode* module.

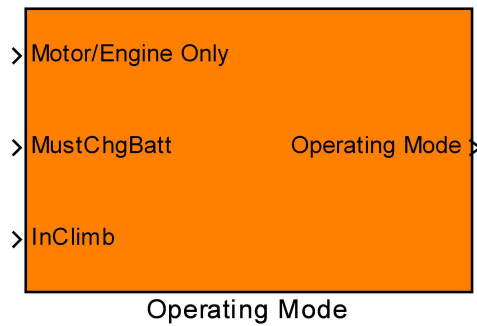


Figure 7.18: The *Operating Mode* block.

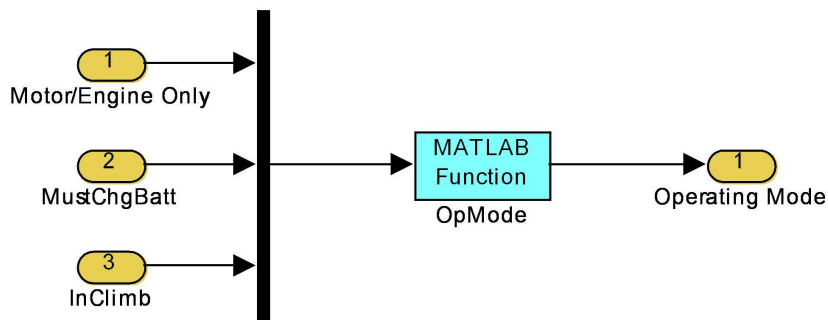


Figure 7.19: Inside the *Operating Mode* block.

7.3.2 Power Demand (PD) Module

The Power Demand (PD) module can be called the *beginning* of the IOL Controller control loop. PD takes the throttle input from the ACM block outside the Aerosonde UAV block, which was determined using a PID controller on the aircraft altitude error, and uses it to determine the amount of power required to be generated by the Powertrain if just the ICE was powering the UAV.

In the original configuration of the Aerosonde UAV, this throttle input is fed directly to the *Piston Engine* block, which uses this, along with other parameters, to determine the power, torque and fuel consumption and their related outputs from the ICE. These outputs are the amounts required for the UAV to operate as desired.

Based on this concept, PD uses parts of this *Piston Engine* block, namely the MAP and Power LUT blocks, to determine the power required to keep the UAV operating as desired.

The functionality of PD in the various operating modes is summarised in Table 7.10.

Operating Mode	Process
Hybrid Normal Motor Only Engine Only	Converts Throttle command from ACM into the Power Demand, using current Propeller speed at the current altitude
Hybrid Charging	Converts Throttle command from ACM into the Power Demand, using current Propeller speed at the current altitude and <u>add</u> the <i>Generator Charging Torque</i>
Hybrid Climbig	Converts Throttle command from ACM into the Power Demand, using current Propeller speed at the current altitude and <u>subtract</u> the <i>Motor Climbing Torque</i>

Table 7.10: Functionality of the *Power Demand* module.

The input/output configuration of PD is listed in Table 7.11.

The Simulink schematics of the PD is shown in Figure 7.20, with its inner schematics in Figure 7.21.

Note that in the *Piston Engine* block, the effect of altitude on the Engine outputs has been taken into account. This is required in calculations for all outputs related to Engine power. Therefore this altitude correction factor is necessary in later power- and

	Identifier	Unit	Description
Input	OpMode	-	Current operating mode - 1 = Hybrid Normal - 2 = Motor Only - 3 = Hybrid Charging - 4 = Engine Only - 5 = Hybrid Climbing
	Thr	-	Desired throttle from ACM (0.01 to 1)
	Atm Pressure	Pa	Static atmospheric pressure at current altitude
	Temp	K	Temperature at current altitude
	Prop Omega	rad/s	Propeller speed
	Eng Omega	rad/s	Engine speed
Output	Power Demand	W	Power demand
	AltCorrection	-	Altitude correction factor

Table 7.11: I/O configuration of *Power Demand* module.

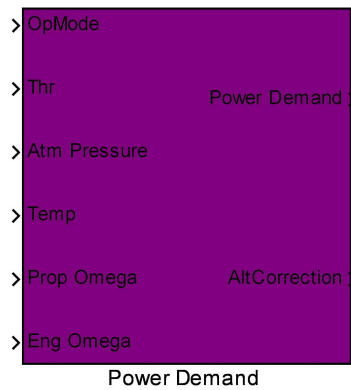


Figure 7.20: The *Power Demand* block.

torque-related calculations.

In the *Piston Engine* block, the torque is derived from the Engine power output following the relationship described in Equation 7.4.

$$Power_{@alt} = Power \cdot altCorrection \quad (7.4)$$

Also:

$$Torque_{@alt} = \frac{Power_{@alt}}{\omega} \quad (7.5)$$

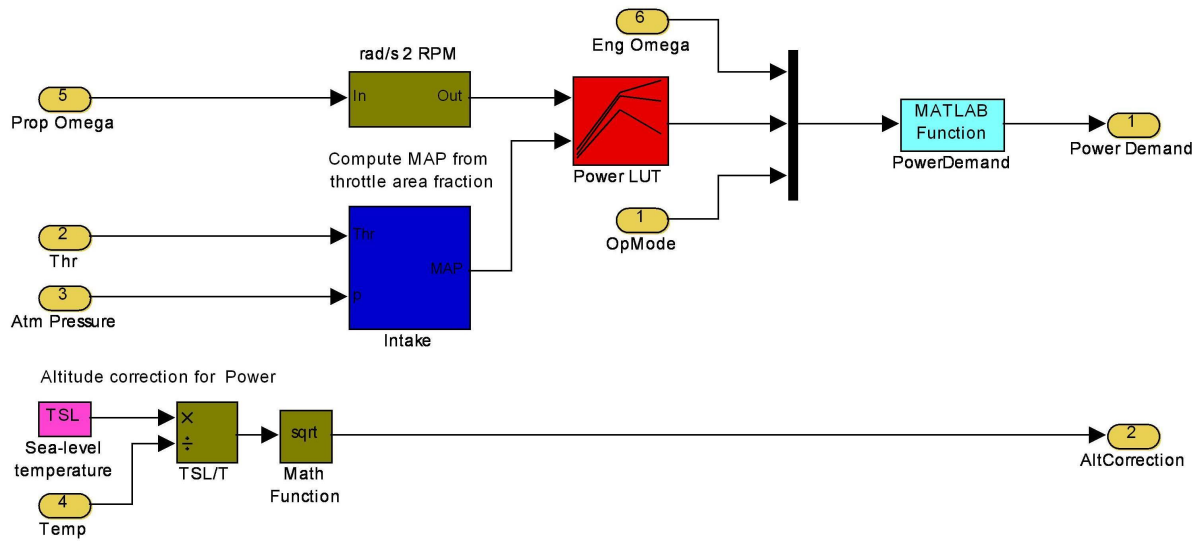


Figure 7.21: Inside the *Power Demand* block.

Substituting Equation 7.4 into Equation 7.5 gives:

$$Torque_{@alt} = \frac{Power}{\omega} \cdot altCorrection \quad (7.6)$$

And lastly, substituting in the definition of Torque into Equation 7.7:

$$Torque_{@alt} = Torque \cdot altCorrection \quad (7.7)$$

In the PM, this altitude correction factor, *altCorrection*, is calculated, but not yet applied to the power value required. This factor is passed on to the Torque Difference Calculation module for further processing.

7.3.3 Engine Operation (EO) Module

A critical step in the IOL control loop is to determine the ICE outputs, in particularly the torque output. During a mission, an ICE can operate either by itself, in which the ICE operates in the normal manner (i.e. not constrained to the IOL), or in conjunction with the EM or Generator, in which the ICE would be operating on the IOL. For this reason, a good approximation of the ICE outputs in each of these operating modes is required in order to proceed to the remainder of the processes in the IOL control loop. The Engine

Operation (EO) module was designed and implemented to perform this approximation of ICE outputs.

The main components in the EO module are the LUTs to determine the torque and manifold pressure (MAP) values for normal ICE and IOL modes of operation. In each time step in the simulation, these outputs are obtained from the current engine speed and, in the case of normal ICE operation, the MAP using atmospheric conditions. These outputs are then passed through a MATLAB function, which takes into consideration the current operating mode (see §7.3.1) and determines the appropriate torque and MAP values to be used in subsequent calculations.

The input/output configuration of the EO module is listed in Table 7.12.

	Identifier	Unit	Description
Input	OpMode	-	Current operating mode - 1 = Hybrid Normal - 2 = Motor Only - 3 = Hybrid Charging - 4 = Engine Only - 5 = Hybrid Climbing
	Thr	-	Desired throttle from ACM (0.01 to 1)
	Atm Pressure	Pa	Static pressure at current altitude
	Eng Omega	rad/s	Engine speed
Output	Eng Torque	Nm	Approximation of ICE torque output
	Eng MAP	kPa	Approximation of ICE MAP

Table 7.12: I/O configuration of *Engine Operation* module.

The Simulink schematics of EO is shown in Figure 7.22, with its inner schematics in Figure 7.23.

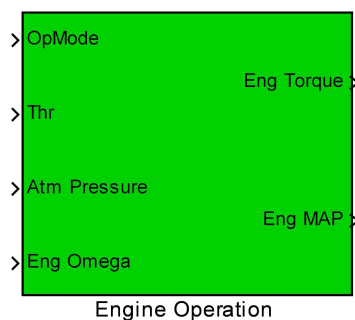


Figure 7.22: The *Engine Operation* block.

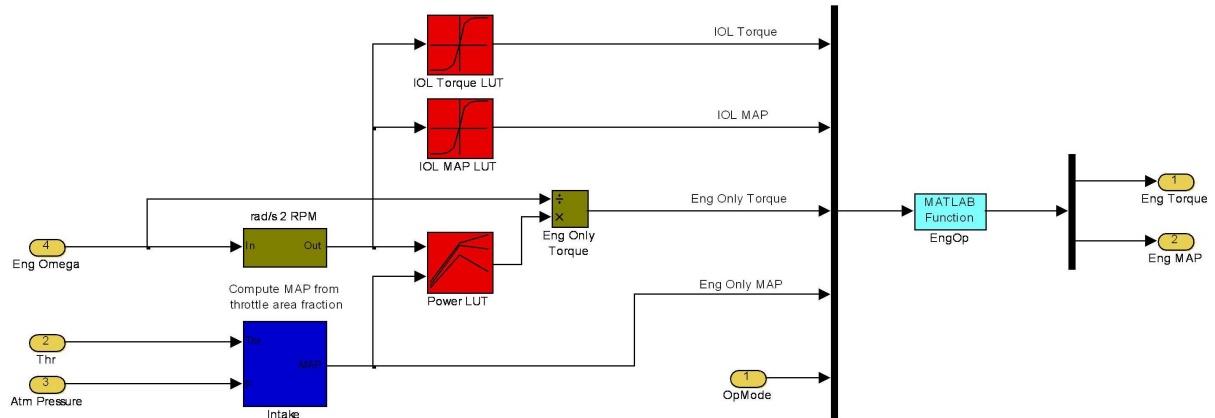


Figure 7.23: Inside the *Engine Operation* block.

7.3.4 Engine Throttle Command (ETC) Module

The Engine Throttle Command (ETC) module computes the Engine throttle required to operate the Engine on or close to the IOL. This module uses Equation 7.8, with the MAP calculated by EO for IOL operation and the static pressure at the current altitude, to calculate the throttle command.

$$thr = \frac{MAP - MAP_{min}}{\frac{p}{1000} - MAP_{min}} \quad (7.8)$$

The ETC module has the following input/output configuration as listed in Table 7.13.

	Identifier	Unit	Description
Input	Engine MAP	kPa	MAP for ICE operation
	Atm Pressure	Pa	Atmospheric pressure at current altitude
Output	Engine Throttle	-	Throttle command for the Engine to operate on the IOL (0.01 to 1)

Table 7.13: I/O configuration of *Engine Throttle Command* module.

The Simulink schematics of the ETC is shown in Figure 7.24, with its inner schematics in Figure 7.25.

The output of ETC, the Throttle Command, is passed to the *Powertrain* subsystem, to be used as input into the *Piston Engine* block.

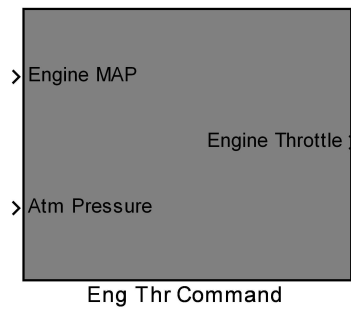


Figure 7.24: The *Engine Throttle Command* block.

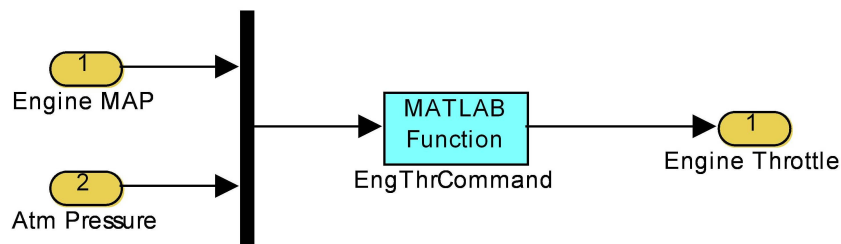


Figure 7.25: Inside the *Engine Throttle Command* block.

7.3.5 Rate of Change of Ratio Command (RCRC) Module

The Rate of Change of Ratio Command (RCRC) module determines the value for the rate of change of ratio (RCR), which is an important parameter in controlling the CVT in order for the HEPS to operate in the desired manner. The computation of the RCR command is dependent on the operating mode from OM (see §7.3.1).

When the HEPS is operating in a Hybrid mode, the RCR is simply computed using the power error and a gain in the following manner:

$$\begin{aligned} \dot{r}_{cmd} &= Gain \cdot PowerError \\ &= Gain \cdot (PowerDemand - EngPower) \end{aligned} \quad (7.9)$$

$$(7.10)$$

where *PowerDemand* is the output from the PD module (see §7.3.2) and *EngPower* from the EO module (see §7.3.3). The *Gain* is taken as 1/2000 in this research, because it is possible for the ICE to produce power up to approximately 2000W, as observed from the engine map for the Aerosonde ICE. As the resulting *RCR Command* value is limited

between -0.1 and 0.1 so that the CVT is not subjected to a large change in its ratio, this *Gain* value needs to be sufficiently small, and the approximate maximum ICE power is an appropriate value to be used in this circumstance.

On the other hand, when the HEPS is operating in either of the *Motor Only* or *Engine Only* modes, it is assumed that the Engine is being operated at the same speed as the Propeller, i.e. $\omega_{eng} = \omega_{prop}$ or $r = 1$. Therefore, when the *Motor Only* or *Engine Only* signal is first detected, the *RCR Command* is set at a value which would increase or decrease the CVT ratio to 1 in 10 time steps. In this research, the simulations are performed with a fixed step size of 0.1 second. This means the CVT reaches a ratio of 1 one second after the *Motor Only* or *Engine Only* signal has been detected. After the 10 time steps, the *RCR Command* is set at zero so that the CVT can maintain a ratio of 1 to keep the Engine operating at the same speed as the Propeller.

The RCRC module has the following input/output configuration as listed in Table 7.14.

	Identifier	Unit	Description
Input	OpMode	-	Current operating mode - 1 = Hybrid Normal - 2 = Motor Only - 3 = Hybrid Charging - 4 = Engine Only - 5 = Hybrid Climbing
	Power Demand	W	Power demand obtained from ACM throttle setting
	Engine Power	W	Approximation of ICE power output
	CVT Ratio	-	CVT ratio
Output	RCR Cmd	-	RCR command

Table 7.14: I/O configuration of *RCR Command* module.

The Simulink schematics of the ETC is shown in Figure 7.26, with its inner schematics in Figure 7.27.

The output of the RCRC module is used mainly in the *Subsystem_CVTDynamics* to control the CVT ratio, but it is also used in the *RCR Compensation Torque* calculations in the *Torque Difference Calculation* module.

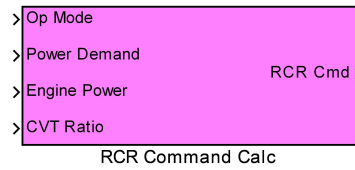


Figure 7.26: The *RCR Command* block.

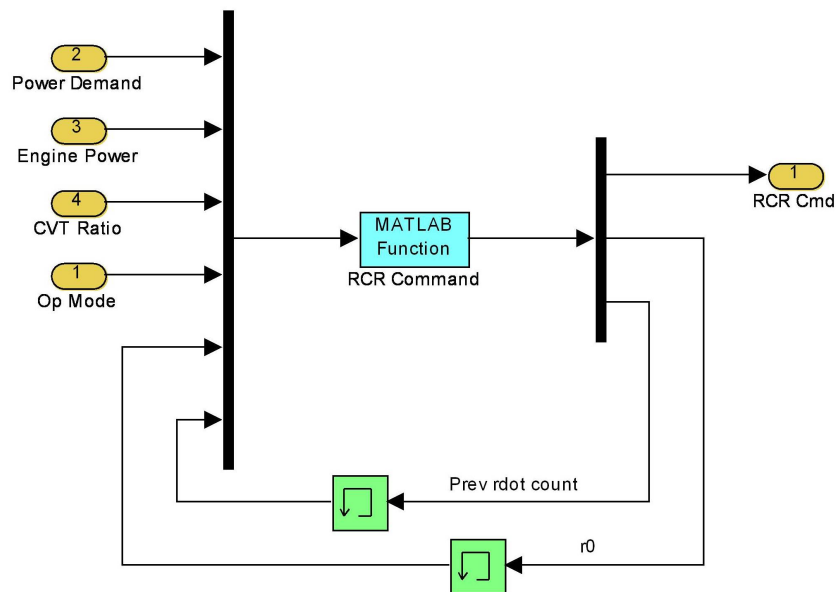


Figure 7.27: Inside the *RCR Command* block.

7.3.6 Torque Difference Calculation (TDC) Module

The Torque Difference Calculation (TDC) module not only computes difference between the desired torque and the IOL torque, it also uses this and other inputs to determine the enable/disable signal to the Engine, EM and Generator modules, as well as to compute the amount of torque available to the Generator or required from the EM.

The use of CVT in the system introduces a torque term, or RCR Torque, which needs to be compensated by the powertrain. This term is given by Equation 7.11, which is extracted from Equation 7.3:

$$RCRTCComp = \dot{r} \omega_{eng} I_p \quad (7.11)$$

This term is negated by the EM, the torque output of which can be expressed as follows:

$$T_M = TorqueDiff + k_2\omega_{eng}\dot{r} \quad (7.12)$$

where k_2 is a constant.

Substituting Equation 7.12 into Equation 7.3 gives:

$$\begin{aligned} \dot{\omega}_{prop} &= \frac{r(T_E + TorqueDiff + k_2\omega_{eng}\dot{r}) - \dot{r}(J_E + J_M) \cdot \omega_{eng} - T_{prop}}{J_{prop} + r^2(J_E + J_M)} \\ &= \frac{r(T_E + TorqueDiff) + rk_2\omega_{eng}\dot{r} - \dot{r}(J_E + J_M) \cdot \omega_{eng} - T_{prop}}{J_{prop} + r^2(J_E + J_M)} \end{aligned}$$

For the EM or Generator to negate the effect of the $-\dot{r}(J_E + J_M) \cdot \omega_{eng}$ term, the following evaluation of k_2 can be assumed:

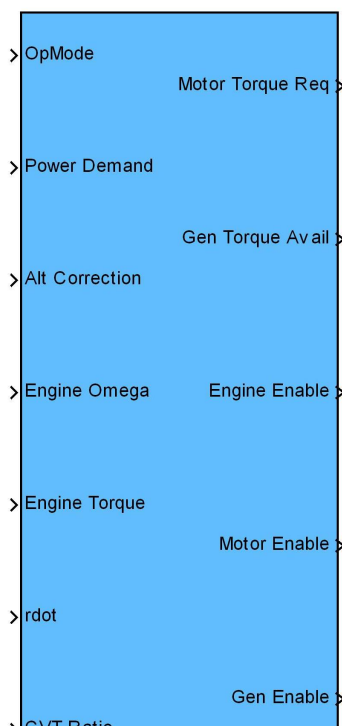
$$\begin{aligned} rk_2\omega_{eng}\dot{r} &= \dot{r}(J_E + J_M) \cdot \omega_{eng} \\ k_2 &= \frac{J_E + J_M}{r} \end{aligned}$$

The input/output configuration of TDCM is as in Table 7.15.

The Simulink schematics of TDC is shown in Figure 7.28, with its inner schematics in Figure 7.29.

	Identifier	Unit	Description
Input	OpMode	-	Current operating mode - 1 = Hybrid Normal - 2 = Motor Only - 3 = Hybrid Charging - 4 = Engine Only - 5 = Hybrid Climbing
	Power Demand	W	Power demanded
	Alt Correction	-	Altitude correction obtained from atmospheric conditions
	Engine Omega	rad/s	Engine speed
	Engine Torque	Nm	Approximation of Engine torque output
	rdot	-	RCR command
	CVT Ratio	-	CVT Ratio
Output	Motor Torque Req	Nm	Torque required to be supplemented by the EM
	Gen Torque Avail	Nm	Torque available to the Generator
	Engine Enable	-	Signal to activate the Engine - 0 = disable - 1 = enable
	Motor Enable	-	Signal to activate the EM - 0 = disable - 1 = enable
	Gen Enable	-	Signal to activate the Generator - 0 = disable - 1 = enable

Table 7.15: I/O configuration of *Torque Difference Calc*



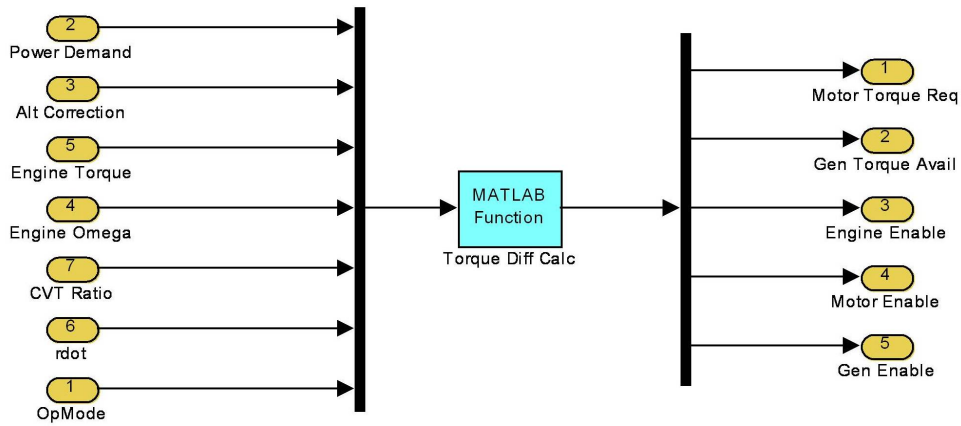


Figure 7.29: Inside the *Torque Difference Calc* block.

All the TDC outputs are passed to the *Powertrain* block to their respective models.

7.3.7 Integration of IOL Controller Components

The integrated IOL Controller is shown in Figure 7.30, with the inner schematics in Figure 7.31.



Figure 7.30: The *IOL Controller* block.

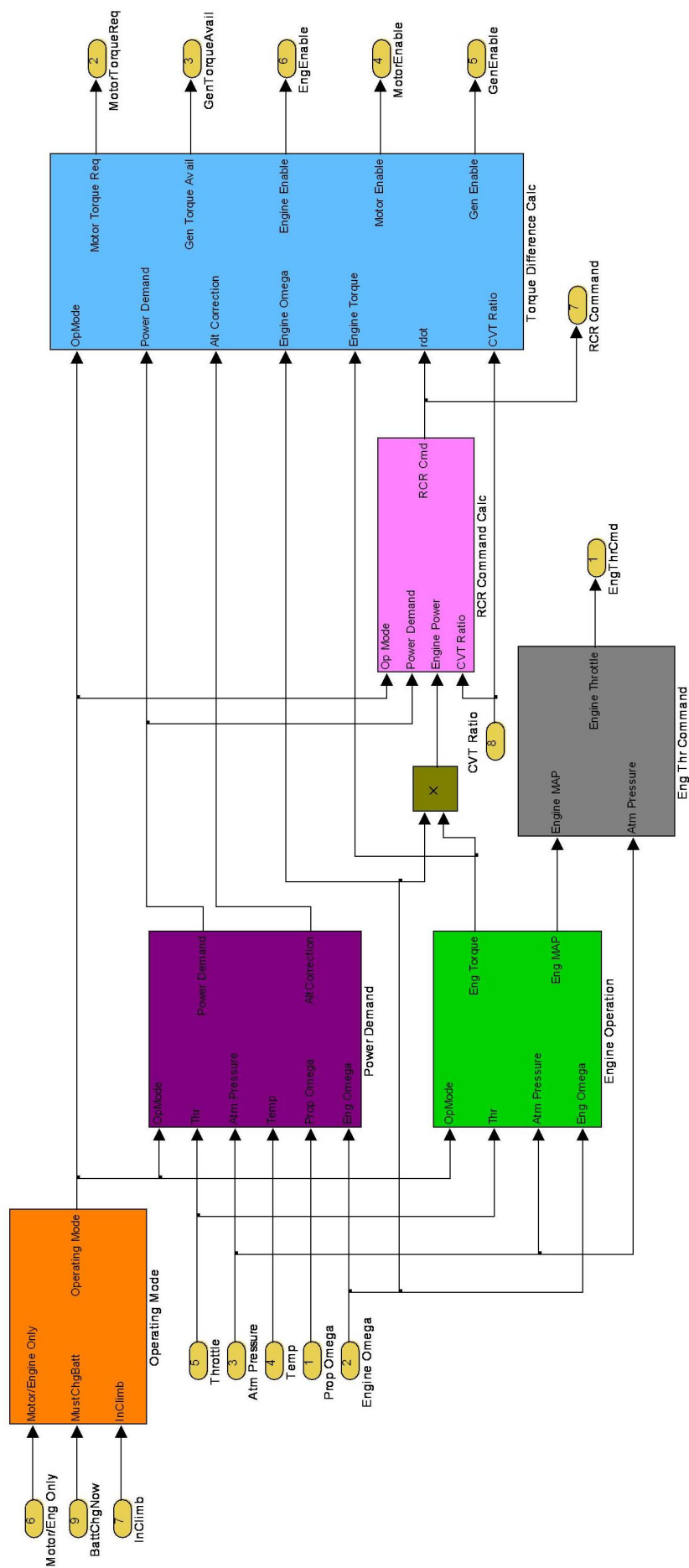


Figure 7.31: Inside the IOL Controller block.

7.4 Integrated HEPS Model

Integrating all the components described in §7.2 and §7.3 gives the integrated HEPS model, which is shown in Figure 7.32, with the inner schematics in Figure 7.33.

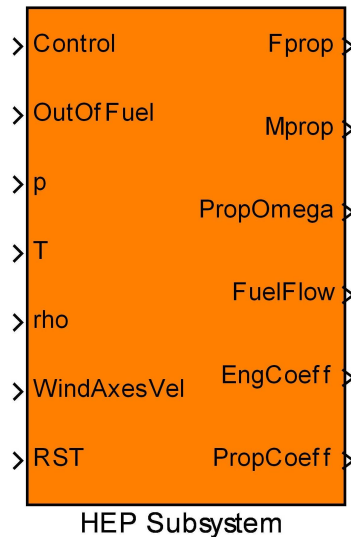


Figure 7.32: The integrated HEPS block.

This integrated HEPS model is integrated into the Aerosonde UAV Block as shown in Figure 7.34.

Using this HEPS-integrated model, a mission simulation using the MS1 waypoints, taken from Chapter 3 and re-iterated in Table 7.16, was performed. The results of the simulation are shown in Figures 7.35 and 7.36. Figures 7.37 and 7.38 show that the flight path generated by the HEPS-integrated model is a close approximate of that of the ICE-only configuration.

At the start of the mission ($t = 0s$ to $t = 95.7s$), the UAV was required to climb to an altitude of 900m. Since the Battery was at full charge (Battery SOC = 100%), this triggered the onset of the “Hybrid Climbing” mode ($OpMode = 5$) and the EM is activated to provide assisting torque to the ICE. The fuel saving effects in this leg is evident in the Fuel Flow plot in Figure 7.36, which showed a fuel flow of approximately $3 \times 10^{-5}g/hr$ for the Hybrid configuration, compared to approximately $4.5 \times 10^{-5}g/hr$ for the ICE Only configuration. However, the UAV had reached the desired altitude of 900m before reaching the desired waypoint coordinates, and since the use of the EM had drained the Battery SOC to below its “Must Charge Now” threshold, the “Engine Only”

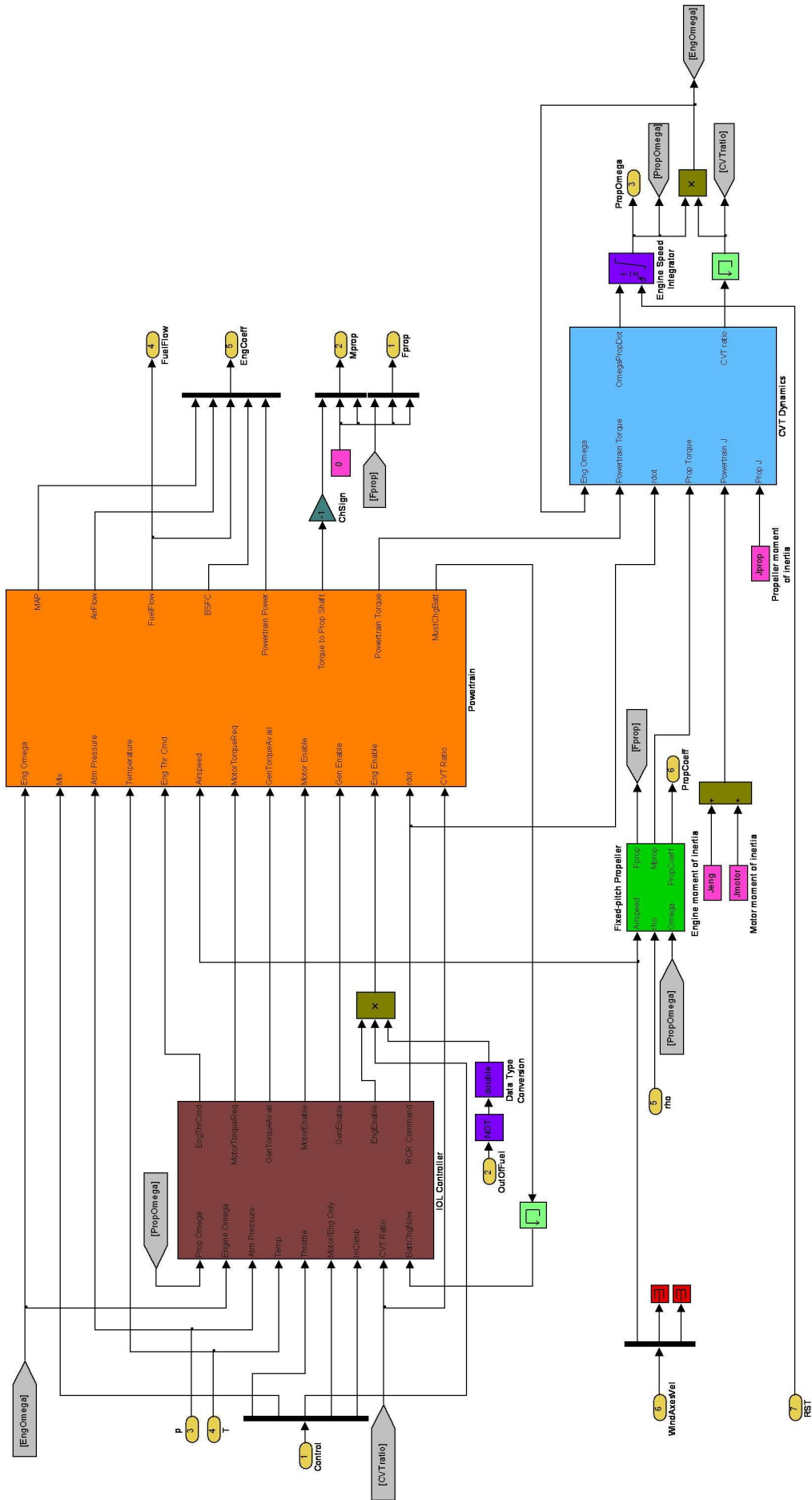


Figure 7.33: Inside the integrated HEPS block.

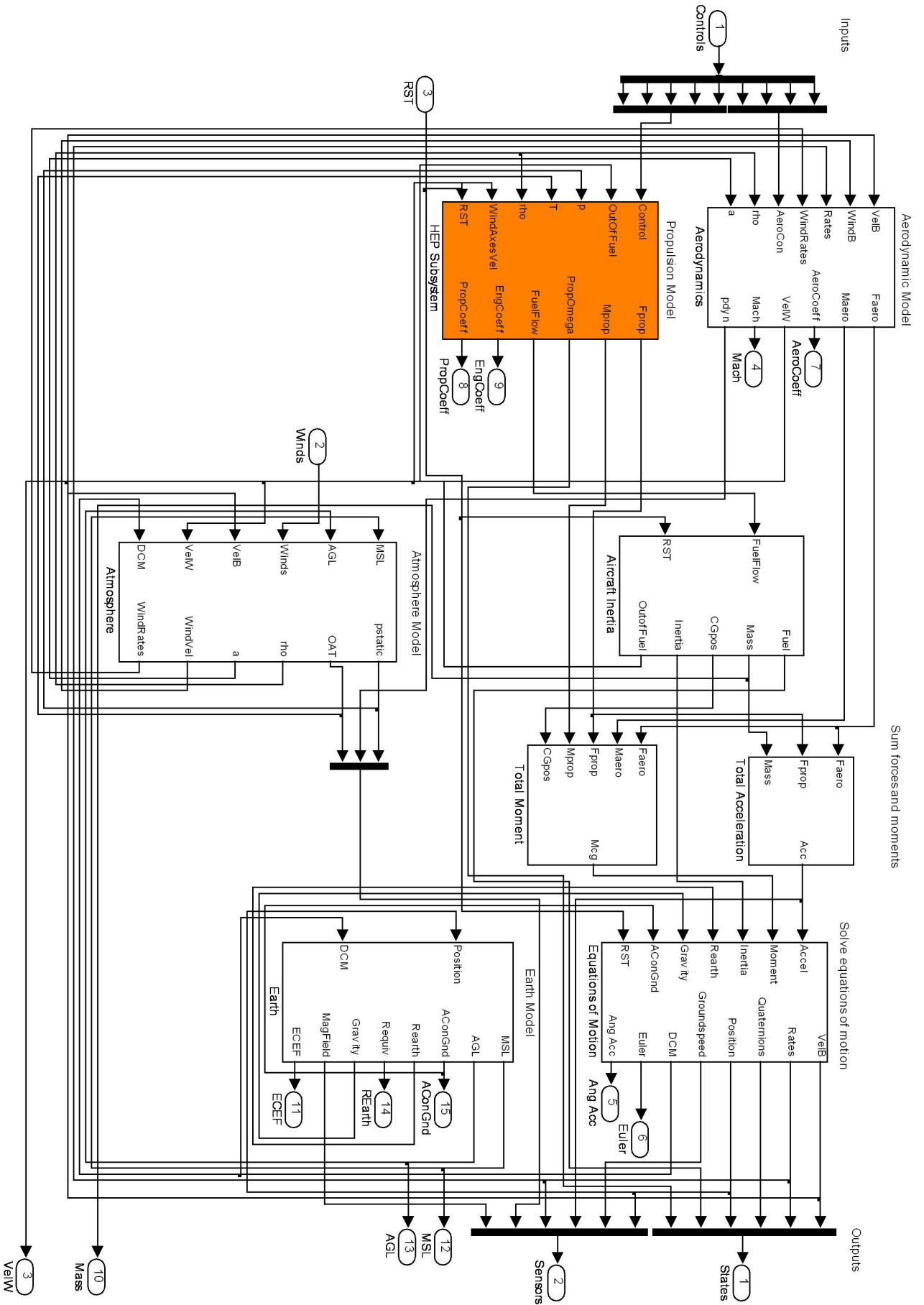


Figure 7.34: Integrating HEPS model (in orange) inside Aeronde UAV block

Waypoint	Waypoint Coordinates		
	Latitude (DD:MM:SS)	Longitude (DD:MM:SS)	Altitude (m)
WP1 (Kingaroy Airport)	26°34'51" S	151°50'28" S	800
WP2	26°33'58" S	151°51'10" E	900
WP3	26°34'08" S	151°51'25" E	900
WP4	26°34'16" S	151°53'13" E	750
WP5	26°34'08" S	151°53'18" E	750
WP6	26°33'59" S	151°53'13" E	750
WP7	26°34'08" S	151°51'25" E	900
WP8	26°34'14" S	151°51'17" E	900
WP9 (Kingaroy Airport)	26°34'51" S	151°40'28" S	800

Table 7.16: Waypoints in MS1.

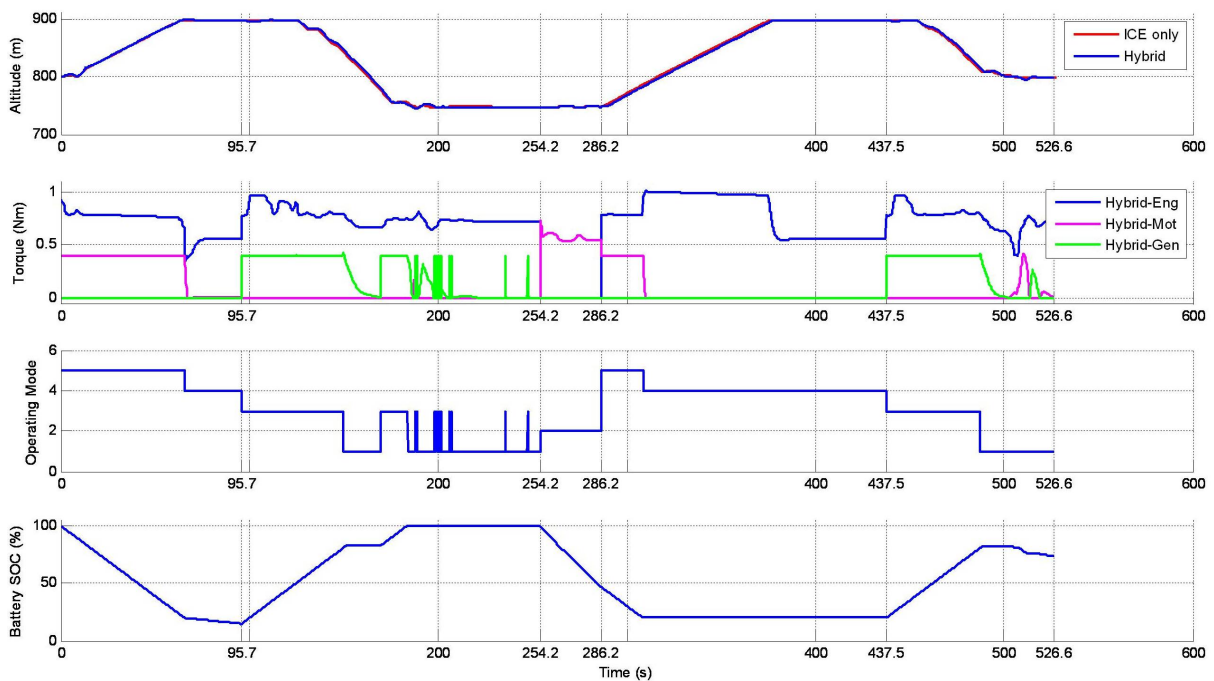


Figure 7.35: Mission simulation using UAVSM with integrated HEPS (Plot 1).

mode ($OpMode = 4$) was triggered, which minimised the use of the EM and thus the Battery.

The next leg ($t = 95.8s$ to $t = 254.2s$) was a cruise phase, during which the UAV was required to hold an altitude of 900m, followed by a descent to and hold at 750m . At the

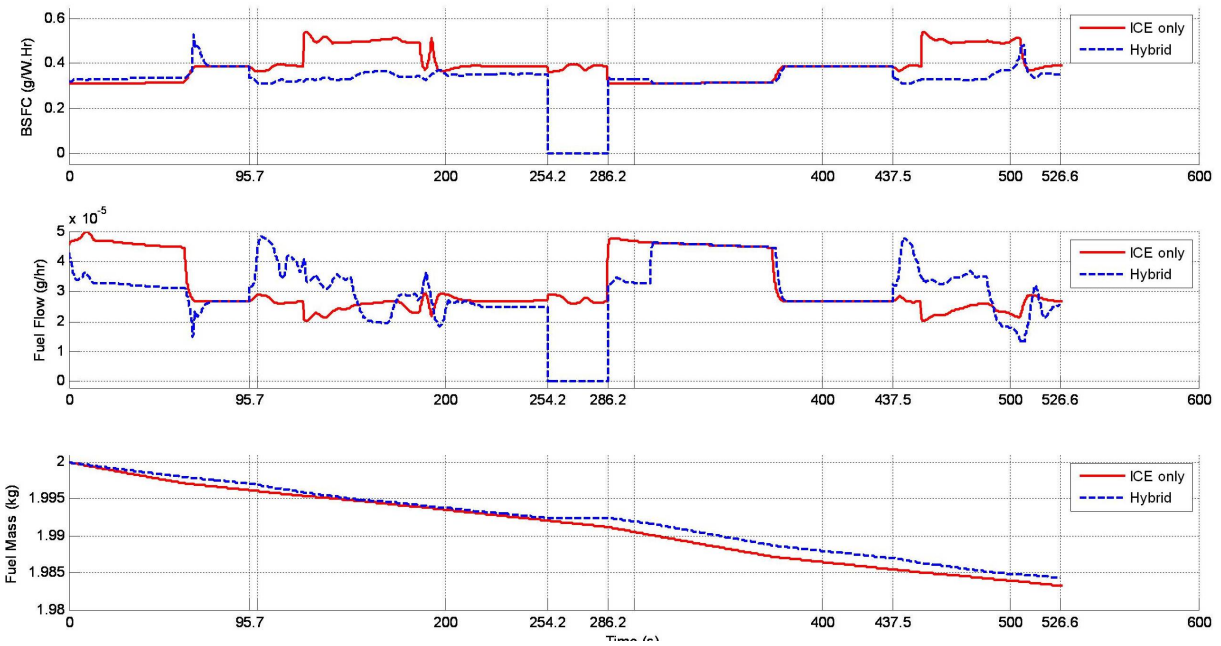


Figure 7.36: Mission simulation using UAVSM with integrated HEPS (Plot 2)

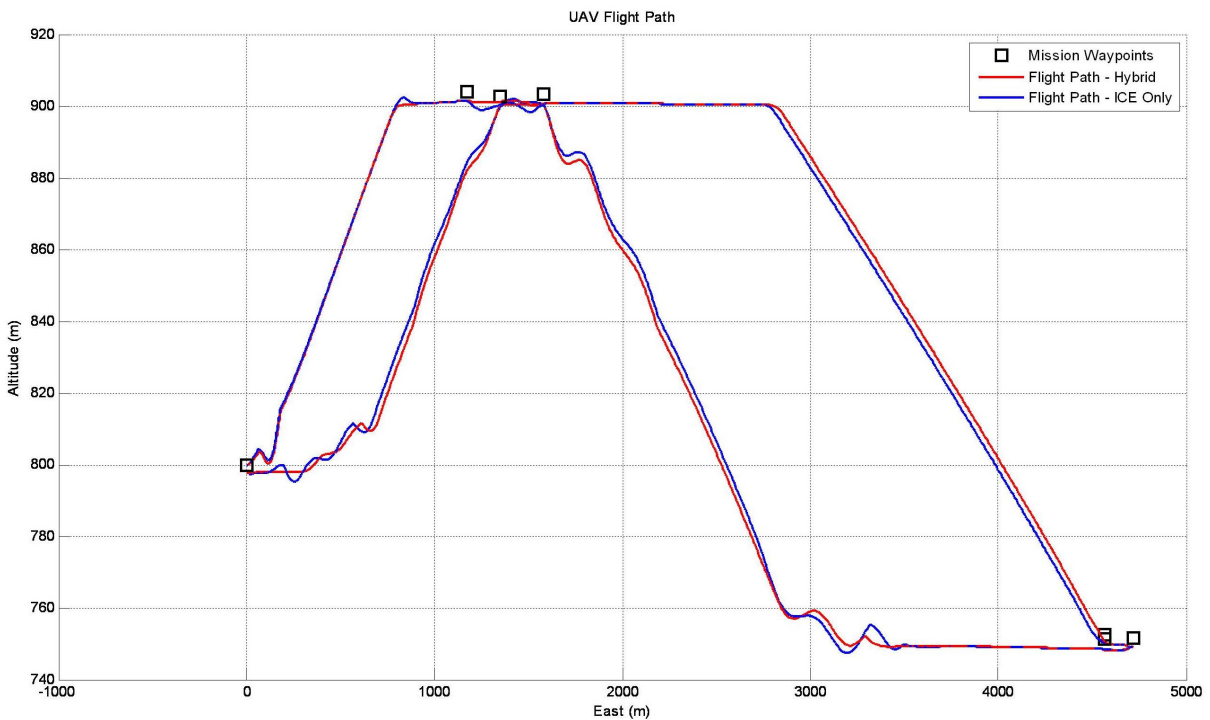


Figure 7.37: Flight path comparison of UAVSM with integrated HEPS and original configuration (side view).

beginning of this leg, the UAV replenished the Battery by entering the “Hybrid Charging” mode ($OpMode = 3$) and the Generator can be seen to receive a constant torque, which charged the Battery. After the Battery SOC had reached the preset threshold of 85%,

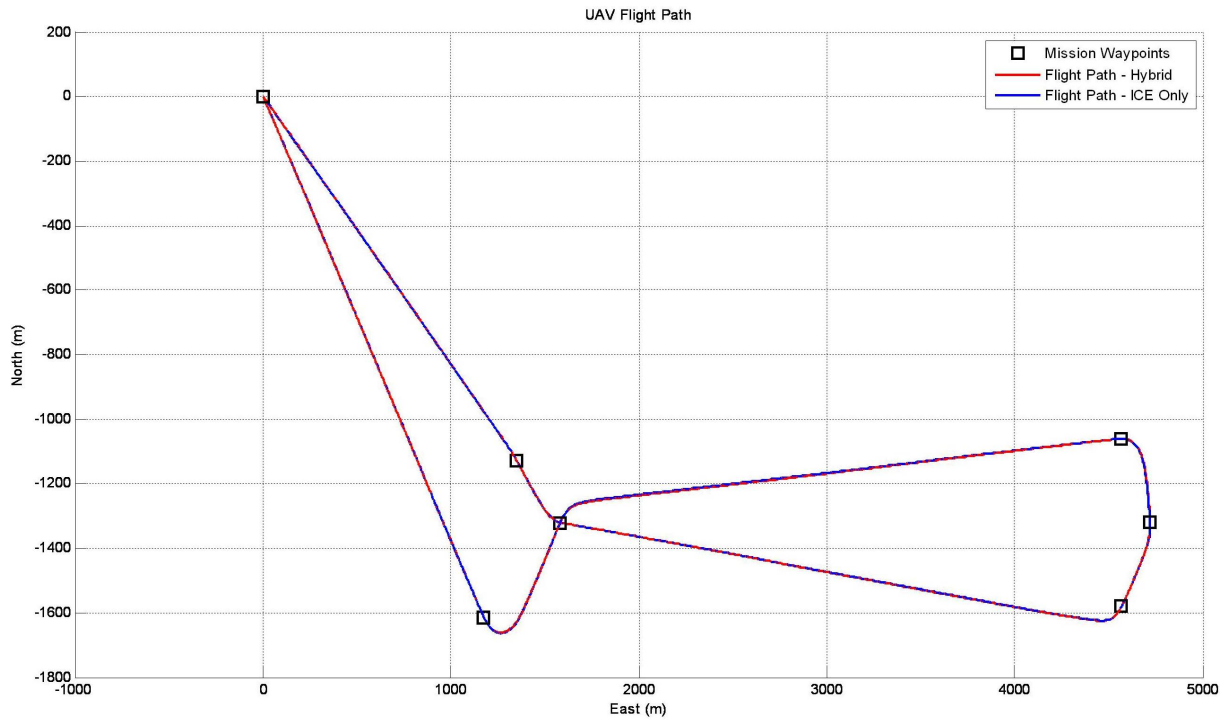


Figure 7.38: Flight path comparison of UAVSM with integrated HEPS and original configuration (top view).

it was able to return to the “Hybrid Normal” mode ($OpMode = 1$) for most of the remainder of this leg. At $t = 169.7s$, because it was desired that the aircraft enter the “Motor Only” phase with the Battery fully charged, the “Hybrid Charging” mode was once again triggered. This period of enforced charging of the Battery did not last long, as the Battery SOC was at a reasonably high level before this occurred. Therefore, at $t = 184s$, the UAV returned to the “Hybrid Normal” mode, with minor switching to the “Hybrid Charging” mode at moments when the EM, and therefore the Battery, had been activated. In terms of fuel, the “Hybrid Charging” mode required significantly more fuel (between 3×10^{-5} to $5 \times 10^{-5}g/hr$) than the ICE Only configuration (from 2×10^{-5} to $2.5 \times 10^{-5}g/hr$). However, in the “Hybrid Normal” mode, the Hybrid configuration showed smaller fuel flow values than that of the ICE Only configuration.

During the “Motor Only” phase ($t = 254.3s$ to $t = 286.2s$), the UAV entered the “Motor Only” mode ($OpMode = 2$), which used no fuel (zero fuel flow, compared to approximately $2.5 \times 10^{-5}g/hr$ in the ICE Only configuration), but the Battery was drained to a SOC of approximately 50%.

After the “Motor Only” phase, the UAV was in another climbing phase to reach an

altitude of 900m ($t = 286.3\text{s}$ to $t = 437.5\text{s}$). At the beginning of this leg, the Battery was adequately charged to power the EM, therefore the “Hybrid Climbing” mode ($OpMode = 5$) was entered. This resulted in a fuel flow of approximately $3.5 \times 10^{-5}\text{g/hr}$ for the Hybrid configuration, compared to approximately $4.5 \times 10^{-5}\text{g/hr}$ for the ICE Only configuration. However, after approximately 20s, the lower threshold of the Battery SOC was reached, which prompted the “Must Charge Battery” signal to be activated. This, combined with the fact that the UAV required to climb to a higher altitude, resulted in the UAV operating in the “Engine Only” mode ($OpMode = 4$). During this period of time, the fuel flow for both configurations was identical.

The last leg of the mission ($t = 437.6\text{s}$ to $t = 526.6\text{s}$) required the UAV to hold at an altitude of 900m, then descend to the final altitude of 800m. Initially, the UAV was in the “Hybrid Charging” mode ($OpMode = 3$) because the Battery SOC had reached the lower threshold, and the fuel flow for the Hybrid configuration in this period of time was significantly greater than that of the ICE Only configuration (between 3.5×10^{-5} to $4.8 \times 10^{-5}\text{g/hr}$ versus approximately $2 \times 10^{-5}\text{g/hr}$ respectively). But at $t = 487.4\text{s}$, the Battery SOC reached the threshold and further charging was not required, the UAV returned to the “Hybrid Normal” mode ($OpMode = 1$). Once again, the Hybrid configuration resulted in a fuel flow smaller in value to that of the ICE Only configuration.

Overall, the fuel consumption for the Hybrid configuration was 0.01567kg or 15.67g, compared to 0.01676kg or 16.76g of the ICE Only configuration, which resulted in a saving of 1.09g, or 6.5%, of fuel. This is achieved with the approximate additional weight of the EM (0.25kg for the Plettenberg), the Battery (0.168kg for two of the Air Thunder LiPo battery packs), and a CVT and controller that may be customised to result in a total additional weight of less than 1.35kg, which is 10% of the mass of the original Aerosonde UAV (13.5kg with a full tank). Therefore, this simulation demonstrated that the implementation of a HEPS on a fixed-wing UAV is capable of reducing the onboard fuel consumption.

7.5 Summary

The aim of this chapter was to develop the components required for the development of a HEPS model for small fixed-wing UAVs. These components include the following:

- Powertrain
 - Engine
 - Fuel
 - Electric Motor (EM)
 - Generator
 - Battery
 - Transmission (CVT)
- IOL Controller
 - Operating Mode
 - Power Demand
 - Engine Operation
 - Engine Throttle Command
 - RCR Command
 - Torque Difference Calculation

In this chapter, simulation models of the Hybrid-Electric Propulsion System (HEPS) components as listed above were designed and implemented in MATLAB Simulink, incorporating the IOL determined in Chapter 6 into the IOL Controller. Integrated models of the Powertrain, IOL Controller and the full HEPS were implemented. Mission simulation using the integrated HEPS were performed and results showed that the HEPS configuration was capable of achieving a fuel saving of 6.5%.

The work described in this chapter satisfied Research Objectives 6 and 7 as specified in §1.2.

Chapter 8

Conclusions

8.1 Summary of the Research

The aim of this thesis was to present an investigation of methods to increase the energy efficiency onboard UAVs. One method explored was the development of a Mission Waypoint Optimisation (MWO) procedure to improve the fuel economy onboard a small fixed-wing UAV. Also, a foundation for the implementation of a parallel HEPS onboard the UAV incorporating an Ideal Operating Line (IOL) control strategy was also developed.

Efficient fuel or energy consumption onboard any aircraft is always crucial in its operations because of the limited weight and space that are available onboard the aircraft. This is even more a problem for small UAVs, therefore it is advantageous to develop methods of further economising the fuel consumption onboard. One way of doing so is by performing Mission Waypoint Optimisation (MWO), or the finding of a set of mission waypoints which, when executed by an aircraft, one or more desired parameters would be optimised. This is different to Flight Mission Planning or trajectory optimisation in that MWO deals with a pre-specified set of waypoints, rather than generating the waypoints, by modifying the given waypoints within certain limits to achieve its optimisation objectives of minimising/maximising specific parameters.

The literature review conducted in Chapter 2 identified the need to develop an accurate simulation model of an UAV in order for MWO to be performed, an UAV's multi-physics nature making it impossible to develop an analytical model. One simulation model, the UAV Simulation Model (UAVSM), was constructed in the MATLAB Simulink

environment, utilising the AeroSim Blockset; see Chapter 3. Test cases confirmed the validity of the UAVSM in tracking a flight plan. This work fulfilled Research Objective 2 as listed in §1.2.

At the same time, the use of a simulation model instead of an analytical one to represent an UAV prompts the need for an optimisation algorithm which can perform simulation optimisation. Two optimisation algorithms – SQP-based and EA-based, respectively – were deemed the most appropriate for this purpose. The selection of these two different methods followed a literature review conducted in Chapter 2 of the optimisation techniques and algorithms that are currently available. Research Objective 1 was satisfied through the completion of this work.

The integration of UAVSM and the HAPMOEA optimiser, a MOEA-based optimisation method that has parallel computing capabilities, was carried out and HAPMOEA's ability to deal with both single-objective and multi-objective MWO problems were demonstrated in Chapters 4 and 5 respectively. The UAVSM was also coupled with the SQP solver, a gradient-based optimisation method as part of the MATLAB Optimization Toolbox, was also performed. Results showed that the SQP solver was more computationally efficient in the optimisation process when the search space is limited, but its capabilities of performing multi-objective optimisation (MOO) was very limited, whereas the HAPMOEA optimiser was able to explore a larger search space more thoroughly and efficiently. Additionally, the HAPMOEA optimiser has the parallel computer capabilities which can further improve the its computational efficiency. The work demonstrated in these chapters met Research Objectives 3 and 4 in §1.2.

Last but not least, simulation models were implemented for the components of a parallel Hybrid-Electric Propulsion System (HEPS). Also, an IOL analysis of the Aerosonde ICE was performed. However, due to the fact that only limited data are available on the performance of the Aerosonde ICE, therefore estimation techniques such as interpolation and extrapolation were required in the computation process for the IOL. Nonetheless, the completion of this analysis established the tools and techniques which were helpful in the process and fulfilled Research Objective 5.

The components required in the implementation of a Hybrid-Electric Propulsion System (HEPS) were identified and individually developed. Test cases confirmed each component to have the expected functionalities. Integration of the Powertrain components – namely Engine, Fuel, Electric Motor, Generator, Battery, CVT Dynamics – was achieved. Similarly, the integration of the IOL Controller components – Operating Mode, Power Demand, Engine Operation, Engine Throttle Command, RCR Command and Torque Difference Calculation – was also carried out. Integration of the IOL Controller and the Powertrain components were successfully carried out, with the resulting HEPS model integrated into the UAVSM and mission simulation was performed. It was demonstrated through simulation that an UAV with the current HEPS configuration was capable of achieving a fuel saving of 6.5%, compared to the ICE-only configuration. These tasks will enable the development of a complete simulation model of an Hybrid-Electric UAV (HEUAV) and thus satisfying Research Objectives 6 and 7.

Overall, the research work described and completed in this thesis has fulfilled the Research Objectives as listed in §1.2.

8.2 Future Work

Some aspects of the research conducted here will require further work. These include the following:

1. UAV Simulation Model (UAVSM)

- Further investigation into the climb and descend capabilities need to be conducted. Currently UAV encounters problems when a descend over a short horizontal distance is required.
- Effects of wind and weather should be investigated and incorporated into the UAVSM.
- The flight mission should be extended to include take-off and landing sequences.

2. Mission Waypoint Optimisation (MWO)

- The capability of parallel computing was not performed in the course of this research. In the interest of reducing computation time, setting up of the parallel

computing aspect of HAPMOEA should be investigated.

3. Hybrid-Electric Propulsion System (HEPS) Modelling

- Improvements in the implementation of the IOL controller may result in great fuel savings.

References

- [1] The application of metaheuristic search techniques to problems in software engineering. [Online]. Available: <http://www.dcs.kcl.ac.uk/projects/seminal/The%20Application%20of%20Metaheuristic%20Search%20Techniques%20to%20Problems%20in%20Software%20Engineering.htm>
- [2] Unmanned aerial vehicles (UAV). [Online]. Available: <http://www.fas.org/irp/program/collect/uav.htm>
- [3] Army unmanned aircraft system operations. [Online]. Available: <http://www.fas.org/irp/doddir/army/fmi3-04-155.pdf>
- [4] Jane's unmanned aerial vehicles and targets. [Online]. Available: <http://juav.janes.com/public/juav/index.shtml>
- [5] "World unmanned aerial vehicle systems - 2010: Market profile and forecast," Teal Group Corporation, Tech. Rep., 2010.
- [6] J. M. Abatti, "Small power: The role of micro and small UAVs in the future," Air Command and Staff College, Air University, Tech. Rep., April 2005.
- [7] M. J. Logan, J. Chu, M. A. Motter, D. L. Carter, M. Ol, and C. Zeune, "Small UAV research and evolution in long endurance electric powered vehicles," in *AIAA Infotech Aerospace 2007 Conference and Exhibit*, Rohnert Park, CA, USA, 7-10 May, 2007.
- [8] R. Glasscock, J. Y. Hung, R. A. Walker, and L. Gonzalez, "Design, modelling and measurement of hybrid powerplant for unmanned aerial systems (UAS)," in *5th Australasian Conference on Applied Mechanics*, Brisbane, Australia, 10-12 December, 2007.

- [9] T. Cox, "Civil UAV capability assessment," NASA, Tech. Rep., 2004.
- [10] M. T. DeGarmo, "Issues concerning integration of unmanned aerial vehicles in civil airspace," MITRE, Center for Advanced Aviation System Development, Tech. Rep., November 2004.
- [11] I. McManus, "A multidisciplinary approach to highly autonomous UAV mission planning and piloting for civilian airspace," Ph.D. Thesis, Queensland University of Technology, Brisbane, Australia, 2004.
- [12] P. P.-Y. Wu, "Multi-objective mission flight planning in civil unmanned aerial systems," Ph.D. Thesis, Queensland University of Technology, Brisbane, Australia, 2009.
- [13] P. P. Narayan, D. A. Campbell, and R. A. Walker, "Computationally adaptive multi-objective trajectory optimization for UAS with variable planning deadlines," in *2009 IEEE Aerospace Conference*, Big Sky, MT, USA, 7-14 March, 2009.
- [14] A. Bhatia, A. Mendiratta, and M. Vaish, "Comparison of proposed six stroke internal combustion engine with four stroke engine using ideal cycle," in *Proceedings of the 2nd International Conference on Mechanical and Electronics Engineering (ICMEE2010)*, vol. 1, Kyoto, Japan, 1-3 August, 2010, pp. 222–225.
- [15] Energy-efficient electric machines. [Online]. Available: <http://www.csiro.au/science/ElectricMachines.html>
- [16] A. B. Francisco, "Implementation of an ideal operating line control strategy for hybrid electric vehicles," M.Sc. Thesis, University of California, Davis, Davis, CA, USA, 2002.
- [17] C. Kim, E. NamGoong, S. Lee, T. Kim, and H. Kim, "Fuel economy optimization for parallel hybrid vehicles with CVT," SAE International, Tech. Rep., 1999.
- [18] F. G. Harmon, "Neural network control of a parallel hybrid-electric propulsion system for a small unmanned aerial vehicle," Ph.D. Thesis, University of California, Davis, Davis, CA, USA, 2005.

-
- [19] K. Aoki, S. Kuroda, S. Kajiwara, H. Sato, and Y. Yamamoto, “Development of Integrated Motor Assist Hybrid System: Development of the ‘Insight’, a personal hybrid coupe,” SAE International, Tech. Rep., 2000.
- [20] K. T. Chau and Y. S. Wong, “Overview of power management in hybrid electric vehicles,” *Energy Conversion and Management*, vol. 43, no. 15, pp. 1953–1968, 2002.
- [21] B. Cho, “Control of a hybrid electric vehicle with predictive journey estimation,” Ph.D. Thesis, Cranfield University, Cranfield, UK, 2008.
- [22] A. A. Frank, “Engine optimization concepts for CVT-hybrid systems to obtain the best performance and fuel efficiency,” in *2004 International Continuously Variable and Hybrid Transmission Congress*, Davis, CA, USA, 23-25 September, 2004.
- [23] Oxford Dictionaries, “optimize”. *Oxford Dictionaries*. Oxford University Press, 2010.
- [24] E. Tekin and I. Sabuncuoglu, “Simulation optimization: A comprehensive review on theory and applications,” *IIE Transactions*, vol. 36, no. 11, pp. 1067–1081, 2004.
- [25] S. Olafsson and J. Kim, “Simulation optimization,” in *Proceedings of the 2002 Winter Simulation Conference*, E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, Eds., San Diego, CA, USA, 8-11 December, 2002, pp. 79–84.
- [26] J. P. C. Kleijnen, W. C. M. van Beers, and I. van Nieuwenhuysse, “Constrained optimization in simulation: A novel approach,” Tilburg University, Center for Economic Research, Discussion Paper 2008-95, 2008.
- [27] F. Azadivar, “Simulation optimization methodologies,” in *Proceedings of the 1999 Winter Simulation Conference*, P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, Eds., vol. 1, Phoenix, AZ, USA, 5-8 December, 1999, pp. 93–100.
- [28] S.-M. Guo, “A fast multi-objective evolutionary algorithm for expensive simulation optimization problems,” in *The 2nd International Conference on Innovative Computing, Information and Control (ICICIC ’07)*, Kumamoto, Japan, 5-7 September, 2007.

- [29] H. Pierreval and J.-L. Paris, "Distributed evolutionary algorithms for simulation optimization," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 30, no. 1, pp. 15–24, 2000.
- [30] D. Erdman and M. Little, "Nonlinear regression analysis and nonlinear simulation models," in *Proceedings of the 19th SAS Users Group International (SUGI 1994)*, Dallas, TX, USA, 10-13 April, 1994.
- [31] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*, 2nd ed. New York: McGraw-Hill, 1991.
- [32] J. April, F. Glover, J. P. Kelly, and M. Laguna, "The exploding domain of simulation optimization," *Newsletter of the INFORMS Computing Society*, vol. 24, no. 2, pp. 1–14, 2003.
- [33] J. R. Swisher, P. D. Hyden, S. H. Jacobson, and L. W. Schruben, "A survey of simulation optimization techniques and procedures," in *Proceedings of the 2000 Winter Simulation Conference*, vol. 1, Orlando, FL, USA, 10-13 December, 2000, pp. 119–128.
- [34] J. D. Hall and R. O. Bowden, "A unified strategy for simulation optimization research and development," in *Proceedings for the 7th Industrial Engineering Research Conference*, Banff, Canada, 9-10 May, 1998, pp. 1–8.
- [35] R. O. Bowden and J. D. Hall, "Simulation optimization research and development," in *Proceedings of the 1998 Winter Simulation Conference*, vol. 2, Washington, DC, USA, 13-16 December, 1998, pp. 1693–1698.
- [36] H. Kargupta and D. E. Goldberg, "Search, blackbox optimisation, and sample complexity," in *Proceedings of the 4th Workshop on Foundations of Genetic Algorithms*, R. K. Belew and M. D. Vose, Eds., San Diego, CA, USA, 5 August, 1996, pp. 291–324.
- [37] M. Laguna, "Metaheuristic optimization with Evolver, Genocop and OptQuest," in *EURO XV / INFORMS XXXIV: Joint International Meeting 1997 Plenaries and Tutorials*, Barcelona, Spain, 14-17 July, 1997.

- [38] M. C. Fu, "Gradient estimation," in *Simulation*, ser. Handbooks in Operations Research and Management Science, Vol. 13, S. G. Henderson and B. L. Nelson, Eds. Amsterdam: Elsevier, 2006.
- [39] F. Glover and G. A. Kochenberger, *Handbook of Metaheuristics*, ser. International Series in Operations Research and Management Science, Vol. 57. Boston: Kluwer Academic Publishers, 2003.
- [40] E.-G. Talbi, *Metaheuristics: From Design to Implementation*, ser. Wiley Series on Parallel and Distributed Computing. Hoboken: John Wiley & Sons, 2009.
- [41] J. April, F. Glover, J. P. Kelly, and M. Laguna, "Practical introduction to simulation optimization," in *Proceedings of the 2003 Winter Simulation Conference*, S. Chick, P. J. Sanchez, D. Ferrin, and D. J. Morrice, Eds., New Orleans, LA, USA, 7-10 December, 2003, pp. 71–78.
- [42] H. Eskandari, "Multiobjective simulation optimization using enhanced evolutionary algorithm approaches," Ph.D. Thesis, University of Central Florida, Orlando, FL, USA, 2006.
- [43] M. C. Fu, C.-H. Chen, and L. Shi, "Some topics for simulation optimization," in *Proceedings of the 2008 Winter Simulation Conference*, S. J. Mason, R. R. Hill, L. Monch, O. Rose, T. Jefferson, and J. W. Fowler, Eds., Austin, TX, USA, 7-10 December, 2008, pp. 27–38.
- [44] J. L. Morales, J. Nocedal, and Y. Wu, "A sequential quadratic programming algorithm with an additional equality constrained phase," Optimization Center, Northwestern University, Tech. Rep., 2008.
- [45] A. Sheta and H. Turabieh, "A comparison between genetic algorithms and sequential quadratic programming in solving constrained optimization problems," *The International Journal of Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 67–74, 2006.
- [46] M. C. Biggs, "Constrained minimization using recursive quadratic programming: Some alternative subproblem formulations," in *Towards Global Optimization*,

- L. C. W. Dixon and G. P. Szergo, Eds. Amsterdam: North-Holland, 1975, pp. 341–349.
- [47] S. P. Han, “A globally convergent method for nonlinear programming,” *Journal of Optimization Theory and Applications*, vol. 22, no. 3, pp. 297–309, 1977.
- [48] M. J. D. Powell, “The convergence of variable metric methods for nonlinearly constrained optimization calculations,” in *Nonlinear Programming 3*, O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, Eds. Academic Press, 1978.
- [49] —, “A fast algorithm for nonlinearly constrained optimization calculations,” in *Numerical Analysis*, ser. Lecture Notes in Mathematics, G. A. Watson, Ed. Berlin: Springer-Verlag, 1978, vol. 630, pp. 144–157.
- [50] P. T. Boggs and J. W. Tolle, “Sequential quadratic programming,” *Acta Numerica*, vol. 4, pp. 1–51, 1995.
- [51] “Optimization ToolboxTM5 user’s guide,” The MathWorks, Inc., Tech. Rep., 2010.
- [52] P. S. Els, P. E. Uys, J. A. Snyman, and M. J. Thoresson, “Gradient-based approximation methods applied to the optimal design of vehicle suspension systems using computational models with severe inherent noise,” *Mathematical and Computer Modeling*, vol. 43, no. 7–8, pp. 787–801, 2006.
- [53] P. E. Gill and E. Wong, “Sequential quadratic programming methods,” University of California, San Diego, Tech. Rep., August 2010.
- [54] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, “User’s guide for NPSOL 5.0: A Fortran package for nonlinear programming,” Systems Optimization Laboratory, Stanford University, Tech. Rep., 1998.
- [55] P. E. Gill, W. Murray, and M. A. Saunders, “SNOPT: An SQP algorithm for large-scale constrained optimization,” *SIAM Journal on Optimization*, vol. 12, pp. 979–1006, 1997.
- [56] —, “User’s guide for SNOPT 7.1: A Fortran package for large-scale nonlinear programming,” Department of Mathematics, University of California, San Diego, Tech. Rep., 2005.

- [57] M. B. Milam, “Real-time optimal trajectory generation for constrained dynamical systems,” Ph.D. Thesis, California Institute of Technology, Pasadena, CA, USA, 2003.
- [58] T. Inanc, K. Misovec, and R. M. Murray, “Nonlinear trajectory generation for unmanned air vehicles with multiple radars,” in *Proceedings of the 43rd IEEE Conference on Decision and Control (CDC 2004)*, vol. 4, Atlantis, Paradise Island, Bahamas, 14-17 December, 2004, pp. 3817–3822.
- [59] K. Misovec, T. Inanc, J. Wohletz, and R. Murray, “Low-observable nonlinear trajectory generation for unmanned air vehicles,” in *Proceedings of the 42nd IEEE Conference on Decision and Control*, Maui, HI, USA, 9-12 December, 2003, pp. 3103–3110.
- [60] F.-L. Lian and R. Murray, “Real-time trajectory generation for the cooperative path planning of multi-vehicle systems,” in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002*, vol. 4, Las Vegas, NV, USA, 10-13 December, 2002, pp. 3766–3769.
- [61] I. M. Garcia, “Nonlinear trajectory optimization with path constraints applied to spacecraft reconfiguration maneuvers,” M.Sc. Thesis, Massachusetts Institute of Technology, Boston, MA, USA, 2005.
- [62] J. C. Meza and I. Olkin, “Numerical procedures for estimating the parameters in a multivariate homogeneous correlation model with unequal variances,” *Sankhya*, vol. 55, no. 3, pp. 506–515, 1993.
- [63] T. H. Bradley, B. A. Moffitt, D. E. Parekh, T. F. Fuller, and D. N. Mavris, “Energy management for fuel cell powered hybrid-electric aircraft,” in *7th International Energy Conversion Engineering Conference*, Denver, CO, USA, 2-5 August, 2009.
- [64] B. Vanek, T. Peni, J. Bokor, and G. Balas, “Practical approach to real-time trajectory tracking of UAV formations,” in *Proceedings of the 2005 American Control Conference*, Portland, OR, USA, 8-10 June, 2005, pp. 122–127.
- [65] B. R. Geiger, J. F. Horn, A. DeLullo, L. N. Long, and A. F. Niessner, “Optimal path planning of UAVs using direct collocation with nonlinear programming,” in

- AIAA Guidance, Navigation, and Controls Conference and Exhibit*, Keystone, CO, USA, 21-24 August, 2006.
- [66] J. A. Kyle, “Optimal soaring by a small autonomous glider,” Ph.D. Thesis, Oregon State University, Corvallis, OR, USA, 2006.
- [67] K. Okuda, K. Yonemoto, and T. Akiyama, “Hybrid optimal trajectory generation using genetic algorithm and sequential quadratic programming,” in *ICCAS-SICE 2009*, Fukuoka, Japan, 18-21 August, 2009, pp. 3233–3238.
- [68] J. Ranta, “Optimal control and flight trajectory optimization applied to evasion analysis,” Licentiate’s Thesis, Helsinki University of Technology, Helsinki, Finland, 2004.
- [69] H. Robbins and S. Monro, “A stochastic approximation method,” *Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951.
- [70] J. E. Gentle, W. Härdle, and Y. Mori, *Handbook of Computational Statistics: Concepts and Methods*. New York: Springer-Verlag, 2004.
- [71] Z. Xu and Y.-H. Dai, “A stochastic approximation frame algorithm with adaptive directions,” *Numerical Mathematics: Theory, Methods and Applications*, vol. 1, no. 4, pp. 460–474, 2008.
- [72] T. L. Lai, “Stochastic approximation,” *Annals of Mathematical Statistics*, vol. 31, no. 2, pp. 391–406, 2003.
- [73] V. S. Borkar, *Stochastic Approximation: A Dynamical Systems Viewpoint*. Cambridge, UK: Cambridge University Press, 2008.
- [74] J. Kiefer and J. Wolfowitz, “Stochastic estimation of the maximum of a regression function,” *Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.
- [75] H. J. Kushner and D. S. Clark, *Stochastic Approximation for Constrained and Unconstrained Systems*, ser. Applied Mathematical Sciences. Berlin: Springer-Verlag, 1978, vol. 26.

-
- [76] J. Theiler and J. Alper, “On the choice of random directions for stochastic approximation algorithms,” *IEEE Transactions on Automatic Control*, vol. 51, no. 3, pp. 476–481, 2006.
- [77] J. C. Spall, “Multivariate stochastic approximation using a simultaneous perturbation gradient approximation,” *IEEE Transactions on Automatic Control*, vol. 37, no. 3, pp. 332–341, 1992.
- [78] H. Kesten, “Accelerated stochastic approximation,” *Annals of Mathematical Statistics*, vol. 29, no. 1, pp. 41–59, 1958.
- [79] B. Delyon and A. Juditsky, “Accelerated stochastic approximation,” *SIAM Journal on Optimization*, vol. 3, no. 4, pp. 868–881, 1993.
- [80] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont: Athena Scientific, 1996.
- [81] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. New York: McMillan, 1998.
- [82] ———, *Adaptive Filter Theory*, 4th ed. Englewood Cliffs: Prentice Hall, 2001.
- [83] D. W. Hutchison and J. C. Spall, “Stopping small-sample stochastic approximation,” in *Proceedings of the 2009 American Control Conference (ACC '09)*, St. Louis, MO, USA, 10-12 June, 2009, pp. 26–31.
- [84] D. W. Hutchison and S. D. Hill, “Simulation optimization of airline delay with constraints,” in *Proceedings of the 2001 Winter Simulation Conference*, B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, Eds., Arlington, VA, USA, 9-12 December, 2001, pp. 1017–1002.
- [85] ———, “Simulation optimization of airline delay with constraints and multiple objectives,” in *Proceedings of the Fourth International Symposium on Uncertainty Modeling and Analysis*, College Park, MD, USA, 21-24 September, 2003, pp. 417–422.
- [86] *SIMMOD Reference Manual*. Sunnyvale, CA: ATAC Corporation, 1995.

- [87] R. Burnett, “Application of stochastic optimization to collision avoidance,” in *Proceedings of the 2004 American Control Conference*, Boston, MA, USA, 30 June - 2 July, 2004, pp. 2789–2794.
- [88] X. Q. Xing and M. Damodaran, “Application of simultaneous perturbation stochastic approximation method for aerodynamic shape design optimization,” *AIAA Journal*, vol. 43, no. 2, pp. 284–294, 2005.
- [89] J. Sobieszczanski-Sobieski and R. T. Haftka, “Multidisciplinary aerospace design optimization: Survey of recent developments,” *Structural and Multidisciplinary Optimization*, vol. 14, no. 1, pp. 1–23, 1997.
- [90] X. Q. Xing and M. Damodaran, “Optimal transonic aerodynamic shape design using simultaneous perturbation stochastic approximation method coupled with global and local optimization methods,” in *21st AIAA Applied Aerodynamics Conference*, Orlando, FL, USA, 23-26 June, 2003.
- [91] —, “Inverse design of transonic airfoils using parallel simultaneous perturbation stochastic approximation,” *Journal of Aircraft*, vol. 42, no. 2, pp. 568–570, 2005.
- [92] X. Wang and M. Damodaran, “Inverse design of transonic airfoils using parallel simulated annealing and CFD,” *AIAA Journal*, vol. 40, no. 4, pp. 791–794, 2002.
- [93] G. Kothandaraman and M. A. Rotea, “Simultaneous-perturbation-stochastic-approximation algorithm for parachute parameter estimation,” *Journal of Aircraft*, vol. 42, no. 5, pp. 1229–1235, 2005.
- [94] N. F. Palumbo, B. E. Reardon, and R. A. Blauwkamp, “Integrated guidance and control for homing missiles,” *Johns Hopkins APL Technical Digest*, vol. 25, no. 2, pp. 121–139, 2004.
- [95] B. E. Reardon, J. M. Lloyd, and R. Y. Perel, “Tuning missile guidance and control algorithms using simultaneous perturbation stochastic approximation,” *Johns Hopkins APL Technical Digest*, vol. 29, no. 1, pp. 85–100, 2010.
- [96] P. Skoglar, “UAV path and sensor planning methods for multiple ground target search and tracking - a literature survey,” Department of Electrical Engineering, Linköping University, Tech. Rep., December 2007.

-
- [97] S. Singh, B.-N. Vo, A. Doucet, and R. Evans, “Stochastic approximation for optimal observer trajectory planning,” in *Proceedings of the 42nd IEEE Conference on Decision and Control, 2003*, vol. 6, Maui, HI, USA, 9-12 December, 2003, pp. 6313–6318.
- [98] G. E. P. Box and K. B. Wilson, “On the experimental attainment of optimum conditions,” *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 13, no. 1, pp. 1–45, 1951.
- [99] R. P. Nicolai, R. Dekker, N. Piersma, and G. J. van Oortmarssen, “Automated response surface methodology for stochastic optimization models with unknown variance,” in *Proceedings of the 2004 Winter Simulation Conference*, R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, Eds., Washington, DC, USA, 5-8 December, 2004, pp. 491–499.
- [100] G. E. P. Box, W. G. Hunter, and J. S. Hunter, *Statistics for Experiment: An Introduction to Design, Data Analysis and Model Building*. New York: John Wiley & Sons, 1978.
- [101] D. C. Montgomery, *Design and Analysis of Experiments: Response Surface Method and Designs*. New York: John Wiley & Sons, 2005.
- [102] M. Jacobsen, “On improving efficiency of flight using optimization,” Ph.D. Thesis, Kungliga Tekniska Högskolan (KTH), Stockholm, Sweden, 2009.
- [103] M. C. Fu, “A tutorial review of techniques for simulation optimization,” in *Proceedings of the 1994 Winter Simulation Conference*, J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, Eds., Lake Buena Vista, FL, USA, 11-14 December, 1994, pp. 149–156.
- [104] J. C. Spall, “Stochastic optimization and the simultaneous perturbation method,” in *Proceedings of the 1999 Winter Simulation Conference*, P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Ewans, Eds., Phoenix, AZ, USA, 5-8 December, 1999, pp. 101–109.
- [105] H. G. Neddermeijer, G. J. van Oortmarssen, N. Piersma, and R. Dekker, “A framework for response surface methodology for simulation optimization,” in

- Proceedings of the 2000 Winter Simulation Conference*, J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, Eds., vol. 1, Orlando, FL, USA, 10-13 December, 2000, pp. 129–136.
- [106] J. P. C. Kleijnen, “Experimental design for sensitivity analysis, optimization, and validation of simulation models,” in *Handbook of Simulation: Principles, Methodology, Advances, Applications and Practice*, J. Banks, Ed. New York: John Wiley & Sons, 1998, pp. 173–223.
- [107] S. S. Joshi, H. D. Sherali, and J. D. Tew, “An enhanced response surface methodology (RSM) algorithm using gradient deflection and second-order search strategies,” *Computers and Operations Research*, vol. 25, no. 7/8, pp. 535–541, 1998.
- [108] M. E. Angun, “Black box simulation optimization: Generalized response surface methodology,” Ph.D. Thesis, Tilburg University, Tilburg, The Netherlands, 2004.
- [109] M. Y. M. Ahmed and N. Qin, “Comparison of response surface and Kriging surrogates in aerodynamic design optimization of hypersonic spiked blunt bodies,” in *13th International Conference on Aerospace Sciences & Aviation Technology (ASAT-13)*, Cairo, Egypt, 26-28 May, 2009.
- [110] R. H. Myers and D. C. Montgomery, *Response Surface Methodology: Process and Product Optimization using Designed Experiments*, 2nd ed. New York: Wiley, 2002.
- [111] A. Cave, S. Nahavandi, and A. Kouzani, “Simulation optimization for process scheduling through simulated annealing,” in *Proceedings of the 2002 Winter Simulation Conference*, E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, Eds., San Diego, CA, US, 8-11 December, 2002, pp. 1909–1913.
- [112] J.-P. Lai, “Surrogate search: A simulation optimization methodology for large-scale systems,” Ph.D. Thesis, University of Pittsburgh, Pittsburgh, PA, USA, 2006.
- [113] S. Kirkpatrick, J. C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
- [114] P. J. M. van Laarhoven and E. H. L. Arts, *Simulated Annealing: Theory and Applications*. Norwell: Kluwer Academic Publishers, 1988.

- [115] E. Arts and J. Korst, *Simulated Annealing and Boltzmann Machines*. New York: John Wiley & Sons, 1989.
- [116] M. H. Alrefaei and S. Andradottir, "A simulated annealing algorithm with constant temperature for discrete stochastic optimization," *Management Science*, vol. 45, no. 5, pp. 748–764, 1999.
- [117] M. H. Alrefaei and A. H. Diabat, "A simulated annealing technique for multi-objective simulation optimization," *Applied Mathematics and Computation*, vol. 215, no. 8, pp. 3029–3035, 2009.
- [118] Global optimization algorithms - theory and application [E Book]. [Online]. Available: <http://www.it-weise.de/projects/book.pdf>
- [119] J. Haddock and J. Mittenthal, "Simulation optimization using simulated annealing," *Computers and Industrial Engineering*, vol. 22, no. 4, pp. 387–395, 1992.
- [120] T. M. Alkhamis, M. A. Ahmed, and V. K. Tuan, "Simulated annealing for discrete optimization with estimation," *European Journal of Operational Research*, vol. 116, no. 3, pp. 530–544, 1999.
- [121] H. Miao and Y. Tian, "Robot path planning in dynamic environments using a simulated annealing based approach," in *Conference on Control, Automation, Robotics and Vision (ICARCV 2008)*, Hanoi, Vietnam, 17-20 December, 2008.
- [122] H. Miao, "A multi-operator based simulated annealing approach for robot navigation in uncertain environments," *International Journal of Computer Science and Security*, vol. 4, no. 1, pp. 50–61, 2010.
- [123] K. Kastella, "Aircraft route optimization using adaptive simulated annealing," in *Proceedings of the 1991 IEEE National Aerospace and Electronics Conference (NAECON 1991)*, Dayton, OH, USA, 20-24 May, 1991, pp. 1123–1129.
- [124] H. Meng and G. Xin, "UAV route planning based on genetic simulated annealing algorithm," in *Proceedings of the 2010 International Conference on Mechatronics and Automation (ICMA)*, Xi'an, China, 4-7 August, 2010, pp. 788–793.

- [125] T. Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. New York: Oxford University Press, 1996.
- [126] T. Bäck, D. B. Fogel, and Z. Michalewics, *Handbook of Evolutionary Computation*, ser. Computational Intelligence Library. Bristol, UK: Oxford University Press in cooperation with the Institute of Physics Publishing, 1997.
- [127] Evolutionary algorithms: Genetic algorithms, evolutionary programming and genetic programming [in “a survey of global optimization methods”]. [Online]. Available: <http://www.cs.sandia.gov/opt/survey/ea.html>
- [128] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press, 1975.
- [129] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. New York: Springer, 2003.
- [130] P. Kochel and U. Nielander, “Kanban optimization by simulation and evolution,” *Production Planning & Control*, vol. 13, no. 8, pp. 725–734, 2002.
- [131] D. L. McWilliams, P. M. Stanfield, and C. D. Geiger, “The parcel hub scheduling problem: A simulation-based solution approach,” *Computers and Industrial Engineering*, vol. 49, no. 3, pp. 393–412, 2005.
- [132] J. Hall, R. Bowden, and J. Usher, “Using evolution strategies and simulation to optimize a pull production system,” *Journal of Materials Processing Technology*, vol. 61, no. 1-2, pp. 47–52, 1996.
- [133] J. Periaux, D. S. Lee, L. F. Gonzalez, and K. Srinivas, “Fast reconstruction of aerodynamic shapes using evolutionary algorithms and virtual Nash strategies in a CFD design environment,” *Journal of Computational and Applied Mathematics*, vol. 232, no. 1, pp. 61–71, 2008.
- [134] Modeling and simulation optimization using evolutionary computation. [Online]. Available: <http://www.cs.ucl.ac.uk/staff/wlangdon/ftp/public/public/papers/06S-SIW-089.pdf>

- [135] S. Mittal and K. Deb, “Three-dimensional offline path planning for UAVs using multiobjective evolutionary algorithms,” in *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC2007)*, Singapore, 25-28 September, 2007, pp. 3195–3202.
- [136] H. Eskandari, L. Rabelo, and M. Mollaghasemi, “Multiobjective simulation optimization using an enhanced genetic algorithm,” in *Proceedings of the 2005 Winter Simulation Conference*, M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, Eds., Orlando, FL, USA, 4-7 December, 2005, pp. 833–841.
- [137] I. K. Nikolos, N. Tsourveloudis, and K. P. Valavanis, “Evolutionary algorithm based 3-D path planner for UAV navigation,” in *Proceedings of the 9th Mediterranean Conference on Control and Automation*, Dubrovnik, Croatia, 27-29 June, 2001.
- [138] I. K. Nikolos, E. S. Zografos, and A. N. Brintaki, “UAV path planning using evolutionary algorithms,” in *Innovations in Intelligent Machines - 1*, ser. Studies in Computational Intelligence, J. S. Chahl, L. C. Jain, A. Mizautani, and M. Sato-Ilic, Eds. New York: Springer-Verlag, 2007, vol. 70.
- [139] G. Sanders and T. Ray, “Optimal offline path planning of a fixed wing unmanned aerial vehicle (UAV) using an evolutionary algorithm,” in *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC 2007)*, 25-28 September, 2007, pp. 4410–4416.
- [140] A. J. Pohl and G. B. Lamont, “Multi-objective UAV mission planning using evolutionary computation,” in *Proceedings of the 2008 Winter Simulation Conference*, S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, and J. W. Fowler, Eds., Austin, TX, USA, 7-10 December, 2008, pp. 1268–1279.
- [141] D. Rathburn, S. Kragelund, A. Pongpunwattana, and B. Capozzi, “An evolution based path planning algorithm for autonomous motion of a UAV through uncertain environments,” in *Proceedings of the 21st Digital Avionics Systems Conference (DASC’02)*, vol. 2, Irvine, CA, USA, 27-31 October, 2002, pp. 8D2–1–8D2–12.
- [142] A. N. Kostaras, I. K. Nikolos, N. C. Tsourveloudis, and K. P. Valavanis, “Evolutionary algorithm based on-line path planner for UAV navigation,” in

- Proceedings of the 10th Mediterranean Conference on Control and Automation (MED2002)*, Lisbon, Portugal, 9-12 July, 2002.
- [143] D. Jia and J. Vagners, “Parallel evolutionary algorithms for UAV path planning,” in *Proceedings of the AIAA 1st Intelligent Systems Technical Conference*, Chicago, IL, USA, 20-22 September, 2004, pp. 1499–1506.
- [144] A. Pongpunwattana and R. Rysdyk, “Evolutino-based dynamic path planning for autonomous vehicles,” in *Innovations in Intelligent Machines - 1*, ser. Studies in Computational Intelligence, J. S. Chahl, L. C. Jain, A. Mizautani, and M. Sato-Ilic, Eds. New York: Springer-Verlag, 2007, vol. 70.
- [145] E. Besada-Portas, L. de la Torre, J. M. de la Cruz, and B. de Andrès-Toro, “Evolutionary trajectory planner for multiple UAVs in realistic scenarios,” *IEEE Transactions on Robotics*, vol. 26, no. 4, pp. 619–634, 2010.
- [146] L. F. Gonzalez, R. J. Whitney, K. Srinivas, K. C. Wong, and J. Periaux, “A framework for multidisciplinary design and optimisation in aeronautics,” in *The 11th Australian International Aerospace Congress (AIAC-11)*, Melbourne, Australia, 13-17 March, 2005.
- [147] L. F. Gonzalez, E. J. Whitney, J. Periaux, M. Sefrioui, and K. Srinivas, “A robust evolutionary technique for inverse aerodynamic design,” in *Proceedings of the 4th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2004)*, P. Neittaanmaki, T. Rossi, S. Korotov, E. Onate, J. Periaux, and D. Knorzner, Eds., vol. 2, Jyvaskyla, Finland, 24-28 July, 2004.
- [148] M. Sefrioui and J. Periaux, “A hierarchical genetic algorithm using multiple models for optimization,” in *Parallel Problem Solving from Nature*, ser. PPSN VI, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, Eds. Springer, 2000, pp. 879–888.
- [149] E. Cantu-Paz, *Efficient and Accurate Parallel Genetic Algorithms*. Norwell, MA: Kluwer Academic Publishers, 2000.
- [150] C. A. C. Coello, D. A. V. Veldhuizen, and G. B. Lamont, *Evolutionary Algorithms*

- for Solving Multi-Objective Problems*. New York: Kluwer Academic Publishers, 2002.
- [151] D. A. V. Veldhuizen, J. B. Zydallis, and G. B. Lamont, “Considerations in engineering parallel multiobjective evolutionary algorithms,” *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 144–173, 2003.
- [152] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading: Addison-Wesley, 1989.
- [153] N. Hansen and A. Ostermeier, “Completely derandomized self-adaptation in evolution strategies,” *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.
- [154] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, ser. Artificial Intelligence. Berlin Heidelberg: Springer-Verlag, 1992.
- [155] J. Wakunda and J. A. Zell, “Median-selection for parallel steady-state evolution strategies,” in *Parallel Problem Solving from Nature*, ser. PPSN VI, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, Eds. Berlin: Springer, 2000, pp. 405–414.
- [156] “AeroSim Blockset, aeronautical simulation blockset, ver. 1.2,” 2005.
- [157] I. McManus, R. Clothier, and R. Walker, “Highly autonomous uav mission planning and piloting for civilian airspace operations,” in *Proceedings of the 11th Australian International Aerospace Congress (AIAC-11), 1st Australian Unmanned Air Vehicles Conference*, Melbourne, Australia, 13-17 March, 2005.
- [158] M. Kayton, “The navigation equations,” in *Avionics Navigations Systems*, M. and W. Fried Kayton, Eds. New York: John Wiley & Sons, 1997, pp. 21–54.
- [159] P. C. S. Eng, L. Mejias, R. A. Walker, and D. L. Fitzgerald, “Simulation of a fixed-wing UAV forced landing with dynamic path planning,” in *The 2007 Australasian Conference on Robotics & Automation*, Brisbane, Australia, 10-12 December, 2007.
- [160] R. C. Nelson, *Flight Stability and Automatic Control*, 2nd ed. Boston: WCB/McGraw-Hill, 1998.

- [161] R. W. Sinnott, "Virtues of the Haversine," *Sky and Telescope*, vol. 68, no. 2, p. 159, 1984 1984.
- [162] C. A. C. Coello, "A comprehensive survey of evolutionary-based multiobjective optimization techniques," *Knowledge and Information Systems*, vol. 1, no. 3, pp. 269–308, 1999.
- [163] M. F. Oudijk, "Optimization of CVT control: For hybrid and conventional drive lines," Masters Thesis, Eindhoven University of Technology; University of California, Davis, Eindhoven, The Netherlands; Davis, CA, USA, 2005.
- [164] *Battery*, ser. MATLAB 7.5 Documentation. The MathWorks, Inc.
- [165] Modeling and simulation of hybrid electric vehicle (HEV). [Online]. Available: <http://www.mathworks.com/mason/tag/proxy.html?dataid=12113&fileid=57988&product=PS&ei=cUOeTPTXK4WsvgOvs5CtDQ&usg=AFQjCNFvo87b0FSb703CwlcZ-M5TxLbI1Q>
- [166] Air Thunder 5000mAh 6-cell lithium-polymer (Li-Po) battery pack. [Online]. Available: <http://www.airthunder.com>
- [167] A. E. Dortland, "Building a dynamometer test-stand," Eindhoven, The Netherlands: Eindhoven University of Technology; Davis, CA, USA: University of California, Davis, Masters Internship Report DCT-2007-134, 22 October 2007.
- [168] C. Brockbank and D. Burt, "Developments in full-toroidal traction drive infinitely and continuously variable transmissions," in *14th Asia Pacific Automotive Engineering Conference (APAC-14)*, Los Angeles, CA, USA, 5-8 August, 2007.
- [169] W. Kriegler, A. Zrim, and G.-J. van Spijk, "IC-Engines and CVTs in passenger cars: A system integration approach," in *IMEchE International Seminar S540: Advanced Vehicle Transmissions and Powertrain Management*, London, UK, 25-26 September, 1997.

Appendix A

Aircraft Configuration for the Aerosonde UAV

AIRCRAFT CONFIGURATION FOR AN AEROSONDE UAV*Sample model from AeroSim Library**Copyright 2002 Unmanned Dynamics, LLC**Revision: 1.0 (13 May, 2002)***Table A.1:** Aircraft configuration for an Aerosonde UAV

Parameter Name	Identifier	Value	Unit
<i>AERODYNAMICS</i>			
Aerodynamic force application point	rAC		
- x-coordinate		0.1425	m
- y-coordinate		0	m
- z-coordinate		0	m
<i>Aerodynamic Parameter Bounds</i>			
Airspeed bounds	VaBnd		
- Upper bound		50	m/s
- Lower bound		15	m/s
Sideslip angle bounds	BetaBnd		
- Upper bound		0.5	rad
- Lower bound		-0.5	rad
Angle of attack bounds	Alpha Bnd		
- Upper bound		0.3	rad
- Lower bound		-0.1	rad
<i>Aerodynamic Reference Parameters</i>			
Mean aerodynamic chord	MAC	0.189941	m
Wing span	b	2.8956	m
Wing area	S	0.55	m ²
<i>Lift Coefficients</i>			
Zero-alpha lift	CL0	0.23	-
Alpha derivative	CLa	5.6106	per radian
Lift control (flap) derivative	CLdf	0.74	per radian
Pitch control (elevator) derivative	CLde	0.13	per radian
Alpha-dot derivative	CLalphadot	1.9724	per radian
Pitch rate derivative	CLq	7.9543	per radian
Mach number derivative	CLM	0	per radian
<i>Drag Coefficients</i>			
Lift at minimum drag	CLmind	0.23	-
Minimum drag	CDmin	0.0434	-
Lift control (flap) derivative	CDdf	0.1467	per radian
Pitch control (elevator) derivative	CDde	0.0135	per radian
Roll control (aileron) derivative	CDda	0.0302	per radian
Yaw control (rudder) derivative	CDdr	0.0303	per radian
Continued on next page			

Table A.1 – continued from previous page

Parameter Name	Identifier	Value	Unit
Mach number derivative	CDM	0	per radian
Oswald's coefficient	osw	0.75	-
<i>Side Force Coefficients</i>			
Sideslip derivative	CYbeta	-0.83	per radian
Roll control derivative	CYda	-0.075	per radian
Yaw control derivative	CYdr	0.1914	per radian
Roll rate derivative	CYp	0	per radian
Yaw rate derivative	CYr	0	per radian
<i>Pitch Moment Coefficients</i>			
Zero-alpha pitch	Cm0	0.135	-
Alpha derivative	Cma	-2.7397	per radian
Lift control derivative	Cmdf	0.0467	per radian
Pitch control derivative	Cmde	-0.9918	per radian
Alpha.dot derivative	Cmalphadot	-10.3796	per radian
Pitch rate derivative	Cmq	-38.2067	per radian
Mach number derivative	CmM	0	per radian
<i>Roll Moment Coefficients</i>			
Sideslip derivative	Clbeta	-0.13	per radian
Roll control derivative	Cl da	-0.1694	per radian
Yaw control derivative	Cl dr	0.0024	per radian
Roll rate derivative	Cl p	-0.5051	per radian
Yaw rate derivative	Cl r	0.2519	per radian
<i>Yaw Moment Coefficients</i>			
Sideslip derivative	Cn beta	0.0726	per radian
Roll control derivative	Cn da	0.0108	per radian
Yaw control derivative	Cn dr	-0.0693	per radian
Roll rate derivative	Cn p	-0.069	per radian
Yaw rate derivative	Cn r	-0.0946	per radian
PROPELLER			
Propulsion force application point	rHub		
- x-coordinate		0	m
- y-coordinate		0	m
- z-coordinate		0	m
Advance ratio	J	(See Table A.2)	-
Coefficient of thrust	CT	(See Table A.2)	-
Coefficient of power	CP	(See Table A.2)	-
Propeller radius	Rprop	0.254	m
Propeller moment of inertia	Jprop	0.002	kg·m ²
ENGINE			
Engine RPM	RPM	(See Tables A.3 & A.4)	rot per min
Manifold pressure	MAP	(See Tables A.3 & A.4)	kPa
Sea-level fuel flow	FuelFlow	(See Table A.3)	g/hr
Continued on next page			

Table A.1 – continued from previous page

Parameter Name	Identifier	Value	Unit
Sea-level power	Power	(see Table A.4)	W
Sea-level pressure (for above data)	pSL	102300	Pa
Sea-level temperature (for above data)	TSL	291.15	K
Engine shaft moment of inertia	Jeng	0.0001	kg·m ²
<i>INERTIA</i>			
Empty aircraft mass (zero-fuel)	mempty	8.5	kg
Gross aircraft mass (full fuel tank)	mgross	13.5	kg
Empty CG location	Cgempty		
- x-coordinate		0.156	m
- y-coordinate		0	m
- z-coordinate		0.079	m
Gross CG location	Cggross		
- x-coordinate		0.159	m
- y-coordinate		0	m
- z-coordinate		0.09	m
Empty moments of inertia	Jempty		
	Jx	0.7795	kg·m ²
	Jy	1.122	kg·m ²
	Jz	1.752	kg·m ²
	Jxz	0.1211	kg·m ²
Gross moments of inertia	Jgross		
	Jx	0.8244	kg·m ²
	Jy	1.135	kg·m ²
	Jz	1.759	kg·m ²
	Jxz	0.1204	kg·m ²

J	CT	CP
-1	0.0492	0.0199
0	0.0286	0.0207
0.1	0.0266	0.0191
0.2	0.0232	0.0169
0.3	0.0343	0.0217
0.35	0.034	0.0223
0.4	0.0372	0.0254
0.45	0.0314	0.0235
0.5	0.0254	0.0212
0.6	0.0117	0.0146
0.7	-0.005	0.0038
0.8	-0.0156	-0.005
0.9	-0.0203	-0.0097
1	-0.0295	-0.018
1.2	-0.04	-0.0273
2	-0.1115	-0.0737

Table A.2: Look-up table for propeller coefficient of thrust and coefficient of power

RPM	MAP								
	60	70	80	90	92	94	96	98	100
1500	31	32	46	53	55	57	65	53	82
2100	40	44	54	69	74	80	92	103	111
2800	50	63	69	92	95	98	126	145	153
3500	66	75	87	110	117	127	150	175	190
4500	83	98	115	143	148	162	191	232	246
5100	93	102	130	159	167	182	208	260	310
5500	100	118	137	169	178	190	232	287	313
6000	104	126	151	184	191	206	253	326	337
7000	123	144	174	210	217	244	321	400	408

Table A.3: Look-up table for Engine fuel flow

RPM	MAP								
	60	70	80	90	92	94	96	98	100
1500	18.85	47.12	65.97	67.54	69.12	67.54	67.54	69.12	86.39
2100	59.38	98.96	127.55	149.54	151.74	160.54	178.13	200.12	224.31
2800	93.83	149.54	187.66	237.50	249.23	255.10	307.88	366.52	398.77
3500	109.96	161.27	245.57	307.88	326.20	351.86	421.50	591.14	531.45
4500	164.93	245.04	339.29	438.25	447.68	494.80	565.49	673.87	772.83
5100	181.58	245.67	389.87	496.69	528.73	571.46	662.25	822.47	993.37
5500	184.31	293.74	403.17	535.64	570.20	622.04	748.75	956.09	1059.80
6000	163.36	276.46	420.97	565.49	609.47	691.15	860.80	1131.00	1193.80
7000	124.62	249.23	417.83	586.43	645.07	762.36	996.93	1246.20	1429.40

Table A.4: Look-up table for Engine power

Appendix B

UAV Simulation Model: Flight Planner

Module MATLAB Code

```

function Navigate = NavigateWaypoints(Argument)

% This function performs waypoint navigation

% Load waypoint list
S = load ('WPTTable.mat');
WPTTable = S.WPTTable;

% Number of waypoints is table
SizeOfTable = size(WPTTable);
NoOfWPT = SizeOfTable(1);

% Allocating inputs
CurrentWaypoint = Argument(1);
CurrentLatitude = Argument(2);
CurrentLongitude = Argument(3);
CurrentAltitude = Argument(4);
CurrentAirspeed = Argument(5);

% Initialise
if (CurrentWaypoint == 0)
    CurrentWaypoint = 1;
end
EndOfMission = 0;

% Get current waypoint coordinates
WPTLat = WPTTable(CurrentWaypoint, 1);
WPTLon = WPTTable(CurrentWaypoint, 2);
WPTAlt = WPTTable(CurrentWaypoint, 3);
WPTAirspeed = WPTTable(CurrentWaypoint, 4);
[Bearing, Distance] = CalculateGC(CurrentLatitude, CurrentLongitude, WPTLat, WPTLon);
Waypoint = CurrentWaypoint;

XTE = 0;

% A waypoint is captured when the aircraft comes with 20m of it
if (Distance > 20)
    if (CurrentWaypoint ~= 1) % Not the first waypoint in table
        LastWPTLat = WPTTable(CurrentWaypoint-1, 1);
        LastWPTLon = WPTTable(CurrentWaypoint-1, 2);
        LastWPTAlt = WPTTable(CurrentWaypoint-1, 3);
        [RequiredTrack, TrackDistance] = ...
            CalculateGC(LastWPTLat, LastWPTLon, WPTLat, WPTLon);
        TrackError = RequiredTrack - Bearing;
        XTE = Distance * sin(TrackError);
        EndOfMission = 0;
    else
        LastWPTLat = 26.5808*pi/180; % Latitude of starting point
        LastWPTLon = 151.8411*pi/180; % Longitude of starting point
        LastWPTAlt = 850; % Altitude of starting point
        [RequiredTrack, TrackDistance] = ...
            CalculateGC(LastWPTLat, LastWPTLon, WPTLat, WPTLon);
        TrackError = RequiredTrack - Bearing;
        XTE = Distance * sin(TrackError);
        EndOfMission = 0;
    end
else % (Distance < 20)
    if (CurrentWaypoint == NoOfWPT) % Last waypoint in table
        if (Distance < 5)
            LastWPTLat = WPTTable(CurrentWaypoint, 1);
            LastWPTLon = WPTTable(CurrentWaypoint, 2);
            LastWPTAlt = WPTTable(CurrentWaypoint, 3);
            [RequiredTrack, TrackDistance] = ...
                CalculateGC(LastWPTLat, LastWPTLon, WPTLat, WPTLon);
            TrackError = RequiredTrack - Bearing;
            XTE = Distance * sin(TrackError);
            EndOfMission = 1;
        end
    else % Other waypoints in table
        WPTLat = WPTTable(CurrentWaypoint+1, 1);
        WPTLon = WPTTable(CurrentWaypoint+1, 2);
        WPTAlt = WPTTable(CurrentWaypoint+1, 3);
        LastWPTLat = WPTTable(CurrentWaypoint, 1);

```

```
        LastWPTLon = WPTTable(CurrentWaypoint, 2);
        LastWPTAlt = WPTTable(CurrentWaypoint, 3);
        [RequiredTrack, TrackDistance] = ...
            CalculateGC(LastWPTLat, LastWPTLon, WPTLat, WPTLon);
        TrackError = RequiredTrack - Bearing;
        XTE = Distance * sin(TrackError);
        Waypoint = CurrentWaypoint + 1;
        EndOfMission = 0;
    end
end

% Target airspeed when reached target altitude at descent
if ((Waypoint == 3) || (Waypoint == 8) || (Waypoint == 11) ...
    || (Waypoint == 16) || (Waypoint == 19) || (Waypoint == 24) ...
    || (Waypoint == 27) || (Waypoint == 32) || (Waypoint == 35) ...
    || (Waypoint == 40))
    if (CurrentAltitude > WPTAlt + 10)
        TargetAirspeed = WPTAirspeed;
    elseif (CurrentAirspeed > 21)
        TargetAirspeed = CurrentAirspeed - 2;
    else
        TargetAirspeed = 20;
    end
else
    TargetAirspeed = WPTAirspeed;
end

% Output of function
Navigate = [Bearing, WPTAlt, TargetAirspeed, XTE, Waypoint, EndOfMission];
```


Appendix C

HAPMOEA Code for One-Objective MWO: Excerpts

C.1 optimisation.parameters: An Example

```

PARAMETERS FOR AEROSIM FLIGHT MISSION OPTIMISATION
1      Number of runs
1 6    Optimisation will be terminated after 6 hours
1      One objective - to minimise fuel consumption
4      Number of waypoints to be optimised
8      Total number of waypoints in WPTTable
2      Number of coordinates for each waypoint (lat & lon)
0.463596392 0.463596392 0.463596392 0.463596392    Lower bounds for WPT lat (rad)
0.46383268 0.46383268 0.46383268 0.46383268    Upper bounds for WPT lat (rad)
2.650312431 2.650312431 2.650312431 2.650312431  Lower bounds for WPT lon (rad)
2.650750363 2.650750363 2.650750363 2.650750363  Upper bounds for WPT lon (rad)
500.0  Live update every 500 seconds
0      Forced synchronous evaluation is OFF (1=YES)
2.00   Migration operation will be after two generations
0.3    Migration ratio set 30%
1      One layer of hierarchical topology is set
- LEVEL ONE -
141    First layer ID is set as 141 = waypoint coordinates
10     Population size
2      Parents of recombination
1      Intermediate Recombination is ON (1=YES)
12     Buffer length
2.0    Tournament-in-Buffer ratio
0.05   Initial Mutation Size

```

C.2 HierarEA.cpp

```

////////////////////////////////////
//                                     //
//                               Main Program                               //
//                                     //
////////////////////////////////////

int
main(int argc, char* argv[])
{
    // Set the priority of the driver executable down.
    int priRet = setpriority(PRIO_PROCESS, 0, +5);
    assert(priRet == 0);

    //
    // Check command line.
    //

    if (argc > 3)
    {
        cerr << "HEAAEROFOIL Usage:" << endl;
        cerr << "\theaerofoil - Uses default input file 'optimisation.parameters'"
" and writes statistics to 'output.statistics'." << endl;
        cerr << "\theaerofoil parameter_file output_file - Reads inputs from"
" 'parameter_file' and writes statistics to 'output_file'." << endl;
        cerr << "\theaerofoil restart - Restarts the run from the point"
" at which it stopped." << endl;
        return 1;
    }

    //
    // Set up optimisation.parameters
    //
    int NumWptsOpt = 4; // Enter a number between 1 to 4
    GenerateOptiParaFile(NumWptsOpt);

    //

```

```

// Determine the type of run.
//

char* inputFile = "optimisation.parameters";
char* outputFile = "output.statistics";

if (argc == 3)
{
    inputFile = argv[1];
    outputFile = argv[2];
}

if (argc == 2)
{
    if (strcasemp(argv[1], "reset") == 0)
{
int retVal = system("rm -f heaerofoil-restart-file-alg*");
if (retVal == 0)
    cout << "All files successfully deleted." << endl;
else
    cout << "Problem calling \"rm -f heaerofoil-restart-file-alg*\"." <<
        endl;
return 0;
}
    else
inputFile = argv[1];
}

if (argc == 3)
{
    inputFile = argv[1];
    outputFile = argv[2];
}

//
// Open the parameters file and read in the data.
//
ifstream fin(inputFile);
if (!fin)
    throw xMain("Couldn't open the input parameters file.");

// Independent parameters.
fin.ignore(255, '\n'); // First line is an identifier.

cout << "*****" << endl;
cout << "*** Hierarchical AeroSim Simulation ***" << endl;
cout << "*****" << endl << endl;

int totalRuns;
fin >> totalRuns; // Number of runs to do
fin.ignore(255, '\n');
cout << "Number of Runs : " << totalRuns << endl;

int stopOption;
double stopNums;
double TerminationCond;
fin >> stopOption >> stopNums; // Maximum time.
fin.ignore(255, '\n');
if ( stopOption == 1 )
{
    TerminationCond = stopNums * 3600.0;
    cout << "Stopping by Time (Hours) For Each Run : " << stopNums << endl;
}
else if ( stopOption == 2 )
{
    TerminationCond = stopNums;
    cout << "Stopping by Function Evaluation For Each Run : " << stopNums << endl;
}
else
{
    TerminationCond = stopNums;
    cout << "Stopping by Pre-defined value For Each Run : " << stopNums << endl;
}
}

```

```

int NumObjectives;
fin >> NumObjectives; // Number of objectives.
fin.ignore(255, '\n');
cout << "Number of Objective : " << NumObjectives << endl;

int numWpt;
fin >> numWpt; // Number of waypoints to be optimised.
fin.ignore(255, '\n');
if (numWpt < 1)
    throw xMain("Must have at least one waypoint.");
cout << "Number of Waypoints to be Optimised : " << numWpt << endl;

int totalNumWpt;
fin >> totalNumWpt; // Total number of waypoints in WPTTable.
fin.ignore(255, '\n');
if (totalNumWpt < 1)
    throw xMain("Must have at least one waypoint in WPTTable.");
cout << "Total Number of Waypoints in WPTTable : " << totalNumWpt << endl;

int numCoord;
fin >> numCoord; // Number of coordinates for each waypoint -- (lat & lon)
fin.ignore(255, '\n');
if (numCoord != 2)
    throw xMain("Must have two coordinates.");
cout << "Number of coordinates (lat & lon) : " << numCoord << endl;

double lbLat;
double ubLat;
vector<double> lowerBoundLat(numWpt);
vector<double> upperBoundLat(numWpt);
// Lower bounds for latitude of waypoints (rad).
for (int i = 0; i < numWpt; ++i)
{
    fin >> lbLat;
    lowerBoundLat[i] = lbLat;
}
fin.ignore(255, '\n');
// Upper bounds for latitude of waypoints (rad).
for (int i = 0; i < numWpt; ++i)
{
    fin >> ubLat;
    upperBoundLat[i] = ubLat;
}
fin.ignore(255, '\n');

double lbLon;
double ubLon;
vector<double> lowerBoundLon(numWpt);
vector<double> upperBoundLon(numWpt);
// Lower bounds for longitude of waypoints (rad).
for (int i = 0; i < numWpt; ++i)
{
    fin >> lbLon;
    lowerBoundLon[i] = lbLon;
}
fin.ignore(255, '\n');
// Upper bounds for longitude of waypoints (rad).
for (int i = 0; i < numWpt; ++i)
{
    fin >> ubLon;
    upperBoundLon[i] = ubLon;
}
fin.ignore(255, '\n');

double liveUpdates;
fin >> liveUpdates; fin.ignore(255, '\n'); // Live update interval (0 = None).
int forceSync;
fin >> forceSync; fin.ignore(255, '\n'); // Force synchronous behaviour (1 = Yes).

//
// Hierarchical parameters
//

```

```

double migrationTime;
// Number of popsize evaluations before migration.
fin >> migrationTime; fin.ignore(255, '\n');
cout << "* PopSize Evaluations Before Migration : " << migrationTime << endl;
if (migrationTime < 0.0)
    throw xMain("Migration time must be >= 0.0.");
// Multiplier of popsize individuals to migrate.
double migrationSize;
fin >> migrationSize; fin.ignore(255, '\n');
cout << "* PopSize Individuals to Migrate : " << migrationSize << endl;
if (migrationSize < 0.0 || migrationSize > 1.0)
    throw xMain("Migration size must be between 0.0 and 1.0 inclusive.");
int levels;
fin >> levels; fin.ignore(255, '\n'); // Number hierarchical layers.
cout << "Number of Hierarchical levels : " << levels << endl;
if (levels < 1)
    throw xMain("Number of hierarchical levels must be >= 1");

// Parameters by layer: Layer 0 = top, layer 1 = 2nd, etc.
vector<int> SidePoints(levels), mu(levels), parents(levels);
vector<int> recoType(levels), buffLength(levels);
vector<double> tournRatio(levels), delta0(levels);

for (int i = 0; i < levels; ++i)
{
    fin.ignore(255, '\n'); // Separator line.
    fin >> SidePoints[i]; fin.ignore(255, '\n'); // Aerofoil side points
    fin >> mu[i]; fin.ignore(255, '\n'); // Size of mu.
    fin >> parents[i]; fin.ignore(255, '\n'); // Parents in recombination.
    fin >> recoType[i]; fin.ignore(255, '\n'); // Recombination type, 1 = intermediate.
    fin >> buffLength[i]; fin.ignore(255, '\n'); // Buffer length.
    fin >> tournRatio[i]; fin.ignore(255, '\n'); // Tournament-in-buffer ratio.
    fin >> delta0[i]; fin.ignore(255, '\n'); // Initial mutation size.
}
fin.close();

cout << "Force Synchronous:\t\t" << (forceSync == 1 ? "Yes" : "No") << endl;

for (int i = 0; i < levels; ++i)
{
    cout << "Layer " << i << ":" << endl;
    cout << "\tSide Points for Solver:\t\t" << SidePoints[i] << endl;
    cout << "\tPopulation Size: \t\t" << mu[i] << endl;
    cout << "\tParents in Recombination:\t" << parents[i] << endl;
    cout << "\tRecombination Type:\t\t" << (recoType[i] == 1 ? "Intermediate" : "Discrete") << endl;
    cout << "\tBuffer Length: \t\t\t" << buffLength[i] << endl;
    cout << "\tTournament-in-Buffer Ratio:\t" << tournRatio[i] << endl;
    cout << "\tInitial Mutation Size:\t\t" << delta0[i] << endl;
}

//
// Run specified number of times.
//
for (int run = 0; run < totalRuns; ++run)
{
    cout << "*** RUN " << (run + 1) << " OF " << totalRuns << " ***" << endl;

    // Start the clock.
    time_t startTime = time(NULL);

    //
    // Establish solvers.
    //

    cout << "Starting solvers:" << endl;
    vector<AeroSimSolver*> solvers;
    for (int i = 0, numThisLevel = 1; i < levels; ++i, numThisLevel *= 2)
{
    cout << "\n\tLevel " << i << ": " << flush;
    for (int j = 0; j < numThisLevel; ++j)
    {
        int minBufferSize = 0;
        if (levels > 1)

```

```

{
    if (i == 0) // Top layer.
        minBufferSize = (int) (2.0*migrationSize*mu[i+1]);
    else if (i == levels - 1) // Bottom layer.
        minBufferSize = (int) (migrationSize*mu[i-1]);
    else // Middle layers.
        minBufferSize = (int) (2.0*migrationSize*mu[i+1] + migrationSize*mu[i-1]);
}

// Must be at least equal to maximum processors.
if (minBufferSize < MaximumTasks)
minBufferSize = MaximumTasks;

if (forceSync == 1) // Must be at least equal to population size.
if (minBufferSize < mu[i])
minBufferSize = mu[i];

// Establish.
solvers.push_back(new AeroSimSolver(SidePoints[i],
NumObjectives,
numWpt,
numCoord,
totalNumWpt,
minBufferSize));
}
}

cout << endl;
//
// Establish the algorithms.
//
vector<char*> fileName;

//Fix alpha

const int N = numWpt*numCoord; // Number of dimensions (waypoint coordinates only).
cout << "N = " << N << endl;
vector<double> lowerBounds(N), upperBounds(N);
double maxSpan = -1.0;

// Setting up lower and upper bound vectors
// Storing lower and upper bounds for latitude
for (int i = 0; i < numWpt; ++i)
{
    lowerBounds[i] = lowerBoundLat[i];
    upperBounds[i] = upperBoundLat[i];

    if (upperBounds[i] - lowerBounds[i] > maxSpan)
        maxSpan = upperBounds[i] - lowerBounds[i];
}

// Storing lower and upper bounds for longitude
for (int i = 0; i < numWpt; ++i)
{
    lowerBounds[i + numWpt] = lowerBoundLon[i];
    upperBounds[i + numWpt] = upperBoundLon[i];

    if (upperBounds[i] - lowerBounds[i] > maxSpan)
        maxSpan = upperBounds[i] - lowerBounds[i];
}

for (int i = 0; i < lowerBounds.size(); ++i)
cout << lowerBounds [i];

vector<MAMDES*> algs;
cout << "Starting algorithms:" << endl;
for (int i = 0, numThisLevel = 1; i < levels; ++i, numThisLevel *= 2)
{
    for (int j = 0; j < numThisLevel; ++j)
    {
        cout << "\tLevel " << i << ": " << endl;

        const int solverRefNum = algs.size();

```

```

    solvers[solverRefNum]->FreeWorkers(); // Clear jobs.
    // Actual tournament size
    const int tourneySize = (int)((double) buffLength[i] / tournRatio[i]);

    char* newName = new char[80];
    sprintf(newName, "heaerofoil-restart-file-alg%d.dat", solverRefNum);
    fileName.push_back(newName);
    try
    {
        algs.push_back(new MAMDES(solvers[solverRefNum],
            fileName[solverRefNum]));
    }
    catch(MAMDESException)
    {
        algs.push_back(new MAMDES(solvers[solverRefNum], // Solver for this algorithm.
            mu[i], // Population size.
            N, // Number of variables.
            lowerBounds, // { Problem }
            upperBounds, // { bounds. }
            parents[i], // Parents in recombination.
            buffLength[i], // Buffer length.
            tourneySize, // Tournament in buffer size.
            delta0[i] // Starting mutation size.
        )
    );
};

algs[solverRefNum]->WriteFile(fileName[solverRefNum]);
}

algs[solverRefNum]->SetMaxFunctionEvaluations(65000); // Arbitrarily high.
algs[solverRefNum]->SetRecombineType(recoType[i] == 1 ? MAMDES::INTERMEDIATE : MAMDES::DISCRETE);
algs[solverRefNum]->SetMaximumVariance(maxSpan / 4.0);
algs[solverRefNum]->SetMinimumVariance(1e-80);
algs[solverRefNum]->SetNoiseSampleInterval(25);
algs[solverRefNum]->ForceSynchronous(forceSync == 1 ? true : false);
algs[solverRefNum]->SetReplacementStrategy(MAMDES::REPLACE_WORST_ALWAYS);

algs[solverRefNum]->SetDDMRefinementParameter(8);
algs[solverRefNum]->SetDDMMaximumEvaluations(2000);
algs[solverRefNum]->SetDDMPowerTerm(0.5);
algs[solverRefNum]->SetDDMProbabilityBounds(0.25,0.75);
algs[solverRefNum]->SetDDMLearningPhaseAccepts(25);
algs[solverRefNum]->SetDDMLearningPhaseMaxTrials(500);
algs[solverRefNum]->UseDDM(true);
}
}

cout << "Proceeding to free workers criteria " << endl;

for (int i = 0; i < (int) solvers.size(); ++i)
solvers[i]->FreeWorkers();

cout << "Proceeding to Reset the migration criteria " << endl;
// Reset the migration criteria.
MigrationCriteria(algs, migrationTime, true);

//
// Run the algorithm.
//
cout << "Running:" << endl;
time_t currentTime = time(NULL);
double elapsed = difftime(currentTime, startTime);

double numVar = numCoord * numWpt;
UpdateProgressFiles(algs, fileName, levels,
    liveUpdates, elapsed,
    numVar, false, true); // Reset the files.
int lastHeadEvals = algs[0]->GetFunctionEvaluations();
double TerminationTaget;

```

```

        if ( stopOption == 1 )
TerminationTaget = elapsed;
        else if ( stopOption == 2 )
TerminationTaget = algs[0]->GetFunctionEvaluations();
        else
TerminationTaget = algs[0]->GetElite().GetFitness()[0];

        while (TerminationTaget < TerminationCond)
{
time_t currentTime = time(NULL);
elapsed = difftime(currentTime, startTime);
Evolve(algs);
if (MigrationCriteria(algs, migrationTime, false))
{
Migrate(algs, solvers, levels, migrationSize);
}

if ( stopOption == 1 )
TerminationTaget = elapsed;
else if ( stopOption == 2 )
TerminationTaget = algs[0]->GetFunctionEvaluations();
else
TerminationTaget = algs[0]->GetElite().GetFitness()[0];

//
// User update.
//
if (algs[0]->GetFunctionEvaluations() - lastHeadEvals > 10)
{
lastHeadEvals = algs[0]->GetFunctionEvaluations();
cout << lastHeadEvals << " Function Evaluations done by Head Node.";
cout << " Elapsed Time: " << elapsed << " seconds." << endl;

solvers[0]->ShowLoadDistribution();
solvers[0]->ResetWorkerStats();
cout << endl;
}
if (liveUpdates > 0.0)
UpdateProgressFiles(algs, fileName, levels, liveUpdates, elapsed, numVar);
}

UpdateProgressFiles(algs, fileName, levels, liveUpdates, elapsed, numVar, true);
// Write final point.

currentTime = time(NULL);
elapsed = difftime(currentTime, startTime);
cout << " Elapsed Time: " << elapsed << " seconds.";

// Clean up.
for (int i = 0; i < (int) fileName.size(); ++i)
{
delete [] (fileName[i]);
fileName[i] = NULL;
}

//
// Close algorithms and solvers.
//

for (int i = 0; i < (int) solvers.size(); ++i)
delete algs[i];
algs.resize(0);
cout << endl;

for (int i = 0; i < (int) solvers.size(); ++i)
delete solvers[i];
solvers.resize(0);
cout << endl;

} // End loop over separate runs.

//

```



```

// Done.
//

return 0;
}

```

C.3 Analyser.cpp

```

//
// Get back the candidate waypoint coordinates in vector form
// - separate into lat and lon (rad) [only lat and lon coordinates are optimised]
//
vector<double> lat(numWaypoints), lon(numWaypoints);
cout << "The waypoints to be evaluated are : " << endl;
for (int i = 0; i < numWaypoints; ++i)
{
    int objPos = i;
    lat[i] = recObj[objPos];
    objPos = objPos + numWaypoints; // move to the next position
    lon[i] = recObj[objPos];
    cout.precision(9);
    cout << "Latitude " << i + 1 << " : " << lat[i] << endl;
    cout << "Longitude " << i + 1 << " : " << lon[i] << endl;
}

//
// Obtain waypoints from WPTTableOrig_numbers.m
//
string wptTable("/home/jane/mywork/aerosimmodel-opti/WPTTableOrig_numbers.m");
ifstream wptTableInput(wptTable.c_str());
if (!wptTableInput)
cerr << "Could not open WPTTableOrig_numbers.m" << endl;

double latitude, longitude, altitude, airspeed;
vector<double> wptTableLat(totalNumWaypoints);
vector<double> wptTableLon(totalNumWaypoints);
vector<double> wptTableAlt(totalNumWaypoints);
vector<double> wptTableAsp(totalNumWaypoints);
for (int i = 0; i < totalNumWaypoints; ++i)
{
    // Waypoint coordinates (lat & lon in radians)
    wptTableInput >> latitude >> longitude >> altitude >> airspeed;
    wptTableInput.ignore(255, '\n');
    wptTableLat[i] = latitude; // Latitude in rad
    wptTableLon[i] = longitude; // Longitude in rad
    wptTableAlt[i] = altitude; // Altitude in metres
    wptTableAsp[i] = airspeed; // Airspeed in m/s
}

    wptTableInput.close();

//
// Create the waypoint table file aerosim_run
//
string aerosimRunFilename = string("aerosim_run_") + stringID + mfile;
string WPTTableName = string("WPTTable_") + stringID;
ofstream aerosimRunFile(aerosimRunFilename.c_str());
aerosimRunFile << "cd /home/jane/mywork/aerosimmodel-opti/" << endl;
aerosimRunFile << "clear;" << endl;
aerosimRunFile << WPTTableName.c_str() << " = [...]" << endl;
// Optimising 1 or 2 WPTs
if (numWaypoints < 3)
{
    // Optimised WPTs
    for (int i = 0; i < numWaypoints; ++i)
    {
        aerosimRunFile.precision(9);
        aerosimRunFile << lat[i] << " ";
    }
}

```

```

    aerosimRunFile << lon[i] << " ";
    aerosimRunFile << wptTableAlt[i] << " ";
    aerosimRunFile << wptTableAsp[i] << ";..." << endl;
}
// The remaining WPTs
for (int i = numWaypoints; i < totalNumWaypoints; ++i)
{
    aerosimRunFile.precision(9);
    aerosimRunFile << wptTableLat[i] << " ";
    aerosimRunFile << wptTableLon[i] << " ";
    aerosimRunFile << wptTableAlt[i] << " ";
    aerosimRunFile << wptTableAsp[i];
    if (i < (totalNumWaypoints - 1))
aerosimRunFile << ";..." << endl;
    else
aerosimRunFile << "]" << endl;
}
}

// If #WPTs to be optimised is 3 or 4, then the WPTTable is formed as follows:
// --WPT 1-2: The optimised WPTs 1 & 2
// --WPT 3-5: The loiter WPTs from original WPTTable (not optimised)
// --WPT 6 and/or 7 : The remaining optimised WPTs
// --WPT 7-8 or just 8: The remaining WPTs from original WPTTable (not optimised)
else
{
// WPT 1-2
for (int i = 0; i < 2; ++i)
{
    aerosimRunFile.precision(9);
    aerosimRunFile << lat[i] << " ";
    aerosimRunFile << lon[i] << " ";
    aerosimRunFile << wptTableAlt[i] << " ";
    aerosimRunFile << wptTableAsp[i] << ";..." << endl;
}
// WPT 3-5 (loiter WPTs, not optimised)
for (int i = 2; i < 5; ++i)
{
    aerosimRunFile.precision(9);
    aerosimRunFile << wptTableLat[i] << " ";
    aerosimRunFile << wptTableLon[i] << " ";
    aerosimRunFile << wptTableAlt[i] << " ";
    aerosimRunFile << wptTableAsp[i] << ";..." << endl;
}
// Remaining optimised WPTs
for (int i = 2; i < numWaypoints; ++i)
{
    aerosimRunFile.precision(9);
    aerosimRunFile << lat[i] << " ";
    aerosimRunFile << lon[i] << " ";
    aerosimRunFile << wptTableAlt[i+3] << " ";
    aerosimRunFile << wptTableAsp[i+3] << ";..." << endl;
}
// Remaining WPTs (not optimised)
for (int i = numWaypoints + 3; i < totalNumWaypoints; ++i)
{
    aerosimRunFile.precision(9);
    aerosimRunFile << wptTableLat[i] << " ";
    aerosimRunFile << wptTableLon[i] << " ";
    aerosimRunFile << wptTableAlt[i] << " ";
    aerosimRunFile << wptTableAsp[i];
    if (i < (totalNumWaypoints - 1))
aerosimRunFile << ";..." << endl;
    else
aerosimRunFile << "]" << endl;
}
}

aerosimRunFile << "WPTTable = " << WPTTableName.c_str() << ";" << endl;
aerosimRunFile << "WPTTable = [WPTTable; WPTTable];" << endl;

//
// Add the run_model code onto the aerosim_run file
//

```



```
// Check for validity of fuel consumption value
// - If an unrealistically small value
//   add 100 to it so that it will not be placed in the buffer
//
// (fuelConsumption < 0.01)
fuelConsumption = fuelConsumption + 100;

//
// Construct the fitness function
// (Currently only one objective : fuel consumption)
// Will need to be modified to accommodate more objectives
//
double fitnesses[ObjNums];
for (int i = 0; i < ObjNums; ++i)
{
    fitnesses[i] = fuelConsumption;
    cout.precision(9);
    cout << "Final Fitness[" << i <<"] = " << fitnesses[i] << endl;
}
}
```

Appendix D

MATLAB SQP Solver Code for Single-Objective MWO

D.1 *fminconOpti* Function for Mission Scenario 1

```

function [history,T1, WPTTableOpti,fuel_consumption,output,T2] = fminconOpti(numWptOpti)
% Runs the fmincon optimisation process using the AeroSim simulation model
% to evaluate the given WPTTable in terms of fuel consumption.
%
% Input:
% - numWptOpti = Number of waypoints to be optimised
%
% Outputs:
% - history = the structure that keeps track of the history of fuel
%           consumptions and function evaluations
% - T1 = summary table of the fmincon process
% - WPTTableOpti = the optimised mission waypoint table
% - fuel_consumption = amount of fuel consumed during mission (kg)

% Set up shared variable with UTFUN
history.fuelconsumption = [];
history.funccount = [];
history.iteration = [];

% Input values
maxIters = 20;           % Maximum number of iterations evaluated

% Start clock
tic

% Baseline waypoint table (in radians)
WPTTableOrig = [...
    (26+33.9722/60)*pi/180  (151+51.1746/60)*pi/180  900 20;...
    (26+34.1315/60)*pi/180  (151+51.4181/60)*pi/180  900 20;...
    (26+34.2729/60)*pi/180  (151+53.2179/60)*pi/180  750 30;...
    (26+34.1322/60)*pi/180  (151+53.3082/60)*pi/180  750 20;...
    (26+33.9915/60)*pi/180  (151+53.2179/60)*pi/180  750 20;...
    (26+34.1315/60)*pi/180  (151+51.4181/60)*pi/180  900 20;...
    (26+34.2358/60)*pi/180  (151+51.2796/60)*pi/180  900 20;...
    26.5808*pi/180          151.8411*pi/180          800 30];
save WPTTableOrig;

% Baseline waypoint table (in radians)
% -- only WPTs to be optimised ---
WPTTableOpt = [WPTTableOrig(1:2,1:3); WPTTableOrig(6:7,1:3)]; % Table of WPTs to be optimised
latOpt = WPTTableOpt(:,1);
lonOpt = WPTTableOpt(:,2);

% Calculate Upper Bounds (in radians)
% -- only WPTs 2-3, 7-8 to be optimised --
% latUB = [0.463693195; 0.463718690; 0.463718124; 0.463762205];
latUB = [0.463693195; 0.463720000; 0.463720000; 0.463762205];
lonUB = [2.650340341; 2.650456443; 2.650456445; 2.650367830];

% Calculate Lower Bounds (in radians)
% -- only WPTs 2-3, 7-8 to be optimised --
% latLB = [0.463642241; 0.463709423; 0.463709989; 0.463741362];
latLB = [0.463642241; 0.463700000; 0.463700000; 0.463741362];
lonLB = [2.650312648; 2.650397006; 2.650400061; 2.650340137];

% Initialise values
numCoord = 2;           % Number of coordinates per waypoint
numVarOpti = numWptOpti * numCoord; % Number of variables to be optimised
x0 = zeros(numVarOpti,1); % Starting points for optimisation
lb = zeros(numVarOpti,1); % Lower bounds
ub = zeros(numVarOpti,1); % Upper bounds

% Set up the vector of starting points for optimisation
% -- only WPTs 2-3, 7-8 to be optimised --
x0(1:numCoord:numVarOpti,1) = latOpt(1:numWptOpti);
x0(2:numCoord:numVarOpti,1) = lonOpt(1:numWptOpti);

% Set up the upper bounds vector
% -- only WPTs 2-3, 7-8 to be optimised --

```

```

ub(1:numCoord:numVarOpti,1) = latUB(1:numWptOpti);
ub(2:numCoord:numVarOpti,1) = lonUB(1:numWptOpti);

% Set up the lower bounds vector
% -- only WPTs 2-3, 7-8 to be optimised --
lb(1:numCoord:numVarOpti,1) = latLB(1:numWptOpti);
lb(2:numCoord:numVarOpti,1) = lonLB(1:numWptOpti);

% Set up the options required for fmincon
options = optimset('OutputFcn',@outfun,'Display','iter',...
    'DiffMinChange',1e-5,'MaxIter',maxIters);

% Run fmincon optimisation process
% -- WPTTableOpti = the optimised waypoints
[T1,WPTTableOpti,fuel_consumption,output] = evalc('fmincon(@run_model,x0,[],[],[],[],lb,ub,[],options)');

% Stop clock
T2 = evalc('toc');

save optiResults

% *****
% ** Set up optimisationResults file **
% *****

% Setting up filename
filenameStem = 'optiResults-';
filenameEnd = '.txt';
filenameNo = num2str(numWptOpti);
filename = strcat(filenameStem, filenameNo, filenameEnd);

fid = fopen(filename,'wt');
fprintf(fid, 'Optimisation Settings:\n\n');
fprintf(fid, '*****\n');
fprintf(fid, 'The initial starting point is at (0.463922478rad, 2.650127135rad, 850)\n\n');
fprintf(fid, 'The original waypoint table used for this mission is:\n\n');
for i = 1 : length(WPTTableOrig)
    fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
end
fprintf(fid, '\n');
fprintf(fid, 'The number of waypoints optimised : %d \n\n', numWptOpti);
fprintf(fid, 'The initial guess for the waypoints to be optimised : \n\n');
i = 1
for i = 1 : numVarOpti
    if (rem(i,numCoord) == 1)
        fprintf(fid, '|%10.9frad ', x0(i,1));
    else
        fprintf(fid, ' %10.9frad|\n', x0(i,1));
    end
end
fprintf(fid, '\n');
fprintf(fid, 'The candidate coordinates are taken from an area bounded by: \n\n');
fprintf(fid, '--- Latitudes of %10.9frad to %10.9frad \n\n', (26+33.8/60)*pi/180, (26+34.5/60)*pi/180);
fprintf(fid, '--- Longitudes of %10.9frad to %10.9frad \n\n', (151.8411)*pi/180, (151+54.5082/60)*pi/180);
fprintf(fid, '*****\n');
fprintf(fid, 'The results of the mission optimisation process is as follows: \n\n');
fprintf(fid, 'The optimised mission waypoint table is : \n\n');
switch numWptOpti
    case 1
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));
        i = 2;
        for i = 2 : 8
            fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
        end
    case 2
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(3), WPTTableOpti(4), WPTTableOrig(2,3));
        i = 3;
        for i = 3 : 8
            fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
        end
    case 3
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));

```

```

fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOpti(3), WPTTableOpti(4), WPTTableOrig(2,3));
i = 3;
for i = 3 : 5
    fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
end
fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOpti(5), WPTTableOpti(6), WPTTableOrig(6,3));
fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOrig(7,1), WPTTableOrig(7,2), WPTTableOrig(7,3));
fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOrig(8,1), WPTTableOrig(8,2), WPTTableOrig(8,3));
case 4
fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));
fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOpti(3), WPTTableOpti(4), WPTTableOrig(2,3));
i = 3;
for i = 3 : 5
    fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
end
fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOpti(5), WPTTableOpti(6), WPTTableOrig(6,3));
fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOrig(7), WPTTableOrig(8), WPTTableOrig(7,3));
fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOrig(8,1), WPTTableOrig(8,2), WPTTableOrig(8,3));
end

fprintf(fid, '\n');
fprintf(fid, '*****\n');
fprintf(fid, 'The process table T1 is : \n\n');
fprintf(fid, '%s', T1);
fprintf(fid, '\n');
fprintf(fid, '*****\n');
fprintf(fid, 'The fuel consumption for this mission is %8.6fkg\n\n', fuel_consumption);
fprintf(fid, '*****\n');
fprintf(fid, '%s', T2);
fclose(fid)

function stop = outfun(x,optimValues,state)
% Function for generating the vector 'history'
% --- Keeps track of the history of fuel consumptions and function
% evaluations ---
stop = false;

switch state
case 'iter'
% Concatenate current fuel consumption and function evaluation
% values with history
history.fuelconsumption = [history.fuelconsumption,optimValues.fval];
history.funccount = [history.funccount,optimValues.funccount];
history.iteration = [history.iteration,optimValues.iteration];
otherwise
end
end

function fuel_cons = run_model(x0)
% Calls the AeroSim model 'aerosonde_mission_v2.mdl' to run through the
% specified flight mission, and obtains the total fuel consumption for
% the mission.
%
% INPUT:
% - x0 = The waypoints to be optimised in the format
%         |lat|
%         |lon|
%
% OUTPUT:
% - fuel_cons = total fuel consumption for the mission

% Initialise
WPTTable = zeros(40,4); % Set up WPTTable
WPTTableTemp = zeros(8,4); % Set up WPTTableTemp

% Rearranging coordinates from x0 into WPTTable format
% [lat lon alt] <-- 'alt' is taken from WPTTableOrig
% --- This is just for one lap of flight
switch numWptOpti
case 1
% only 1 waypoint (WPT 2) is being optimised
WPTTableTemp(1,1) = x0(1);

```



```

        WPTTableTemp(1,2)    = x0(2);
        WPTTableTemp(1,3)    = WPTTableOrig(1,3);
        WPTTableTemp(2:8,1:3) = WPTTableOrig(2:8,1:3);
        WPTTableTemp(1:8,4)  = WPTTableOrig(1:8,4); % Airspeed
    case 2
        % 2 waypoints (WPT 2 & 3) are being optimised
        WPTTableTemp(1:2,1)  = x0(1:2:4);
        WPTTableTemp(1:2,2)  = x0(2:2:4);
        WPTTableTemp(1:2,3)  = WPTTableOrig(1:2,3);
        WPTTableTemp(3:8,1:3) = WPTTableOrig(3:8,1:3);
        WPTTableTemp(1:8,4)  = WPTTableOrig(1:8,4); % Airspeed
    case 3
        % 3 waypoints (WPTs 2, 3 & 7) are being optimised
        WPTTableTemp(1:2,1)  = x0(1:2:4);
        WPTTableTemp(1:2,2)  = x0(2:2:4);
        WPTTableTemp(1:2,3)  = WPTTableOrig(1:2,3);
        WPTTableTemp(3:5,1:3) = WPTTableOrig(3:5,1:3);
        WPTTableTemp(6,1)    = x0(5);
        WPTTableTemp(6,2)    = x0(6);
        WPTTableTemp(6,3)    = WPTTableOrig(6,3);
        WPTTableTemp(7:8,1:3) = WPTTableOrig(7:8,1:3);
        WPTTableTemp(1:8,4)  = WPTTableOrig(1:8,4); % Airspeed
    case 4
        % 4 waypoints (WPTs 2, 3, 7 & 8) are being optimised
        WPTTableTemp(1:2,1)  = x0(1:2:4);
        WPTTableTemp(1:2,2)  = x0(2:2:4);
        WPTTableTemp(1:2,3)  = WPTTableOrig(1:2,3);
        WPTTableTemp(3:5,1:3) = WPTTableOrig(3:5,1:3);
        WPTTableTemp(6:7,1)  = x0(5:2:8);
        WPTTableTemp(6:7,2)  = x0(6:2:8);
        WPTTableTemp(6:7,3)  = WPTTableOrig(6:7,3);
        WPTTableTemp(8,1:3)  = WPTTableOrig(8,1:3);
        WPTTableTemp(1:8,4)  = WPTTableOrig(1:8,4); % Airspeed
    end

    % The mission consists of 5 laps
    WPTTable = [WPTTableTemp; WPTTableTemp; WPTTableTemp; WPTTableTemp; WPTTableTemp];

    save WPTTable WPTTable
    S = load('WPTTable.mat');
    WPTTable = S.WPTTable;

    % Run simulation with the given waypoints
    sim('aerosonde_mission_v2');

    % Extract Fuel Mass information
    size_array = size(fuel_mass);
    fuel_m = zeros(1,size_array(3));
    fuel_m(1:size_array(3)) = fuel_mass(:,1:size_array(3));

    % Calculate fuel consumption
    num = size(fuel_m);
    fuel_cons = fuel_m(1) - fuel_m(num(2));
end

end

```

D.2 *fminconOpti* Function for Mission Scenario 2

```

function [history,T1, WPTTableOpti,fuel_consumption,output,T2] = fminconOpti(numWptOpti)
% Runs the fmincon optimisation process using the AeroSim simulation model
% to evaluate the given WPTTable in terms of fuel consumption.
%
% Input:
% - numWptOpti = Number of waypoints to be optimised
%
% Outputs:

```

```

% - history = the structure that keeps track of the history of fuel
%       consumptions and function evaluations
% - T1 = summary table of the fmincon process
% - WPTTableOpti = the optimised mission waypoint table
% - fuel_consumption = amount of fuel consumed during mission (kg)

% Set up shared variable with OUTFUN
history.fuelconsumption = [];
history.funccount = [];
history.iteration = [];

% Input values
maxIters = 20;           % Maximum number of iterations evaluated

% Start clock
tic

% Baseline waypoint table (in radians)
WPTTableOrig = [...
    (26+33.7693/60)*pi/180  (151+51.1585/60)*pi/180  900 20;...
    (26+33.9588/60)*pi/180  (151+51.5236/60)*pi/180  900 20;...
    (26+35.0415/60)*pi/180  (151+53.6234/60)*pi/180  750 30;...
    (26+34.8070/60)*pi/180  (151+53.7740/60)*pi/180  750 20;...
    (26+34.5725/60)*pi/180  (151+53.6234/60)*pi/180  750 20;...
    (26+33.9588/60)*pi/180  (151+51.5236/60)*pi/180  900 20;...
    (26+34.2132/60)*pi/180  (151+51.4206/60)*pi/180  900 20;...
    26.5808*pi/180          151.8411*pi/180          800 30];
save WPTTableOrig;

% Baseline waypoint table (in radians)
% -- only WPTs to be optimised ---
WPTTableOpt = [WPTTableOrig(1:2,1:3); WPTTableOrig(6:7,1:3)]; % Table of WPTs to be optimised
latOpt = WPTTableOpt(:,1);
lonOpt = WPTTableOpt(:,2);

% Calculate Upper Bounds (in radians)
% -- only WPTs 2-3, 7-8 to be optimised --
% latUB = [0.463693195; 0.463718690; 0.463718124; 0.463762205];
latUB = [0.463693195; 0.463720000; 0.463720000; 0.463762205];
lonUB = [2.650340341; 2.650456443; 2.650456445; 2.650367830];

% Calculate Lower Bounds (in radians)
% -- only WPTs 2-3, 7-8 to be optimised --
% latLB = [0.463642241; 0.463709423; 0.463709989; 0.463741362];
latLB = [0.463642241; 0.463700000; 0.463700000; 0.463741362];
lonLB = [2.650312648; 2.650397006; 2.650400061; 2.650340137];

% Initialise values
numCoord = 2;           % Number of coordinates per waypoint
numVarOpti = numWptOpti * numCoord; % Number of variables to be optimised
x0 = zeros(numVarOpti,1); % Starting points for optimisation
lb = zeros(numVarOpti,1); % Lower bounds
ub = zeros(numVarOpti,1); % Upper bounds

% Set up the vector of starting points for optimisation
% -- only WPTs 2-3, 7-8 to be optimised --
x0(1:numCoord:numVarOpti,1) = latOpt(1:numWptOpti);
x0(2:numCoord:numVarOpti,1) = lonOpt(1:numWptOpti);

% Set up the upper bounds vector
% -- only WPTs 2-3, 7-8 to be optimised --
ub(1:numCoord:numVarOpti,1) = latUB(1:numWptOpti);
ub(2:numCoord:numVarOpti,1) = lonUB(1:numWptOpti);

% Set up the lower bounds vector
% -- only WPTs 2-3, 7-8 to be optimised --
lb(1:numCoord:numVarOpti,1) = latLB(1:numWptOpti);
lb(2:numCoord:numVarOpti,1) = lonLB(1:numWptOpti);

% Set up the options required for fmincon
options = optimset('OutputFcn',@outfun,'Display','iter',...
    'DiffMinChange',1e-5,'MaxIter',maxIters);

```

```

% Run fmincon optimisation process
% -- WPTTableOpti = the optimised waypoints
[T1,WPTTableOpti,fuel_consumption,output] = evalc('fmincon(@run_model,x0,[],[],[],[],lb,ub,@distconstr,options)');

% Stop clock
T2 = evalc('toc');

save optiResults

% *****
% ** Set up optimisationResults file **
% *****

% Setting up filename
filenameStem = 'optiResults-';
filenameEnd = '.txt';
filenameNo = num2str(numWptOpti);
filename = strcat(filenameStem, filenameNo, filenameEnd);

fid = fopen(filename,'wt');
fprintf(fid, 'Optimisation Settings:\n\n');
fprintf(fid, '*****\n');
fprintf(fid, 'The initial starting point is at (0.463922478rad, 2.650127135rad, 850)\n\n');
fprintf(fid, 'The original waypoint table used for this mission is:\n\n');
for i = 1 : length(WPTTableOrig)
    fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
end
fprintf(fid, '\n');
fprintf(fid, 'The number of waypoints optimised : %d \n\n', numWptOpti);
fprintf(fid, 'The initial guess for the waypoints to be optimised : \n\n');
i = 1
for i = 1 : numVarOpti
    if (rem(i,numCoord) == 1)
        fprintf(fid, '|%10.9frad ', x0(i,1));
    else
        fprintf(fid, ' %10.9frad|\n', x0(i,1));
    end
end
fprintf(fid, '\n');
fprintf(fid, 'The candidate coordinates are taken from an area bounded by: \n\n');
fprintf(fid, '--- Latitudes of %10.9frad to %10.9frad \n\n', (26+33.8/60)*pi/180, (26+34.5/60)*pi/180);
fprintf(fid, '--- Longitudes of %10.9frad to %10.9frad \n\n', (151.8411)*pi/180, (151+54.5082/60)*pi/180);
fprintf(fid, '*****\n');
fprintf(fid, 'The results of the mission optimisation process is as follows: \n\n');
fprintf(fid, 'The optimised mission waypoint table is : \n\n');
switch numWptOpti
    case 1
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));
        i = 2;
        for i = 2 : 8
            fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
        end
    case 2
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(3), WPTTableOpti(4), WPTTableOrig(2,3));
        i = 3;
        for i = 3 : 8
            fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
        end
    case 3
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(3), WPTTableOpti(4), WPTTableOrig(2,3));
        i = 3;
        for i = 3 : 5
            fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
        end
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(5), WPTTableOpti(6), WPTTableOrig(6,3));
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(7,1), WPTTableOrig(7,2), WPTTableOrig(7,3));
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(8,1), WPTTableOrig(8,2), WPTTableOrig(8,3));
    case 4
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(3), WPTTableOpti(4), WPTTableOrig(2,3));

```

```

    i = 3;
    for i = 3 : 5
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
    end
    fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(5), WPTTableOpti(6), WPTTableOrig(6,3));
    fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(7), WPTTableOrig(8), WPTTableOrig(7,3));
    fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(8,1), WPTTableOrig(8,2), WPTTableOrig(8,3));
end

fprintf(fid, '\n');
fprintf(fid, '*****\n');
fprintf(fid, 'The process table T1 is : \n\n');
fprintf(fid, '%s', T1);
fprintf(fid, '\n');
fprintf(fid, '*****\n');
fprintf(fid, 'The fuel consumption for this mission is %8.6fkg\n\n', fuel_consumption);
fprintf(fid, '*****\n');
fprintf(fid, '%s', T2);
fclose(fid)

function stop = outfun(x,optimValues,state)
    % Function for generating the vector 'history'
    % --- Keeps track of the history of fuel consumptions and function
    % evaluations ---
    stop = false;

    switch state
        case 'iter'
            % Concatenate current fuel consumption and function evaluation
            % values with history
            history.fuelconsumption = [history.fuelconsumption,optimValues.fval];
            history.funccount = [history.funccount,optimValues.funccount];
            history.iteration = [history.iteration,optimValues.iteration];
        otherwise
    end
end

function fuel_cons = run_model(x0)
    % Calls the AeroSim model 'aerosonde_mission_v2.mdl' to run through the
    % specified flight mission, and obtains the total fuel consumption for
    % the mission.
    %
    % INPUT:
    % - x0 = The waypoints to be optimised in the format
    %       |lat|
    %       |lon|
    %
    % OUTPUT:
    % - fuel_cons = total fuel consumption for the mission

    % Initialise
    WPTTable = zeros(40,4); % Set up WPTTable
    WPTTableTemp = zeros(8,4); % Set up WPTTableTemp

    % Rearranging coordinates from x0 into WPTTable format
    % [lat lon alt] <-- 'alt' is taken from WPTTableOrig
    % --- This is just for one lap of flight
    switch numWptOpti
        case 1
            % only 1 waypoint (WPT 2) is being optimised
            WPTTableTemp(1,1) = x0(1);
            WPTTableTemp(1,2) = x0(2);
            WPTTableTemp(1,3) = WPTTableOrig(1,3);
            WPTTableTemp(2:8,1:3) = WPTTableOrig(2:8,1:3);
            WPTTableTemp(1:8,4) = WPTTableOrig(1:8,4); % Airspeed
        case 2
            % 2 waypoints (WPT 2 & 3) are being optimised
            WPTTableTemp(1:2,1) = x0(1:2:4);
            WPTTableTemp(1:2,2) = x0(2:2:4);
            WPTTableTemp(1:2,3) = WPTTableOrig(1:2,3);
            WPTTableTemp(3:8,1:3) = WPTTableOrig(3:8,1:3);
            WPTTableTemp(1:8,4) = WPTTableOrig(1:8,4); % Airspeed
    end
end

```

```

case 3
    % 3 waypoints (WPTs 2, 3 & 7) are being optimised
    WPTTableTemp(1:2,1) = x0(1:2:4);
    WPTTableTemp(1:2,2) = x0(2:2:4);
    WPTTableTemp(1:2,3) = WPTTableOrig(1:2,3);
    WPTTableTemp(3:5,1:3) = WPTTableOrig(3:5,1:3);
    WPTTableTemp(6,1) = x0(5);
    WPTTableTemp(6,2) = x0(6);
    WPTTableTemp(6,3) = WPTTableOrig(6,3);
    WPTTableTemp(7:8,1:3) = WPTTableOrig(7:8,1:3);
    WPTTableTemp(1:8,4) = WPTTableOrig(1:8,4); % Airspeed
case 4
    % 4 waypoints (WPTs 2, 3, 7 & 8) are being optimised
    WPTTableTemp(1:2,1) = x0(1:2:4);
    WPTTableTemp(1:2,2) = x0(2:2:4);
    WPTTableTemp(1:2,3) = WPTTableOrig(1:2,3);
    WPTTableTemp(3:5,1:3) = WPTTableOrig(3:5,1:3);
    WPTTableTemp(6:7,1) = x0(5:2:8);
    WPTTableTemp(6:7,2) = x0(6:2:8);
    WPTTableTemp(6:7,3) = WPTTableOrig(6:7,3);
    WPTTableTemp(8,1:3) = WPTTableOrig(8,1:3);
    WPTTableTemp(1:8,4) = WPTTableOrig(1:8,4); % Airspeed
end

% The mission consists of 5 laps
WPTTable = [WPTTableTemp; WPTTableTemp; WPTTableTemp; WPTTableTemp; WPTTableTemp];

save WPTTable WPTTable
S = load('WPTTable.mat');
WPTTable = S.WPTTable;

% Run simulation with the given waypoints
sim('aerosonde_mission_v2');

% Extract Fuel Mass information
size_array = size(fuel_mass);
fuel_m = zeros(1,size_array(3));
fuel_m(1:size_array(3)) = fuel_mass(:,1:size_array(3));

% Calculate fuel consumption
num = size(fuel_m);
fuel_cons = fuel_m(1) - fuel_m(num(2));
end

function [c,ceq] = distconstr(x0)
    % Sets the constraints for the waypoints to be optimised
    % -- minimum distance from other fixed waypoints

    % Rearranging coordinates from x0 into WPTTable format
    % [lat lon alt] <-- 'alt' is taken from WPTTableOrig
    % --- This is just for one lap of flight

    % Define starting WPT (Kingaroy)
    startLat = 26.5808*pi/180; % Latitude of Kingaroy
    startLon = 151.8411*pi/180; % Longitude of Kingaroy

    switch numWptOpti
        case 1
            % only 1 waypoint (WPT 2) is being optimised
            % -- distance(WPT1 -> WPT2) >= 2000m
            c(1) = 2000 - CalcGCDist(startLat,startLon,x0(1),x0(2));
            % -- distance(WPT2 -> WPT3) >= 300m
            c(2) = 300 - CalcGCDist(x0(1),x0(2),WPTTableOrig(2,1),WPTTableOrig(2,2));
            % No nonlinear equality constraints
            ceq = [];
        case 2
            % 2 waypoints (WPT 2 & 3) are being optimised
            % -- distance(WPT1 -> WPT2) >= 2000m
            c(1) = 2000 - CalcGCDist(startLat,startLon,x0(1),x0(2));
            % -- distance(WPT2 -> WPT3) >= 300m
            c(2) = 300 - CalcGCDist(x0(1),x0(2),x0(3),x0(4));
            % -- distance(WPT3 -> WPT4) >= 3000m
            c(3) = 3000 - CalcGCDist(x0(3),x0(4),WPTTableOrig(3,1),WPTTableOrig(3,2));
    end

```

```
% No nonlinear equality constraints
ceq = [];
case 3
% 3 waypoints (WPTs 2, 3 & 7) are being optimised
% -- distance(WPT1 -> WPT2) >= 2000m
c(1) = 2000 - CalcGCDist(startLat,startLon,x0(1),x0(2));
% -- distance(WPT2 -> WPT3) >= 300m
c(2) = 300 - CalcGCDist(x0(1),x0(2),x0(3),x0(4));
% -- distance(WPT3 -> WPT4) >= 3000m
c(3) = 3000 - CalcGCDist(x0(3),x0(4),WPTTableOrig(3,1),WPTTableOrig(3,2));
% -- distance(WPT6 -> WPT7) >= 3000m
c(4) = 3000 - CalcGCDist(WPTTableOrig(5,1),WPTTableOrig(5,2),x0(5),x0(6));
% -- distance(WPT7 -> WPT8) >= 300m
c(5) = 300 - CalcGCDist(x0(5),x0(6),WPTTableOrig(7,1),WPTTableOrig(7,2));
% No nonlinear equality constraints
ceq = [];
case 4
% 4 waypoints (WPTs 2, 3, 7 & 8) are being optimised
% -- distance(WPT1 -> WPT2) >= 2000m
c(1) = 2000 - CalcGCDist(startLat,startLon,x0(1),x0(2));
% -- distance(WPT2 -> WPT3) >= 300m
c(2) = 300 - CalcGCDist(x0(1),x0(2),x0(3),x0(4));
% -- distance(WPT3 -> WPT4) >= 3000m
c(3) = 3000 - CalcGCDist(x0(3),x0(4),WPTTableOrig(3,1),WPTTableOrig(3,2));
% -- distance(WPT6 -> WPT7) >= 3000m
c(4) = 3000 - CalcGCDist(WPTTableOrig(5,1),WPTTableOrig(5,2),x0(5),x0(6));
% -- distance(WPT7 -> WPT8) >= 300m
c(5) = 300 - CalcGCDist(x0(5),x0(6),x0(7),x0(8));
% -- distance(WPT8 -> WPT9) >= 1700m
c(6) = 1700 - CalcGCDist(x0(7),x0(8),WPTTableOrig(8,1),WPTTableOrig(8,2));
% No nonlinear equality constraints
ceq = [];
end
end
end
```

Appendix E

HAPMOEA Code for Two-Objective MWO: Excerpts

E.1 optimisation.parameters: An Example

```

PARAMETERS FOR AEROSIM FLIGHT MISSION OPTIMISATION
1      Number of runs
1 6    Optimisation will be terminated after 6 hours
2      Two objectives
4      Number of waypoints to be optimised
8      Total number of waypoints in WPTTable
2      Number of coordinates for each waypoint (lat & lon)
0.463596392 0.463596392 0.463596392 0.463596392    Lower bounds for WPT lat (rad)
0.46383268 0.46383268 0.46383268 0.46383268    Upper bounds for WPT lat (rad)
2.650312431 2.650312431 2.650312431 2.650312431  Lower bounds for WPT lon (rad)
2.650750363 2.650750363 2.650750363 2.650750363  Upper bounds for WPT lon (rad)
500.0  Live update every 500 seconds
0      Forced synchronous evaluation is OFF (1=YES)
2.00  Migration operation will be after two generations
0.3    Migration ratio set 30%
1      One layer of hierarchical topology is set
- LEVEL ONE -
141   First layer ID is set as 141 = waypoint coordinates
40    Population size
2     Parents of recombination
1     Intermediate Recombination is ON (1=YES)
42   Buffer length
2.0  Tournament-in-Buffer ratio
0.05 Initial Mutation Size

```

E.2 HierarEA.cpp

```

////////////////////////////////////
//                                     //
//                               Main Program                               //
//                                     //
////////////////////////////////////

int
main(int argc, char* argv[])
{
    // Set the priority of the driver executable down.
    int priRet = setpriority(PRIO_PROCESS, 0, +5);
    assert(priRet == 0);

    //
    // Check command line.
    //

    if (argc > 3)
    {
        cerr << "HEAAEROFOIL Usage:" << endl;
        cerr << "\theaerofoil - Uses default input file 'optimisation.parameters' and writes statistics to"
" 'output.statistics'." << endl;
        cerr << "\theaerofoil parameter_file output_file - Reads inputs from 'parameter_file' and writes"
" statistics to 'output_file'." << endl;
        cerr << "\theaerofoil restart - Restarts the run from the point at which it stopped." << endl;
        return 1;
    }

    //
    // Set up optimisation.parameters
    // First try: 1 waypoint
    //
    int NumWptsOpt = 4;
    GenerateOptiParaFile(NumWptsOpt);

    //

```



```

// Determine the type of run.
//

char* inputFile = "optimisation.parameters";
char* outputFile = "output.statistics";

if (argc == 3)
{
    inputFile = argv[1];
    outputFile = argv[2];
}

if (argc == 2)
{
    if (strcasecmp(argv[1], "reset") == 0)
{
    int retVal = system("rm -f heaaerofoil-restart-file-alg*");
    if (retVal == 0)
        cout << "All files successfully deleted." << endl;
    else
        cout << "Problem calling \"rm -f heaaerofoil-restart-file-alg*\"." <<
            endl;
    return 0;
}
    else
inputFile = argv[1];
}

if (argc == 3)
{
    inputFile = argv[1];
    outputFile = argv[2];
}

//
// Open the parameters file and read in the data.
//
ifstream fin(inputFile);
if (!fin)
    throw xMain("Couldn't open the input parameters file.");

// Independent parameters.
fin.ignore(255, '\n'); // First line is an identifier.

cout << "*****" << endl;
cout << "*** Hierarchical AeroSim Simulation ***" << endl;
cout << "*****" << endl << endl;

int totalRuns;
fin >> totalRuns; // Number of runs to do
fin.ignore(255, '\n');
cout << "Number of Runs : " << totalRuns << endl;

int stopOption;
double stopNums;
double TerminationCond;
fin >> stopOption >> stopNums; // Maximum time.
fin.ignore(255, '\n');
if ( stopOption == 1 )
{
    TerminationCond = stopNums * 3600.0;
    cout << "Stopping by Time (Hours) For Each Run : " << stopNums << endl;
}
else if ( stopOption == 2 )
{
    TerminationCond = stopNums;
    cout << "Stopping by Function Evaluation For Each Run : " << stopNums << endl;
}
else
{
    TerminationCond = stopNums;
    cout << "Stopping by Pre-defined value For Each Run : " << stopNums << endl;
}
}

```

```

int NumObjectives;
fin >> NumObjectives; // Number of objectives.
fin.ignore(255, '\n');
cout << "Number of Objective : " << NumObjectives << endl;

int numWpt;
fin >> numWpt; // Number of waypoints to be optimised.
fin.ignore(255, '\n');
if (numWpt < 1)
    throw xMain("Must have at least one waypoint.");
cout << "Number of Waypoints to be Optimised : " << numWpt << endl;

int totalNumWpt;
fin >> totalNumWpt; // Total number of waypoints in WPTTable.
fin.ignore(255, '\n');
if (totalNumWpt < 1)
    throw xMain("Must have at least one waypoint in WPTTable.");
cout << "Total Number of Waypoints in WPTTable : " << totalNumWpt << endl;

int numCoord;
fin >> numCoord; // Number of coordinates for each waypoint -- (lat & lon)
fin.ignore(255, '\n');
if (numCoord != 2)
    throw xMain("Must have two coordinates.");
cout << "Number of coordinates (lat & lon) : " << numCoord << endl;

double lbLat;
double ubLat;
vector<double> lowerBoundLat(numWpt);
vector<double> upperBoundLat(numWpt);
// Lower bounds for latitude of waypoints (rad).
for (int i = 0; i < numWpt; ++i)
{
    fin >> lbLat;
    lowerBoundLat[i] = lbLat;
}
fin.ignore(255, '\n');
// Upper bounds for latitude of waypoints (rad).
for (int i = 0; i < numWpt; ++i)
{
    fin >> ubLat;
    upperBoundLat[i] = ubLat;
}
fin.ignore(255, '\n');

double lbLon;
double ubLon;
vector<double> lowerBoundLon(numWpt);
vector<double> upperBoundLon(numWpt);
// Lower bounds for longitude of waypoints (rad).
for (int i = 0; i < numWpt; ++i)
{
    fin >> lbLon;
    lowerBoundLon[i] = lbLon;
}
fin.ignore(255, '\n');
// Upper bounds for longitude of waypoints (rad).
for (int i = 0; i < numWpt; ++i)
{
    fin >> ubLon;
    upperBoundLon[i] = ubLon;
}
fin.ignore(255, '\n');

double liveUpdates;
fin >> liveUpdates; fin.ignore(255, '\n'); // Live update interval (0 = None).
int forceSync;
fin >> forceSync; fin.ignore(255, '\n'); // Force synchronous behaviour (1 = Yes).

// Hierarchical parameters.
double migrationTime;
fin >> migrationTime; fin.ignore(255, '\n'); // Number of popsize evaluations before migration.

```

```

cout << "* Popsiz Evaluations Before Migration : " << migrationTime << endl;
if (migrationTime < 0.0)
    throw xMain("Migration time must be >= 0.0.");
double migrationSize;
fin >> migrationSize; fin.ignore(255, '\n'); // Multiplier of popsize individuals to migrate.
cout << "* Popsiz Individuals to Migrate : " << migrationSize << endl;
if (migrationSize < 0.0 || migrationSize > 1.0)
    throw xMain("Migration size must be between 0.0 and 1.0 inclusive.");
int levels;
fin >> levels; fin.ignore(255, '\n'); // Number hierarchical layers.
cout << "Number of Hiearchical levels : " << levels << endl;
if (levels < 1)
    throw xMain("Number of hierarchical levels must be >= 1");

// Parameters by layer: Layer 0 = top, layer 1 = 2nd, etc.
vector<int> SidePoints(levels), mu(levels), parents(levels), recoType(levels), buffLength(levels);
vector<double> tournRatio(levels), delta0(levels);

for (int i = 0; i < levels; ++i)
{
    fin.ignore(255, '\n'); // Separator line.
    fin >> SidePoints[i]; fin.ignore(255, '\n'); // Aerofoil side points
    fin >> mu[i]; fin.ignore(255, '\n'); // Size of mu.
    fin >> parents[i]; fin.ignore(255, '\n'); // Parents in recombination.
    fin >> recoType[i]; fin.ignore(255, '\n'); // Recombination type, 1 = intermediate.
    fin >> buffLength[i]; fin.ignore(255, '\n'); // Buffer length.
    fin >> tournRatio[i]; fin.ignore(255, '\n'); // Tournament-in-buffer ratio.
    fin >> delta0[i]; fin.ignore(255, '\n'); // Initial mutation size.
}
fin.close();

cout << "Force Synchronous:\t\t" << (forceSync == 1 ? "Yes" : "No") << endl;

for (int i = 0; i < levels; ++i)
{
    cout << "Layer " << i << ":" << endl;
    cout << "\tSide Points for Solver:\t\t" << SidePoints[i] << endl;
    cout << "\tPopulation Size: \t\t" << mu[i] << endl;
    cout << "\tParents in Recombination:\t" << parents[i] << endl;
    cout << "\tRecombination Type:\t\t" << (recoType[i] == 1 ? "Intermediate" : "Discrete") << endl;
    cout << "\tBuffer Length: \t\t\t" << buffLength[i] << endl;
    cout << "\tTournament-in-Buffer Ratio:\t" << tournRatio[i] << endl;
    cout << "\tInitial Mutation Size:\t\t" << delta0[i] << endl;
}

//
// Run specified number of times.
//
for (int run = 0; run < totalRuns; ++run)
{
    cout << "*** RUN " << (run + 1) << " OF " << totalRuns << " ***" << endl;

    // Start the clock.
    time_t startTime = time(NULL);

    //
    // Establish solvers.
    //

    cout << "Starting solvers:" << endl;
    vector<AeroSimSolver*> solvers;
    for (int i = 0, numThisLevel = 1; i < levels; ++i, numThisLevel *= 2)
{
    cout << "\n\tLevel " << i << ": " << flush;
    for (int j = 0; j < numThisLevel; ++j)
    {
        int minBufferSize = 0;
        if (levels > 1)
        {
            if (i == 0) // Top layer.
                minBufferSize = (int) (2.0*migrationSize*mu[i+1]);
            else if (i == levels - 1) // Bottom layer.
                minBufferSize = (int) (migrationSize*mu[i-1]);
        }
    }
}
}

```

```

else // Middle layers.
    minBufferSize = (int) (2.0*migrationSize*mu[i+1] + migrationSize*mu[i-1]);
}

// Must be at least equal to maximum processors.
if (minBufferSize < MaximumTasks)
minBufferSize = MaximumTasks;

if (forceSync == 1) // Must be at least equal to population size.
if (minBufferSize < mu[i])
    minBufferSize = mu[i];

// Establish.
solvers.push_back(new AeroSimSolver(SidePoints[i],
NumObjectives,
numWpt,
numCoord,
totalNumWpt,
minBufferSize));
}
}

cout << endl;
//
// Establish the algorithms.
//
vector<char*> fileName;

//Fix alpha

const int N = numWpt*numCoord; // Number of dimensions (waypoint coordinates only).
cout << "N = " << N << endl;
vector<double> lowerBounds(N), upperBounds(N);
double maxSpan = -1.0;

// Setting up lower and upper bound vectors
// Storing lower and upper bounds for latitude
for (int i = 0; i < numWpt; ++i)
{
    lowerBounds[i] = lowerBoundLat[i];
    upperBounds[i] = upperBoundLat[i];

    if (upperBounds[i] - lowerBounds[i] > maxSpan)
        maxSpan = upperBounds[i] - lowerBounds[i];
}

// Storing lower and upper bounds for longitude
for (int i = 0; i < numWpt; ++i)
{
    lowerBounds[i + numWpt] = lowerBoundLon[i];
    upperBounds[i + numWpt] = upperBoundLon[i];

    if (upperBounds[i] - lowerBounds[i] > maxSpan)
        maxSpan = upperBounds[i] - lowerBounds[i];
}

for (int i = 0; i < lowerBounds.size(); ++i)
cout << lowerBounds [i];

vector<MAMDES*> algs;
cout << "Starting algorithms:" << endl;
for (int i = 0, numThisLevel = 1; i < levels; ++i, numThisLevel *= 2)
{
for (int j = 0; j < numThisLevel; ++j)
{
    cout << "\tLevel " << i << ": " << endl;

    const int solverRefNum = algs.size();

    solvers[solverRefNum]->FreeWorkers(); // Clear jobs.
    const int tourneySize = (int)((double) buffLength[i] / tournRatio[i]); // Actual tournament size.

    char* newName = new char[80];

```

```

        sprintf(newName, "heaerofoil-restart-file-alg%d.dat", solverRefNum);
        fileName.push_back(newName);
        try
    {
        algs.push_back(new MAMDES(solvers[solverRefNum],
            fileName[solverRefNum]));
    }
        catch(MAMDESException)
    {
        algs.push_back(new MAMDES(solvers[solverRefNum],          // Solver for this algorithm.
            mu[i],                // Population size.
            N,                    // Number of variables.
            lowerBounds,          // { Problem }
            upperBounds,          // { bounds. }
            parents[i],           // Parents in recombination.
            buffLength[i],        // Buffer length.
            tourneySize,          // Tournament in buffer size.
            delta0[i]             // Starting mutation size.
        )
    );

    algs[solverRefNum]->WriteFile(fileName[solverRefNum]);
}
    algs[solverRefNum]->SetMaxFunctionEvaluations(65000); // Arbitrarily high.
    algs[solverRefNum]->SetRecombineType(recoType[i] == 1 ? MAMDES::INTERMEDIATE : MAMDES::DISCRETE);
    algs[solverRefNum]->SetMaximumVariance(maxSpan / 4.0);
    algs[solverRefNum]->SetMinimumVariance(1e-80);
    algs[solverRefNum]->SetNoiseSampleInterval(25);
    algs[solverRefNum]->ForceSynchronous(forceSync == 1 ? true : false);
    algs[solverRefNum]->SetReplacementStrategy(MAMDES::REPLACE_WORST_ALWAYS);

    algs[solverRefNum]->SetDDMRefinementParameter(8);
    algs[solverRefNum]->SetDDMMaximumEvaluations(2000);
    algs[solverRefNum]->SetDDMPowerTerm(0.5);
    algs[solverRefNum]->SetDDMProbabilityBounds(0.25,0.75);
    algs[solverRefNum]->SetDDMLearningPhaseAccepts(25);
    algs[solverRefNum]->SetDDMLearningPhaseMaxTrials(500);
    algs[solverRefNum]->UseDDM(true);
}
}

    cout << "Proceeding to free workers criteria " << endl;

    for (int i = 0; i < (int) solvers.size(); ++i)
solvers[i]->FreeWorkers();

    cout << "Proceeding to Reset the migration criteria " << endl;
    // Reset the migration criteria.
    MigrationCriteria(algs, migrationTime, true);

    //
    // Run the algorithm.
    //
    cout << "Running:" << endl;
    time_t currentTime = time(NULL);
    double elapsed = difftime(currentTime, startTime);

    double numVar = numCoord * numWpt;
    UpdateProgressFiles(algs, fileName, levels, liveUpdates, elapsed, numVar, false, true); // Reset the files.
    int lastHeadEvals = algs[0]->GetFunctionEvaluations();
    double TerminationTaget;

    if ( stopOption == 1 )
TerminationTaget = elapsed;
        else if ( stopOption == 2 )
TerminationTaget = algs[0]->GetFunctionEvaluations();
        else
TerminationTaget = algs[0]->GetElite().GetFitness()[0];

    while (TerminationTaget < TerminationCond)

```

```

{
    time_t currentTime = time(NULL);
    elapsed = difftime(currentTime, startTime);
    Evolve(algs);
    if (MigrationCriteria(algs, migrationTime, false))
    {
        Migrate(algs, solvers, levels, migrationSize);
    }

    if ( stopOption == 1 )
        TerminationTaget = elapsed;
    else if ( stopOption == 2 )
        TerminationTaget = algs[0]->GetFunctionEvaluations();
    else
        TerminationTaget = algs[0]->GetElite().GetFitness()[0];

    //
    // User update.
    //
    if (algs[0]->GetFunctionEvaluations() - lastHeadEvals > 10)
    {
        lastHeadEvals = algs[0]->GetFunctionEvaluations();
        cout << lastHeadEvals << " Function Evaluations done by Head Node.";
        cout << " Elapsed Time: " << elapsed << " seconds." << endl;

        solvers[0]->ShowLoadDistribution();
        solvers[0]->ResetWorkerStats();
        cout << endl;
    }
    if (liveUpdates > 0.0)
        UpdateProgressFiles(algs, fileName, levels, liveUpdates, elapsed, numVar);
}

    UpdateProgressFiles(algs, fileName, levels, liveUpdates, elapsed, numVar, true);
    // Write final point.

    currentTime = time(NULL);
    elapsed = difftime(currentTime, startTime);
    cout << " Elapsed Time: " << elapsed << " seconds.";

    // Clean up.
    for (int i = 0; i < (int) fileName.size(); ++i)
{
    delete [] (fileName[i]);
    fileName[i] = NULL;
    }

    //
    // Close algorithms and solvers.
    //

    for (int i = 0; i < (int) solvers.size(); ++i)
delete algs[i];
    algs.resize(0);
    cout << endl;

    for (int i = 0; i < (int) solvers.size(); ++i)
delete solvers[i];
    solvers.resize(0);
    cout << endl;

    } // End loop over separate runs.

    //
    // Done.
    //

    return 0;
}

```

E.3 Analyser.cpp

E.3.1 Mission Scenario 1

```
//
// Get back the candidate waypoint coordinates in vector form
// - separate into lat and lon (rad) [only lat and lon coordinates are optimised]
//
vector<double> lat(numWaypoints), lon(numWaypoints);
cout << "The waypoints to be evaluated are : " << endl;
for (int i = 0; i < numWaypoints; ++i)
{
    int objPos = i;
    lat[i] = recObj[objPos];
    objPos = objPos + numWaypoints;      // move to the next position
    lon[i] = recObj[objPos];
    cout.precision(9);
    cout << "Latitude " << i + 1 << " : " << lat[i] << endl;
    cout << "Longitude " << i + 1 << " : " << lon[i] << endl;
}

//
// Obtain waypoints from WPTTableOrig_numbers.m
//
string wptTable("/home/jane/mywork/aerosimmodel-opti/WPTTableOrig_numbers.m");
ifstream wptTableInput(wptTable.c_str());
if (!wptTableInput)
cerr << "Could not open WPTTableOrig_numbers.m" << endl;

double latitude, longitude, altitude, airspeed;
vector<double> wptTableLat(totalNumWaypoints);
vector<double> wptTableLon(totalNumWaypoints);
vector<double> wptTableAlt(totalNumWaypoints);
vector<double> wptTableAsp(totalNumWaypoints);
for (int i = 0; i < totalNumWaypoints; ++i)
{
// Waypoint coordinates (lat & lon in radians)
wptTableInput >> latitude >> longitude >> altitude >> airspeed;
wptTableInput.ignore(255, '\n');
wptTableLat[i] = latitude; // Latitude in rad
wptTableLon[i] = longitude; // Longitude in rad
wptTableAlt[i] = altitude; // Altitude in metres
wptTableAsp[i] = airspeed; // Airspeed in m/s
}

wptTableInput.close();

//
// Create the waypoint table file aerosim_run
//
string aerosimRunFilename = string("aerosim_run_") + stringID + mfile;
string WPTTableName = string("WPTTable_") + stringID;
ofstream aerosimRunFile(aerosimRunFilename.c_str());
aerosimRunFile << "cd /home/jane/mywork/aerosimmodel-opti/" << endl;
aerosimRunFile << "clear;" << endl;
aerosimRunFile << WPTTableName.c_str() << " = [...]" << endl;
// Optimising 1 or 2 WPTs
if (numWaypoints < 3)
{
// Optimised WPTs
for (int i = 0; i < numWaypoints; ++i)
{
aerosimRunFile.precision(9);
aerosimRunFile << lat[i] << " ";
aerosimRunFile << lon[i] << " ";
aerosimRunFile << wptTableAlt[i] << " ";
aerosimRunFile << wptTableAsp[i] << ";..." << endl;
}
// The remaining WPTs
for (int i = numWaypoints; i < totalNumWaypoints; ++i)
```

```

    {
        aerosimRunFile.precision(9);
        aerosimRunFile << wptTableLat[i] << " ";
        aerosimRunFile << wptTableLon[i] << " ";
        aerosimRunFile << wptTableAlt[i] << " ";
        aerosimRunFile << wptTableAsp[i];
        if (i < (totalNumWaypoints - 1))
aerosimRunFile << "... " << endl;
        else
aerosimRunFile << "]" << endl;
    }
}

// If #WPTs to be optimised is 3 or 4, then the WPTTable is formed as follows:
// --WPT 1-2: The optimised WPTs 1 & 2
// --WPT 3-5: The loiter WPTs from original WPTTable (not optimised)
// --WPT 6 and/or 7 : The remaining optimised WPTs
// --WPT 7-8 or just 8: The remaining WPTs from original WPTTable (not optimised)
else
{
// WPT 1-2
for (int i = 0; i < 2; ++i)
    {
        aerosimRunFile.precision(9);
        aerosimRunFile << lat[i] << " ";
        aerosimRunFile << lon[i] << " ";
        aerosimRunFile << wptTableAlt[i] << " ";
        aerosimRunFile << wptTableAsp[i] << "... " << endl;
    }
// WPT 3-5 (loiter WPTs, not optimised)
for (int i = 2; i < 5; ++i)
    {
        aerosimRunFile.precision(9);
        aerosimRunFile << wptTableLat[i] << " ";
        aerosimRunFile << wptTableLon[i] << " ";
        aerosimRunFile << wptTableAlt[i] << " ";
        aerosimRunFile << wptTableAsp[i] << "... " << endl;
    }
// Remaining optimised WPTs
for (int i = 2; i < numWaypoints; ++i)
    {
        aerosimRunFile.precision(9);
        aerosimRunFile << lat[i] << " ";
        aerosimRunFile << lon[i] << " ";
        aerosimRunFile << wptTableAlt[i+3] << " ";
        aerosimRunFile << wptTableAsp[i+3] << "... " << endl;
    }
// Remaining WPTs (not optimised)
for (int i = numWaypoints + 3; i < totalNumWaypoints; ++i)
    {
        aerosimRunFile.precision(9);
        aerosimRunFile << wptTableLat[i] << " ";
        aerosimRunFile << wptTableLon[i] << " ";
        aerosimRunFile << wptTableAlt[i] << " ";
        aerosimRunFile << wptTableAsp[i];
        if (i < (totalNumWaypoints - 1))
aerosimRunFile << "... " << endl;
        else
aerosimRunFile << "]" << endl;
    }
}

aerosimRunFile << "WPTTable = " << WPTTableName.c_str() << ";" << endl;
aerosimRunFile << "WPTTable = [WPTTable; WPTTable; WPTTable; WPTTable; WPTTable];" << endl;

//
// Add the run_model code onto the aerosim_run file
//
aerosimRunFile << endl;
aerosimRunFile << "save WPTTable WPTTable;" << endl;
aerosimRunFile << "S = load('WPTTable.mat');" << endl;
aerosimRunFile << "WPTTable = S.WPTTable;" << endl;
aerosimRunFile << "[fuel_cons,fl_time] = run_model_2obj(WPTTable);" << endl;
aerosimRunFile << "total_fuel_cons = fuel_cons*20;" << endl;

```



```

//
if (fuelConsumption < 0.01)
fuelConsumption = fuelConsumption + 100;
double invFlightTime = 1/flightTime;
if (invFlightTime < 0.001)
invFlightTime = invFlightTime + 100;

//
// Construct the fitness function
// - Currently two objectives: fuel consumption & flight time
//
double fitnesses[ObjNums];
fitnesses[0] = fuelConsumption;
fitnesses[1] = invFlightTime;
cout.precision(9);
cout << "Final Fitness = [" << fitnesses[0] << " " << fitnesses[1] << "]" << endl;

```

E.3.2 Mission Scenario 2

```

//
// Get back the candidate waypoint coordinates in vector form
// - separate into lat and lon (rad) [only lat and lon coordinates are optimised]
//
vector<double> lat(numWaypoints), lon(numWaypoints);
cout << "The waypoints to be evaluated are : " << endl;
for (int i = 0; i < numWaypoints; ++i)
{
int objPos = i;
lat[i] = recObj[objPos];
objPos = objPos + numWaypoints; // move to the next position
lon[i] = recObj[objPos];
cout.precision(9);
cout << "Latitude " << i + 1 << " : " << lat[i] << endl;
cout << "Longitude " << i + 1 << " : " << lon[i] << endl;
}

//
// Obtain waypoints from WPTTableOrig_numbers.m
//
string wptTable("/home/jane/mywork/aerosimmodel-opti/WPTTableOrig_numbers.m");
ifstream wptTableInput(wptTable.c_str());
if (!wptTableInput)
cerr << "Could not open WPTTableOrig_numbers.m" << endl;

double latitude, longitude, altitude, airspeed;
vector<double> wptTableLat(totalNumWaypoints);
vector<double> wptTableLon(totalNumWaypoints);
vector<double> wptTableAlt(totalNumWaypoints);
vector<double> wptTableAsp(totalNumWaypoints);
for (int i = 0; i < totalNumWaypoints; ++i)
{
wptTableInput >> latitude >> longitude >> altitude >> airspeed; // Waypoint coordinates (lat & lon in radians)
wptTableInput.ignore(255, '\n');
wptTableLat[i] = latitude; // Latitude in rad
wptTableLon[i] = longitude; // Longitude in rad
wptTableAlt[i] = altitude; // Altitude in metres
wptTableAsp[i] = airspeed; // Airspeed in m/s
}

wptTableInput.close();

//
// Create the waypoint table file aerosim_run
//
string aerosimRunFilename = string("aerosim_run_") + stringID + mfile;
string WPTTableName = string("WPTTable_") + stringID;
ofstream aerosimRunFile(aerosimRunFilename.c_str());
aerosimRunFile << "cd /home/jane/mywork/aerosimmodel-opti/" << endl;
aerosimRunFile << "clear;" << endl;

```

```

aerosimRunFile << WPTTableName.c_str() << " = [...]" << endl;
// Optimising 1 or 2 WPTs
if (numWaypoints < 3)
{
// Optimised WPTs
for (int i = 0; i < numWaypoints; ++i)
{
aerosimRunFile.precision(9);
aerosimRunFile << lat[i] << " ";
aerosimRunFile << lon[i] << " ";
aerosimRunFile << wptTableAlt[i] << " ";
aerosimRunFile << wptTableAsp[i] << ";..." << endl;
}
// The remaining WPTs
for (int i = numWaypoints; i < totalNumWaypoints; ++i)
{
aerosimRunFile.precision(9);
aerosimRunFile << wptTableLat[i] << " ";
aerosimRunFile << wptTableLon[i] << " ";
aerosimRunFile << wptTableAlt[i] << " ";
aerosimRunFile << wptTableAsp[i];
if (i < (totalNumWaypoints - 1))
aerosimRunFile << ";..." << endl;
else
aerosimRunFile << "]" << endl;
}
}

// If #WPTs to be optimised is 3 or 4, then the WPTTable is formed as follows:
// --WPT 1-2: The optimised WPTs 1 & 2
// --WPT 3-5: The loiter WPTs from original WPTTable (not optimised)
// --WPT 6 and/or 7 : The remaining optimised WPTs
// --WPT 7-8 or just 8: The remaining WPTs from original WPTTable (not optimised)
else
{
// WPT 1-2
for (int i = 0; i < 2; ++i)
{
aerosimRunFile.precision(9);
aerosimRunFile << lat[i] << " ";
aerosimRunFile << lon[i] << " ";
aerosimRunFile << wptTableAlt[i] << " ";
aerosimRunFile << wptTableAsp[i] << ";..." << endl;
}
// WPT 3-5 (loiter WPTs, not optimised)
for (int i = 2; i < 5; ++i)
{
aerosimRunFile.precision(9);
aerosimRunFile << wptTableLat[i] << " ";
aerosimRunFile << wptTableLon[i] << " ";
aerosimRunFile << wptTableAlt[i] << " ";
aerosimRunFile << wptTableAsp[i] << ";..." << endl;
}
// Remaining optimised WPTs
for (int i = 2; i < numWaypoints; ++i)
{
aerosimRunFile.precision(9);
aerosimRunFile << lat[i] << " ";
aerosimRunFile << lon[i] << " ";
aerosimRunFile << wptTableAlt[i+3] << " ";
aerosimRunFile << wptTableAsp[i+3] << ";..." << endl;
}
// Remaining WPTs (not optimised)
for (int i = numWaypoints + 3; i < totalNumWaypoints; ++i)
{
aerosimRunFile.precision(9);
aerosimRunFile << wptTableLat[i] << " ";
aerosimRunFile << wptTableLon[i] << " ";
aerosimRunFile << wptTableAlt[i] << " ";
aerosimRunFile << wptTableAsp[i];
if (i < (totalNumWaypoints - 1))
aerosimRunFile << ";..." << endl;
else
aerosimRunFile << "]" << endl;
}
}

```

```

}
}

aerosimRunFile << "WPTTable = " << WPTTableName.c_str() << ";" << endl;

//
// Determine the distance constraints
//
aerosimRunFile << "numWptOpti = " << numWaypoints << ";" << endl;
aerosimRunFile << "StartPtLat = 26.5808*pi/180;" << endl;
aerosimRunFile << "StartPtLon = 151.8411*pi/180;" << endl;
aerosimRunFile << endl;
aerosimRunFile << "switch numWptOpti" << endl;
aerosimRunFile << "    case 1    % Only WPT 2 is being optimised" << endl;
aerosimRunFile << "        dist1to2 = CalcGCDist(StartPtLat,StartPtLon,..." << endl;
aerosimRunFile << "            WPTTable(1,1),WPTTable(1,2));" << endl;
aerosimRunFile << "        dist2to3 = CalcGCDist(WPTTable(1,1),WPTTable(1,2),..." << endl;
aerosimRunFile << "            WPTTable(2,1),WPTTable(2,2));" << endl;
aerosimRunFile << "        if ((dist1to2 < 2000) || (dist2to3 < 300))" << endl;
aerosimRunFile << "            DistDiff = max([(2000-dist1to2),(300-dist2to3)]);" << endl;
aerosimRunFile << "            penalty = 100*DistDiff;" << endl;
aerosimRunFile << "        else" << endl;
aerosimRunFile << "            penalty = 0;" << endl;
aerosimRunFile << "        end" << endl;
aerosimRunFile << "    case 2    % WPTs 2 and 3 are being optimised" << endl;
aerosimRunFile << "        dist1to2 = CalcGCDist(StartPtLat,StartPtLon,..." << endl;
aerosimRunFile << "            WPTTable(1,1),WPTTable(1,2));" << endl;
aerosimRunFile << "        dist2to3 = CalcGCDist(WPTTable(1,1),WPTTable(1,2),..." << endl;
aerosimRunFile << "            WPTTable(2,1),WPTTable(2,2));" << endl;
aerosimRunFile << "        dist3to4 = CalcGCDist(WPTTable(2,1),WPTTable(2,2),..." << endl;
aerosimRunFile << "            WPTTable(3,1),WPTTable(3,2));" << endl;
aerosimRunFile << "        if ((dist1to2 < 2000) || (dist2to3 < 300) || (dist3to4 < 3000))" << endl;
aerosimRunFile << "            DistDiff = max([(2000-dist1to2),(300-dist2to3),(3000-dist3to4)]);" << endl;
aerosimRunFile << "            penalty = 100*DistDiff;" << endl;
aerosimRunFile << "        else" << endl;
aerosimRunFile << "            penalty = 0;" << endl;
aerosimRunFile << "        end" << endl;
aerosimRunFile << "    case 3    % WPTs 2, 3 and 7 are being optimised" << endl;
aerosimRunFile << "        dist1to2 = CalcGCDist(StartPtLat,StartPtLon,..." << endl;
aerosimRunFile << "            WPTTable(1,1),WPTTable(1,2));" << endl;
aerosimRunFile << "        dist2to3 = CalcGCDist(WPTTable(1,1),WPTTable(1,2),..." << endl;
aerosimRunFile << "            WPTTable(2,1),WPTTable(2,2));" << endl;
aerosimRunFile << "        dist3to4 = CalcGCDist(WPTTable(2,1),WPTTable(2,2),..." << endl;
aerosimRunFile << "            WPTTable(3,1),WPTTable(3,2));" << endl;
aerosimRunFile << "        dist6to7 = CalcGCDist(WPTTable(5,1),WPTTable(5,2),..." << endl;
aerosimRunFile << "            WPTTable(6,1),WPTTable(6,2));" << endl;
aerosimRunFile << "        dist7to8 = CalcGCDist(WPTTable(6,1),WPTTable(6,2),..." << endl;
aerosimRunFile << "            WPTTable(7,1),WPTTable(7,2));" << endl;
aerosimRunFile << "        if ((dist1to2 < 2000) || (dist2to3 < 300) || (dist3to4 < 3000) ..." << endl;
aerosimRunFile << "            || (dist6to7 < 3000) || (dist7to8 < 300))" << endl;
aerosimRunFile << "            DistDiff = max([(2000-dist1to2),(300-dist2to3),(3000-dist3to4),..." << endl;
aerosimRunFile << "                (3000-dist6to7),(300-dist7to8)]);" << endl;
aerosimRunFile << "            penalty = 100*DistDiff;" << endl;
aerosimRunFile << "        else" << endl;
aerosimRunFile << "            penalty = 0;" << endl;
aerosimRunFile << "        end" << endl;
aerosimRunFile << "    case 4    % WPTs 2, 3, 7 and 8 are being optimised" << endl;
aerosimRunFile << "        dist1to2 = CalcGCDist(StartPtLat,StartPtLon,..." << endl;
aerosimRunFile << "            WPTTable(1,1),WPTTable(1,2));" << endl;
aerosimRunFile << "        dist2to3 = CalcGCDist(WPTTable(1,1),WPTTable(1,2),..." << endl;
aerosimRunFile << "            WPTTable(2,1),WPTTable(2,2));" << endl;
aerosimRunFile << "        dist3to4 = CalcGCDist(WPTTable(2,1),WPTTable(2,2),..." << endl;
aerosimRunFile << "            WPTTable(3,1),WPTTable(3,2));" << endl;
aerosimRunFile << "        dist6to7 = CalcGCDist(WPTTable(5,1),WPTTable(5,2),..." << endl;
aerosimRunFile << "            WPTTable(6,1),WPTTable(6,2));" << endl;
aerosimRunFile << "        dist7to8 = CalcGCDist(WPTTable(6,1),WPTTable(6,2),..." << endl;
aerosimRunFile << "            WPTTable(7,1),WPTTable(7,2));" << endl;
aerosimRunFile << "        dist8to9 = CalcGCDist(WPTTable(7,1),WPTTable(7,2),..." << endl;
aerosimRunFile << "            WPTTable(8,1),WPTTable(8,2));" << endl;
aerosimRunFile << "        if ((dist1to2 < 2000) || (dist2to3 < 300) || (dist3to4 < 3000) || ..." << endl;
aerosimRunFile << "            (dist6to7 < 3000) || (dist7to8 < 300) || (dist8to9 < 1700))" << endl;
aerosimRunFile << "            DistDiff = max([(2000-dist1to2),(300-dist2to3),(3000-dist3to4),..." << endl;
aerosimRunFile << "                (3000-dist6to7),(300-dist7to8),(1700-dist8to9)]);" << endl;

```



```
//
// Obtain the mission outputs from the missionOutputs file
//
ifstream mOutput(missionOutputs.c_str());
if (!mOutput)
cout << "Couldn't open the mission outputs file." << endl;

double fuelConsumption;
double minDistanceToA1, minDistanceToA2;
mOutput >> fuelConsumption >> minDistanceToA1 >> minDistanceToA2; // Outputs of the simulation
mOutput.ignore(255, '\n');
cout << "Total Fuel Consumption of Simulation (kg) : " << fuelConsumption << endl;
cout << "Minimum Distance to Hazard Area 1 (m) : " << minDistanceToA1 << endl;
cout << "Minimum Distance to Hazard Area 2 (m) : " << minDistanceToA2 << endl;
mOutput.close();

//
// Check for validity of the fuel consumption value
// - If fuel consumption is an unrealistically small value
//   -> add 100 to it so that it will not be placed in the buffer
//
if (fuelConsumption < 0.01)
fuelConsumption = fuelConsumption + 100;

//
// Construct the fitness function
// - Currently two objectives: fuel consumption & total minimum distance to Hazard Areas
//
double fitnesses[ObjNums];
double totalMinDistToHA = 1000 * (minDistanceToA1 + minDistanceToA2) / (minDistanceToA1 * minDistanceToA2);
fitnesses[0] = fuelConsumption;
fitnesses[1] = totalMinDistToHA;
cout.precision(9);
cout << "Final Fitness = [" << fitnesses[0] << " " << fitnesses[1] << "]" << endl;
```

Appendix F

MATLAB SQP Solver Code for Two-Objective MWO

F.1 Mission Scenario 1

F.1.1 *twoOptiPareto* Function

```

function [history,T1,WPTTableOpti,missionOutputs,T2] = twoOptiPareto(numWptOpti,w1,w2)
% Runs the fmincon optimisation process using the AeroSim simulation model
% to evaluate the given WPTTable in terms of a combination of fuel
% consumption AND flight time (two objectives).
%
% This combination of the two objectives uses an aggregate method, in which
% different weights are used with each objective, and are then added
% together to form ONE overall objective.
%
% Inputs:
% - numWptOpti = number of waypoints to be optimised
% - w1         = weight coefficient for objective #1
% - w2         = weight coefficient for objective #2
%
% Outputs:
% - history    = structure that keeps track of the history of fuel
%              consumptions, flight times, objective function values
%              and function evaluations
% - T1        = summary table of the fmincon process
% - WPTTableOpti = optimised mission waypoint table
% - missionOutputs = vector containing the optimised values for fuel
%              consumption (kg), flight time (min), and objective
%              function.
% -- fuel_consumption = amount of fule consumed during mission (kg)
% -- flight_time     = time taken for the mission (hrs)
% - T2            = time taken

% Set up shared variable with OUTFUN
history.objFunc = [];
history.funccount = [];
history.iteration = [];

% Input values
maxIters = 20;      % Maximum number of iterations evaluated

% Start clock
tic

% Baseline waypoint table (in radians)
WPTTableOrig = [...
    (26+33.9722/60)*pi/180 (151+51.1746/60)*pi/180 900 20;...
    (26+34.1315/60)*pi/180 (151+51.4181/60)*pi/180 900 20;...
    (26+34.2729/60)*pi/180 (151+53.2179/60)*pi/180 750 30;...
    (26+34.1322/60)*pi/180 (151+53.3082/60)*pi/180 750 20;...
    (26+33.9915/60)*pi/180 (151+53.2179/60)*pi/180 750 20;...
    (26+34.1315/60)*pi/180 (151+51.4181/60)*pi/180 900 20;...
    (26+34.2358/60)*pi/180 (151+51.2796/60)*pi/180 900 20;...
    26.5808*pi/180        151.8411*pi/180        800 30];
save WPTTableOrig;

% Baseline waypoint table (in radians)
% -- only WPTs to be optimised ---
WPTTableOpt = [WPTTableOrig(1:2,1:3); WPTTableOrig(6:7,1:3)]; % Table of WPTs to be optimised
latOpt = WPTTableOpt(:,1);
lonOpt = WPTTableOpt(:,2);

% Calculate Upper Bounds (in radians)
% -- only WPTs 2-3, 7-8 to be optimised --
latUB = [0.463693195; 0.463718690; 0.463718124; 0.463762205];
latUB = [0.463693195; 0.463720000; 0.463720000; 0.463762205];
lonUB = [2.650340341; 2.650456443; 2.650456445; 2.650367830];

% Calculate Lower Bounds (in radians)
% -- only WPTs 2-3, 7-8 to be optimised --
latLB = [0.463642241; 0.463709423; 0.463709989; 0.463741362];

```



```

latLB = [0.463642241; 0.463700000; 0.463700000; 0.463741362];
lonLB = [2.650312648; 2.650397006; 2.650400061; 2.650340137];

% Initialise values
numCoord = 2; % Number of coordinates per waypoint
numVarOpti = numWptOpti * numCoord; % Number of variables to be optimised
x0 = zeros(numVarOpti,1); % Starting points for optimisation
lb = zeros(numVarOpti,1); % Lower bounds
ub = zeros(numVarOpti,1); % Upper bounds

% Set up the vector of starting points for optimisation
% -- only WPTs 2-3, 7-8 to be optimised --
x0(1:numCoord:numVarOpti,1) = latOpt(1:numWptOpti);
x0(2:numCoord:numVarOpti,1) = lonOpt(1:numWptOpti);

% Set up the upper bounds vector
% -- only WPTs 2-3, 7-8 to be optimised --
ub(1:numCoord:numVarOpti,1) = latUB(1:numWptOpti);
ub(2:numCoord:numVarOpti,1) = lonUB(1:numWptOpti);

% Set up the lower bounds vector
% -- only WPTs 2-3, 7-8 to be optimised --
lb(1:numCoord:numVarOpti,1) = latLB(1:numWptOpti);
lb(2:numCoord:numVarOpti,1) = lonLB(1:numWptOpti);

% Set up the options required for fmincon
options = optimset('OutputFcn',@outfun,'Display','iter',...
    'DiffMinChange',1e-5,'MaxIter',maxIters);

% Run fmincon optimisation process
% -- WPTTableOpti = the optimised waypoints
[T1,WPTTableOpti,objFuncVal,output] = evalc('fmincon(@run_model_2obj,x0,[],[],[],[],lb,ub,[],options)');

out1 = load('fuel_cons');
fuel_cons = out1.fuel_cons;
fuelConsumption = fuel_cons(2,length(fuel_cons));
out2 = load('flight_time');
flight_time = out2.flight_time;
flightTime = flight_time(2,length(flight_time));

missionOutputs = [fuelConsumption, flightTime, objFuncVal];

% Stop clock
T2 = evalc('toc');

% *****
% ** Set up 2objOptiResults file **
% *****

% Setting up filename
filenameStem = '2objOptiResults_';
filenameTxtEnd = '.txt';
filenameMATEnd = '.mat';
filenameNo = num2str(w1/0.05);
filenameTxt = strcat(filenameStem, filenameNo, filenameTxtEnd);
filenameMAT = strcat(filenameStem, filenameNo, filenameMATEnd);

save(filenameMAT,'history','T1','WPTTableOpti','missionOutputs','T2')

fid = fopen(filenameTxt,'wt');
fprintf(fid, 'Two-objective Optimisation Settings:\n\n');
fprintf(fid, '*****\n');
fprintf(fid, 'The initial starting point is at (0.463922478rad, 2.650127135rad, 850)\n\n');
fprintf(fid, 'The original waypoint table used for this mission is:\n\n');
for i = 1 : length(WPTTableOrig)
    fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
end
fprintf(fid, '\n');
fprintf(fid, 'The number of waypoints optimised : %d \n\n', numWptOpti);
fprintf(fid, 'The initial guess for the waypoints to be optimised : \n\n');
i = 1
for i = 1 : numVarOpti
    if (rem(i,numCoord) == 1)

```

```

        fprintf(fid, '%10.9frad ', x0(i,1));
    else
        fprintf(fid, '%10.9frad\n', x0(i,1));
    end
end
fprintf(fid, '\n');
fprintf(fid, 'The candidate coordinates are taken from an area bounded by: \n\n');
fprintf(fid, '--- Latitudes of %10.9frad to %10.9frad \n\n', (26+33.8/60)*pi/180, (26+34.5/60)*pi/180);
fprintf(fid, '--- Longitudes of %10.9frad to %10.9frad \n\n', (151.8411)*pi/180, (151+54.5082/60)*pi/180);
fprintf(fid, '*****\n');
fprintf(fid, 'The results of the mission optimisation process is as follows: \n\n');
fprintf(fid, 'The optimised mission waypoint table is : \n\n');
switch numWptOpti
    case 1
        fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));
        i = 2;
        for i = 2 : 8
            fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
        end
    case 2
        fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));
        fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOpti(3), WPTTableOpti(4), WPTTableOrig(2,3));
        i = 3;
        for i = 3 : 8
            fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
        end
    case 3
        fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));
        fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOpti(3), WPTTableOpti(4), WPTTableOrig(2,3));
        i = 3;
        for i = 3 : 5
            fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
        end
        fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOpti(5), WPTTableOpti(6), WPTTableOrig(6,3));
        fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOrig(7,1), WPTTableOrig(7,2), WPTTableOrig(7,3));
        fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOrig(8,1), WPTTableOrig(8,2), WPTTableOrig(8,3));
    case 4
        fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));
        fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOpti(3), WPTTableOpti(4), WPTTableOrig(2,3));
        i = 3;
        for i = 3 : 5
            fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
        end
        fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOpti(5), WPTTableOpti(6), WPTTableOrig(6,3));
        fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOrig(7), WPTTableOrig(8), WPTTableOrig(7,3));
        fprintf(fid, '%10.9frad %10.9frad %6.3fm\n', WPTTableOrig(8,1), WPTTableOrig(8,2), WPTTableOrig(8,3));
end

fprintf(fid, '\n');
fprintf(fid, '*****\n');
fprintf(fid, 'The process table T1 is : \n\n');
fprintf(fid, '%s', T1);
fprintf(fid, '\n');
fprintf(fid, '*****\n');
fprintf(fid, 'The fuel consumption for this mission is %8.6fkg\n\n', fuelConsumption);
fprintf(fid, 'The flight time for this mission is %8.6f hours \n\n', flightTime);
fprintf(fid, '*****\n');
fprintf(fid, '%s', T2);
fclose(fid)

function stop = outfun(x,optimValues,state)
    % Function for generation the vector 'history'
    % --- Keeps track of the history of the objective function values and
    % function evaluations ---
    stop = false;

    switch state
        case 'iter'
            % Concatenate current objective function value and function
            % evaluation value with history
            history.objFunc = [history.objFunc,optimValues.fval];
            history.funccount = [history.funccount,optimValues.funccount];
            history.iteration = [history.iteration,optimValues.iteration];
    end
end

```

```

        otherwise
    end
end

function objFuncVal = run_model_2obj(x0)
% Calls the AeroSim model 'aerosonde_mission_v3_2obj.mdl' to run
% through the specified flight mission, and obtains the objective
% function value for the mission
%
% INPUT:
% - x0 = The waypoints to be optimised in the format
%       |lat|
%       |lon|
%
% OUTPUT:
% - objFunc = value of the objective function, a combination of fuel
%             consumption and flight time

% Initialise
WPTTable = zeros(40,4); % Set up WPTTable
WPTTableTemp = zeros(8,4); % Set up WPTTableTemp

% Rearranging coordinates from x0 into WPTTable format
% [lat lon alt] <-- 'alt' is taken from WPTTableOrig
% --- This is just for one lap of flight
switch numWptOpti
    case 1
        % only 1 waypoint (WPT 2) is being optimised
        WPTTableTemp(1,1) = x0(1);
        WPTTableTemp(1,2) = x0(2);
        WPTTableTemp(1,3) = WPTTableOrig(1,3);
        WPTTableTemp(2:8,1:3) = WPTTableOrig(2:8,1:3);
        WPTTableTemp(1:8,4) = WPTTableOrig(1:8,4); % Airspeed
    case 2
        % 2 waypoints (WPT 2 & 3) are being optimised
        WPTTableTemp(1:2,1) = x0(1:2:4);
        WPTTableTemp(1:2,2) = x0(2:2:4);
        WPTTableTemp(1:2,3) = WPTTableOrig(1:2,3);
        WPTTableTemp(3:8,1:3) = WPTTableOrig(3:8,1:3);
        WPTTableTemp(1:8,4) = WPTTableOrig(1:8,4); % Airspeed
    case 3
        % 3 waypoints (WPTs 2, 3 & 7) are being optimised
        WPTTableTemp(1:2,1) = x0(1:2:4);
        WPTTableTemp(1:2,2) = x0(2:2:4);
        WPTTableTemp(1:2,3) = WPTTableOrig(1:2,3);
        WPTTableTemp(3:5,1:3) = WPTTableOrig(3:5,1:3);
        WPTTableTemp(6,1) = x0(5);
        WPTTableTemp(6,2) = x0(6);
        WPTTableTemp(6,3) = WPTTableOrig(6,3);
        WPTTableTemp(7:8,1:3) = WPTTableOrig(7:8,1:3);
        WPTTableTemp(1:8,4) = WPTTableOrig(1:8,4); % Airspeed
    case 4
        % 4 waypoints (WPTs 2, 3, 7 & 8) are being optimised
        WPTTableTemp(1:2,1) = x0(1:2:4);
        WPTTableTemp(1:2,2) = x0(2:2:4);
        WPTTableTemp(1:2,3) = WPTTableOrig(1:2,3);
        WPTTableTemp(3:5,1:3) = WPTTableOrig(3:5,1:3);
        WPTTableTemp(6:7,1) = x0(5:2:8);
        WPTTableTemp(6:7,2) = x0(6:2:8);
        WPTTableTemp(6:7,3) = WPTTableOrig(6:7,3);
        WPTTableTemp(8,1:3) = WPTTableOrig(8,1:3);
        WPTTableTemp(1:8,4) = WPTTableOrig(1:8,4); % Airspeed
end

% The mission consists of 5 laps
WPTTable = [WPTTableTemp; WPTTableTemp; WPTTableTemp; WPTTableTemp; WPTTableTemp];

save WPTTable WPTTable
S = load('WPTTable.mat');
WPTTable = S.WPTTable;

% Run simulation with the given waypoints
sim('aerosonde_mission_2obj');

```

```

    % Extract objective function value
    S = load('objFunc');
    objFunc = S.objFunc;
    objFuncVal = objFunc(2,length(objFunc));

end

end

```

F.2 Mission Scenario 2

F.2.1 *twoOptiPareto* Function

```

function [history,T1,WPTTableOpti,misionOutputs,T2] = twoOptiPareto(numWptOpti,w1,w2)
% Runs the fmincon optimisation process using the AeroSim simulation model
% to evaluate the given WPTTable in terms of a combination of fuel
% consumption AND flight time (two objectives).
%
% This combination of the two objectives uses an aggregate method, in which
% different weights are used with each objective, and are then added
% together to form ONE overall objective.
%
% Inputs:
% - numWptOpti = number of waypoints to be optimised
% - w1         = weight coefficient for objective #1
% - w2         = weight coefficient for objective #2
%
% Outputs:
% - history    = structure that keeps track of the history of fuel
%              consumptions, flight times, objective function values
%              and function evaluations
% - T1        = summary table of the fmincon process
% - WPTTableOpti = optimised mission waypoint table
% - misionOutputs = vector containing the optimised values for fuel
%              consumption (kg), flight time (min), and objective
%              function.
% -- fuel_consumption = amount of fule consumed during mission (kg)
% -- totalDist2HA    = time taken for the mission (hrs)
% - T2              = time taken

% Set up shared variable with OUTFUN
history.objFunc = [];
history.funccount = [];
history.iteration = [];

% Set up x0_history variable
x0_history = [];
save('x0_history','x0_history')

% Start optimisation
startSim = 1;
save startSim;

% Input values
maxIters = 20; % Maximum number of iterations evaluated

% Start clock
tic

% Baseline waypoint table (in radians)
WPTTableOrig = [...
    (26+33.7693/60)*pi/180 (151+51.1585/60)*pi/180 900 20;...
    (26+33.9588/60)*pi/180 (151+51.5236/60)*pi/180 900 20;...
    (26+35.0415/60)*pi/180 (151+53.6234/60)*pi/180 750 30;...
    (26+34.8070/60)*pi/180 (151+53.7740/60)*pi/180 750 20;...

```

```

(26+34.5725/60)*pi/180 (151+53.6234/60)*pi/180 750 20;...
(26+33.9588/60)*pi/180 (151+51.5236/60)*pi/180 900 20;...
(26+34.2132/60)*pi/180 (151+51.4206/60)*pi/180 900 20;...
26.5808*pi/180 151.8411*pi/180 800 30];
save WPTTableOrig;

% Baseline waypoint table (in radians)
% -- only WPTs to be optimised ---
WPTTableOpt = [WPTTableOrig(1:2,1:3); WPTTableOrig(6:7,1:3)]; % Table of WPTs to be optimised
latOpt = WPTTableOpt(:,1);
lonOpt = WPTTableOpt(:,2);

% Calculate Upper Bounds (in radians)
% -- only WPTs 2-3, 7-8 to be optimised --
latUB = ones(numWptOpti,1)*(26+34.5393/60)*pi/180;
lonUB = ones(numWptOpti,1)*(151+52.6085/60)*pi/180;

% Calculate Lower Bounds (in radians)
% -- only WPTs 2-3, 7-8 to be optimised --
latLB = ones(numWptOpti,1)*(26+33.727/60)*pi/180;
lonLB = ones(numWptOpti,1)*(151+51.103/60)*pi/180;

% Initialise values
numCoord = 2; % Number of coordinates per waypoint
numVarOpti = numWptOpti * numCoord; % Number of variables to be optimised
x0 = zeros(numVarOpti,1); % Starting points for optimisation
lb = zeros(numVarOpti,1); % Lower bounds
ub = zeros(numVarOpti,1); % Upper bounds

% Set up the vector of starting points for optimisation
% -- only WPTs 2-3, 7-8 to be optimised --
x0(1:numCoord:numVarOpti,1) = latOpt(1:numWptOpti);
x0(2:numCoord:numVarOpti,1) = lonOpt(1:numWptOpti);

% Set up the upper bounds vector
% -- only WPTs 2-3, 7-8 to be optimised --
ub(1:numCoord:numVarOpti,1) = latUB(1:numWptOpti);
ub(2:numCoord:numVarOpti,1) = lonUB(1:numWptOpti);

% Set up the lower bounds vector
% -- only WPTs 2-3, 7-8 to be optimised --
lb(1:numCoord:numVarOpti,1) = latLB(1:numWptOpti);
lb(2:numCoord:numVarOpti,1) = lonLB(1:numWptOpti);

% Set up the options required for fmincon
options = optimset('OutputFcn',@outfun,'Display','iter',...
'DiffMinChange',1e-5,'DiffMaxChange',1e-3,'MaxIter',maxIters,...
'TolX',1e-6);

% Run fmincon optimisation process
% -- WPTTableOpti = the optimised waypoints
[T1,WPTTableOpti,objFuncVal,output] = evalc('fmincon(@run_model_2obj,x0,[],[],[],[],lb,ub,@distconstr,options)');

out1 = load('fuel_cons');
fuel_cons = out1.fuel_cons;
fuelConsumption = fuel_cons(2,length(fuel_cons));
out2 = load('minDist2A1')
minDist2A1 = out2.minDist2A1;
minDistToA1 = minDist2A1(2,length(minDist2A1));
out3 = load('minDist2A2')
minDist2A2 = out3.minDist2A2;
minDistToA2 = minDist2A2(2,length(minDist2A2));

missionOutputs = [fuelConsumption, minDistToA1, minDistToA2, objFuncVal];

% Stop clock
T2 = evalc('toc');

% *****
% ** Set up 2objOptiResults file **
% *****

% Setting up filename

```

```

filenameStem = '2objOptiResults_';
filenameTxtEnd = '.txt';
filenameMATEnd = '.mat';
filenameNo = num2str(w1/0.05);
filenameTxt = strcat(filenameStem, filenameNo, filenameTxtEnd);
filenameMAT = strcat(filenameStem, filenameNo, filenameMATEnd);

save(filenameMAT, 'history', 'T1', 'WPTTableOpti', 'missionOutputs', 'T2')

fid = fopen(filenameTxt, 'wt');
fprintf(fid, 'Two-objective Optimisation Settings:\n\n');
fprintf(fid, '*****\n');
fprintf(fid, 'The initial starting point is at (0.463922478rad, 2.650127135rad, 850)\n\n');
fprintf(fid, 'The original waypoint table used for this mission is:\n\n');
for i = 1 : length(WPTTableOrig)
    fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
end
fprintf(fid, '\n');
fprintf(fid, 'The number of waypoints optimised : %d \n\n', numWptOpti);
fprintf(fid, 'The initial guess for the waypoints to be optimised : \n\n');
i = 1
for i = 1 : numVarOpti
    if (rem(i,numCoord) == 1)
        fprintf(fid, '|%10.9frad ', x0(i,1));
    else
        fprintf(fid, ' %10.9frad|\n', x0(i,1));
    end
end
fprintf(fid, '\n');
fprintf(fid, 'The candidate coordinates are taken from an area bounded by: \n\n');
fprintf(fid, '--- Latitudes of %10.9frad to %10.9frad \n\n', (26+33.8/60)*pi/180, (26+34.5/60)*pi/180);
fprintf(fid, '--- Longitudes of %10.9frad to %10.9frad \n\n', (151.8411)*pi/180, (151+54.5082/60)*pi/180);
fprintf(fid, '*****\n');
fprintf(fid, 'The results of the mission optimisation process is as follows: \n\n');
fprintf(fid, 'The optimised mission waypoint table is : \n\n');
switch numWptOpti
    case 1
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));
        i = 2;
        for i = 2 : 8
            fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
        end
    case 2
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(3), WPTTableOpti(4), WPTTableOrig(2,3));
        i = 3;
        for i = 3 : 8
            fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
        end
    case 3
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(3), WPTTableOpti(4), WPTTableOrig(2,3));
        i = 3;
        for i = 3 : 5
            fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
        end
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(5), WPTTableOpti(6), WPTTableOrig(6,3));
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(7,1), WPTTableOrig(7,2), WPTTableOrig(7,3));
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(8,1), WPTTableOrig(8,2), WPTTableOrig(8,3));
    case 4
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(1), WPTTableOpti(2), WPTTableOrig(1,3));
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(3), WPTTableOpti(4), WPTTableOrig(2,3));
        i = 3;
        for i = 3 : 5
            fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(i,1), WPTTableOrig(i,2), WPTTableOrig(i,3));
        end
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(5), WPTTableOpti(6), WPTTableOrig(6,3));
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOpti(7), WPTTableOpti(8), WPTTableOrig(7,3));
        fprintf(fid, '|%10.9frad %10.9frad %6.3fm|\n', WPTTableOrig(8,1), WPTTableOrig(8,2), WPTTableOrig(8,3));
end

fprintf(fid, '\n');
fprintf(fid, '*****\n');

```

```

fprintf(fid, 'The process table T1 is : \n\n');
fprintf(fid, '%s', T1);
fprintf(fid, '\n');
fprintf(fid, '*****\n');
fprintf(fid, 'The fuel consumption for this mission is %8.6fkg\n\n', fuelConsumption);
fprintf(fid, 'The minimum distance to Hazard Area 1 is %8.6fm\n\n', minDistToA1);
fprintf(fid, 'The minimum distance to Hazard Area 2 is %8.6fm\n\n', minDistToA2);
fprintf(fid, '*****\n');
fprintf(fid, '%s', T2);
fclose(fid)

function stop = outfun(x,optimValues,state)
    % Function for generation the vector 'history'
    % --- Keeps track of the history of the objective function values and
    % function evaluations ---
    stop = false;

    switch state
        case 'iter'
            % Concatenate current objective function value and function
            % evaluation value with history
            history.objFunc = [history.objFunc,optimValues.fval];
            history.funccount = [history.funccount,optimValues.funccount];
            history.iteration = [history.iteration,optimValues.iteration];
        otherwise
            end
    end
end

function objFuncVal = run_model_2obj(x0)
    % Calls the AeroSim model 'aerosonde_mission_v3_2obj.mdl' to run
    % through the specified flight mission, and obtains the objective
    % function value for the mission
    %
    % INPUT:
    % - x0 = The waypoints to be optimised in the format
    %       |lat|
    %       |lon|fprintf(fid, 'The minimum distance to Hazard Area 1 is \n\n', minDist2A1);
    %
    % OUTPUT:
    % - objFunc = value of the objective function, a combination of fuel
    %             consumption and flight time

    % Initialise
    WPTTable = zeros(40,4); % Set up WPTTable
    WPTTableTemp = zeros(8,4); % Set up WPTTableTemp

    % Rearranging coordinates from x0 into WPTTable format
    % [lat lon alt] <-- 'alt' is taken from WPTTableOrig
    % --- This is just for one lap of flight
    switch numWptOpti
        case 1
            % only 1 waypoint (WPT 2) is being optimised
            WPTTableTemp(1,1) = x0(1);
            WPTTableTemp(1,2) = x0(2);
            WPTTableTemp(1,3) = WPTTableOrig(1,3);
            WPTTableTemp(2:8,1:3) = WPTTableOrig(2:8,1:3);
            WPTTableTemp(1:8,4) = WPTTableOrig(1:8,4); % Airspeed
        case 2
            % 2 waypoints (WPT 2 & 3) are being optimised
            WPTTableTemp(1:2,1) = x0(1:2:4);
            WPTTableTemp(1:2,2) = x0(2:2:4);
            WPTTableTemp(1:2,3) = WPTTableOrig(1:2,3);
            WPTTableTemp(3:8,1:3) = WPTTableOrig(3:8,1:3);
            WPTTableTemp(1:8,4) = WPTTableOrig(1:8,4); % Airspeed
        case 3
            % 3 waypoints (WPTs 2, 3 & 7) are being optimised
            WPTTableTemp(1:2,1) = x0(1:2:4);
            WPTTableTemp(1:2,2) = x0(2:2:4);
            WPTTableTemp(1:2,3) = WPTTableOrig(1:2,3);
            WPTTableTemp(3:5,1:3) = WPTTableOrig(3:5,1:3);
            WPTTableTemp(6,1) = x0(5);
            WPTTableTemp(6,2) = x0(6);
            WPTTableTemp(6,3) = WPTTableOrig(6,3);
    end
end

```

```

    WPTTableTemp(7:8,1:3) = WPTTableOrig(7:8,1:3);
    WPTTableTemp(1:8,4) = WPTTableOrig(1:8,4); % Airspeed
case 4
    % 4 waypoints (WPTs 2, 3, 7 & 8) are being optimised
    WPTTableTemp(1:2,1) = x0(1:2:4);
    WPTTableTemp(1:2,2) = x0(2:2:4);
    WPTTableTemp(1:2,3) = WPTTableOrig(1:2,3);
    WPTTableTemp(3:5,1:3) = WPTTableOrig(3:5,1:3);
    WPTTableTemp(6:7,1) = x0(5:2:8);
    WPTTableTemp(6:7,2) = x0(6:2:8);
    WPTTableTemp(6:7,3) = WPTTableOrig(6:7,3);
    WPTTableTemp(8,1:3) = WPTTableOrig(8,1:3);
    WPTTableTemp(1:8,4) = WPTTableOrig(1:8,4); % Airspeed
end

% The mission consists of 5 laps
WPTTable = [WPTTableTemp; WPTTableTemp; WPTTableTemp; WPTTableTemp; WPTTableTemp];

save WPTTable WPTTable
S = load('WPTTable.mat');
WPTTable = S.WPTTable;

% Run simulation with the given waypoints
sim('aerosonde_mission_2obj');

% Extract objective function value
S = load('objFunc');
objFunc = S.objFunc;
objFuncVal = objFunc(2,length(objFunc));

end

function [c,ceq] = distconstr(x0)
% Sets the constraints for the waypoints to be optimised
% -- minimum distance from other fixed waypoints

% Rearranging coordinates from x0 into WPTTable format
% [lat lon alt] <-- 'alt' is taken from WPTTableOrig
% --- This is just for one lap of flight

% Define starting WPT (Kingaroy)
startLat = 26.5808*pi/180; % Latitude of Kingaroy
startLon = 151.8411*pi/180; % Longitude of Kingaroy

switch numWptOpti
case 1
    % only 1 waypoint (WPT 2) is being optimised
    % -- distance(WPT1 -> WPT2) >= 2000m
    c(1) = 2000 - CalcGCDist(startLat,startLon,x0(1),x0(2));
    % -- distance(WPT2 -> WPT3) >= 300m
    c(2) = 300 - CalcGCDist(x0(1),x0(2),WPTTableOrig(2,1),WPTTableOrig(2,2));
    % No nonlinear equality constraints
    ceq = [];
case 2
    % 2 waypoints (WPT 2 & 3) are being optimised
    % -- distance(WPT1 -> WPT2) >= 2000m
    c(1) = 2000 - CalcGCDist(startLat,startLon,x0(1),x0(2));
    % -- distance(WPT2 -> WPT3) >= 300m
    c(2) = 300 - CalcGCDist(x0(1),x0(2),x0(3),x0(4));
    % -- distance(WPT3 -> WPT4) >= 3000m
    c(3) = 3000 - CalcGCDist(x0(3),x0(4),WPTTableOrig(3,1),WPTTableOrig(3,2));
    % No nonlinear equality constraints
    ceq = [];
case 3
    % 3 waypoints (WPTs 2, 3 & 7) are being optimised
    % -- distance(WPT1 -> WPT2) >= 2000m
    c(1) = 2000 - CalcGCDist(startLat,startLon,x0(1),x0(2));
    % -- distance(WPT2 -> WPT3) >= 300m
    c(2) = 300 - CalcGCDist(x0(1),x0(2),x0(3),x0(4));
    % -- distance(WPT3 -> WPT4) >= 3000m
    c(3) = 3000 - CalcGCDist(x0(3),x0(4),WPTTableOrig(3,1),WPTTableOrig(3,2));
    % -- distance(WPT6 -> WPT7) >= 3000m
    c(4) = 3000 - CalcGCDist(WPTTableOrig(5,1),WPTTableOrig(5,2),x0(5),x0(6));

```



```

    % -- distance(WPT7 -> WPT8) >= 300m
    c(5) = 300 - CalcGCDist(x0(5),x0(6),WPTTableOrig(7,1),WPTTableOrig(7,2));
    % No nonlinear equality constraints
    ceq = [];
case 4
    % 4 waypoints (WPTs 2, 3, 7 & 8) are being optimised
    % -- distance(WPT1 -> WPT2) >= 2000m
    c(1) = 2000 - CalcGCDist(startLat,startLon,x0(1),x0(2));
    % -- distance(WPT2 -> WPT3) >= 300m
    c(2) = 300 - CalcGCDist(x0(1),x0(2),x0(3),x0(4));
    % -- distance(WPT3 -> WPT4) >= 3000m
    c(3) = 3000 - CalcGCDist(x0(3),x0(4),WPTTableOrig(3,1),WPTTableOrig(3,2));
    % -- distance(WPT6 -> WPT7) >= 3000m
    c(4) = 3000 - CalcGCDist(WPTTableOrig(5,1),WPTTableOrig(5,2),x0(5),x0(6));
    % -- distance(WPT7 -> WPT8) >= 300m
    c(5) = 300 - CalcGCDist(x0(5),x0(6),x0(7),x0(8));
    % -- distance(WPT8 -> WPT9) >= 1700m
    c(6) = 1700 - CalcGCDist(x0(7),x0(8),WPTTableOrig(8,1),WPTTableOrig(8,2));
    % No nonlinear equality constraints
    ceq = [];
end
end
end

```

F.2.2 Updated *Navigate Waypoints* Function

```

function Navigate = NavigateWaypoints(Argument)
% This function performs waypoint navigation

% Load waypoint list
S = load ('WPTTable.mat');
WPTTable = S.WPTTable;

% Number of waypoints is table
SizeOfTable = size(WPTTable);
NoOfWPT = SizeOfTable(1);

% Allocating inputs
CurrentWaypoint = Argument(1);
CurrentLatitude = Argument(2);
CurrentLongitude = Argument(3);
CurrentAltitude = Argument(4);
CurrentAirspeed = Argument(5);
PrevMinDist2A1 = Argument(6);
PrevMinDist2A2 = Argument(7);

% Initialise
if (CurrentWaypoint == 0)
    CurrentWaypoint = 1;
end
EndOfMission = 0;

% Get current waypoint coordinates
WPTLat = WPTTable(CurrentWaypoint, 1);
WPTLon = WPTTable(CurrentWaypoint, 2);
WPTAlt = WPTTable(CurrentWaypoint, 3);
WPTAirspeed = WPTTable(CurrentWaypoint, 4);
[Bearing, Distance] = CalculateGC(CurrentLatitude, CurrentLongitude, WPTLat, WPTLon);
Waypoint = CurrentWaypoint;

XTE = 0;

% A waypoint is captured when the aircraft comes with 20m of it
if (Distance > 20)
    if (CurrentWaypoint ~= 1) % Not the first waypoint in table
        LastWPTLat = WPTTable(CurrentWaypoint-1, 1);
        LastWPTLon = WPTTable(CurrentWaypoint-1, 2);
        LastWPTAlt = WPTTable(CurrentWaypoint-1, 3);
    end
end

```

```

    [RequiredTrack, TrackDistance] = CalculateGC>LastWPTLat, LastWPTLon, WPTLat, WPTLon);
    TrackError = RequiredTrack - Bearing;
    XTE = Distance * sin(TrackError);
    EndOfMission = 0;
else
    LastWPTLat = 26.5808*pi/180;    % Latitude of starting point (Kingaroy)
    LastWPTLon = 151.8411*pi/180;  % Longitude of starting point (Kingaroy)
    LastWPTAlt = 850;              % Altitude of starting point (Kingaroy @ 850m)
    [RequiredTrack, TrackDistance] = CalculateGC>LastWPTLat, LastWPTLon, WPTLat, WPTLon);
    TrackError = RequiredTrack - Bearing;
    XTE = Distance * sin(TrackError);
    EndOfMission = 0;
end
else % (Distance < 20)
    if (CurrentWaypoint == NoOfWPT)    % Last waypoint in table
        if (Distance < 5)
            LastWPTLat = WPTTable(CurrentWaypoint, 1);
            LastWPTLon = WPTTable(CurrentWaypoint, 2);
            LastWPTAlt = WPTTable(CurrentWaypoint, 3);
            [RequiredTrack, TrackDistance] = CalculateGC>LastWPTLat, LastWPTLon, WPTLat, WPTLon);
            TrackError = RequiredTrack - Bearing;
            XTE = Distance * sin(TrackError);
            EndOfMission = 1;
        end
    else % Other waypoints in table
        WPTLat = WPTTable(CurrentWaypoint+1, 1);
        WPTLon = WPTTable(CurrentWaypoint+1, 2);
        WPTAlt = WPTTable(CurrentWaypoint+1, 3);
        LastWPTLat = WPTTable(CurrentWaypoint, 1);
        LastWPTLon = WPTTable(CurrentWaypoint, 2);
        LastWPTAlt = WPTTable(CurrentWaypoint, 3);
        [RequiredTrack, TrackDistance] = CalculateGC>LastWPTLat, LastWPTLon, WPTLat, WPTLon);
        TrackError = RequiredTrack - Bearing;
        XTE = Distance * sin(TrackError);
        Waypoint = CurrentWaypoint + 1;
        EndOfMission = 0;
    end
end

% Target airspeed when reached target altitude at descent
if ((Waypoint == 3) || (Waypoint == 8) || (Waypoint == 11) ...
    || (Waypoint == 16) || (Waypoint == 19) || (Waypoint == 24) ...
    || (Waypoint == 27) || (Waypoint == 32) || (Waypoint == 35) ...
    || (Waypoint == 40))
    if (CurrentAltitude > WPTAlt + 10)
        TargetAirspeed = WPTAirspeed;
    elseif (CurrentAirspeed > 21)
        TargetAirspeed = CurrentAirspeed - 2;
    else
        TargetAirspeed = 20;
    end
else
    TargetAirspeed = WPTAirspeed;
end

% Calculate the distances to nearest point in Hazard Areas 1 and 2
% respectively, and compare to previous distances to obtain the minimum
% distances
PrevDistancesToHA = [PrevMinDist2A1, PrevMinDist2A2];
distances = distToHazard(CurrentLatitude, CurrentLongitude, PrevDistancesToHA);
minDist2A1 = distances(1);
minDist2A2 = distances(2);

% Output of function
Navigate = [Bearing, WPTAlt, TargetAirspeed, XTE, Waypoint, EndOfMission, minDist2A1, minDist2A2];

```

F.2.3 *distToHazard* Function

```
function distances = distToHazard(CurrentLatitude, CurrentLongitude, prevDistances)
```

```

% This function determines the distance from the current position to each
% of the hazard areas, A1 and A2 (using the given coordinates), as well as
% the minimum distances to these A1 and A2.
%
% Inputs:
% - CurrentLatitude = latitude of current position (rad)
% - CurrentLongitude = longitude of current position (rad)
% - prevDistances = minimum distances from the previous time step (m)
%
% Outputs:
% - minDist_to_A1 = minimum distance between the aircraft and Area 1 during
%                 the entire mission (m)
% - minDist_to_A2 = minimum distance between the aircraft and Area 2 during
%                 the entire mission (m)

% Set up previous distances
prevMinDistToA1 = prevDistances(1);
prevMinDistToA2 = prevDistances(2);

latLB_A = (26+33.778/60)*pi/180;
latUB_A = (26+33.9405/60)*pi/180;
lonLB_A = (151+50.842/60)*pi/180;
lonUB_A = (151+51.0407/60)*pi/180;

if (CurrentLatitude > latUB_A)
    if (CurrentLongitude < lonLB_A)
        % A1S1
        dist_to_A1 = CalcGCDist(CurrentLatitude,CurrentLongitude,latUB_A,lonLB_A);
    elseif ((CurrentLongitude >= lonLB_A) && (CurrentLongitude <= lonUB_A))
        % A1S2
        dist_to_A1 = CalcGCDist(CurrentLatitude,CurrentLongitude,latUB_A,CurrentLongitude);
    elseif (CurrentLongitude > lonUB_A)
        % A1S3
        dist_to_A1 = CalcGCDist(CurrentLatitude,CurrentLongitude,latUB_A,lonUB_A);
    end
elseif ((CurrentLatitude >= latLB_A) && (CurrentLatitude <= latUB_A))
    if (CurrentLongitude < lonLB_A)
        % A1S4
        dist_to_A1 = CalcGCDist(CurrentLatitude,CurrentLongitude,CurrentLatitude,lonLB_A);
    elseif (CurrentLongitude > lonUB_A)
        % A1S5
        dist_to_A1 = CalcGCDist(CurrentLatitude,CurrentLongitude,CurrentLatitude,lonUB_A);
    end
elseif (CurrentLatitude < latLB_A)
    if (CurrentLongitude < lonLB_A)
        % A1S6
        dist_to_A1 = CalcGCDist(CurrentLatitude,CurrentLongitude,latLB_A,lonLB_A);
    elseif ((CurrentLongitude >= lonLB_A) && (CurrentLongitude <= lonUB_A))
        % A1S7
        dist_to_A1 = CalcGCDist(CurrentLatitude,CurrentLongitude,latLB_A,CurrentLongitude);
    elseif (CurrentLongitude > lonUB_A)
        % A1S8
        dist_to_A1 = CalcGCDist(CurrentLatitude,CurrentLongitude,latLB_A,lonUB_A);
    end
end

% Determine minimum distance to Area 1
minDist_to_A1 = min(dist_to_A1,prevMinDistToA1);

latLB_B = (26+34.8484/60)*pi/180;
latUB_B = (26+34.973/60)*pi/180;
lonLB_B = (151+52.443/60)*pi/180;
lonUB_B = (151+52.6538/60)*pi/180;

if (CurrentLatitude > latUB_B)
    if (CurrentLongitude < lonLB_B)
        % A2S1
        dist_to_A2 = CalcGCDist(CurrentLatitude,CurrentLongitude,latUB_B,lonLB_B);
    elseif ((CurrentLongitude >= lonLB_B) && (CurrentLongitude <= lonUB_B))
        % A2S2
        dist_to_A2 = CalcGCDist(CurrentLatitude,CurrentLongitude,latUB_B,CurrentLongitude);
    elseif (CurrentLongitude > lonUB_B)
        % A2S3

```

```
        dist_to_A2 = CalcGCDist(CurrentLatitude,CurrentLongitude,latUB_B,lonUB_B);
    end
elseif ((CurrentLatitude >= latLB_B) && (CurrentLatitude <= latUB_B))
    if (CurrentLongitude < lonLB_B)
        % A2S4
        dist_to_A2 = CalcGCDist(CurrentLatitude,CurrentLongitude,CurrentLatitude,lonLB_B);
    elseif (CurrentLongitude > lonUB_B)
        % A2S5
        dist_to_A2 = CalcGCDist(CurrentLatitude,CurrentLongitude,CurrentLatitude,lonUB_B);
    end
elseif (CurrentLatitude < latLB_B)
    if (CurrentLongitude < lonLB_B)
        % A2S6
        dist_to_A2 = CalcGCDist(CurrentLatitude,CurrentLongitude,latLB_B,lonLB_B);
    elseif ((CurrentLongitude >= lonLB_B) && (CurrentLongitude <= lonUB_B))
        % A2S7
        dist_to_A2 = CalcGCDist(CurrentLatitude,CurrentLongitude,latLB_B,CurrentLongitude);
    elseif (CurrentLongitude > lonUB_B)
        % A2S8
        dist_to_A2 = CalcGCDist(CurrentLatitude,CurrentLongitude,latLB_B,lonUB_B);
    end
end

% Determine minimum distance to Area 2
minDist_to_A2 = min(dist_to_A2,prevMinDistToA2);

% Outputs
distances = [minDist_to_A1,minDist_to_A2];
```

Appendix G

Aerosonde ICE - Open Throttle Calculations

Static Pressure (kPa)	MAP								
	1500	2100	2800	3500	4500	5100	5500	6000	7000
101325	97.8314	240.3359	420.1356	558.1554	838.3910	1106.5913	1128.4414	1235.4415	1550.8231
100129.439	87.5077	225.8756	400.8572	534.0588	779.2346	1004.4305	1066.4694	1197.8770	1441.2798
98945.3256	77.2829	211.5537	381.7634	510.1930	720.6447	903.2481	1005.0910	1160.6721	1332.7855
97772.5771	68.9403	197.6195	359.8520	483.2211	661.5460	804.2512	932.5131	1100.2486	1217.8286
96611.1094	68.0228	184.8491	325.7977	442.7788	598.6060	711.2060	812.1037	943.3517	1073.0865
95460.8393	67.5400	173.3881	293.6515	402.7264	546.4334	637.7748	714.5915	815.0657	933.6945
94321.6841	67.5400	163.3692	263.5892	363.0610	506.1699	586.0628	642.4203	718.4369	800.0887
93193.5613	68.1771	156.9917	252.7331	341.5134	475.8003	554.2304	601.1371	658.2150	715.0664
92076.389	69.0597	152.0761	249.4542	327.1801	449.4797	530.3621	572.1800	612.5897	649.5498
90970.0858	68.3064	150.6071	243.1896	316.7660	442.8240	512.2308	552.4031	586.8222	614.8729
89874.5705	67.5203	149.2642	236.8749	307.0984	437.0087	495.3502	533.9784	563.6773	584.3153
88789.7625	67.3500	146.8787	231.4682	300.3390	426.2735	483.7622	519.6080	547.9996	566.0254
87715.5816	67.1813	144.5166	226.1145	293.6458	415.6434	472.2878	505.3783	532.4756	547.9147
86641.9479	67.0144	142.1776	220.8133	287.0183	405.1177	460.9261	491.2884	517.1040	529.9818
85598.7819	66.8490	139.8617	215.5643	280.4560	394.6955	449.6762	477.3371	501.8836	512.2255
84556.0048	66.6853	137.5687	210.3671	273.9585	384.3762	438.5372	463.5234	486.8134	494.6442
83523.538	66.5232	135.2983	205.2213	267.5252	374.1589	427.5084	449.8463	471.8922	477.2369
82501.3031	66.3627	133.0504	200.1265	261.1556	364.0429	416.5889	436.3048	457.1188	460.0020
81489.2226	66.2038	130.8248	195.0823	254.8493	354.0273	405.7779	422.8977	442.4922	442.9383
80487.2191	66.0465	128.6214	190.0883	248.6059	344.1115	395.0745	409.6242	428.0113	426.0445
79495.2155	65.0185	126.1068	185.7358	241.3147	334.5324	382.5910	397.6461	413.6754	409.3193
78513.1354	63.1673	123.2991	181.9921	233.0357	325.2763	368.4294	386.8992	399.4833	392.7615
77540.9027	61.3346	120.5194	178.2859	224.8398	316.1130	354.4098	376.2601	385.4336	376.3696
76578.4417	59.5204	117.7678	174.6170	216.7263	307.0418	340.5311	365.7279	371.5251	360.1425
75625.677	57.7244	115.0438	170.9851	208.6945	298.0620	326.7923	355.3018	357.7567	344.0789
74682.5337	55.9466	112.3474	167.3898	200.7438	289.1729	313.1921	344.9810	344.1273	328.1775
73748.9373	54.1867	109.6782	163.8309	192.8735	280.3737	299.7297	334.7646	330.6359	312.4371
72824.8137	52.4448	107.0361	160.3082	185.0832	271.6639	286.4038	324.6519	317.2814	296.8564
71910.0892	50.7205	104.4209	156.8213	177.3721	263.0426	273.2135	314.6421	304.0627	281.4341
71004.6905	49.0138	101.8324	153.3699	169.7395	254.5092	260.1576	304.7343	290.9788	266.1691
70108.5447	47.3246	99.2703	149.9538	162.1850	246.0630	247.2352	294.9278	278.0286	251.0601
69221.5792	44.9194	95.8790	145.2034	157.2759	238.8041	240.6811	285.2217	267.6561	239.5301
68343.722	42.4377	92.4045	140.3129	152.7716	231.7716	235.0549	275.6153	257.7275	228.5911
67474.9012	39.9815	88.9657	135.4727	148.3137	224.8114	229.4866	266.1078	247.9011	217.7647
66615.0457	37.5507	85.5624	130.6824	143.9018	217.9231	223.9758	256.6984	238.1762	207.0501
65764.0844	35.1451	82.1942	125.9417	139.5355	211.1061	218.5220	247.3864	228.5518	196.4463
64921.9467	32.7631	78.8611	121.2502	135.2145	204.3597	213.1248	238.1709	219.0272	185.9524
64088.5626	30.4084	75.5625	116.6074	130.9384	197.6835	207.7836	229.0511	209.6016	175.5676
63263.8621	28.0769	72.2984	112.0130	126.7069	191.0768	202.4981	220.0264	200.2743	165.2910
62447.7761	25.7699	69.0683	107.4666	122.5195	184.5391	197.2678	211.0960	191.0443	155.1217
61640.2353	23.4869	65.8721	102.9678	118.3760	178.0699	192.0923	202.2591	181.9111	145.0590
60841.1712	21.2280	62.7094	98.5162	114.2760	171.6686	186.9711	193.5149	172.8736	135.1018
60050.5157	18.9928	59.5799	94.1114	110.2192	165.3347	181.9038	184.8628	163.9313	125.2495

Table G.1: Look-up table for Engine power (open throttle calculations)

Static Pressure (kPa)	MAP								
	1500	2100	2800	3500	4500	5100	5500	6000	7000
101325	87.9625	116.3000	158.3000	199.9375	255.2750	343.1250	330.2250	344.2875	413.3000
100129.439	82.5825	111.5178	153.5178	190.9708	246.9061	313.2360	314.6827	337.7119	408.5178
98945.3256	77.2540	106.7813	148.7813	182.0899	238.6173	283.6331	299.2892	331.1993	403.7813
97772.5771	72.0903	101.7492	142.8395	172.1572	227.3378	254.0870	280.7459	317.6991	391.0168
96611.1094	67.4444	95.3611	131.8055	157.6389	203.5277	223.8888	248.8055	275.3055	345.1388
95460.8393	62.8434	88.7650	118.4518	143.7997	183.1822	200.9909	220.6776	240.3297	300.2423
94321.6841	58.2867	81.9301	102.5036	130.6994	166.6644	186.1819	196.7554	213.5596	256.3848
93193.5613	56.1936	77.5807	96.7903	122.9678	156.3549	175.9517	185.1614	199.9517	233.1131
92076.389	55.0764	74.2292	95.1146	117.3819	148.5347	167.5729	178.4583	191.5729	218.0313
90970.0858	53.9701	71.4252	93.4551	113.3953	145.4252	162.8803	173.3654	187.3953	213.3953
89874.5705	52.9122	68.8119	91.7115	109.7115	142.6488	158.6363	168.5986	183.5861	209.5485
88789.7625	52.1528	67.1846	89.2165	107.2165	139.6113	155.4903	165.1272	180.0062	205.6431
87715.5816	51.4009	65.5734	86.7458	104.7458	136.6036	152.3752	161.6899	176.4614	201.7761
86641.9479	50.6564	63.9779	84.2995	102.2995	133.6255	149.2906	158.2862	172.9514	197.9470
85598.7819	49.9191	62.3982	81.8772	99.8772	130.6766	146.2365	154.9161	169.4760	194.1556
84556.0048	49.1892	60.8340	79.4788	97.4788	127.7568	143.2124	151.5792	166.0348	190.4016
83523.538	48.4665	59.2853	77.1041	95.1041	124.8659	140.2183	148.2753	162.6277	186.6847
82501.3031	47.7509	57.7520	74.7530	92.7530	122.0036	137.2538	145.0042	159.2543	183.0047
81489.2226	47.0425	56.2338	72.4252	90.4252	119.1698	134.3187	141.7655	155.9144	179.3612
80487.2191	46.3411	54.7308	70.1206	88.1206	116.3642	131.4129	138.5591	152.6078	175.7540
79495.2155	45.2933	53.4952	68.6971	86.3943	114.1419	128.5866	136.0409	149.7380	172.4856
78513.1354	43.9184	52.5131	68.1079	85.2158	112.4723	125.8368	134.1750	147.2828	169.5394
77540.9027	42.5573	51.5409	67.5245	84.0491	110.8195	123.1145	132.3277	144.8523	166.6227
76578.4417	41.2098	50.5784	66.9471	82.8941	109.1834	120.4196	130.4990	142.4461	163.7353
75625.677	39.8759	49.6257	66.3754	81.7508	107.5637	117.7519	128.6888	140.0642	160.8770
74682.5337	38.5555	48.6825	65.8095	80.6190	105.9603	115.1111	126.8968	137.7063	158.0476
73748.9373	37.2485	47.7489	65.2494	79.4987	104.3732	112.4970	125.1230	135.3723	155.2468
72824.8137	35.9547	46.8248	64.6949	78.3898	102.8022	109.9095	123.3671	133.0620	152.4744
71910.0892	34.6741	45.9101	64.1461	77.2921	101.2472	107.3482	121.6292	130.7752	149.7303
71004.6905	33.4066	45.0047	63.6028	76.2056	99.7080	104.8131	119.9089	128.5117	147.0141
70108.5447	32.1520	44.1085	63.0651	75.1303	98.1845	102.3039	118.2062	126.2714	144.3256
69221.5792	31.9222	43.6886	62.9881	74.2994	96.8324	101.2994	116.5988	124.2875	142.3653
68343.722	31.8344	43.3375	60.8468	73.5093	95.5156	100.5093	115.0187	122.3562	140.5218
67474.9012	31.7475	42.9900	59.7174	72.7274	94.2124	99.7274	113.4548	120.4448	138.6973
66615.0457	31.6615	42.6460	58.5996	71.9535	92.9226	98.9535	111.9071	118.5531	136.8916
65764.0844	31.5764	42.3056	57.4933	71.1877	91.6461	98.1877	110.3754	116.6810	135.1046
64921.9467	31.4922	41.9688	56.3985	70.4298	90.3829	97.4298	108.8595	114.8283	133.3361
64088.5626	31.4089	41.6354	55.3151	69.6797	89.1328	96.6797	107.3594	112.9948	131.5860
63263.8621	31.3264	41.3055	54.2430	68.9375	87.8958	95.9375	105.8750	111.1805	129.8541
62447.7761	31.2448	40.9791	53.1821	68.2030	86.6717	95.2030	104.4060	109.3851	128.1403
61640.2353	31.1640	40.6561	52.1323	67.4762	85.4604	94.4762	102.9524	107.6085	126.4445
60841.1712	31.0841	40.3365	51.0935	66.7571	84.2618	93.7571	101.5141	105.8506	124.7665
60050.5157	31.0051	40.0202	50.0657	66.0455	83.0758	93.0455	100.0909	104.1111	123.1061

Table G.2: Look-up table for Engine fuel flow (open throttle calculations)

Static Pressure (kPa)	MAP								
	1500	2100	2800	3500	4500	5100	5500	6000	7000
101325	0.6228	1.0929	1.4329	1.5229	1.7791	2.0720	1.9592	1.9663	2.1156
100129.439	0.5571	1.0271	1.3671	1.4571	1.6536	1.8807	1.8516	1.9065	1.9662
98945.3256	0.4920	0.9620	1.3020	1.3920	1.5293	1.6913	1.7451	1.8473	1.8182
97772.5771	0.4389	0.8986	1.2273	1.3184	1.4038	1.5059	1.6191	1.7511	1.6613
96611.1094	0.4330	0.8406	1.1111	1.2081	1.2703	1.3317	1.4100	1.5014	1.4639
95460.8393	0.4300	0.7884	1.0015	1.0988	1.1596	1.1942	1.2407	1.2972	1.2737
94321.6841	0.4300	0.7429	0.8990	0.9906	1.0741	1.0974	1.1154	1.1434	1.0915
93193.5613	0.4340	0.7139	0.8619	0.9318	1.0097	1.0377	1.0437	1.0476	0.9755
92076.389	0.4396	0.6915	0.8508	0.8927	0.9538	0.9931	0.9934	0.9750	0.8861
90970.0858	0.4349	0.6849	0.8294	0.8643	0.9397	0.9591	0.9591	0.9340	0.8388
89874.5705	0.4298	0.6787	0.8079	0.8379	0.9274	0.9275	0.9271	0.8971	0.7971
88789.7625	0.4288	0.6679	0.7894	0.8194	0.9046	0.9058	0.9022	0.8722	0.7722
87715.5816	0.4277	0.6572	0.7712	0.8012	0.8820	0.8843	0.8775	0.8475	0.7475
86641.9479	0.4266	0.6465	0.7531	0.7831	0.8597	0.8630	0.8530	0.8230	0.7230
85598.7819	0.4256	0.6360	0.7352	0.7652	0.8376	0.8420	0.8288	0.7988	0.6988
84556.0048	0.4245	0.6256	0.7174	0.7475	0.8157	0.8211	0.8048	0.7748	0.6748
83523.538	0.4235	0.6152	0.6999	0.7299	0.7940	0.8005	0.7810	0.7510	0.6510
82501.3031	0.4225	0.6050	0.6825	0.7125	0.7725	0.7800	0.7575	0.7275	0.6275
81489.2226	0.4215	0.5949	0.6653	0.6953	0.7513	0.7598	0.7343	0.7042	0.6042
80487.2191	0.4205	0.5849	0.6483	0.6783	0.7302	0.7397	0.7112	0.6812	0.5812
79495.2155	0.4139	0.5734	0.6334	0.6584	0.7099	0.7164	0.6904	0.6584	0.5584
78513.1354	0.4021	0.5607	0.6207	0.6358	0.6903	0.6899	0.6717	0.6358	0.5358
77540.9027	0.3905	0.5480	0.6080	0.6134	0.6708	0.6636	0.6533	0.6134	0.5134
76578.4417	0.3789	0.5355	0.5955	0.5913	0.6516	0.6376	0.6350	0.5913	0.4913
75625.677	0.3675	0.5231	0.5831	0.5694	0.6325	0.6119	0.6169	0.5694	0.4694
74682.5337	0.3562	0.5109	0.5709	0.5477	0.6136	0.5864	0.5990	0.5477	0.4477
73748.9373	0.3450	0.4987	0.5587	0.5262	0.5950	0.5612	0.5812	0.5262	0.4262
72824.8137	0.3339	0.4867	0.5467	0.5050	0.5765	0.5363	0.5637	0.5050	0.4050
71910.0892	0.3229	0.4748	0.5348	0.4839	0.5582	0.5116	0.5463	0.4839	0.3839
71004.6905	0.3120	0.4631	0.5231	0.4631	0.5401	0.4871	0.5291	0.4631	0.3631
70108.5447	0.3013	0.4514	0.5114	0.4425	0.5222	0.4629	0.5121	0.4425	0.3425
69221.5792	0.2860	0.4360	0.4952	0.4291	0.5068	0.4507	0.4952	0.4260	0.3268
68343.722	0.2702	0.4202	0.4785	0.4168	0.4918	0.4401	0.4785	0.4102	0.3118
67474.9012	0.2545	0.4046	0.4620	0.4047	0.4771	0.4297	0.4620	0.3945	0.2971
66615.0457	0.2391	0.3891	0.4457	0.3926	0.4624	0.4194	0.4457	0.3791	0.2825
65764.0844	0.2237	0.3738	0.4295	0.3807	0.4480	0.4092	0.4295	0.3638	0.2680
64921.9467	0.2086	0.3586	0.4135	0.3689	0.4337	0.3991	0.4135	0.3486	0.2537
64088.5626	0.1936	0.3436	0.3977	0.3572	0.4195	0.3891	0.3977	0.3336	0.2395
63263.8621	0.1787	0.3288	0.3820	0.3457	0.4055	0.3792	0.3820	0.3187	0.2255
62447.7761	0.1641	0.3141	0.3665	0.3343	0.3916	0.3694	0.3665	0.3041	0.2116
61640.2353	0.1495	0.2995	0.3512	0.3230	0.3779	0.3597	0.3512	0.2895	0.1979
60841.1712	0.1351	0.2852	0.3360	0.3118	0.3643	0.3501	0.3360	0.2751	0.1843
60050.5157	0.1209	0.2709	0.3210	0.3007	0.3509	0.3406	0.3210	0.2609	0.1709

Table G.3: Look-up table for Engine torque (open throttle calculations)

Static Pressure (kPa)	MAP								
	1500	2100	2800	3500	4500	5100	5500	6000	7000
101325	0.8991	0.4839	0.3768	0.3582	0.3045	0.3101	0.2926	0.2787	0.2665
100129.439	0.9437	0.4937	0.3830	0.3576	0.3169	0.3119	0.2951	0.2819	0.2834
98945.3256	0.9996	0.4057	0.3897	0.3569	0.3311	0.3140	0.2978	0.2854	0.3030
97772.5771	1.0457	0.5149	0.3969	0.3563	0.3436	0.3159	0.3011	0.2888	0.3211
96611.1094	0.9915	0.5159	0.4046	0.3560	0.3400	0.3158	0.3064	0.2918	0.3216
95460.8393	0.9305	0.5119	0.4034	0.3571	0.3352	0.3151	0.3088	0.2949	0.3216
94321.6841	0.8630	0.5015	0.3889	0.3600	0.3293	0.3177	0.3063	0.2973	0.3204
93193.5613	0.8242	0.4942	0.3830	0.3601	0.3286	0.3175	0.3080	0.3038	0.3260
92076.389	0.7975	0.4881	0.3813	0.3588	0.3305	0.3160	0.3119	0.3127	0.3357
90970.0858	0.7901	0.4742	0.3843	0.3580	0.3284	0.3180	0.3138	0.3193	0.3471
89874.5705	0.7836	0.4610	0.3872	0.3573	0.3264	0.3203	0.3157	0.3257	0.3586
88789.7625	0.7744	0.4574	0.3854	0.3570	0.3275	0.3214	0.3178	0.3285	0.3633
87715.5816	0.7651	0.4537	0.3836	0.3567	0.3287	0.3226	0.3199	0.3314	0.3683
86641.9479	0.7559	0.4500	0.3818	0.3564	0.3298	0.3239	0.3222	0.3345	0.3735
85598.7819	0.7467	0.4461	0.3798	0.3561	0.3311	0.3252	0.3245	0.3377	0.3790
84556.0048	0.7376	0.4422	0.3778	0.3558	0.3324	0.3266	0.3270	0.3411	0.3849
83523.538	0.7286	0.4382	0.3757	0.3555	0.3337	0.3280	0.3296	0.3446	0.3912
82501.3031	0.7195	0.4341	0.3735	0.3552	0.3351	0.3295	0.3323	0.3484	0.3978
81489.2226	0.7106	0.4298	0.3713	0.3548	0.3366	0.3310	0.3352	0.3524	0.4049
80487.2191	0.7016	0.4255	0.3689	0.3545	0.3382	0.3326	0.3383	0.3566	0.4125
79495.2155	0.6966	0.4242	0.3699	0.3580	0.3412	0.3361	0.3421	0.3620	0.4214
78513.1354	0.6953	0.4259	0.3742	0.3657	0.3458	0.3415	0.3468	0.3687	0.4317
77540.9027	0.6939	0.4277	0.3787	0.3738	0.3506	0.3474	0.3517	0.3758	0.4427
76578.4417	0.6924	0.4295	0.3834	0.3825	0.3556	0.3536	0.3568	0.3834	0.4546
75625.677	0.6908	0.4314	0.3882	0.3917	0.3609	0.3603	0.3622	0.3915	0.4676
74682.5337	0.6891	0.4333	0.3932	0.4016	0.3664	0.3675	0.3678	0.4002	0.4816
73748.9373	0.6874	0.4354	0.3983	0.4122	0.3723	0.3753	0.3738	0.4094	0.4969
72824.8137	0.6856	0.4375	0.4036	0.4235	0.3784	0.3838	0.3800	0.4194	0.5136
71910.0892	0.6836	0.4397	0.4090	0.4358	0.3849	0.3929	0.3866	0.4301	0.5320
71004.6905	0.6816	0.4419	0.4147	0.4490	0.3918	0.4029	0.3935	0.4417	0.5523
70108.5447	0.6794	0.4443	0.4206	0.4632	0.3990	0.4138	0.4008	0.4542	0.5749
69221.5792	0.7107	0.4557	0.4269	0.4724	0.4055	0.4209	0.4088	0.4644	0.5944
68343.722	0.7501	0.4690	0.4337	0.4812	0.4121	0.4276	0.4173	0.4748	0.6147
67474.9012	0.7941	0.4832	0.4408	0.4904	0.4191	0.4346	0.4263	0.4859	0.6369
66615.0457	0.8432	0.4984	0.4484	0.5000	0.4264	0.4418	0.4359	0.4978	0.6612
65764.0844	0.8985	0.5147	0.4565	0.5102	0.4341	0.4493	0.4462	0.5105	0.6877
64921.9467	0.9612	0.5322	0.4651	0.5209	0.4423	0.4571	0.4571	0.5243	0.7170
64088.5626	1.0329	0.5510	0.4744	0.5322	0.4509	0.4653	0.4687	0.5391	0.7495
63263.8621	1.1157	0.5713	0.4843	0.5441	0.4600	0.4738	0.4812	0.5551	0.7856
62447.7761	1.2125	0.5933	0.4949	0.5567	0.4697	0.4826	0.4946	0.5726	0.8261
61640.2353	1.3269	0.6172	0.5063	0.5700	0.4799	0.4918	0.5090	0.5915	0.8717
60841.1712	1.4643	0.6432	0.5186	0.5842	0.4908	0.5015	0.5246	0.6123	0.9235
60050.5157	1.6325	0.6717	0.5320	0.5992	0.5025	0.5115	0.5414	0.6351	0.9829

Table G.4: Look-up table for Engine BSFC (open throttle calculations)

Appendix H

Plettenberg HP220/25 Motor Data

I (A)	MotorPowerOut (W)	MotorTorqueOut (Nm)
19.8	292.3	0.316
20.8	307.8	0.335
22.1	330.5	0.362
22.9	346.9	0.381
23.9	361.4	0.399
24.6	371.0	0.411
25.6	389.9	0.435
26.8	406.7	0.456
28.3	432.0	0.489
28.8	441.7	0.501
29.9	459.2	0.524
30.6	468.7	0.537
31.7	487.3	0.561
32.9	505.4	0.586
33.6	516.9	0.602
35.0	536.9	0.629
35.6	547.2	0.644
36.8	566.0	0.670
37.6	575.7	0.684
39.0	594.5	0.712
39.7	605.3	0.728
40.9	625.0	0.756
41.7	633.6	0.771
42.5	642.5	0.785
43.8	660.5	0.813
44.6	670.5	0.829
46.5	695.8	0.870
46.6	697.1	0.872
48.0	715.3	0.902
48.8	723.5	0.917
49.4	732.6	0.932

Table H.1: Look-up table for Plettenberg HP220/25 Motor Data

Appendix I

MATLAB Code for HEPS Powertrain Components

I.1 Motor Function

```

function MotorOutputs = Subsystem_Mot(Argument)
% This function simulates the operation of a Motor
%
% Inputs:
% - MotorTorqueReq: Torque required from Motor (Nm)
% - MotorCurrent: Current to be drawn by the Motor from the Battery (A)
% - MotorPower: Motor Power (W)
% - MotorEnable: Motor enable signal
%   MotorEnable = 0 -- Motor not operational
%   MotorEnable = 1 -- Motor in operation
% - BattSOC: Battery SOC (%)
% - BattVoltage: Battery Voltage (V)
%
% Outputs:
% - M_Power: Motor power output (W)
% - M_Torque: Motor torque output (Nm)
% - M_Current: Motor current required (A)

% Allocate inputs
MotorTorqueReq = Argument(1);
MotorCurrent = Argument(2);
MotorPower = Argument(3);
MotorEnable = Argument(4);
BattSOC = Argument(5);
BattVoltage = Argument(6);

% Set up parameters
maxTorque = 0.932; % Max value from Motor Torque LUT

% Calculate outputs
if ((MotorEnable == 0) || (MotorTorqueReq == 0))
    % Either Motor Enable signal or torque required is ZERO.
    M_Power = 0;
    M_Torque = 0;
    M_Current = 0;
elseif ((BattVoltage < 18) && (BattSOC < 5))
    M_Power = 0;
    M_Torque = 0;
    M_Current = 0;
else
    % Set up upper limit for Motor Torque
    if (MotorTorqueReq > maxTorque)
        MotorTorqueReq = maxTorque;
    end
    M_Power = MotorPower;
    M_Torque = MotorTorqueReq;
    M_Current = MotorCurrent;
end

% Output of function
MotorOutputs = [M_Power, M_Torque, M_Current];

```

I.2 Generator Function

```

function GenOutCurrent = Subsystem_Gen(Argument)
% This function simulates the operation of a Generator
% -- The Generator is assumed to have the same response as the Motor, but
% just in reverse.
%
% Inputs:
% - GenEnable: Generator enable signal (0 or 1)
% - GenTorqueAvail: Torque available to power the Generator (Nm)
%

```

```

% Output:
% - GenOutCurrent: Generator output current (A)

% Load Generator LUT
I = [19.8 20.8 22.1 22.9 23.9 24.6 25.6 26.8 28.3 28.8 29.9 30.6 31.7 ...
     32.9 33.6 35 35.6 36.8 37.6 39 39.7 40.9 41.7 42.5 43.8 44.6 46.5 ...
     46.6 48 48.8 49.4]; % A
MotorTorqueOut = [0.316 0.335 0.362 0.381 0.399 0.411 0.435 0.456 ...
                  0.489 0.501 0.524 0.537 0.561 0.586 0.602 0.629 0.644 0.67 0.684 ...
                  0.712 0.728 0.756 0.771 0.785 0.813 0.829 0.87 0.872 0.902 0.917 ...
                  0.932];

% Allocate inputs
GenEnable = Argument(1);
GenTorqueAvail = Argument(2);

% Determine Generator output current
if (GenEnable == 0) % Generator not activated
    GenOutCurrent = 0;
elseif (GenTorqueAvail < MotorTorqueOut(1))
    % If GenTorqueAvail is not enough to drive the Generator
    GenOutCurrent = 0;
else
    % Set upper limit
    if (GenTorqueAvail > MotorTorqueOut(length(MotorTorqueOut)))
        GenTorqueAvail = MotorTorqueOut(length(MotorTorqueOut));
    end

    % Determine output current
    GenOutCurrent = interp1(MotorTorqueOut, I, GenTorqueAvail);
end

```

I.3 Battery Function

```

function BattOutputs = Subsystem_Batt(Argument)
% This function simulates the operation of a Battery
%
% Inputs:
% - BattCurrent: Battery discharging/charging current (A)
%   BattCurrent = +ve -- Discharging
%   BattCurrent = -ve -- Charging
% - PrevAccCap: Previously accumulated extracted capacity (A.sec)
% - PrevBattSOC: Previous battery State-Of-Charge (%)
%
% Outputs:
% - B_SOC: Battery State-Of-Charge (%)
% - B_Voltage: Battery output voltage (V)
% - NextAccCap: Next accumulated extracted capacity (Ah)

% Battery parameters
% Current required vector
I_req = [5 50 75 100 125]; % A
% Q value Look-Up Table: Q_value = Q_value(I_req)
Q_value = [5.5 5.38172 5.26344 5.02688 4.67204]; % Amp-Hr
% Maximum Q value
Q_max = max(Q_value); % Amp-Hr
% A value Look-Up Table: A_value = A_value(I_req)
A_value = [3.224 3.453 4.054 4.166 4.446]; % V
% B value Look-Up Table: B_value = B_value(I_req)
B_value = [0.4426 1.424 4.224 6.596 9.428]; % A/h
% K value Look-Up Table: K_value = K_value(I_req)
K_value = [0.03167 0.043 0.04075 0.03957 0.04082]; % V

% Allocate inputs
PrevBattSOC = Argument(1);
BattCurrent = Argument(2);
PrevAccCap = Argument(3);

```

```

% Calculate accumulated extracted capacity
NextAccCap = PrevAccCap + BattCurrent;

% Set upper & lower limits for extracted capacity
if (NextAccCap > Q_max*0.9999*3600)
    NextAccCap = Q_max*0.9999*3600;
elseif (NextAccCap < 0)
    NextAccCap = 0;
end

% Convert capacity from Amp-sec to Amp-hr
NextAccCap = NextAccCap/3600;

% Determine Battery parameters
if (BattCurrent == 0)
    if (PrevAccCap == 0) % First iteration
        B_SOC = 100;
        B_Voltage = 0;
    else
        B_SOC = PrevBattSOC;
        B_Voltage = 0;
    end
else
    % Determine parameters Q, K, A, B
    abs_BattCurrent = abs(BattCurrent);
    Q = interp1(I_req, Q_value, abs_BattCurrent, 'pchip', 'extrap');
    K = interp1(I_req, K_value, abs_BattCurrent, 'pchip', 'extrap');
    A = interp1(I_req, A_value, abs_BattCurrent, 'pchip', 'extrap');
    B = interp1(I_req, B_value, abs_BattCurrent, 'pchip', 'extrap');

    % Determine Battery SOC
    B_SOC = 100*(1-NextAccCap/Q);
    if (B_SOC > 100) % Set upper and lower limits
        B_SOC = 100;
    elseif (B_SOC < 0)
        B_SOC = 0;
    end

    % Determine Battery Voltage
    B_Voltage = 24+K-A-K*Q/(Q-NextAccCap)+A*exp(-B*NextAccCap);
    if (B_Voltage > 24) % Set upper and lower limits
        B_Voltage = 24;
    elseif (B_Voltage < 0)
        B_Voltage = 0;
    elseif (B_SOC == 0)
        B_Voltage = 0;
    end
end

% Output of function
BattOutputs = [B_SOC, B_Voltage, NextAccCap*3600];

```

I.4 Powertrain Output Allocation Function

```

function PowertrainOutputs = PowertrainOutAlloc(Arguments)
% This function determines the powertrain outputs by combining the outputs
% from the Engine, Motor and Generator.
%
% Inputs:
% - EngTorque = Engine torque available (Nm)
% - EngPower = Engine power available (W)
% - EngOmega = Engine speed (rad/s)
% - EngEnable = Engine enable (0 or 1)
% - MotorTorque = Motor torque available (Nm)
% - MotorPower = Motor power available (W)
% - MotorEnable = Motor enable (0 or 1)

```



```

% - GenTorque    = Generator torque required (Nm)
% - GenEnable    = Generator enable (0 or 1)
% - rdot         = RCR command
% - CVTratio     = CVT ratio (0 to 1)
%
% Outputs:
% - PowertrainTorque = Powertrain torque (Nm)
% - TorqueToPropShaft = Powertrain torque transmitted to Prop shaft (Nm)
% - PowertrainPower  = Powertrain power (W)

% Allocate inputs
EngTorque    = Arguments(1);
EngPower     = Arguments(2);
EngOmega     = Arguments(3);
EngEnable    = Arguments(4);
MotorTorque  = Arguments(5);
MotorPower   = Arguments(6);
MotorEnable  = Arguments(7);
GenTorque    = Arguments(8);
GenEnable    = Arguments(9);
rdot         = Arguments(10);
CVTratio     = Arguments(11);

% RCR Compensation Torque
J_eng = 0.0001; % Engine shaft moment of inertia (kg*m^2)
J_mot = 2.25e-06; % Motor shaft moment of inertia (kg*m^2)
k2_gain = (J_eng + J_mot) / (CVTratio + rdot);
RCRCompTorque = k2_gain * EngOmega * rdot;

if ((EngEnable == 1) && (MotorEnable == 1)) % Engine + Motor
    PowertrainTorque = EngTorque + MotorTorque;
    TorqueToPropShaft = EngTorque + MotorTorque - RCRCompTorque;
    PowertrainPower = EngPower + MotorPower;
elseif ((EngEnable == 1) && (GenEnable == 1)) % Engine + Generator
    PowertrainTorque = EngTorque - GenTorque;
% TorqueToPropShaft = EngTorque - GenTorque - RCRCompTorque;
    TorqueToPropShaft = EngTorque - GenTorque;
    PowertrainPower = PowertrainTorque / EngOmega;
elseif ((EngEnable == 0) && (MotorEnable == 1)) % Motor only
    PowertrainTorque = MotorTorque;
    TorqueToPropShaft = MotorTorque - RCRCompTorque;
    % This should become just MotorTorque 10 time steps after detecting
    % Motor Only signal
    PowertrainPower = MotorPower;
else % Engine only
    PowertrainTorque = EngTorque;
    TorqueToPropShaft = EngTorque + MotorTorque - RCRCompTorque;
    % This should become just EngTorque 10 time steps after detecting
    % Engine Only signal
    PowertrainPower = EngPower;
end

PowertrainOutputs = [PowertrainTorque,TorqueToPropShaft,PowertrainPower];

```

I.5 *ChargeBattNow* Function

```

function Outputs = ChargeBattNow(Inputs)
% Determines if the Battery requires immediate charging
% Inputs:
% - CurrentMustChargeBatt = Current "Must Charge Battery" signal
%   - CurrentMustChargeBatt = 1: Must charge battery
%   - CurrentMustChargeBatt = 0: Battery doesn't need charging
% - CurrentBattSOC = Current Battery SOC, from 0% to 100%
% - CurrentAirspeed = Current aircraft airspeed (m/s)
% - CurrentDistRem = Current distance remaining until "Motor Only" (m)
% - BattInputCurrent = Input current into the Battery (A)
% - AvgChgCurrent = Average charge current (A)

```

```

% - ChgCurrentCount = Number of recorded charge currents
% - SimClock = Current simulation time (s)
%
% Outputs:
% - NextMustChargeBatt = Next "Must Charge Battery" signal
%   - NextMustChargeBatt = 1: Must charge battery
%   - NextMustChargeBatt = 0: Battery doesn't need charging
% - NextDistRem = Next distance remaining until "Motor Only" (m)
% - NextAvgChgCurrent = Next average charge current (A)
% - NextChgCurrentCount = Updated number of recorded charge currents
%
% Other data required:
% - TOTALDIST2MO = Total distance from start of mission to "Motor Only"
%   mode; calculated in "WPTTable.m".
% - I_req = Battery current vector for Q-value calculations
% - Q_value = Q-value vector for the Battery

% Allocate inputs
CurrentMustChargeBatt = Inputs(1);
CurrentBattSOC        = Inputs(2);
CurrentAirspeed       = Inputs(3);
BattInputCurrent      = Inputs(4);
CurrentDistRem        = Inputs(5);
AvgChgCurrent         = Inputs(6);
ChgCurrentCount       = Inputs(7);
SimClock              = Inputs(8);

% Load data
TOTALDIST2MO = 5495.623946181784; % Initial distance to Motor Only mode
I_req = [5 50 75 100 125]; % Battery current vector
Q_value = [5.5 5.38172 5.26344 5.02688 4.67204]; % Q-value vector

% Initialisation
NextMustChargeBatt = CurrentMustChargeBatt; % Default: let sleeping Battery sleep
t_to_MO = 0; % Time remaining until "Motor Only" mode (s)
t_to_FC = 0; % Time required for Battery to be fully charged (s)
delta_time = 0.1; % Time interval for one iteration; taken from Aerosonde UAV block (s)

% Calculate the time until "Motor Only"
if (SimClock < delta_time)
    % In the first iteration
    CurrentDistRem = TOTALDIST2MO;
end

t_to_MO = CurrentDistRem/CurrentAirspeed;

% Calculate the distance remaining until "Motor Only" after this iteration
NextDistRem = CurrentDistRem - CurrentAirspeed*delta_time;
% Keep this value above zero
if (NextDistRem < 0)
    NextDistRem = 0;
end

% Calculate average Battery charging current

NextAvgChgCurrent = AvgChgCurrent; % Initialise, carry over prev value
NextChgCurrentCount = ChgCurrentCount; % Initialise, carry over prev value

if (BattInputCurrent < 0)
    % Battery is charging (-ve = charging)
    if (ChgCurrentCount == 0)
        % No value in vector yet
        NextChgCurrentCount = 1;
        NextAvgChgCurrent = abs(BattInputCurrent);
    else
        NextChgCurrentCount = ChgCurrentCount + 1;
        NextAvgChgCurrent = (AvgChgCurrent*ChgCurrentCount+abs(BattInputCurrent))/NextChgCurrentCount;
    end
else
    % Battery is not charging
    NextChgCurrentCount = ChgCurrentCount;
    NextAvgChgCurrent = AvgChgCurrent;
end
end

```

```

% Calculate the time required to fully charge the Battery
if (ChgCurrentCount > 0) % Battery has undergone charging
    Q_AvgCurrent = interp1(I_req, Q_value, NextAvgChgCurrent, 'pchip', 'extrap');
    t_to_FC = Q_AvgCurrent*(1-CurrentBattSOC/100)/NextAvgChgCurrent;
else % Battery hasn't required charging
    t_to_FC = 0; % SOC is near 100% even if not having charged before
end

% Determine the value for NextMustChargeBatt
% if (CurrentBattSOC > 90)
%     NextMustChargeBatt = 0; % No charging required
%     ChgCase = 5;
if ((t_to_MO-t_to_FC) < 60) && (CurrentDistRem > 0)
    % Almost not enough time to bring Battery to full charge and "Motor
    % Only" mode has not been reached
    if (CurrentBattSOC > 99.5)
        NextMustChargeBatt = 0; % No charging required
        ChgCase = 5;
    else
        NextMustChargeBatt = 1; % Battery needs charging now
        ChgCase = 1;
    end
elseif (CurrentBattSOC < 20)
    % SOC_charge less than 20%
    NextMustChargeBatt = 1;
    ChgCase = 2;
elseif ((CurrentBattSOC < 80) && (CurrentMustChargeBatt == 1))
    % Battery has been charging, but SOC not yet 80%
    NextMustChargeBatt = 1;
    ChgCase = 3;
elseif ((CurrentBattSOC >= 85) && (CurrentDistRem <= 0))
    % Battery SOC >= 85% after the "Motor Only" mode has been reached
    NextMustChargeBatt = 0; % No charging required
    ChgCase = 4;
else
    NextMustChargeBatt = 0;
    ChgCase = 0;
end

% Define the output vector
Outputs = [NextMustChargeBatt, NextDistRem, NextAvgChgCurrent, NextChgCurrentCount, ChgCase];

```


Appendix J

MATLAB Code for HEPS IOL Controller Components

J.1 *OpMode* Function

```

function CurrentOpMode = OpMode(Argument)
% This function determines the mode in which the HEPS is currently
% operating in
%
% Inputs:
% - MotEngOnly: The Motor Only or Engine Only signal
%   MotEngOnly = 0 -- HEPS operating in Hybrid mode
%   MotEngOnly = 1 -- HEPS in Motor Only mode
%   MotEngOnly = 2 -- HEPS in Engine Only mode
% - MustChgBatt: If the Battery needs to charging
%   MustChgBatt = 1 -- Charge Battery NOW!
%   MustChgBatt = 0 -- No need to charge Battery
% - InClimb: If the aircraft is in a climb
%   InClimb = 1 -- Aircraft is climbing
%   InClimb = 0 -- Aircraft is not climbing
%
% Output:
% - CurrentOpMode: The current mode of operation for the HEPS
%   CurrentOpMode = 1 -- Hybrid Normal
%   CurrentOpMode = 2 -- Motor Only
%   CurrentOpMode = 3 -- Hybrid Charging
%   CurrentOpMode = 4 -- Engine Only
%   CurrentOpMode = 5 -- Hybrid Climbing

% Allocate inputs
MotEngOnly = Argument(1);
MustChgBatt = Argument(2);
InClimb = Argument(3);

% Determine the operating mode
if (MotEngOnly == 1)
    % Motor Only mode
    CurrentOpMode = 2;
elseif (MotEngOnly == 2)
    % Engine Only mode
    CurrentOpMode = 4;
elseif (MustChgBatt == 1)
    if (InClimb == 1)
        % Engine Only mode
        CurrentOpMode = 4;
    else
        % Hybrid Charging mode
        CurrentOpMode = 3;
    end
elseif (InClimb == 1)
    % Hybrid Climbing mode
    CurrentOpMode = 5;
else
    % Hybrid Normal mode (default)
    CurrentOpMode = 1;
end
end

```

J.2 *PowerDemandAlloc* Function

```

function PowerDemand = PowerDemAlloc(Argument)
% This function determines the Power Demand which is required to maintain
% UAV flight.
%
% Inputs:
% - EngOmega: Engine speed (rad/s)
% - PowerReq: Power required obtained from throttle setting (W)
% - OpMode: Current mode of operation for the HEPS
%   OpMode = 1 -- Hybrid Normal

```

```

%      OpMode = 2 -- Motor Only
%      OpMode = 3 -- Hybrid Charging
%      OpMode = 4 -- Engine Only
%      OpMode = 5 -- Hybrid Climbing
%
% Output:
% - PowerDemand: Total Power Demand from the HEPS (W)

% Allocate inputs
EngOmega = Argument(1);
PowerReq = Argument(2);
OpMode = Argument(3);

% Parameter Setup
GenChgT = 0.4;      % Generator charging torque (Nm)
                  % Uses a value slightly greater than the minimum
                  % Motor/Gen torque value
MotClimbT = 0.4;   % Motor torque to assist when climbing (Nm)

% Determine the Power Demand
switch OpMode
    case {1,2,4}
        % Hybrid Normal, Motor Only and Engine Only modes
        PowerDemand = PowerReq;
    case 3
        % Hybrid Charging mode
        PowerDemand = PowerReq + GenChgT * EngOmega;
    case 5
        % Hybrid Climbing mode
        PowerDemand = PowerReq - MotClimbT * EngOmega;
end

```

J.3 *EngOp* Function

```

function EngOutputs = EngOp(Argument)
% This function determines the Torque, Power and Manifold Pressure (MAP)
% values that are required to be produced by the Engine
%
% Inputs:
% - Torque_IOL: Engine IOL torque (Nm)
% - MAP_IOL: MAP required to produce the Engine IOL torque (kPa)
% - Torque_EngNorm: Engine torque in normal operations (Nm)
% - MAP_EngNorm: Engine MAP in normal operations (kPa)
% - OpMode: Current mode of operation for the HEPS
%      OpMode = 1 -- Hybrid Normal
%      OpMode = 2 -- Motor Only
%      OpMode = 3 -- Hybrid Charging
%      OpMode = 4 -- Engine Only
%      OpMode = 5 -- Hybrid Climbing
%
% Outputs:
% - EngTorque: Torque required to be produced by the Engine (Nm)
% - EngMAP: MAP required at the Engine (kPa)

% Allocate inputs
Torque_IOL = Argument(1);
MAP_IOL = Argument(2);
Torque_EngNorm = Argument(3);
MAP_EngNorm = Argument(4);
OpMode = Argument(5);

% Determine the Engine outputs required
switch OpMode
    case {1,3,5}
        % Hybrid Normal, Hybrid Charging and Hybrid Climbing modes
        EngTorque = Torque_IOL;
        EngMAP = MAP_IOL;

```

```

    case 2
        % Motor Only mode
        EngTorque = 0;
        EngMAP = 0;
    case 4
        % Engine Only mode
        EngTorque = Torque_EngNorm;
        EngMAP = MAP_EngNorm;
end

% Define the outputs vector
EngOutputs = [EngTorque,EngMAP];

```

J.4 *EngThrCommand* Function

```

function EngThr = EngThrCommand(Arguments)
% This function determines the engine throttle command required to operate
% the Engine at the desired Torque value
%
% Inputs:
% - EngMAP = Engine MAP required (kPa)
% - p_alt = atmosphere pressure at current altitude (Pa)
%
% Outputs:
% - EngThr = engine throttle command (0.01 to 1)

% Allocating inputs
EngMAP = Arguments(1);
p_alt = Arguments(2);

% Set up
MAPmin = 60;    % From Aerosonde config file

% Calculating Engine throttle command
if (EngMAP == 0)
    % Motor Only mode; no Engine output required
    EngThr = 0;
else
    % Hybrid Normal, Hybrid Charging and Engine Only modes
    EngThr = (EngMAP-MAPmin)/(p_alt/1000-MAPmin);
    % Put upper and lower limits on EngThr
    if (EngThr < 0.01)
        EngThr = 0.01;
    elseif (EngThr > 1)
        EngThr = 1;
    end
end
end

```

J.5 *RCRCommand* Function

```

function Outputs = RCRCommand(Arguments)
% This function determines the Rate of Change of Ratio (RCR), or rdot,
% command
%
% Inputs:
% - PowerDemand: Power demand (W)
% - EngPower: Power required from the Engine (W)
% - CVT_ratio: Current CVT ratio
% - OpMode: Current mode of operation for the HEPS
%   OpMode = 1 -- Hybrid Normal
%   OpMode = 2 -- Motor Only
%   OpMode = 3 -- Hybrid Charging

```



```

%      OpMode = 4 -- Engine Only
%      OpMode = 5 -- Hybrid Climbing
% - PrevRdotCount: Current value of a counter for RCR determination
% - Prev_r0: The record of the CVT ratio when Engine Only or Motor Only is
%           first detected
%
% Outputs:
% - rdot = RCR command
% - NextRdotCount: Next value of a counter for RCR determination
% - Next_r0: The record of the CVT ratio when Engine Only or Motor Only is
%           first detected

% Allocating inputs
PowerDemand = Arguments(1);
EngPower = Arguments(2);
CVT_ratio = Arguments(3);
OpMode = Arguments(4);
PrevRdotCount = Arguments(5);
Prev_r0 = Arguments(6);

% Set up parameter
RCR_gain = 1/2000;

% Determine the RCR Command
switch OpMode
    case {1,3,5}
        % Hybrid Normal, Hybrid Charging and Hybrid Climbing modes
        Next_r0 = 0;
        rdot = RCR_gain*(PowerDemand-EngPower);
        % Upper and lower limits on value of rdot
        % -0.1 < rdot < 0.1
        if (rdot < -0.1)
            rdot = -0.1;
        elseif (rdot > 0.1)
            rdot = 0.1;
        end
        NextRdotCount = 0;
    case {2,4}
        % Motor Only or Engine Only modes
        % When Motor Only or Engine Only is first detected
        if (PrevRdotCount == 0)
            Next_r0 = CVT_ratio;
            rdot = 1-Next_r0;
            NextRdotCount = 1;
        elseif ((PrevRdotCount > 0) && (PrevRdotCount < 10))
            % Within the first 10 time steps of detecting Motor Only or
            % Engine Only signals
            % -- Simulation step size = 0.1 sec --> 10 time steps = 1 sec
            Next_r0 = Prev_r0;
            rdot = 1-Next_r0;
            NextRdotCount = PrevRdotCount + 1;
        else
            % After CVT ratio has settled at 1
            Next_r0 = 0;
            rdot = 0;          % No more changes to CVT ratio required
            NextRdotCount = PrevRdotCount;
        end
end

% Define the outputs vector
Outputs = [rdot,NextRdotCount,Next_r0];

```

J.6 TorqueDiffCalc Function

```

function Outputs = TorqueDiffCalc(Arguments)
% This function determines the amount of torque required from the Motor, or
% available for the Generator. Also, the on/off signals for Engine, Motor

```

```

% and Generator components are also determined here.
%
% Inputs:
% - PowerDemand: Total Power Demand required (W)
% - AltCorrection: Altitude correction
% - EngTorque: Torque required to be produced by the Engine (Nm)
% - EngOmega: Engine speed (rad/s)
% - CVT_Ratio: Current CVT ratio
% - rdot: RCR command
% - OpMode: Current mode of operation for the HEPS
%   OpMode = 1 -- Hybrid Normal
%   OpMode = 2 -- Motor Only
%   OpMode = 3 -- Hybrid Charging
%   OpMode = 4 -- Engine Only
%   OpMode = 5 -- Hybrid Climbing
%
% Outputs:
% - MotorTorqueReq: Torque required to be supplemented by the Motor (Nm)
% - GenTorqueAvail: Torque available to be used by the Generator (Nm)
% - EngEnable: Engine Enable signal
%   EngEnable = 0 -- Engine OFF
%   EngEnable = 1 -- Engine ON
% - MotorEnable: Motor Enable signal
%   MotorEnable = 0 -- Motor OFF
%   MotorEnable = 1 -- Motor ON
% - GenEnable: Generator Enable signal
%   GenEnable = 0 -- Generator OFF
%   GenEnable = 1 -- Generator ON

% Allocate inputs
PowerDemand = Arguments(1);
AltCorrection = Arguments(2);
EngTorque = Arguments(3);
EngOmega = Arguments(4);
CVT_Ratio = Arguments(5);
rdot = Arguments(6);
OpMode = Arguments(7);

% Parameter Setup
J_eng = 0.0001; % Engine shaft moment of inertia (kg*m^2)
J_mot = 2.25e-06; % Motor shaft moment of inertia (kg*m^2)
GenChgT = 0.4; % Generator charging torque (Nm)
MotClimbT = 0.4; % Motor torque to assist when climbing (Nm)

% RCR Compensation Torque
k2_gain = (J_eng + J_mot) / (CVT_Ratio + rdot);
RCRCompTorque = k2_gain * EngOmega * rdot;

% Determine the Motor/Generator Torque values
switch OpMode
case 1
    % Hybrid Normal mode
    TorqueDiff = (PowerDemand / EngOmega - EngTorque) * AltCorrection + RCRCompTorque;
    if (TorqueDiff >= 0)
        % Motor is required to supplement torque
        MotorTorqueReq = TorqueDiff;
        GenTorqueAvail = 0;
        EngEnable = 1; % Engine ON
        MotorEnable = 1; % Motor ON
        GenEnable = 0; % Generator OFF
    else
        % Excess torque is available to charge the Battery
        MotorTorqueReq = 0;
        GenTorqueAvail = -TorqueDiff;
        EngEnable = 1; % Engine ON
        MotorEnable = 0; % Motor OFF
        GenEnable = 1; % Generator ON
    end
case 2
    % Motor Only mode
    MotorTorqueReq = PowerDemand / EngOmega * AltCorrection + RCRCompTorque;
    GenTorqueAvail = 0;
    EngEnable = 0; % Engine OFF

```

```
    MotorEnable = 1;          % Motor ON
    GenEnable = 0;           % Generator OFF
case 3
    % Hybrid Charging mode
    MotorTorqueReq = 0;
    GenTorqueAvail = GenChgT - RCRCCompTorque;
    EngEnable = 1;          % Engine ON
    MotorEnable = 0;       % Motor OFF
    GenEnable = 1;         % Generator ON
case 4
    % Engine Only mode
    MotorTorqueReq = (PowerDemand / EngOmega - EngTorque) * AltCorrection + RCRCCompTorque; % This should be small
    GenTorqueAvail = 0;
    EngEnable = 1;         % Engine ON
    if (abs(MotorTorqueReq) < 0.001)
        MotorEnable = 0;   % Motor OFF
    else
        MotorEnable = 1;   % Motor ON
    end
    GenEnable = 0;         % Generator OFF
case 5
    % Hybrid Climbing mode
    MotorTorqueReq = MotClimbT + RCRCCompTorque;
    GenTorqueAvail = 0;
    EngEnable = 1;         % Engine ON
    MotorEnable = 1;       % Motor ON
    GenEnable = 0;         % Generator OFF
end

% Define output vector
Outputs = [MotorTorqueReq, GenTorqueAvail, EngEnable, MotorEnable, GenEnable];
```