CRANFIELD UNIVERSITY

R. G. DRURY

TRAJECTORY GENERATION FOR AUTONOMOUS UNMANNED AIRCRAFT USING INVERSE DYNAMICS

SCHOOL OF ENGINEERING

PhD THESIS

Academic Year 2009-2010

Supervisors: Professor A. Tsourdos and Dr A. K. Cooke

September 2010

ABSTRACT

The problem addressed in this research is the in-flight generation of trajectories for autonomous unmanned aircraft, which requires a method of generating pseudo-optimal trajectories in near-real-time, on-board the aircraft, and without external intervention. The focus of this research is the enhancement of a particular inverse dynamics direct method that is a candidate solution to the problem. This research introduces the following contributions to the method.

A quaternion-based inverse dynamics model is introduced that represents all orientations without singularities, permits smooth interpolation of orientations, and generates more accurate controls than the previous Euler-angle model.

Algorithmic modifications are introduced that: overcome singularities arising from parameterization and discretization; combine analytic and finite difference expressions to improve the accuracy of controls and constraints; remove roll ill-conditioning when the normal load factor is near zero, and extend the method to handle negative-g orientations. It is also shown in this research that quadratic interpolation improves the accuracy and speed of constraint evaluation.

The method is known to lead to a multimodal constrained nonlinear optimization problem. The performance of the method with four nonlinear programming algorithms was investigated: a differential evolution algorithm was found to be capable of over 99% successful convergence, to generate solutions with better optimality than the quasi-Newton and derivative-free algorithms against which it was tested, but to be up to an order of magnitude slower than those algorithms. The effects of the degree and form of polynomial airspeed parameterization on optimization performance were investigated, and results were obtained that quantify the achievable optimality as a function of the parameterization degree.

Overall, it was found that the method is a potentially viable method of on-board near-real-time trajectory generation for unmanned aircraft but for this potential to be realized in practice further improvements in computational speed are desirable. Candidate optimization strategies are identified for future research.

Keywords:

Autonomy; differential evolution; direct methods; inverse dynamics; near-real-time; negative-g; nonlinear programming; numerical optimization; optimal control; quaternions; trajectory generation; UAV; unmanned aircraft.

ACKNOWLEDGEMENTS

My partner, Fiona Pearson, has given unstinting and generous support that has enabled me to fulfil a long-standing ambition. She has provided encouragement and advice throughout, without which I could not have undertaken this adventure. (She is also the most significant financial supporter of this work!)

My supervisors, Professor Antonios Tsourdos and Dr Alastair Cooke, have both provided excellent help and support whenever I have needed it. Antonios' knowledge of optimal control and trajectory generation has been combined with excellent advice on the process of research and publication which has had a big positive impact on the quality of my work. I would not have completed this work without him and am very grateful. Alastair's advice on flight dynamics and Cranfield's processes is similarly appreciated. Mike Cook provided valuable advice at the outset and in my first year, particularly on flight control, as well as arranging the EPSRC CASE award under which this work was formally funded. Professor Rafał Żbikowski gave me advice on optimization which I had struggled to find: his exposition was clear, enthusiastic and one of the most useful contributions that I have received in the last 3 years.

I was very fortunate to find good friends among my fellow students, who have shared their own experiences and not only helped me in my work but also enriched the experience for me. Mike Riley tolerated my rambling descriptions of my work, frequently came up with questions and answers that enabled me to move forward, introduced me to Differential Evolution, and freely shared his ideas and advice. Without his advice on hardware and software it is unlikely that I would have been able to carry out my experiments: his espresso coffee also helped! Deborah Saban made me feel welcome from my first day, helped me to get started and throughout she gave freely of her own findings and experiences. Most importantly, she showed me a very different perspective on life and shared some thoughts and ideas which I very much value. Pierre-Daniel Jameson and Stuart Andrews also helped me with their comments and advice. I am most grateful to Mike, Deborah, Pierre-Daniel and Stuart and wish them well in their careers.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

AR	=	aspect ratio
$\mathbf{a}$	=	acceleration
a	=	acceleration magnitude; spatial coefficient; quadrature interval factor
a_1, a_2, a_A	=	temporary acceleration variables (Chapter 5)
a_x	=	tangential acceleration
b	=	Hooke-Jeeves base exploration flag
C	=	Bernstein form coefficient
C_D	=	drag coefficient
C_{D0}	=	drag coefficient at minimum drag
C_L	=	lift coefficient
C_{LminD}	=	lift coefficient at minimum drag
$C_{L\alpha}$	=	lift curve slope
C^n	=	at least n times continuously differentiable
C_r	=	differential evolution crossover probability
$\mathbf{c}$	=	constraint vector
c_i	=	i-th constraint violation
D	=	drag; NLP dimension
D_1, D_2	=	temporary drag variables (Chapter 5)
d	=	positive or negative-g alignment (Chapter 7)
d_v	=	airspeed polynomial degree
$\mathbf{E}$	=	4x3 quaternion propagation matrix
E	=	set of equality constraints
$\mathbf{e}$	=	unit quaternion $(e_0, e_1, e_2, e_3)^T$
F	=	differential evolution mutation scale factor
$\mathbf{f}_M$	=	total force per unit mass
f	=	optimization objective; degree $N-1$ Chebyshev interpolant
f_b	=	best objective in differential evolution population (Chapter 6)
f_{base}	=	base point objective in Hooke-Jeeves algorithm (Chapter 6)
f_{exp}	=	explore point objective in Hooke-Jeeves algorithm (Chapter 6)
f_{max}	=	best objective in Nelder-Mead simplex (Chapter 6)
f_{min}	=	worst objective in Nelder-Mead simplex (Chapter 6)
f_w	=	worst objective in differential evolution population (Chapter 6)
$\mathbf{g}$	=	gravity vector $(0, 0, g)^T$ in flat Earth frame
g_c	=	mean gravitational acceleration over inter-node segment
$\mathbf{g}_z$	=	gravity vector component in wind frame z-axis
g	=	gravitational acceleration
$\mathbf{H}$	=	rotation matrix
H	=	Hamiltonian
$\mathbf{h}$	=	quaternion representation of 180º rotation in bank
h	=	Hooke-Jeeves step reduction count; load factor limit $v_s \leq v \leq v_a$
h_{max}	=	Maximum count of Hooke-Jeeves step reductions
I	=	set of inequality constraints
j	=	node index
k	=	penalty weight
L	=	lift

$\mathbf{l}_z$	=	normal load factor
l_{struct}	=	structural load factor limit
l_z	=	normal load factor magnitude
l^+	=	positive-g normal load factor limit
l^-	=	negative-g normal load factor limit (Chapter 7)
M	=	aircraft mass
m	=	local quadratic interpolation function
N	=	number of discretization nodes per trajectory
N_F	=	feasible set size in differential evolution population (Chapter 6)
N_p	=	differential evolution population size (Chapter 6)
P	=	penalty function
p, q, r	=	roll, pitch and yaw angular velocities in the wind frame
p_{max}	=	maximum rate of change of bank angle
R	=	residual
$\mathbf{r}$	=	position vector $(x, y, z)^T$
S	=	wing reference area
S^3	=	3-sphere
s	=	path length
T	=	thrust in wind frame x-axis
T_{max}	=	maximum thrust, wind frame
T_{min}	=	minimum thrust, wind frame
t_f	=	final flight time
t_I	=	time required for inversion within angular velocity constraints (Chapter 7)
t_{Ref}	=	reference final flight time at v_{max} (Chapter 5)
$\mathbf{u}$	=	control vector
v	=	airspeed
v_a	=	manoeuvring airspeed
v_1, v_2, v_{lim}	=	temporary airspeed variables (Chapter 5)
v_B	=	lower limit on airspeed during manoeuvre (Chapter 5)
v_l	=	maximum airspeed that satisfies the load factor constraint (Chapter 5)
v_{max}	=	maximum airspeed that satisfies all path constraints (Chapter 5)
v_{min}	=	minimum airspeed that satisfies all path constraints (Chapter 5)
v_{ne}	=	never-exceed airspeed
v_p	=	maximum airspeed that satisfies bank rate constraint (Chapter 5)
v_s	=	stall speed, straight and level
v_{Tmax}	=	maximum airspeed that satisfies T_{max} constraint (Chapter 5)
v_{Tmin}	=	maximum airspeed that satisfies T_{min} constraint (Chapter 5)
w	=	quadrature weight
$\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}$	=	coordinate axes (unit vectors)
x, y, z	=	position components, flat Earth axes unless otherwise specified
α	=	angle of attack
α_s	=	stall limit of angle of attack
$\alpha_0, \alpha_2, \alpha_5$	=	optimality scores (Chapter 6)
β	=	sideslip angle; optimization success ratio
γ	=	flight path angle
Δ	=	quaternion interpolation step size; optimization tolerance
$\Delta_{f\,0}, \Delta_{f\ min}$	=	initial and minimum objective ranges (Chapter 6)

Δ_f	=	termination objective tolerance (Chapter 6)
$\Delta_{f\ thresh}$	=	objective threshold (Chapter 6)
Δ_χ	=	maximum ∞-norm of differential evolution population (Chapter 6)
δs_j	=	segment distance, node *j* -1 to node *j*
δt_j	=	segment duration, node *j* -1 to node *j*
$\delta \tau_j$	=	segment virtual arc, node *j* -1 to node *j*
$\hat{\boldsymbol{\varepsilon}}$	=	Frenet frame coordinate axis unit vector
ε	=	Oswald efficiency factor
ζ	=	multiplication or reduction factor (Chapter 6)
η	=	constraint violation; inversion permissible flag
η_{max}	=	maximum permitted constraint violation
η_{tol}	=	SNOPT post-optimization constraint tolerance (Chapter 6)
θ	=	angle between quaternions
κ	=	curvature
Λ	=	inversion cost function (Chapter 7)
λ	=	speed factor
μ	=	bank angle
ξ	=	heading
ρ	=	air density; radius of curvature; penalty weighting parameter
ρ_{max}	=	maximum penalty weighting parameter
σ	=	count of trajectory evaluations
σ_{max}	=	termination limit on σ
σ_g	=	differential evolution generation count limit (Chapter 6)
τ	=	virtual arc independent variable
ϕ	=	optimization performance profile factor; roll angle
$\boldsymbol{\chi}$	=	vector of optimization parameters
$\boldsymbol{\chi}_{\Delta 0}$	=	initial step size for Hooke-Jeeves or Nelder-Mead (Chapter 6)
$\boldsymbol{\chi}_b$	=	best vector in differential evolution population (Chapter 6)
$\boldsymbol{\chi}_w$	=	worst vector in differential evolution population (Chapter 6)
$\boldsymbol{\chi}_{max}$	=	best vector in Nelder-Mead simplex (Chapter 6)
$\boldsymbol{\chi}_{min}$	=	worst vector in Nelder-Mead simplex (Chapter 6)
χ	=	inversion demand flag (Chapter 7)
ψ	=	set of contiguous intervals between two or more nodes (Chapter 7)
$\boldsymbol{\Omega}$	=	4x4 quaternion propagation matrix
$\tilde{\boldsymbol{\omega}}$	=	angular velocity cross product equivalent matrix
$\boldsymbol{\omega}$	=	angular velocity in wind frame $(p, q, r)^T$

Superscripts or subscripts

a	=	index; manoeuvring speed
b	=	index; best solution in differential evolution population
E	=	inertial flat-Earth frame, north-east-down axes
f	=	final boundary point; optimization function value
i, j, k	=	indices
m	=	node index at end of inversion
ne	=	never-exceed speed
s	=	stall speed; node index at start of inversion

seg	=	segment variable
t, *n*	=	Frenet frame tangential and normal axes
V	=	velocity frame
W	=	wind frame
w	=	worst solution in differential evolution population
0	=	initial boundary point; initialization value
*	=	optimal or pseudo-optimal value
+	=	nonnegative constraint violation; positive-g
-	=	negative-g

Operators

*	=	quaternion multiplication; as superscript quaternion conjugation
′	=	derivative with respect to τ
×	=	vector cross product
$\|\cdot\|$	=	2-norm

All quantities are expressed in the International System of Units. Scalars use italic Latin or Greek fonts; vectors and matrices use bold Latin or Greek fonts.

Chapter 2 uses chapter-specific variables, not those listed above.

1 INTRODUCTION

Unmanned air vehicles (UAVs) are in operational use for intelligence, surveillance, target acquisition, reconnaissance, and ground attack missions supporting our Armed Forces. They are also being used for civil purposes including security patrolling and environmental monitoring. Operational trajectories are typically 2D or 3D paths generated on the ground before the flight (or mission segment) and flown as waypoint sequences with pre-programmed manoeuvres such as standard rate turns. Many current UAVs require more than one ground-based operator per aircraft to plan the mission and flight path, fly the aircraft, and operate the sensors and weapons[23, 97]. Further, the associated communications between operator and aircraft reduce the bandwidth available to the sensor and weapon systems, which is a critical resource for UAV operations.

Autonomous generation of pseudo-optimal trajectories on board the aircraft would enable

- The communications bandwidth required to operate the UAV to be reduced.
- The operator workload to be reduced.
- Trajectories closer to optimal to be generated.

It should also enable the UAV to react autonomously to changes in its operating environment such as moving or unforeseen targets, obstacles, or threats, and to changes to the mission such as air-air refuelling or time-varying rendezvous points.

As well as minimizing an objective such as time of flight or fuel consumed, trajectories must satisfy constraints arising from boundary conditions, aircraft dynamics, sensor dynamics and from the changing environment, such as obstacles. To provide the inputs required by a flight control system, the trajectory generator should produce not only the desired flight path but also suitable control trajectories. The trajectory optimization problem may therefore be formulated as an optimal control problem, which is then transcribed to a parameter optimization problem and solved by the application of a constrained nonlinear programming (NLP) algorithm. Sufficient computing power to solve the NLP problem in real time and within typical UAV weight and power

limitations is not yet available for operational use, but the architecture shown in Figure 1-1, adapted from Åström and Murray[3], shows a system architecture that separates the trajectory generation task from the trajectory following task. The timing demand on the trajectory generator is relaxed from real time to near real time and, in isolation, the trajectory generator itself becomes open loop, making the problem potentially achievable if a suitably fast and reliable method of solving the open loop optimal control problem is used.

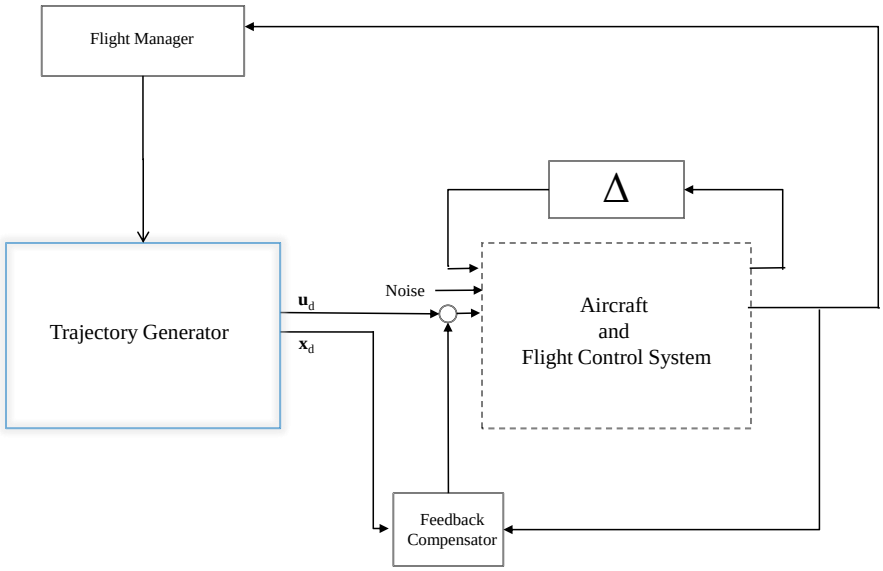

Figure 1-1. Two Degrees of Freedom Architecture

Indirect methods of the calculus of variations are not currently feasible for real-time implementation by on-board aircraft systems. They solve the optimal control problem by formulating the first order optimality conditions, applying Pontryagin's Maximum Principle and using NLP to solve the resulting two-point boundary value problem numerically. Direct methods, which seek to directly minimize the objective, are not guaranteed to result in an optimal solution but compared to indirect methods will usually generate a pseudo-optimal trajectory more robustly (it is not necessary to solve

potentially ill-conditioned combinations of state, costate, and stationarity/Pontryagin conditions) and have a larger radius of convergence.

Comparing ground-based generation of a pre-planned spacecraft manoeuvre with on-board generation of a UAV manoeuvre highlights the relative strengths of these methods. The spacecraft manoeuvre typically requires a one-off solution, with hours or days available to solve the problem, and the solution must be of high accuracy. Indirect methods are therefore well suited to this problem (although pseudospectral and inverse dynamics methods have also been used). In contrast the UAV problem requires repeated solution of time-varying problems in near real time, whilst wind, noise, uncertainty and continual updating reduce the importance of high accuracy. Direct methods are well suited to this type of problem.

A particular direct method that is computationally fast, does not require large data storage, and guarantees satisfaction of spatial and airspeed boundary conditions, has been developed from ideas introduced by Taranenko in Russia in 1968[126]. In 1999-2000 Yakimenko introduced an inverse dynamics variant of the method to the West[133, 134]. As a direct method it is computationally cheaper than indirect methods and does not require good initial guesses of constrained arcs or of non-intuitive costate variables. Numerical simulations[5, 17, 114, 135] suggest that it is faster than other candidate direct methods such as shooting or pseudospectral methods. However, it has a number of limitations and only small samples of data on its performance with NLP algorithms have been published. In this document it is referred to as "the inverse dynamics method" and is applied to the minimization of the flight time of a conventional fixed wing aeroplane.

The thesis that motivates this research is that the inverse dynamics method is a potentially viable method of on-board near-real-time trajectory generation for unmanned aircraft, and research into its four main parts can further improve its capabilities. The contributions from this research therefore address the four parts of the method:

- Objective, controls, and constraints evaluation algorithm.
- Aircraft inverse dynamics model.

- State vector parameterization.
- Constrained nonlinear optimization.

The following contributions to the inverse dynamics method have been introduced in this research:

- A quaternion-based inverse dynamics model that represents all orientations without singularities, permits smooth interpolation of orientations, and generates more accurate controls than the previous Euler-angle model.
- Algorithmic modifications that overcome singularities arising from zeros of the spatial parametric speed, airspeed, and normal load factor.
- Combinations of analytic and finite difference expressions that improve the accuracy of controls and constraints.
- Local quadratic interpolation of constraints that improve accuracy and computational speed.
- An algorithm that evaluates maximum feasible airspeed without using numerical optimization.
- Algorithmic modifications, in conjunction with the quaternion-based model, that remove roll ill-conditioning when the normal load factor is near zero, and extend the method to handle negative-g orientations.
- Quantification of the effects on optimality, robustness and computational speed of polynomial airspeed parameterization.
- Comparison of the optimality, robustness and computational speed achieved with four nonlinear programming algorithms. A differential evolution algorithm was found to be capable of over 99% successful convergence, to generate solutions with better optimality than the quasi-Newton and derivative-free algorithms against which it was tested, but to be up to an order of magnitude slower than them.
- Identification of candidate optimization strategies for future research.

This document describes the research and contributions as follows.

Chapter 2 provides background and contextual material from the literature on methods of trajectory generation and optimization, focusing on direct methods of optimal control and showing how the inverse dynamics method relates to them, to underpin the thesis

stated above. The chapter also provides background material on relevant NLP algorithms.

Chapter 3 describes the inverse dynamics method as previously published and analyses it to identify limitations. The chapter then introduces modifications to address the limitations and increase the accuracy and robustness of evaluation of the objective, controls, and constraints. The contributions of this chapter are: the handling of the algorithmic singularities (Sections 3.4.2 - 3.4.6); the observations on constraint discontinuities and the examples of multimodality (Section 3.4.8); the combination of node and segment-based expressions (Section 3.4.9); the observations in Section 3.4.10; and the introduction in Section 3.5 of local quadratic interpolation of constraints.

Chapter 4 introduces a new quaternion-based point-mass inverse dynamics aircraft model that removes the singularities inherent in Euler-angle orientation representation and the discontinuities in the controls associated with those singularities. The main contribution of this chapter is the quaternion inverse dynamics model, specifically the derivation of the model in Sections 4.2, 4.3, 4.4, and 4.5, the test results of Sections 4.6 and 4.7, and the improved control expressions derived and tested in Section 4.8. Section 4.9 describes a minor contribution to the method.

Chapter 5 describes an investigation into the effects of the degree and form of airspeed polynomial parameterization on the robustness, optimality, and computational speed of optimization. The chapter presents results for Chebyshev, Bernstein, barycentric Lagrange and power series polynomials. The contributions of this chapter are: the introduction in Section 5.2 of an algorithm for evaluating maximum feasible airspeed without requiring airspeed parameterization or optimization; and the quantification of the effects of the degree and form of polynomial airspeed parameterization on optimization performance presented in Sections 5.3.3 and 5.4.

Chapter 6 extends the optimization research of Chapter 5 by investigating the combined effects of spatial and airspeed parameterizations and the choice of NLP algorithm on robustness, optimality, and computational speed. It provides comparative numerical assessments of several NLP algorithms when applied to the minimization of final flight time using the inverse dynamics method and quaternion model. The contributions of

this chapter are: the comparison of the performance of four different NLP algorithms applied to the inverse dynamics method over a range of degrees of airspeed parameterization as presented in Sections 6.3 and 6.4, in particular the performance of differential evolution compared to quasi-Newton and derivative-free NLP algorithms; and the identification of candidate optimization strategies in Section 6.4.5.

Chapter 7 describes an extension of the method to include negative-g trajectories. This overcomes the ill-conditioning in roll which otherwise arises when the normal load factor transitions through zero, thus removing unwanted step changes in bank angle at zero normal load factor and making the generated trajectories more suitable for aircraft operating as surveillance platforms. The contributions of this chapter are described in Section 7.2: improving platform stability around zero normal load factor; and permitting negative-g trajectories to be generated.

The following papers[29-33] have been published, accepted, or submitted for publication as a result of the research described in this document.

Journal papers:

Drury, R. G., Tsourdos, A., and Cooke, A. K., "Negative-g Trajectory Generation Using Quaternion-Based Inverse Dynamics", Journal of Guidance, Control, and Dynamics, Vol. 34, No. 1, 2011, (to appear)..

Drury, R. G., and Whidborne, J. F., "Quaternion-Based Inverse Dynamics Model for Evaluating Aerobatic Aircraft Trajectories," Journal of Guidance, Control, and Dynamics, Vol. 32, No. 4, 2009, pp. 1388-1391.

Conference papers:

Drury, R. G., Tsourdos, A., and Cooke, A. K., "Negative-g Trajectory Generation Using Quaternion-Based Inverse Dynamics," Proceedings of the AIAA Atmospheric Flight Mechanics Conference, Toronto, 2010.

Drury, R. G., Tsourdos, A., and Cooke, A. K., "Real-Time Trajectory Generation: Improving the Optimality and Speed of an Inverse Dynamics Method," Proceedings of the IEEE Aerospace Conference, Big Sky, 2010.

Drury, R. G., and Whidborne, J. F., "A Quaternion-Based Inverse Dynamics Model for Real-Time Trajectory Generation," Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit, Chicago, 2009.

2 BACKGROUND: TRAJECTORY GENERATION METHODS

Many approaches to trajectory generation have appeared, particularly in recent years as computational power has increased so that near-real-time trajectory generators are becoming feasible. This chapter provides the contextual background to the inverse dynamics direct method to support the thesis stated in Chapter 1.

2.1 Graph-Based Approaches

Most aircraft rely on pre-flight preparation of flight paths, which are typically prepared manually (or with basic computer support) as a sequence of waypoints, times, and altitudes interconnected by linear tracks, pre-determined airspeeds, and standard rate or procedural turns. These paths are input manually or electronically to the on-board flight management system or autopilot which then generates the control demands to track the desired flight path using feedback controllers. The dynamic constraints of the equations of motion and aircraft limits such as maximum normal load factor, maximum thrust and control limits are taken into account indirectly by restricting the paths to known feasible rates of climb/descent, airspeeds, and manoeuvres such as standard rate turns.

Graph theory and combinatorics can be applied effectively to path planning problems because such problems do not usually directly impose dynamic constraints (which is one way of distinguishing between path planning and trajectory generation). The field of robotics has given rise to path planning techniques such as subdivision of the overall path into cells, each of which is relatively simple to solve, and reconstituting the path by linking the cells to each other and to the boundary conditions using graph-theoretic ("roadmap") methods such as Voronoi diagrams or visibility graphs[8]. Roadmap methods construct a subset of paths that span the feasible space, then seek to select the path that is closest to optimal from the subset. Krozel and Andrisani[82] applied Voronoi diagrams to path planning in mountainous terrain. Bortoff[15] used a combination of Voronoi diagrams to generate an initial estimate followed by a potential field approach to model combinations of threats (hostile radar) and dynamic constraints on the trajectory; resulting in a nonlinear programming problem. Frazzoli et al.[51] described a

probabilistic approach to motion planning that incorporated dynamic constraints, using motion primitives and rapidly-exploring random trees; the approach assumes that the vehicle can be guided from any state to any desired final state, which itself requires the solution of an optimal control problem. Yang and Zhao[136] approached the problem of incorporating dynamic constraints within a graph-based method by discretizing the path and representing the dynamic constraints as bounds on consecutive nodes; the A* direct search method was then applied to the 4D search space with a linear objective function.

Gu et al.[59] gave an overview of five path planning approaches, including Voronoi diagrams, visibility lines (constructing straight line segments that do not intersect with obstacles), and Mixed Integer Linear Programming (MILP), and concluded that visibility lines was an efficient path planning method, although it did not take account of dynamic constraints. Kamal et al.[70] combined MILP with branch-and-bound algorithms to solve path planning problems but concluded that MILP was too computationally expensive for real-time application.

Eele and Richards[35] described a method for generating globally optimal 2D trajectories that avoid fixed obstacles, whilst taking nonlinear dynamics into account. They modified a branch-and-bound algorithm to decide which side of an obstacle to pass, then solved the resulting sub-problem using interior point optimization. This method has elements of both graph-based and optimal control approaches.

2.2 Geometric Guidance and Navigation

In 1957 Dubins[34] showed that in two dimensions the shortest distance between two boundary conditions which include position and orientation, subject to a path constraint on maximum curvature, is a combination of straight lines and circular arcs. Anderson et al.[1] described a method of generating 2D trajectories in two stages: firstly a waypoint sequence is produced without time constraints then a Dubins set is constructed and optimized, using curvature and airspeed as constraints on the turn radius and minimizing the deviation from the straight line between waypoints. Shanmugavel et al.[117] used Dubins sets as the basis for generating 2D trajectories for multiple cooperating UAVs, and extended that work into 3D[118], optimizing the path length using curvature as the primary constraint.

Classical missile guidance is a differential-geometric approach using line of sight (LOS) information as a primary input to the control law. To obtain LOS information semi-active missile sensors rely on a separate emitter to illuminate the target, passive sensors detect emissions generated by the target, and active missiles illuminate the target themselves. Active and passive missiles therefore have a degree of autonomy after launch. Proportional navigation (PN) is widely-used to guide the missile to the target. Ho et al.[63] showed that PN is an optimal control in that missile normal acceleration is minimized. Lewis and Syrmos[88] also give a derivation of PN as the solution to an optimal control problem. White et al.[131] categorized PN as True PN or Pure PN, which are distinguished by the direction in which missile normal acceleration is oriented: perpendicular to the line of sight for True PN and perpendicular to missile velocity for Pure PN. However, PN design is based on a constant target velocity vector, and does not take account of obstacles conflicting with the flight path. White et al.[132] examined the application of differential geometry to missile guidance for manoeuvring targets, also using LOS information. Ariff et al.[2] identified three shortcomings of PN: dependence on line of sight information; limited effectiveness against manoeuvring targets (which is dependent on permitted missile lateral acceleration; Ariff et al. also noted that this limitation can be partially mitigated by modifications to PN); and lack of direct control over the missile trajectory. They applied differential geometry, modelling the target trajectory by its curvature and torsion, to overcome these shortcomings without resorting to optimal control.

2.3 Dynamic Programming

In 1957 Bellman[9] published the dynamic programming approach to optimization based on the "principle of optimality". Considering a time sequence of decisions, Bellman's principle of optimality states that on an optimal path remaining decisions must be optimal irrespective of the preceding decisions or of the initial state. Therefore a decision path can be decomposed, starting from the end point, into shorter paths and working backwards in time optimal decisions can be calculated sequentially until the initial time is reached. Application of this principle to the continuous-time optimal control problem results in a system of first order nonlinear partial differential equations, known as the Hamilton-Jacobi-Bellman equation. The Hamilton-Jacobi-Bellman

equation leads to an optimal control via a two point boundary value problem which is not, in general, solvable analytically.

The Euler-Lagrange equations, Pontryagin's Maximum Principle[105] and Hamilton-Jacobi-Bellman equation are inter-related and can be derived from each other[22, 88].

The computational and storage loads imposed by dynamic programming are subject to the "curse of dimensionality" (Bellman[9]) and the method is not "computationally competitive" (Betts[12]) and is therefore not a promising candidate for on-board near-real-time trajectory generation.

2.4 Optimal Control

The optimal control approach to trajectory generation is to treat it as a calculus of variations problem: the optimization of a functional in a continuous, infinite dimensional space subject to nonlinear constraints. However, in general the resulting differential-algebraic system is not solvable analytically and the problem is transformed by discretization and parameterization to a finite-dimensional parameter optimization problem, which is then solved by the application of an NLP algorithm.

2.4.1 Statement of the Optimal Control Problem

The continuous optimal control problem may be written (Bryson and Ho[22], Lewis and Syrmos[88], Subchan and Żbikowski[125]) as the task of finding the admissible control **u** (and, optionally, the associated state **x**) that minimizes the scalar objective function

$$J = \phi(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\mathbf{x}(t), \mathbf{u}(t), t)\, dt \tag{2.1}$$

subject to

$$\mathbf{x}(t) \in \mathbb{R}^n \text{ and } \mathbf{u}(t) \in \mathbb{R}^m \tag{2.2}$$

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{2.3}$$

$$\mathbf{x}(t_0) = \mathbf{x}_0 \text{ specified} \tag{2.4}$$

$$\boldsymbol{\psi}(\mathbf{x}(t_f), t_f) = 0, \quad \boldsymbol{\psi} : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^q, q \leq n \tag{2.5}$$

Eq. (2.1) is written in the Bolza form with scalar final cost function ϕ and an integral general cost function. Eqs. (2.3) are the state equations of the dynamical system, and the problem is subject to q constraints on the final state $\boldsymbol{\psi}$ (Eq. (2.5)).

In practical problems additional algebraic path constraints **c** on the state and control variables are also likely to apply:

$$\begin{aligned} c_i(\mathbf{x}, \mathbf{u}, t) &= 0, \quad i \in E \\ c_i(\mathbf{x}, \mathbf{u}, t) &\leq 0, \quad i \in I \end{aligned} \tag{2.6}$$

with

$$E = \{1, \ldots m_e\} \tag{2.7}$$

$$I = \{m_e + 1, \ldots m_c\} \tag{2.8}$$

In Eqs. (2.6) each c_i is either an equality or inequality path constraint and may or may not be explicitly dependent on **x** or **u**.

2.4.2 Indirect Methods

The calculus of variations and Lagrange theory[14] provide a theoretical basis for solving the optimal control problem. As is well-known (e.g. Bryson and Ho[22]) the state equations can be adjoined to the objective function to form the Hamiltonian

$$H(\mathbf{x}, \mathbf{u}, t) = L(\mathbf{x}, \mathbf{u}, t) + \boldsymbol{\lambda}^T \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{2.9}$$

Path constraints can also be adjoined to the Hamiltonian using Lagrange multipliers. Setting the first variation of the Hamiltonian to zero gives the first order optimality conditions, comprising the state equations (Eq. (2.3)), the differential costate equation

$$\frac{\partial H}{\partial \mathbf{x}} = -\dot{\boldsymbol{\lambda}} \tag{2.10}$$

and the algebraic stationarity condition

$$\frac{\partial H}{\partial \mathbf{u}} = 0 \tag{2.11}$$

When control constraints are present, as will be the case in practical problems, the stationarity condition must be modified by applying Pontryagin's Maximum Principle[105] which states that on the optimal trajectory the Hamiltonian is minimal with respect to the controls, which can be written

$$H(\mathbf{x}^*, \mathbf{u}^*, \boldsymbol{\lambda}^*, t) \leq H(\mathbf{x}^*, \mathbf{u}, \boldsymbol{\lambda}^*, t), \quad \mathbf{u} \in U \tag{2.12}$$

Eqs. (2.3), (2.10), and (2.12), together with algebraic boundary (transversality) conditions at the two end points, form a differential-algebraic system that can, in principle, be solved as a two-point boundary value problem. However, solving the two-point boundary value problem analytically is difficult, and not feasible for most practical problems. Numerical methods are therefore required, as described in the next section. Unfortunately these are not currently fast enough to produce solutions in near real time. An initial estimate of the costate variables is also required.

2.4.2.1 Numerical Methods for Indirect Methods

Betts[12] described shooting and collocation methods for transforming the optimality conditions, path constraints and boundary conditions into an NLP problem. Shooting methods treat the two-point boundary value problem as a series of initial value problems and use well-known numerical integration schemes, such as the Runge-Kutta 4th order scheme or multi-step schemes such as Adams-Moulton or Adams-Bashforth[65, 124], to solve the initial value problems. Path constraints require that the sequence of constrained and unconstrained arcs be determined a priori, and that the trajectory be divided into corresponding phases. This in turn requires the introduction of additional variables and junction constraints. Betts[12] and Bryson and Ho[22] noted that shooting can produce "wild" trajectories and is very sensitive to a good initial guess, due to the instability of the integration which coupled with a poor initial guess can lead to divergence. In addition, Betts noted that small changes early in the trajectory have a disproportionate effect on the trajectory than similar changes late in the trajectory, and that this can have a "catastrophic" effect on the shooting method.

In 1989 Oberle and Grimm[100] described a software package, BNDSCO, that implements an indirect multiple shooting method in which the trajectory is divided into segments and shooting is used to solve each segment. The dimension of the problem is larger than for single-segment shooting since additional variables and constraints must be introduced at inter-segment boundaries. (The segments are not the same as the phases of constrained/unconstrained arcs: the segments are introduced to overcome the numerical limitations of single-segment shooting, and within each segment there may be multiple phases[12].) Subchan and Żbikowski[125] compared BNDSCO with various direct collocation methods, and found it to produce more accurate solutions than the direct methods, but that it was sensitive to initial guesses: direct methods were used to generate good initial guesses for the BNDSCO method.

In their 1992 paper, von Stryk and Bulirsch[130] used a hybrid direct/indirect method. A direct local collocation method was used to provide good initial guesses of the variables including the costates, constrained and unconstrained arcs. They found that the accuracy of their direct method was typically 1%. The BNDSCO package was then used to provide a more accurate solution. Pesch's 1994 paper[102] also described the use of multiple shooting for offline solution of a number of aerospace problems, and suggested that parallel implementations of multiple shooting might be fast enough for on-line trajectory generation if computational power increased sufficiently.

Collocation is a technique in which a residual function is minimized at a sequence of nodes, and it has been applied to indirect methods since at least the 1970s. Fahroo and Ross[39] described an indirect collocation method. Collocation avoids the computationally expensive numerical integration of the shooting methods, but when used as part of an indirect method it still requires good initial guesses of the costates and the constrained and unconstrained arcs[12, 102, 130]. It is widely used as part of a direct method, and to generate initial guesses for indirect multiple shooting. Collocation is described in the context of pseudospectral methods below.

2.4.2.2 Attributes of Indirect Methods

The key attributes of indirect methods (Betts[12, 13], Bryson and Ho[22], and Subchan and Żbikowski[125]) are:

1. Indirect methods seek a solution to the first order optimality conditions, and therefore, when a solution is found it will be accurate to the tolerance of the chosen NLP algorithm.
2. The radius of convergence is small.
3. Initial estimates of the costate variables, which are not physical variables and therefore are not intuitive, are required.
4. Initial estimates of the number and locations of the constrained and unconstrained arcs are required.
5. The combination of the state, costate and Maximum Principle equations can be ill-conditioned.
6. It is necessary to formulate the first order optimality conditions using the Euler-Lagrange equations and boundary conditions, and to solve the resulting two point boundary value problem.
7. Analytic derivatives are required in order to formulate the optimality conditions.
8. The dimension of the problem is increased by the inclusion of the costate variables.
9. Singular arcs may arise.

The main difficulties with indirect methods are: the creation of good initial guesses, including the costates, constrained and unconstrained arcs; small radii of convergence; the potential ill-conditioning of the equations; and the computational load of evaluating the state, costate and transversality conditions.

2.4.3 Direct Methods

For ease of comparison with the indirect methods of the previous sub-section, the key attributes of direct methods are listed here and discussed in the next sub-sections:

1. Direct methods seek to minimize the objective function directly, by transforming the functional optimization to a parameter optimization without formulating and solving the first order optimality conditions; therefore the solutions are not guaranteed to be accurate.
2. Direct methods in general result in multimodal optimization problems, so a solution is not guaranteed to be globally optimal.

3. The radius of convergence is larger than for indirect methods.
4. No initial guess of the costate variables or constrained arcs is required.
5. The methods are more robust because it is not necessary to solve potentially ill-conditioned state, costate and Maximum Principle systems.
6. The accuracy of a direct method depends on the approximations used to represent the states and controls, and on the number and distribution of discretization nodes (and collocation nodes if collocation is used).
7. The dimension of a direct method depends on the actual method chosen, but does not have to include the costate variables.
8. Singular arcs may, depending on the actual method, cause similar problems to those caused to indirect methods.
9. Estimates of the costates are not available as outputs from all direct methods.
10. The methods are computationally simpler.

For on-board near-real-time trajectory generation when the overall system accuracy is limited by atmospheric conditions, sensor precision, the accuracy of the aircraft model, and noise, high accuracy is less important than timely generation of a feasible solution that approximates an optimal solution. Moreover, the task is to repeatedly solve a sequence of related time-critical problems defined by time-varying boundary conditions and path constraints, rather than to solve one problem to high accuracy: each solution has only a short period of validity. The priority is to generate pseudo-optimal solutions within the required time intervals, a task for which fast direct methods are well suited.

Various authors have used different taxonomies for direct methods, with the same names being used with different meanings. In 1996, Rutherford and Thomson[114] categorized inverse simulation direct methods as either integration or differentiation methods according to whether time-stepping numerical integration was used in the method; this categorization can be applied to other direct methods. They also presented numerical results showing that differentiation methods could be an order of magnitude faster than integration methods. Boyd[18] describes a similar categorization.

A useful categorization was described by Hull[65] in 1997. He categorized direct methods by the subset of state and control variables that are parameterized. If only control and some of the state variables are parameterized then time-stepping numerical integration

of the remaining state equations is required (e.g. direct shooting or direct multiple shooting), but if all state variables are parameterized, then time-stepping numerical integration is not required (e.g. direct transcription[60], pseudospectral[10, 40, 60], differential inclusion[83, 116] or inverse dynamics). The next three sections use this categorization.

2.4.3.1 Control Parameterization: Shooting Methods

If only controls are parameterized the state equations are numerically integrated and iteration is used to find the control parameters that produce a pseudo-optimal trajectory that satisfies the state equations, boundary conditions and path constraints. Direct shooting and direct multiple shooting are analogous to the corresponding indirect methods, use the same numerical integration algorithms, and are also at risk of divergence and instability. If gradients must be obtained by finite differencing then n trajectories must be numerically integrated for each gradient at each NLP iteration[12]. Hence although they typically result in a lower dimension NLP problem than pseudospectral methods, they are slower due to numerical integration of many candidate trajectories with small time steps. Hull also noted that path constraints could be incorporated into shooting and collocation methods but in general were only satisfied at discrete nodes in the trajectory and not across inter-node segments. In these methods constraint satisfaction may be improved by increasing the number of nodes, and in practice if the segment duration is sufficiently short an infeasible trajectory will violate control constraints.

Shooting methods are therefore not well suited to on-board near-real-time trajectory generation due to their poor stability and slow computation.

Betts[12] gives a detailed description of direct shooting, and further references can be found in von Stryk and Bulirsch[130]. A modification of the shooting methods, the Adjoint method, is described in Polak[104].

2.4.3.2 State and Control Parameterization

Parameterization of the states and controls leads to a system of algebraic equations from which the parameters may be determined. The states and controls are parameterized in the form

$$\mathbf{y}(t) = \sum_{i=0}^{N} \alpha_i \zeta_i(t) \tag{2.13}$$

and, for a set of coefficients α and trial functions ζ, the parameterized candidate solution is substituted into the state equations (2.3) to give a residual error vector **R** at each node. Then the inner product of **R** with test functions **w** provides constraints:

$$\langle \mathbf{w}_k, \mathbf{R}_k \rangle = 0, \quad k \in K \tag{2.14}$$

where K is a set of discrete nodes along the trajectory.

Three widely-used methods for implementing Eq. (2.14) are:

- Lanczos-τ. Lanczos introduced the τ method for determining the coefficients in 1938[86]; he used Chebyshev polynomials as the test functions **w** with the Chebyshev inner product.
- Galerkin. In the Galerkin method the original basis functions are transformed into new basis functions that satisfy the boundary conditions and then the new basis functions are also used as the test functions **w**, i.e. the test functions are the same as the trial functions[25].
- Collocation. In the collocation method, the test functions are Dirac Delta functions, which simplifies Eq. (2.14) to setting the residuals to zero[18]:

$$\mathbf{R}_k = 0, \quad k \in K \tag{2.15}$$

Whichever method is chosen, the resulting parameter optimization problem is solved using an NLP algorithm.

Hargraves and Paris[60] and von Stryk and Bulirsch[130] described direct collocation methods using local parameterization functions. The NTG algorithm[96] used B-splines because of their local support property, and allowed the user to specify the distribution of the collocation nodes.

Spectral methods[18, 129] are widely used for solving ordinary and partial differential equations, particularly in fluid dynamics, and use global parameterization functions.

Spectral methods that use collocation were termed pseudospectral methods by Orszag in 1972[101]. Fahroo and Ross[44] described pseudospectral methods as spectral methods with global interpolants and collocation at Gauss, Gauss-Radau or Gauss-Lobatto points, i.e. the roots or extrema of orthogonal polynomials ("orthogonal collocation") together with 0,1, or 2 end points. Hence orthogonal collocation and pseudospectral are often used synonymously; Enright and Conway[38] explained the relationship between direct transcription and direct collocation as two perspectives on the same approach, and Hull[65] equated direct transcription with collocation. There are many variations on local, spectral and pseudospectral methods, and due to the long history and wide range of application of these methods the terminology varies considerably across the literature. The preface in Trefethen[129] and Chapters 3 and 21 of Boyd[18] provide further background on the origin and development of spectral and pseudospectral methods for solving partial and ordinary differential equations.

In a pseudospectral method for optimal control on the interval $\tau \in$ [-1,1], the states and controls are interpolated over two sets of interpolation nodes *N* and *M* respectively

$$\tilde{\mathbf{x}}(\tau) = \sum_{i \in N} \mathbf{x}(\tau_i) L_i(\tau) \tag{2.16}$$

$$\tilde{\mathbf{u}}(\tau) = \sum_{i \in M} \mathbf{u}(\tau_i) L_i^*(\tau) \tag{2.17}$$

where the ~ accent denotes the parameterized approximations, and the basis functions L and L^* are Lagrange polynomials (Fahroo and Ross[113] showed that in general these need not be Lagrange polynomials). Then

$$\dot{\tilde{\mathbf{x}}}(\tau) = \sum_{i \in N} \mathbf{x}(\tau_i) \dot{L}_i(\tau) \tag{2.18}$$

The residual is evaluated at a set of collocation nodes *K*

$$\mathbf{R}_k = \dot{\tilde{\mathbf{x}}}(\tau_k) - \mathbf{f}(\tilde{\mathbf{x}}(\tau_k), \tilde{\mathbf{u}}(\tau_k), \tau_k), \quad k \in K \tag{2.19}$$

For any given choice of nodes *K*, the derivatives of **x** at the nodes *K* can be expressed as functions only of the values of **x** at the interpolation nodes *N* using a differentiation matrix **D**

$$\dot{\tilde{\mathbf{x}}}(\tau_k) = \sum_{i \in N} D_{ki} \mathbf{x}(\tau_i) \tag{2.20}$$

The differentiation matrix is a key feature of pseudospectral methods and is readily determined (Press et al.[108], Berrut and Trefethen[11], Fahroo and Ross[44])

$$D_{ki} = \dot{L}_i(\tau_k) = \sum_{i \in N} \frac{\prod_{j \in N, j \neq i,k} (\tau_k - \tau_j)}{\prod_{j \in N, j \neq i} (\tau_i - \tau_j)}, \quad k \in K \tag{2.21}$$

Clearly D_{ki} is only dependent on the choice of two sets of nodes: the state interpolation nodes *N* and the collocation nodes *K*, so it can be computed offline once, and does not need to be updated during the optimization.

The residual conditions then become

$$\mathbf{R}_k = \sum_{i=1}^{N} D_{ki} \mathbf{x}(\tau_i) - \mathbf{f}(\tilde{\mathbf{x}}(\tau_k), \tilde{\mathbf{u}}(\tau_k), \tau_k) = 0, \quad k \in K \tag{2.22}$$

The integral term in Eq. (2.1) is evaluated by numerical quadrature, so the nodes used to evaluate the objective are chosen to minimize the error in the quadrature formula, i.e. Gaussian quadrature is used at the roots or extrema of orthogonal polynomials[18, 129]. If the interpolation nodes are chosen in this way, the ill-conditioning that arises with polynomial interpolation on uniformly-spaced grids is avoided. Since the Gauss nodes do not include the end points, many methods employ Gauss-Radau or Gauss-Lobatto nodes to explicitly include the initial and/or final boundary points. The interpolation nodes *N*, *M* and the collocation nodes *K* need not be identical sets, and some methods (e.g. the Gauss pseudospectral method[10]) use multiple overlapping node sets to enhance accuracy and generate costate estimates.

Parameterizing the states and controls, using quadrature to approximate the integral in the objective, and requiring that the residual be zero at the collocation nodes together

with the path constraints and boundary conditions, transforms the optimal control problem into a constrained parameter optimization problem.

In the aerospace field pseudospectral methods are widely used for trajectory generation, usually distinguished by the choice of the collocation points. In 1995 Elnagar et al.[37] described a Legendre pseudospectral method using Lagrange polynomial parameterization with Legendre-Gauss-Lobatto points, and in 1998 Elnagar and Kazemi[36] described a variant using Chebyshev-Gauss-Lobatto points. Fahroo and Ross have published a number of papers on both Chebyshev-Gauss-Lobatto[42] and Legendre[40] pseudospectral methods, including the use of Legendre-Gauss-Radau points[43, 44]. Another variant is the Gauss pseudospectral method described by Benson et al.[10] in 2006 which uses Legendre-Gauss points.

An advantage of the Gauss and Legendre pseudospectral methods is the development of methods to produce estimates of the costates[10, 40, 55], by parameterizing the costates in a similar form to **x** and **u**. These estimates are useful in two ways: they enable the accuracy of the solution to be assessed, and if required they can be used as part of an initial guess for an indirect method, typically multiple shooting.

A disadvantage of pseudospectral methods compared to inverse dynamics is the dimension of the resulting NLP problem. For the Gauss pseudospectral method the dimension is $n(K+2)+mK$ (Huntington[66]) which, even with a low number of nodes (typically 5-20), typically leads to dimensions of $\mathcal{O}(100)$; for example Yakimenko et al.[5] showed that using an Euler-angle point-mass aircraft model with only 6 states and 3 controls, for 200 nodes the dimension was 1801. For the inverse dynamics method the dimension is dependent on the degrees of the spatial and airspeed parameterizations but independent of the number of nodes, typically leading to a 4-15 dimension NLP problem.

2.4.3.3 State Parameterization

Differential Inclusions

In 1994 Seywald[116] introduced a direct method using differential inclusions, as an alternative to collocation methods. The differential inclusion method does not require

the controls to be parameterized because a change of variables is used to replace the controls with approximations to the states and state rates of change, and the control constraints are replaced by constraining the state rates over each inter-node segment to the attainable set. The states are parameterized and the required approximations of the states are defined as functions of the parameterized values at the nodes

$$\bar{\mathbf{x}}_i = \frac{\mathbf{x}_{i+1} + \mathbf{x}_i}{2}; \quad \dot{\bar{\mathbf{x}}}_i = \frac{\mathbf{x}_{i+1} - \mathbf{x}_i}{\delta t_i} \tag{2.23}$$

The state equations are solved for **u** as functions of the states and state rates

$$\mathbf{u} = \mathbf{u}(\dot{\mathbf{x}}, \mathbf{x}) \tag{2.24}$$

Eqs. (2.23) are used to approximate **u** as

$$\mathbf{u} = \mathbf{u}(\dot{\bar{\mathbf{x}}}, \bar{\mathbf{x}}) \tag{2.25}$$

Eqs. (2.23) and (2.25) are substituted into the control constraints and state equations to give algebraic constraint equations which, with any state constraints, form a set of constraint equations for input to an NLP algorithm. Kumar and Seywald[83] showed that the dimension of an NLP problem in differential inclusion form was smaller, for the same number of nodes, than the equivalent collocation form, and hence argued that the solution would converge more quickly. Conway and Larson[26] showed that the state rate approximation Eq. (2.23) was equivalent to using implicit Euler quadrature, and argued that collocation with higher order quadrature rules would achieve the same accuracy as differential inclusions but require fewer nodes, and hence have lower NLP dimension. Fahroo and Ross[41] presented an analysis of the application of differential inclusions with the Legendre pseudospectral method[40], in which the differentiation matrix increased the accuracy of the state derivative without needing to increase the number of nodes; they concluded that this approach to discretizing the differential inclusions was computationally competitive.

An advantage of the differential inclusion method is that it does not require **u** to be parameterized, nor any initial guess of **u**. However, irrespective of the dimension of the

NLP problem compared to a collocation formulation, the differential inclusion method suffers from four disadvantages:

- Unless the state equations are linear in **u**, solving them for **u** is difficult and likely to require an iterative method[61, 83].
- It is more difficult to obtain analytical derivatives of the constraints than for the collocation formulation[83].
- It does not provide a solution for **u**, which must be evaluated once the state solution has been found.
- A Bolza problem has to be transformed to a Mayer problem (which can usually be accomplished by introducing one additional state variable)[41].

In passing, it is noted that the well-known point-mass model of wind-axes aircraft equations of motion using Euler-angles to represent orientation (e.g. Stengel[122]) is not linear in all the controls (Section 3.2.2). The quaternion-based point-mass model introduced in Chapter 4 is linear in **u** and is therefore better suited to a differential inclusion formulation than the Euler-angle model.

Inverse Simulation

In the 1990s two classes of "inverse simulation" methods to solve trajectory generation problems appeared in the literature: integration-based and differentiation-based methods. In 1991 Hess et al.[61] described an integration-based method, in which an output trajectory is discretized, an initial guess of $\mathbf{u}(t_0)$ is applied to a numerical integration routine to generate an output at the next discretization node, the error between the desired output trajectory and the generated output is evaluated and Newton's method is used to minimize this error over admissible **u**. The process is then repeated for the next discretization segment. Although the method avoided the inherent noisiness of numerical differentiation, the method was susceptible to oscillations[89]. It also required numerical integration, which is computationally expensive. The similarity to shooting methods is clear.

In 1996 Rutherford and Thomson[114] described and compared integration-based and differentiation-based inverse simulation. The differentiation-based approach used numerical differentiation to derive the state rates and controls directly from the desired

states using the state equations, again using Newton's method for solving the system of equations. They concluded that both the integration and differentiation inverse simulation methods suffered from instabilities in certain cases, and that the differentiation method was an order of magnitude faster.

Inverse Dynamics

Inverse dynamics has been used for many years for flight control (Hess et al. [61], Lane and Stengel[87]) to evaluate the controls that drive a system to follow a desired state trajectory.

In 1993 Lu[91] described how inverse dynamics could be extended to trajectory optimization applied to a low-Earth orbit ascent trajectory; he used a point mass vehicle model in polar coordinates, cubic splines as parameterization functions, and parameterized one state and one control thus also using numerical integration. In their 1996 paper, Kato and Sugiura[74] described an inverse dynamics approach to trajectory generation in which the control vector was calculated from the parameterized output trajectory by inverting the state equations. They used a body-axes aircraft model, with angle of attack (α), sideslip (β) and roll angle (ϕ) as controls, and manipulated the force and moment equations to obtain a system of nonlinear equations in α, β, and ϕ, which was then solved using an iterative method. They used finite differences for numerical differentiation. Sentoh and Bryson[115] adapted Kato and Sugiura's method as an initial guess for an optimal control method. Lou and Bryson[90] used a wind-axes aircraft model with α, bank angle (μ) and thrust as controls; since they parameterized thrust but did not parameterize a sufficient subset of the states (see next but one paragraph), their method required numerical integration. They used the inverse dynamics output as an initial guess for an optimal control problem formulated as a differential inclusion problem.

In 1999 Jaddu and Shimemura[68, 69] used state parameterization by Chebyshev polynomials to transform an optimal control problem with quadratic performance index

$$J = \mathbf{x}(t_f)^T \mathbf{S}\mathbf{x}(t_f) + \int_0^{t_f} (\mathbf{x}^T \mathbf{Q}\mathbf{x} + \mathbf{u}^T \mathbf{R}\mathbf{u}) dt \tag{2.26}$$

subject to state equations (2.3), boundary conditions (2.4) and (2.5), and linear bounds on **u**, into a quadratic NLP problem with **u** as a function of the coefficients of the state

parameterization coefficients, i.e. the problem was treated as an inverse dynamics trajectory generation problem.

For a consistent set of *n* state equations with *m* controls, if $m \le n$ then the state equations form an over-determined system for the *m* controls and, provided that sufficient states are parameterized with suitably smooth functions to enable the remaining states to be expressed as a function of the parameterized states, all the controls are uniquely defined by the set of parameterized states and their analytic derivatives, without numerical integration of any of the state equations. The parameterization functions must be chosen to be sufficiently continuous that all required derivatives are available analytically. Therefore, it is desirable that the state equations allow a set of parameterized states to be chosen such that each control can be expressed as

$$u_j = u_j\left(\mathbf{x}_P, \dot{\mathbf{x}}_P, \ddot{\mathbf{x}}_P, ...\mathbf{x}_P^{(b)}\right) \tag{2.27}$$

where

$$\mathbf{x}_P := \left\{x_i \mid i \in \mathbf{P}\right\}, \quad \mathbf{P} := \left\{1, ...p\right\}, \quad p \le n \tag{2.28}$$

so that the controls can be evaluated analytically from $\mathbf{x}_p$ and its derivatives. If the controls cannot be evaluated analytically, then an iterative approach is required.

Fliess et al.[48, 49] introduced the idea of differential flatness, which is a generalized set of conditions for evaluating the controls from outputs that are functions of the states and controls:

- the state and control vectors must be directly expressible, without integrating any differential equations, as real analytic functions of a flat output **y** and a finite number of its derivatives:

$$\mathbf{x} = \mathbf{g}(\mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}, ...\mathbf{y}^{(s)}); \quad \mathbf{u} = \mathbf{h}(\mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}, ...\mathbf{y}^{(q)}); \quad s, q \in \mathbb{Z}^+ \tag{2.29}$$

and

- any component of the flat output **y** must be expressible as a real analytic function of the state vector, control vector, and a finite number of derivatives of the control vector:

$$\mathbf{y} = \mathbf{w}(\mathbf{x}, \mathbf{u}, \dot{\mathbf{u}}, \ddot{\mathbf{u}}, \ldots \mathbf{u}^{(r)}); \quad r \in \mathbb{Z}^{+} \tag{2.30}$$

The same authors[49] also showed that a differentially flat system is controllable, and Martin[93] showed that a conventional fixed wing aircraft in forward flight is differentially flat.

The key benefit of differential flatness in this context is that it guarantees that, for a given **y** trajectory, the corresponding **x** and **u** trajectories can be evaluated without integrating any of the state equations. The inverse dynamics method for solving a trajectory generation problem is then reduced to the NLP problem of finding an output trajectory $\mathbf{y}^{*}$ that satisfies the boundary conditions (Eqs. (2.4) and (2.5)) and

$$\mathbf{y}^{*} = \arg\min_{\mathbf{y} \in \mathbf{Y}} J(\mathbf{y}) \tag{2.31}$$

where **Y** is the set of admissible output trajectories and path constraints are expressed as functions of the output space

$$\mathbf{s}(\mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}, \ldots) \leq \mathbf{0} \tag{2.32}$$

Ross and Fahroo[112] note that the choice of outputs **y** instead of states **x** might make the objective, boundary conditions, and constraints more complicated, and hence might worsen real time trajectory generation. In the inverse dynamics method investigated in this research the outputs are simply a subset of the states, which avoids this potential problem.

In aircraft state-space models the state vector usually includes the aircraft position. It is obvious that, when position variables are parameterized with respect to time, the time derivatives of position are then also defined, i.e. the velocity of the dynamical system is defined and cannot be optimized independently of the position variables. Lu[91] addressed this problem by transforming the aircraft model so that polar angle was the independent variable. In his 2000 paper that introduced the inverse dynamics method to

the West[133] Yakimenko described work carried out in Russia by Taranenko and colleagues[126, 127], originating in the 1960s. In this work Taranenko introduced, as part of a collocation method, the concept of a real nonnegative strictly monotonically increasing "virtual arc" $\tau \in [0, \tau_f]$: parameterizing the states with respect to τ allows the speed to be parameterized (also with respect to τ) independently of position. Taranenko also defined the scalar "speed factor" $\lambda := d\tau/dt$ (not costates or Lagrange multipliers) which defines the relationship between functions of τ and functions of time.

Taranenko used low-order polynomials and trigonometric functions for parameterization, in collaboration with Momdzhi he also used cubic splines; other Russian research described by Yakimenko used trigonometric functions for helical manoeuvres.

In his 2000, 2008 and 2010 papers[5, 133, 135] Yakimenko used the virtual arc with degree 7 global polynomials to parameterize the position states of a point-mass aircraft model that satisfied Eq. (2.27). He discussed using polynomials to parameterize *v*, but in his examples chose to parameterize one control (throttle) as a bang-bang control with two switching points and to numerically integrate the corresponding state equation. Kaminer et al.[71-73] applied the method to generate trajectories for multiple unmanned aircraft and described parameterizing airspeed explicitly, or implicitly via λ, thus parameterizing a sufficient subset of states and obviating the need for integration of any state equations to evaluate **u**.

Numerical simulations[5, 17, 114, 135] suggest that this inverse dynamics method is faster than other candidate direct methods such as shooting or pseudospectral methods, and fast enough for on-board near-real-time aircraft trajectory generation. However, the literature typically examines a few examples of the performance of the method in detail, rather than a statistical analysis of performance across a larger sample.

2.5 Nonlinear Programming Algorithms

Indirect and direct methods transcribe the optimal control problem from a functional optimization (infinite-dimensional) problem into a parameter optimization (finite-dimensional) problem which is solved using a nonlinear programming algorithm. The

choice of NLP algorithm is critical to the robustness, optimality, and computational speed of the method in which it is used. This choice is separate from the choice of indirect or direct method, but the two decisions are inter-related, by for example the availability of analytic gradients and second derivatives, by the quality of initial guesses, or by choice of merit function for a line search[12].

The parameter optimization problem may be written

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbf{X}} J(\mathbf{x}) \tag{2.33}$$

subject to

$$\mathbf{X} = \{\mathbf{x} \mid \mathbf{x}_l \le \mathbf{x} \le \mathbf{x}_u, \mathbf{x} \in \mathbb{R}^n\} \tag{2.34}$$

$$\begin{aligned} f_i(\mathbf{x}) &= 0, \quad i \in E \\ f_i(\mathbf{x}) &\le 0, \quad i \in I \end{aligned} \tag{2.35}$$

with

$$E = \{1, \ldots m_e\} \tag{2.36}$$

$$I = \{m_e + 1, \ldots m\} \tag{2.37}$$

In this section the nomenclature is different to that of the preceding sections; in particular n, $\mathbf{x}$, $\boldsymbol{\lambda}$, $\mathbf{f}$, and J have different meanings: n is the dimension of the problem, $\mathbf{x}$ is a vector of n parameters (free variables), $\mathbf{x}_l$ and $\mathbf{x}_u$ are upper and lower bounds on $\mathbf{x}$, $\mathbf{f}$ is a vector of constraint functions, $\boldsymbol{\lambda}$ (to appear below) is a vector of Lagrange multipliers (not costates), and J is a scalar objective function of $\mathbf{x}$.

2.5.1 Gradient Algorithms

For an unconstrained problem $E=I=\varnothing$, the classical Newton's method (e.g. Fletcher[47]) uses the iteration

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{p}^{(k)} \tag{2.38}$$

where the Newton search direction **p** is defined by

$$\mathbf{p}^{(k)} = -\left(\mathbf{G}^{(k)}\right)^{-1}\mathbf{g}^{(k)} \tag{2.39}$$

The superscript denotes the iteration count, **g** is the gradient of *J* and **G** is the Hessian of *J*. For a quadratic function, Newton's method converges in a single iteration to a solution that satisfies the first and second order necessary conditions for optimality

$$\begin{aligned}\mathbf{g}(\mathbf{x}) &:= \nabla J(\mathbf{x}^*) = \mathbf{0}\\ \mathbf{G}(\mathbf{x}^*) &:= \nabla^2 J(\mathbf{x}^*) \geq 0\end{aligned} \tag{2.40}$$

The second order sufficient condition is that $\mathbf{G}(\mathbf{x}^*)$ be positive definite. Clearly Newton's method requires that *J* be at least twice continuously differentiable.

Newton's method is quadratically convergent close to a minimum provided that $\mathbf{G}(\mathbf{x}^{(k)})$ is positive definite, but the method may be slow to converge, or may diverge (because although the Newton direction at a local minimum $\mathbf{x}^*$ is zero it may be uphill at $\mathbf{x}^{(k)}$, if for example the underlying quadratic model is not a good approximation to *J* at $\mathbf{x}^{(k)}$, and $\mathbf{G}(\mathbf{x}^{(k)})$ may not be positive definite). This restricts the radius of convergence of the method. Two approaches to modifying Eqs. (2.38)-(2.39) to stabilize the method are line searches and trust regions (Nocedal and Wright[99]).

In the line search approach (Powell[106], Gill[54]) a step length α is introduced

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha\mathbf{p}^{(k)} \tag{2.41}$$

The step length is usually determined either by a minimum of a local cubic or quadratic model, and to satisfy a sufficient decrease condition. Trust region methods handle non-symmetric-positive-definite Hessians (Kelley[75]), and modify not just the step length but also the search direction to minimize a model function within an *n*-dimensional region around the current iterate.

The requirement to evaluate the inverse of the Hessian to solve Eq. (2.39) is a major limitation. First order gradient methods use the gradient but not the Hessian: the method of steepest descent is equivalent to replacing the Hessian in Eq. (2.39) with the

identity matrix. Other methods such as conjugate gradient and Levenberg-Marquadt use modified search directions based on the gradient but not on the Hessian. Although these methods retain global convergence (convergence to a local minimum from any starting point, but not necessarily convergence to a global minimum), they are generally less efficient than quasi-Newton methods.

Quasi-Newton methods seek to retain the efficiency of Newton's method without the problems of evaluating **G** by replacing it in Eq. (2.39) with a symmetric-positive-definite approximation that is updated iteratively and converges to **G**; the most widely-used technique in known as the BFGS update after Broyden, Fletcher, Goldfarb, and Shanno who independently proposed the update in 1970 (Fletcher[47]). Hence quasi-Newton methods seek solutions that satisfy first order optimality conditions, utilizing function and gradient information and an approximation to the Hessian.

When *m* constraints apply (including inequality and equality constraints), Newton or quasi-Newton methods can be applied by replacing *J* with an augmented objective, the Lagrangian, using *m* Lagrange multipliers $\boldsymbol{\lambda}$

$$J_a = J + \boldsymbol{\lambda}^T \mathbf{f} \tag{2.42}$$

In this case, subject to appropriate constraint qualifications (Bazaraa et al.[7]), the first order optimality conditions become the Karush-Kuhn-Tucker (KKT) necessary conditions

$$\begin{aligned} &\mathbf{g}(\mathbf{x}) := \nabla J_a(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0} \\ &f_i(x^*) = 0, \quad i \in E \\ &f_i(x^*) \le 0, \quad i \in I \\ &\lambda_i^* \ge 0, \quad i \in I \\ &\lambda_i^* f_i(x^*) = 0, \quad i \in E \cup I \end{aligned} \tag{2.43}$$

Under suitable convexity and differentiability conditions[7] on *J* and **f**, the KKT conditions are also sufficient for **x** to be a local minimum of *J*. The KKT conditions can be used to determine the search direction for a line search method and to verify that an

optimal solution has been found. The Lagrange multipliers can be used to identify the active set and assess the sensitivity of the optimal solution to changes in the constraints.

The quasi-Newton update is usually implemented as part of a sequential quadratic programming (SQP) method. At each major iteration the objective is approximated by a quadratic function with linearized constraints and an estimate of the active set. Since the objective, constraints, and active set are approximations, the solution of the sub-problem is only an approximation to the solution of the original problem, which is used to generate a step length and direction, and updated active set estimate, for the next major iteration. The BFGS update is applied at each major iteration (the quadratic sub-problem assumes a constant Hessian by definition). Maintaining estimates of the active set is computationally expensive, since there are 2^a possible sets of a inequality constraints.

Sequential quadratic programming methods have the advantages of global convergence, rapid convergence through exploitation of the Hessian approximation, accuracy through satisfaction of the KKT conditions, and direct handling of equality and inequality constraints, but they also require expensive arithmetic operations to

- Update the Hessian approximation.
- Update the quadratic objective, linearized constraints and active set.
- Solve the quadratic sub-problems.
- Apply line search or trust region stabilization of step length and direction.
- Evaluate the KKT conditions.

These methods are sensitive to the smoothness and convexity of the objective and constraints. If analytic derivatives are not available and must be evaluated by finite differences then the concomitant errors will, at best, slow convergence, and may cause non-convergence. Nocedal and Wright[99] also note that SQP methods tend to be most efficient when there are almost as many active constraints as dimensions.

Quasi-Newton SQP algorithms such as SNOPT[52, 53] have been used successfully in well-known implementations of the Gauss, Legendre, and Chebyshev pseudospectral methods described in Section 2.4.3.2.

2.5.2 Derivative-Free Algorithms

Trajectory generation objectives and constraints are in general nonsmooth, analytic derivatives are impracticable, and the requirement that the objective and constraints be at least one or two times continuously differentiable may not be satisfied. Derivative-free unconstrained optimization methods are characterized by not requiring the gradient or Hessian of the objective. They do not evaluate the KKT conditions or maintain active set estimates, and as such are computationally simpler, may have a larger radius of convergence, and can be more robust on nonsmooth or discontinuous problems, but the solution may not be an accurate minimum or a KKT point. Since they do not exploit derivatives the search directions may be less optimal than for gradient methods, especially those that use exact or approximate Hessians: convergence is therefore slower. Derivative-free methods have been published since at least the 1950s, but analysis and proofs of multi-dimensional convergence are still appearing[81, 84]. Two of the most successful derivative-free methods are the Nelder-Mead simplex algorithm[98] and the Hooke-Jeeves pattern search[64], introduced in 1961 and 1965 respectively.

The Nelder-Mead simplex algorithm has been widely used in many fields, despite a lack of convergence proofs until the late 1990s. The algorithm builds a simplex of points, initially regular, about the initial guess and evaluates the objective at each point of the simplex. It then uses a search direction away from the worst point, through the centroid of the remaining points, with a user-defined step length. If this "reflection" point is an improvement on the second worst point in the simplex, but not on the best simplex point, it is accepted. If the reflection is better than the best simplex point an expansion point is evaluated by stepping further in the same direction and the best of the reflection or expansion points is accepted. Otherwise contraction points are evaluated by using reduced step lengths with the same search direction. If a better point than the worst point is found, it is accepted. Once a point has been accepted, the accepted point replaces the worst point in the simplex; if no point is accepted, the simplex volume is shrunk. The process is then iterated until the termination criteria are satisfied. The 1997 paper by Lagarias et al.[84] provides an updated description of the algorithm which removes some ambiguities of the original paper, and provides a convergence analysis.

In 2002 Price et al.[109] described a provably-convergent variant (for C^1 functions subject to certain conditions) of the Nelder-Mead algorithm, building on Lagarias et al.[84]. Prior to Price's paper there was little convergence theory for the algorithm despite its wide use, and in 1998 McKinnon (see Powell[107]) demonstrated that it failed on a class of convex smooth functions in two dimensions, due to degeneration of the simplex. Price modified the algorithm to measure the volume of the simplex, evaluate the function at an additional point and reshape the simplex using a new basis set of orthogonal vectors if the simplex failed to meet volume requirements, so as to guarantee convergence. The shrink step of the original algorithm was removed. These modifications improve theoretical convergence, but complicate the algorithm; however it remains less complex than the SQP methods.

The Hooke-Jeeves algorithm explores around a base point (initialized to the initial guess) along n fixed search directions starting with a user-specified step length (which may be different in each direction). In the original algorithm, the exploration search directions are the dimensions of $\mathbf{x}$. If a reduction in the objective is found, the algorithm makes a pattern move from the base point of twice the vector difference of the base point and the current best point. An exploration is then made around this pattern point; if a reduction is found compared to the base point, the base point is updated with the latest best point and a further pattern move is made. When a pattern move fails the previous best point becomes the base point and an exploration is made around this new base point; if a base point exploration fails the step length is reduced, and a new exploration around the same base point is made. When the improvement in the objective and the step length are less than user-specified values the algorithm terminates. The algorithm is easily implemented and code is readily available, e.g. Kelley[75], who also provided a convergence proof and implemented a caching strategy to reduce re-evaluation of points, or Bunday[24]. In 1997 Torczon[128] provided a convergence proof that relied only on simple decrease in the objective (as used in the original algorithm) rather than sufficient decrease. These proofs assume that the objective has continuous first derivatives (Powell[107]), although the derivatives are not used by the algorithm.

Each Hooke-Jeeves exploration requires up to 2*n* function evaluations; the number of reflection, expansion and contraction points in each iteration of the Nelder-Mead algorithm is not explicitly dependent on *n*. Hence the number of function evaluations required by the Nelder-Mead algorithm is less sensitive to *n* than that of the Hooke-Jeeves algorithm.

The Nelder-Mead and Hooke-Jeeves algorithms require adaptation to apply to constrained optimization. To incorporate nonlinear constraints the constrained problem may be transformed to a sequence of unconstrained problems by adding a barrier or penalty function to the objective.

Barrier functions seek to keep **x** within a feasible region by increasing the modified objective as a constraint boundary is approached; common barrier functions are the logarithm or inverse[16]. These methods may fail when a constraint is violated because the barrier function makes it difficult for the algorithm to re-enter a feasible region (Price et al.[110]). A second disadvantage of these methods is that the edges of the feasible regions, where an optimal solution might be expected to lie, are penalized.

Penalty functions include a term for each constraint which is zero when the constraint is not violated and positive otherwise (Nocedal and Wright[99]), thus not penalizing the edges of feasible regions and creating a pressure towards feasibility. The standard transformation[56] of the parameter optimization problem (Eqs. (2.33)-(2.37)) using penalty functions may be written

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbf{X}}(J(\mathbf{x}) + \rho P(\mathbf{c}^+)) \tag{2.44}$$

subject to

$$\mathbf{x}_l \le \mathbf{x} \le \mathbf{x}_u \tag{2.45}$$

where

$$c_i^+ = \begin{cases} c_i(\mathbf{x}), & i \in E \\ \max(0, c_i(\mathbf{x})), & i \in I \end{cases} \tag{2.46}$$

The penalty function is a weighted sum of the terms in $\mathbf{c}^+$; the penalty weights k_i are chosen by trial and error so as to balance the relative importance of each constraint. From an engineering perspective, the penalty weights may also be considered as ensuring that the dimensions (mass, distance, time, etc) of each term are consistent with the other terms.

Common penalty functions are the quadratic or squared two-norm

$$P = \sum_{i \in E \cup I} k_i \left\| c_i^+ \right\|_2^2 \tag{2.47}$$

or the one, two or infinity norms

$$P = \sum_{i \in E \cup I} k_i \left\| c_i^+ \right\|_a, \quad a \in \{1, 2, \infty\} \tag{2.48}$$

The squared two-norm is smooth but not exact: exact penalty functions have the property that a finite value of ρ exists at which the solution $\mathbf{x}^*$ of Eq. (2.44) equals the solution of the constrained problem Eq. (2.33). However, exact penalty functions are nonsmooth, hence the optimization may not converge to a KKT point. Griffin and Kolda[56] described a number of exact, smooth, and "smoothed exact" penalty functions in their 2007 paper, and presented comparative results using the Asynchronous Parallel Pattern Search (APPS) algorithm (Kolda[80] and Kolda et al[81]).

Griffin and Kolda also described an algorithmic framework for implementing a sequential derivative-free algorithm; this framework is the basis for all of the sequential derivative-free optimization results in this document and is described in Section 6.2.6.2. It is based on the convergence requirement (Nocedal and Wright[99]) that a sequence of unconstrained problems Eq. (2.44) is solved with $\rho \to \infty$ so that the sequence of solutions of Eq. (2.44) converges to the minimizer of the constrained problem. The rate at which ρ is increased is not critical to the convergence proof, but does affect the rate of convergence.

2.5.3 Stochastic and Evolutionary Algorithms

The convergence analyses of the SQP and derivative-free algorithms rely on the convexity and smoothness of the objective and constraints. They are not global optimization algorithms: a local minimum produced by them may not be a global minimum and will depend on the relative locations of the initial guess and local minima. Methods that depend on a single initial guess are therefore not well suited to optimization of multimodal functions unless the initial guess can be tuned to the specific instance of the problem. Von Stryk and Bulirsch[130] reported examples, from their own work and others', of discretized optimal control problems with multiple local minima, and noted that direct methods may only find a local minimum well away from the true solution of the optimality conditions of the two point boundary value problem. Examples are given in Sections 3.4.4, 3.4.7 and 3.4.8 below that show that the objective and constraints of the inverse dynamics method may in general be nonsmooth, non-convex, and multimodal. Hence an algorithm that accepts only downhill moves from a single initial guess may not converge to the true solution. This motivates consideration of algorithms that either accept uphill moves or that use multiple initial guesses.

Simulated Annealing (SA) (Kirkpatrick et al.[78]) is a widely-used stochastic algorithm that accepts uphill moves starting with a single initial guess. At each iteration k, with current iterate $\mathbf{x}_b$, a user-specified move function M is used to generate a candidate iterate $\mathbf{x}$. The selection criterion for choosing whether to replace $\mathbf{x}_b$ with $\mathbf{x}$ requires two user-specified functions: an acceptance probability function P (distinct from the penalty function) and a "temperature" function T, where

$$P = P(J(\mathbf{x}_b), J(\mathbf{x}), T(k)), \quad P \in [0,1] \tag{2.49}$$

and T is monotonically decreasing with k, $P > 0$ when $J(\mathbf{x}_b) > J(\mathbf{x})$, and $P \to 0$ as $(T \to 0 \cap J(\mathbf{x}_b) > J(\mathbf{x}))$; this ensures that the probability of accepting an uphill move decreases as T decreases. The original P function was

$$P = \min\left(1, \exp\left(-\frac{J(\mathbf{x}) - J(\mathbf{x}_b)}{T}\right)\right) \tag{2.50}$$

and it has been shown (see Ingber[67] who also describes the origins of the algorithm and numerous improvements to it including ensembles to exploit parallel processing) that if *M* uses a Gaussian distribution then the algorithm statistically converges, but slowly, if *T* is given by

$$T = \frac{T_0}{\ln k} \tag{2.51}$$

The user must therefore provide three functions: *M*, *T* and *P*. If a random search is used for *M*, the algorithm does not exploit any structure of the objective or constraints and will therefore be slow but, for problems with discontinuities or in which function values away from the minimum convey little information about the location of the minimum, will be robust. The rate of convergence and robustness is also dependent on the choice of annealing schedule and acceptance probability functions, which must usually be determined empirically. Parameter bounds may be incorporated in *M*, and nonlinear problem constraints are typically incorporated via penalty functions.

Lu[92] used a continuous variant of SA, for which convergence to a global minimum had been proved, to solve three examples of aircraft trajectory generation problems using a six degrees of freedom aircraft model. The variant of SA randomly generated a search direction and a step length, constrained to lie in a feasible parameter space, with a modified *T* function based on a chi-squared distribution and the closeness of the iterate to the optimal. Controls were parameterized, equality constraints were included by penalty functions, and direct shooting was used to generate the trajectory. Lu found that the algorithm was not sensitive to penalty weights and that the solutions were more accurate than those obtained using a Nelder-Mead algorithm (it is inferred that the Nelder-Mead algorithm was not applied sequentially). He noted that Nelder-Mead, and two unnamed SQP algorithms, often failed to converge to any solution whereas SA converged in each case.

Differential Evolution (DE)[110] is a global optimization algorithm introduced by Price and Storn in 1995 that uses multiple pseudo-random initial guesses. It is different from genetic algorithms in that it uses floating-point encoding and arithmetic operations, rather than bit encoding and logical operations, so that it is designed to handle

continuous problems. It uses an initial population of N_p values of **x** generated by a pseudo-random number generator. Any initial distribution may be used, but the most common approach is a uniform distribution.

In the most widely-used unconstrained version, denoted by Price as "DE/Rand/1/Bin", the objectives at each **x** of a generation are evaluated once; for each **x** (the target) three other distinct points ($\mathbf{r}_0$, $\mathbf{r}_1$, and $\mathbf{r}_2$) are randomly chosen from the current population and a new vector **v** is derived by combining (mutation) $\mathbf{r}_0$, $\mathbf{r}_1$, and $\mathbf{r}_2$. A scale factor F is used as a weighting function in mutation. The new vector **v** is then combined with the target **x** by selecting elements from **x** or **v** (crossover) depending on whether a pseudo-random number (generated separately for each element) is less than a user-defined crossover probability C_r, to create a trial vector **u**. "Rand" denotes that $\mathbf{r}_0$ is chosen randomly from the population, "1" denotes that a single vector difference is used in mutation, and "Bin" denotes that crossover selects trial vector elements pseudo-randomly from each of **x** and **v.** In the unconstrained version, **x** or **u** is selected (selection) to be added to the next generation according to which has the lowest objective value. The algorithm uses random numbers to generate the initial population, to select $\mathbf{r}_0$, $\mathbf{r}_1$, and $\mathbf{r}_2$ for each **x** from the current generation, and as comparators with C_r to decide, for each element of **u**, whether the element is copied from **x** or **v** (the algorithm ensures that at least one element comes from **v**).

Lampinen[85] modified the algorithm to take account of equality and inequality nonlinear constraints in 2002, directly using the feasibility of candidate vectors rather than requiring penalty functions or associated weights. Bounds on **x** are easily applied, a simple technique recommended by Price[110] is to replace any element of **u** which exceeds a bound by a randomly-chosen value between the corresponding value of **x** and the exceeded bound.

DE has a significant drawback for use in aerospace applications: a lack of convergence proofs despite empirical success.

2.5.4 Discussion

There are very many other nonlinear programming algorithms in the literature, including variants of gradient algorithms, interior-point algorithms, derivative-free algorithms, augmented Lagrangian algorithms and genetic and evolutionary algorithms, which might also be applicable to aircraft trajectory generation. Each algorithm requires user-specified settings which have a significant effect on whether or not the algorithm converges, its rate of convergence, and the optimality of the solution: SNOPT has approximately 70 settings although typically only a few will need to be explicitly chosen by the user; the derivative-free algorithms and DE each have 10-20 settings dependent on implementation.

SNOPT is reasonably claimed by Yakimenko et al.[5, 135] to be likely to improve the computational speed of the inverse dynamics method over the Hooke-Jeeves method used in their work, but there is no specific published data to support this claim. In his 2000 paper[133] Yakimenko recommended the Nelder-Mead or Hooke-Jeeves algorithms for convergence robustness, and a two-stage approach in which a feasible solution is obtained in the first stage that is then used as an initial guess for the second stage. SNOPT, Sequential Hooke-Jeeves (SHJ), and Sequential Nelder-Mead (SNM) are therefore chosen to test these claims and recommendations and to provide additional data with which to assess the performance of the inverse dynamics method.

DE is naturally well suited to parallel implementation, handles parameter bounds and nonlinear constraints directly without penalty functions or weights, and does not rely on a single initial guess. The lack of convergence proofs is a disadvantage, but Nelder-Mead and Hooke-Jeeves were widely used without such proofs for many years, and most proofs rely on convexity and differentiability of the objective and constraints; these conditions are not, in general, true for the inverse dynamics method (Chapter 3). DE has therefore been included in the comparative numerical performance analysis of Chapter 6.

SA is also attractive due to its ability to accept uphill moves and hence find a global solution, but requires a user-specified *M* function (which may be pseudo-random) to generate a search step and direction and a sufficiently slow annealing schedule: it may

be slow to converge and less robust than DE; analysis of its performance with the inverse dynamics method is left as possible future research.

An obvious and well-known two-stage approach would be to use a global algorithm to generate initial guesses for a deterministic local algorithm, e.g. to use DE or SA to generate initial guesses for SNOPT, Hooke-Jeeves or Nelder-Mead algorithms. Parallel DE is attractive in this scenario.

The performance of SNOPT, SNM, SHJ, and DE with the inverse dynamics method is described in Chapter 6. Table 2-1 summarizes the key attributes on which these algorithms were selected.

Algorithm	Attributes
SNOPT	Quasi-Newton; single initial guess; local not global solution; utilizes gradient and the BFGS Hessian approximation for efficiency; satisfies KKT conditions for accuracy; requires good initial guess; sensitive to scaling and conditioning; explicitly handles parameter bounds and nonlinear constraints without penalty weights; computationally complex and hence less easy to implement with limited on-board hardware; can use finite differences for gradients; disadvantaged by non-availability of analytic derivatives; many user-specified settings; algorithm of choice for Gauss, Legendre, and Chebyshev pseudospectral methods.
Hooke-Jeeves (sequential)	Derivative-free; single initial guess; local not global solution; less efficient than gradient methods; does not explicitly satisfy KKT conditions; convergence proofs exist for smooth objectives; easy to apply parameter bounds; requires penalty weights to handle constraints; requires sequential implementation to handle constraints; search direction set cannot degenerate; more robust on nonsmooth problems than quasi-Newton; used by Yakimenko and others with the inverse dynamics method.
Nelder-Mead (sequential)	As for Hooke-Jeeves, except that the simplex can degenerate so Price's modification is required; more complex than Hooke-Jeeves; difficult to apply parameter bounds; number of points evaluated per iteration is less dependent than Hooke-Jeeves on problem dimension.
Differential Evolution	Global solution; multiple initial guesses; easy to implement in parallel; few, simple, user-specified settings; stochastic therefore slow; lack of convergence proofs; explicitly handles parameter bounds and nonlinear constraints without penalty weights; easy to implement with limited on-board hardware.

Table 2-1. Attributes of Selected NLP Algorithms

3 THE INVERSE DYNAMICS METHOD

3.1 Introduction

This chapter focuses on the algorithm of the inverse dynamics method. The next two sections are preparatory: Section 3.2 describes the inverse dynamics method as previously published, and Section 3.3 describes the hardware and software environment on which the experiments described in this document were carried out.

Section 3.4 presents the author's analysis of the method, including numerical results obtained using the environment of Section 3.3, and introduces modifications to overcome some of the limitations identified in the analysis. The analysis is presented in the following sequence although the order of the sub-sections is not critical: Section 3.4.1, in preparation for Chapter 4, describes, in the context of the inverse dynamics method, the well-known limitations of the Euler-angle model; Sections 3.4.2 - 3.4.6 describe problems and limitations arising from assumptions, parameterization, and discretization; Section 3.4.7 describes constraint discontinuities; Section 3.4.8 describes the multimodality of the constrained objective; Section 3.4.9 describes the need to use node and segment variables; and Section 3.4.10 describes further observations on the method.

Section 3.5 describes research into the use of local quadratic and cubic interpolation for evaluating constraints, leading to the finding that using local quadratic interpolation improves computational speed and constraint accuracy compared to the previous reliance on node values only.

The main contributions in this chapter are: the handling of the algorithmic singularities in Sections 3.4.2 - 3.4.6; the observations on constraint discontinuities and the examples of multimodality in Section 3.4.8; the combination of node and segment-based expressions in Section 3.4.9; the observations in Sections 3.4.10; and the introduction in Section 3.5 of local quadratic interpolation of constraints.

3.2 The Baseline Algorithm

Yakimenko combined the following ideas in his seminal 2000 paper[133] on the inverse dynamics method:

- Taranenko's virtual arc and speed factor thereby allowing independent parameterization of position and airspeed.
- Degree 7 power series polynomials as global parameterization functions for *x*, *y*, and *z*.
- A wind frame Euler-angle-based point-mass aircraft model that satisfies Eq. (2.27) when *x*, *y*, *z*, and *v* are parameterized, so that **u** may be evaluated analytically.

These attributes allow analytic evaluation of the controls from the state equations, and bring three potential advantages over other direct methods:

- When the airspeed is explicitly parameterized by a global polynomial that satisfies velocity and acceleration conditions at the boundary points, the NLP dimension of this method is $1+n_r+n_v$: 1 for the value of τ at the final boundary, n_r for the free spatial coefficients (for degree 7 polynomials n_r =6), and n_v for the free airspeed coefficients (e.g. for a quintic polynomial n_v =2), leading to a typical dimension of 9.
- In pseudospectral methods, the state equations are enforced at collocation nodes by explicit constraints on residuals, i.e. the search space includes trajectories that violate the state equations. The search space of the inverse dynamics method is automatically restricted to a region that satisfies the state equations. (In both methods control constraints are applied to ensure control feasibility.) Hence the ratio of feasible trajectory space to the search space should be larger for the inverse dynamics method than for a pseudospectral method.
- No numerical integration or differentiation of the state equations is required, although quadrature is required for evaluation of any integral terms in the objective function.

The following assumptions are applied

Assumption 1	$v > 0$
Assumption 2	$\hat{\mathbf{z}}_W = -\mathbf{l}_z / \|\mathbf{l}_z\|, \quad \|\mathbf{l}_z\| \neq 0$
Assumption 3	$\beta = 0$

i.e. positive airspeed, wind frame *z*-axis aligned with the negative direction of the non-zero normal load factor, and zero sideslip. Zero wind is also assumed. Assumption 1 is required physically for aerodynamic lift and mathematically for excluding singularities. Assumption 2 sets the bank angle, and in this form limits the method to positive-g trajectories; Chapter 7 introduces an extension to the method to allow negative-g trajectories.

Without loss of generality, $t_0 \coloneqq 0$ and $\tau_0 \coloneqq 0$. Each trajectory is discretized at *N* nodes $j \in \{1,...N\}$; each inter-node interval is denoted as a "segment". For uniformly-spaced nodes

$$\delta\tau = \frac{\tau_f}{N-1} \tag{3.1}$$

3.2.1 Virtual Arc

With λ defined as

$$\lambda \coloneqq \frac{d\tau}{dt} \tag{3.2}$$

then, as described by Kaminer et al.[73]

$$v = \left\| \frac{d\mathbf{r}}{dt} \right\| = \lambda \left\| \frac{d\mathbf{r}}{d\tau} \right\| \tag{3.3}$$

and since λ can be varied independently of $\mathbf{r}$, $\mathbf{r}'$ is not a function of $\dot{\mathbf{r}}$ and position and airspeed can be parameterized independently with respect to τ.

For an arbitrary variable ζ Eq. (3.2) leads directly to

$$\dot{\zeta} = \lambda \zeta' \tag{3.4}$$

$$\ddot{\zeta} = \lambda\left(\zeta''\lambda + \zeta'\lambda'\right), \tag{3.5}$$

$$\dddot{\zeta} = \lambda^3\zeta''' + 3\lambda^2\lambda'\zeta'' + (\lambda^2\lambda'' + \lambda\lambda'^2)\zeta' \tag{3.6}$$

3.2.2 Aircraft Dynamical Model

The state space model of the aircraft dynamics is chosen to balance fidelity against computational load; simple point-mass models of conventional aircraft flight dynamics have been well known since at least the 1960s (e.g. Miele[95]); Menon[94] described a point-mass dynamical model for aircraft pursuit-evasion modelling using Cartesian coordinates and Euler-angle orientation representation, and Lou and Bryson[90] investigated point mass models for precision aerobatic manoeuvres. These wind frame models are simpler than body frame models because angle of attack and sideslip do not appear explicitly, and the models satisfy Eq. (2.27): **u** can therefore be evaluated analytically.

A well-known Euler-angle system with state vector $(x, y, z, v, \gamma, \xi)^T$ and control vector $(a_x, l_z, \mu)^T$ is defined by the state equations

$$\begin{aligned} &\dot{x} = v\cos\gamma\cos\xi, && \dot{y} = v\cos\gamma\sin\xi, && \dot{z} = -v\sin\gamma \\ &\dot{v} = a_x, && \dot{\gamma} = \frac{(l_z\cos\mu - g\cos\gamma)}{v}, && \dot{\xi} = \frac{l_z\sin\mu}{v\cos\gamma} \end{aligned} \tag{3.7}$$

with controls

$$a_x = \frac{T - D}{M} - g\sin\gamma \tag{3.8}$$

$$l_z = \|\mathbf{l}_z\| = \left(\left(v\dot{\gamma} + g\cos\gamma\right)^2 + \left(v\dot{\xi}\cos\gamma\right)^2\right)^{1/2} \tag{3.9}$$

$$\mu = \arctan\left(\frac{v\dot{\xi}\cos\gamma}{v\dot{\gamma} + g\cos\gamma}\right) \tag{3.10}$$

where arctan is the 4-quadrant inverse tangent. Clearly, with outputs $(x, y, z)^T$ the above model is differentially flat, subject to assumptions 1-3, except when $\gamma = \pm\pi/2$.

3.2.3 Spatial and Airspeed Parameterization

Spatial parameterization by a global degree 7 power-series polynomial in each dimension is guaranteed to meet position, velocity and acceleration boundary conditions, and allows the third derivatives of the position vector ("jerk") at the boundary points as a vector of six free optimization variables[133].

Position in each dimension is parameterized as

$$x(\tau) = \sum_{i=0}^{7} a_i \tau^i \tag{3.11}$$

where the coefficients are defined by third-order Hermite interpolation at the boundary points

$$\begin{aligned}
&a_0 = x_0, \quad a_1 = x_0', \quad a_2 = x_0''/2, \quad a_3 = x_0'''/6, \\
&a_4 = \frac{1}{12}\left(-\frac{2x_f''' + 8x_0'''}{\tau_f} + \frac{30x_f'' - 60x_0''}{\tau_f^2} - \frac{180x_f' + 240x_0'}{\tau_f^3} + \frac{420(x_f - x_0)}{\tau_f^4}\right) \\
&a_5 = \frac{1}{20}\left(\frac{10x_f''' + 20x_0'''}{\tau_f^2} - \frac{140x_f'' - 200x_0''}{\tau_f^3} + \frac{780x_f' + 900x_0'}{\tau_f^4} - \frac{1680(x_f - x_0)}{\tau_f^5}\right) \\
&a_6 = \frac{1}{30}\left(-\frac{15x_f''' + 20x_0'''}{\tau_f^3} + \frac{195x_f'' - 225x_0''}{\tau_f^4} - \frac{1020x_f' + 1080x_0'}{\tau_f^5} - \frac{2100(x_f - x_0)}{\tau_f^6}\right) \\
&a_7 = \frac{1}{42}\left(\frac{7(x_f''' + x_0''')}{\tau_f^4} - \frac{84(x_f'' - x_0'')}{\tau_f^5} + \frac{420(x_f' + x_0')}{\tau_f^6} - \frac{840(x_f - x_0)}{\tau_f^7}\right)
\end{aligned} \tag{3.12}$$

Airspeed may be parameterized explicitly by a global polynomial of degree d_v. If airspeed and acceleration are defined and higher derivatives are free, then airspeed requires at least a cubic polynomial to satisfy the boundary conditions with zero degrees of freedom, and for $d_v > 3$ there are $d_v - 3$ additional free airspeed parameters. The NLP dimension is then $1 + 6 + d_v - 3 = d_v + 4$ and the vector of free optimization parameters is

$$\boldsymbol{\chi} = (\tau_f, \mathbf{r}_0''', \mathbf{r}_f''', \mathbf{h}) \tag{3.13}$$

where $\mathbf{r} = (x, y, z)^T$ and $\mathbf{h}$ is the vector of $d_v - 3$ airspeed free variables.

Alternatively airspeed may be parameterized indirectly by parameterizing thrust: since thrust appears linearly in the state equations ((3.7) and (3.8)), for a minimum-time problem Yakimenko[133] parameterized thrust as a bang-bang control with 2 switching points and reduced the NLP dimension further by setting

$$\mathbf{r}_f''' = (0,0,0)^T \text{ and } \mathbf{r}_0''' = (k\sin\xi_0, k\cos\xi_0, 0)^T \tag{3.14}$$

where k is a free variable. For a minimum-fuel problem he suggested using relative throttle or thrust as a free variable.

It is also possible to parameterize λ instead of v, since from Eq. (3.6)

$$v = \dot{s} = \lambda s' \tag{3.15}$$

where

$$\dot{s} := \|\dot{\mathbf{r}}\|; \quad s' = \|\mathbf{r}'\|; \quad s'' := \frac{ds'}{d\tau} \tag{3.16}$$

Since airspeed is an intuitive variable, and for many light aircraft (such as many UAVs) an on-off thrust profile is impracticable, in this work airspeed has been parameterized explicitly as a global polynomial and investigation of λ-parameterization is left as an open question (Boyarko et al.[17] described one form of λ-parameterization for spacecraft reorientation).

Horner's algorithm[108] was used to evaluate all power series polynomials and their derivatives.

3.2.4 Boundary Conditions

For spatial parameterization by degree 7 polynomials, boundary conditions on the states may be defined by conditions at $\tau \in \{0, \tau_f\}$ on any set of variables from which position,

velocity, and acceleration in each dimension may be uniquely derived. It is intuitive to choose position, airspeed, tangential acceleration, normal load factor, and orientation, i.e. $\{x, y, z, v, a_x, l_z, \xi, \gamma, \mu\}$. With the values of these variables defined at the initial and final boundary points it is straightforward to evaluate the boundary values of the first and second time derivatives $\{\dot{\mathbf{r}}, \ddot{\mathbf{r}}, \dot{v}\}$ and, and Eqs. (3.4)-(3.6) are applied to transform the boundary conditions to derivatives with respect to τ. The third derivatives $\mathbf{r}'''$ at each boundary point are defined as part of the optimization vector, and these are then substituted into Eq. (3.12) (and a corresponding equation for v) to determine the coefficients of the spatial parameterization. The coefficients of the airspeed parameterization is similarly derived from the airspeed boundary conditions.

The transformation requires λ_0 and λ_f to be defined. Yakimenko[133] suggested

$$\lambda_0 := v_0, \quad \text{and} \quad \lambda_f := v_f \tag{3.17}$$

from which

$$\lambda_0' = \frac{\dot{v}_0}{v_0}, \quad \text{and} \quad \lambda_f' = \frac{\dot{v}_f}{v_f} \tag{3.18}$$

3.2.5 Trajectory Evaluation

Using the Euler-angle model and given $\mathbf{r}$, v and their derivatives with respect to τ from the parameterization functions, the following expressions[133] are evaluated at each node $j \in [2,N]$

$$\delta s_j = \left\| \mathbf{r}_j - \mathbf{r}_{j-1} \right\| \tag{3.19}$$

$$\delta t_j = \frac{\delta s_j}{\left(v_j + v_{j-1}\right)/2} \delta\tau \tag{3.20}$$

$$\lambda_j = \frac{\delta\tau}{\delta t_j} \tag{3.21}$$

$$\xi_j = \arctan\left(\frac{y'_j}{x'_j}\right) \tag{3.22}$$

$$\gamma_j = \arcsin\left(\frac{-z'_j}{\delta s_j}\right) \tag{3.23}$$

$$a_{xj} = \lambda_j v'_j \tag{3.24}$$

$$l_{zj} = \left(\left(v_j \lambda_j \gamma'_j + g\cos\gamma_j\right)^2 + \left(v_j \lambda_j \xi'_j \cos\gamma_j\right)^2\right)^{1/2} \tag{3.25}$$

$$\mu_j = \arctan\left(\frac{v_j \lambda_j \xi'_j \cos\gamma_j}{v_j \lambda_j \gamma'_j + g\cos\gamma_j}\right) \tag{3.26}$$

3.2.6 Penalty Function and Initial Guesses

In his 2000 paper Yakimenko used a derivative-free NLP algorithm with a penalty function of the form

$$P = \left(k_0(v_N - v_f) + \sum_{i\in I} k_i \max(0, c_i)^2\right)^{1/2} \tag{3.27}$$

where

$$c_i = \max_{j=1,\ldots N} c_{ij} \tag{3.28}$$

and k_i, $i \in I$, are penalty weights that scale the penalty terms relative to each other to achieve the desired balance (Section 2.5.2).

When v is parameterized so as to satisfy the boundary conditions, the first term in Eq. (3.27) is identically zero.

Each of the local NLP algorithms requires an initial guess of the free variables $\boldsymbol{\chi}$; Yakimenko[133] suggested the following for τ_f

$$\tau_f = \left(1 + 0.3\left(\left|\xi_f - \xi_0\right|\right)\right)\left(\left\|\mathbf{r}_f - \mathbf{r}_0\right\|\right) \tag{3.29}$$

3.3 Hardware and Software Environment

The hardware used in the experiments described in this document (except for Section 4.6) was a 64-bit Intel quad core i7 975 CPU clocked at 4.2 GHz, with 6 GB of RAM, running the 64-bit Windows 7 operating system. All the code was run under the Matlab® R2009a 64-bit environment. Timings were made using a single processor. The machine was not connected to any network and no anti-virus or other monitoring software was installed.

Estimates of floating point operations (flops) were made using timing tests: addition, subtraction, multiplication, division, square, square root, tangent, sine, cosine, 4-quadrant inverse tangent, inverse sine, and inverse cosine functions were repeated at least 10^9 times in non-vectorized Matlab loops, and the elapsed times were recorded; the process was repeated 10 times. Function tests were interleaved with empty loops, the times for which were subtracted from the results. The times for each operation were divided by the times for the addition operation, to give estimates of the flop value, with one flop ≔ one addition (including an assignment to a variable). Table 3-1 shows the floating point estimates for each operation produced by this combination of hardware and software.

	-	**×**	**/**	**^2**	**√**	**sin**	**cos**	**asin**	**acos**	**atan2**
Mean	1.00	1.50	5.77	0.99	38.22	29.31	29.09	37.99	39.72	30.92
Std Dev	0.00	0.00	0.00	0.00	3.28	0.01	0.01	0.01	0.01	0.01
Max	1.00	1.50	5.77	0.99	48.06	29.32	29.10	38.02	39.73	30.94
Min	1.00	1.50	5.77	0.99	37.10	29.29	29.08	37.97	39.69	30.90

Table 3-1. Estimates of Floating Point Operations on a 64-bit Processor

3.4 Analysis

Sections 3.4.1 - 3.4.6 describe problems and limitations arising from the Euler-angle model, assumptions, discretization, and the spatial and airspeed parameterizations.

3.4.1 Singularity of the Euler-Angle Model

It is well-known that any model dependent on an Euler angle representation of orientation will exhibit singularities. For the conventional aerospace Euler angles these singularities are at $\gamma = \pm\pi/2$, hence the model cannot represent sustained vertical flight. Although in theory a trajectory which transitions through the vertical can be evaluated using an Euler-angle model if the discretization nodes do not coincide with vertical flight, in practice ξ and μ may be discontinuous over the transition, and γ may be nonsmooth, leading to step inputs to the trajectory-following flight controller: Figure 3-1 shows the discontinuous Euler-angle model controls for a vertical loop. A unit quaternion-based model that overcomes these limitations is described in Chapter 4.

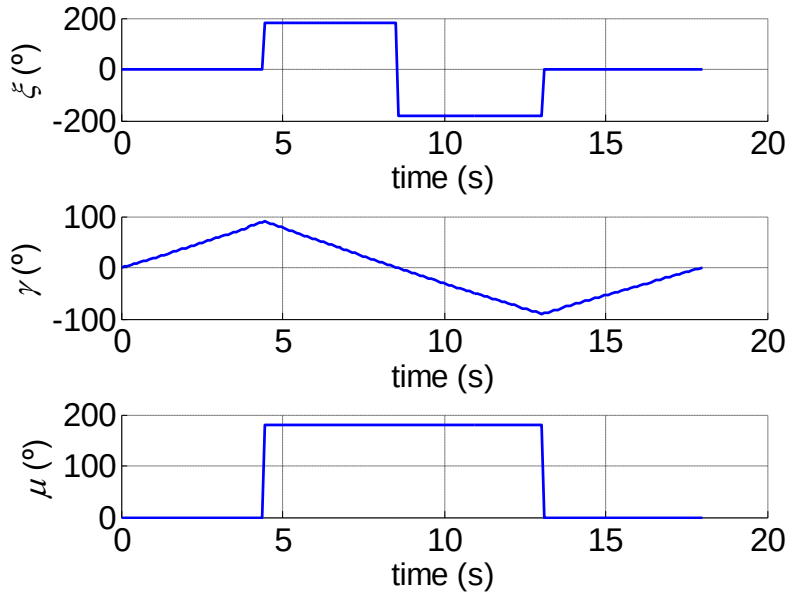

Figure 3-1. Discontinuous Euler-Angle States and Controls for a Vertical Loop

3.4.2 Assumption 2: Non-Zero Normal Load Factor

Assumption 2 relies on $l_z \neq 0$, whilst Eq. (3.25) shows that $l_z = 0$ will occur during "free fall" or steady vertical flight. Apart from the Euler-angle singularity, μ and $\hat{\mathbf{z}}_W$ become indeterminate when $l_z = 0$. Hence specifying that, if $l_{z\,j} = 0$

$$\mu_j = \mu_{j-1}, \quad \hat{\mathbf{z}}_{W\,j} = \hat{\mathbf{z}}_{W\,j-1} \tag{3.30}$$

is sufficient to handle $l_z = 0$ (it is assumed that $l_z \neq 0$ at τ_0, a condition that is easily checked). The ill-conditioning in bank angle that occurs when $\mathbf{l}_z$ reverses direction is discussed and overcome in Chapter 7.

3.4.3 Ill-Conditioning of Spatial Parameterization

Eq. (3.12) can be written in matrix form

$$\begin{pmatrix} 1 & 0 & . & . & . & . & . & 0 \\ 0 & 1 & 0 & . & . & . & . & 0 \\ 0 & 0 & 2 & 0 & . & . & . & 0 \\ 0 & 0 & 0 & 6 & . & . & . & 0 \\ 1 & \tau_f & \tau_f^2 & . & . & . & . & \tau_f^7 \\ 0 & 1 & 2\tau_f & 3\tau_f^2 & . & . & . & 7\tau_f^6 \\ 0 & 0 & 2 & 6\tau_f & . & . & . & 42\tau_f^5 \\ 0 & 0 & 0 & 6 & 24\tau_f & . & . & 210\tau_f^4 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ . \\ . \\ . \\ . \\ . \\ a_7 \end{pmatrix} = \begin{pmatrix} x_0 \\ x_0' \\ x_0'' \\ x_0''' \\ x_f \\ x_f' \\ x_f'' \\ x_f''' \end{pmatrix} \tag{3.31}$$

from which it is clear that the parameterization is ill-conditioned (the minimum condition number of the matrix is 2.9×10^4 at $\tau_f \approx 1.97$).

Eq. (3.31) could be rewritten to reduce the condition number of the matrix by defining the coefficients as coefficients of x''' instead of as coefficients of x

$$\begin{pmatrix} 1 & 0 & . & . & . & . & . & 0 \\ 0 & 1 & 0 & . & . & . & . & 0 \\ 0 & 0 & 1 & 0 & . & . & . & 0 \\ 0 & 0 & 0 & 1 & . & . & . & 0 \\ 1 & \tau_f & \frac{1}{2}\tau_f^2 & . & . & . & . & \frac{1}{210}\tau_f^7 \\ 0 & 1 & \tau_f & \frac{1}{2}\tau_f^2 & . & . & . & \frac{1}{30}\tau_f^6 \\ 0 & 0 & 1 & \tau_f & . & . & . & \frac{1}{5}\tau_f^5 \\ 0 & 0 & 0 & 1 & \tau_f & . & . & \tau_f^4 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ . \\ . \\ . \\ . \\ . \\ a_7 \end{pmatrix} = \begin{pmatrix} x_0 \\ x_0' \\ x_0'' \\ x_0''' \\ x_f \\ x_f' \\ x_f'' \\ x_f''' \end{pmatrix} \tag{3.32}$$

but this would only reduce the minimum condition number to 7.9×10^3 at $\tau_f \approx 2.77$. Similarly, defining the coefficients as the coefficients of x' or x'' does not solve the ill-conditioning problem.

Although Eq. (3.12) will give values of the coefficients even when the parameterization is ill-conditioned, small changes in the boundary conditions will cause large changes in the coefficients, and therefore in the trajectory, which will make it difficult for the NLP algorithm to find the optimal solution accurately and efficiently. A value of τ_f greater than 2, but otherwise as small as possible, therefore appears to be a good initial guess; this is investigated further in Chapter 6.

The ill-conditioning can be reduced by transforming the power series to the interval [0,1], or by using Chebyshev, Bernstein, or other basis functions; this is left as future research.

3.4.4 A Pathological Example: Course Reversal

A further effect of τ_f is shown in Figure 3-2 which plots the effect of various τ_f values on spatial parameterization for an example trajectory from the origin to a point 1000 m due north, in level flight with zero bank angle at a constant airspeed of 25 m/s with the remaining free variables set to zero. At low τ_f the parametric speed (x') starts low, increases, then decreases symmetrically about the mid-point and is always positive; the path is a straight line with varying parametric speed. At a critical value of τ_f (74 in this example), $x' = 0$ at the mid-point (and $s' = 0$ since $y' = z' = 0$ for this trajectory). For higher τ_f, $x' < 0$ for a portion of the path around the mid-point. Hence for $\tau_f > 74$, the

path is a straight line with reducing parametric speed (but constant airspeed) until the aircraft reverses direction, then at a later point it again reverses direction: there are two instantaneous 180° course reversals at constant airspeed interconnected by straight and level flight and at each reversal $s' = 0$ and s' is nonsmooth. For finite N and floating-point arithmetic each reversal will take place entirely within a segment even as $N \to \infty$.

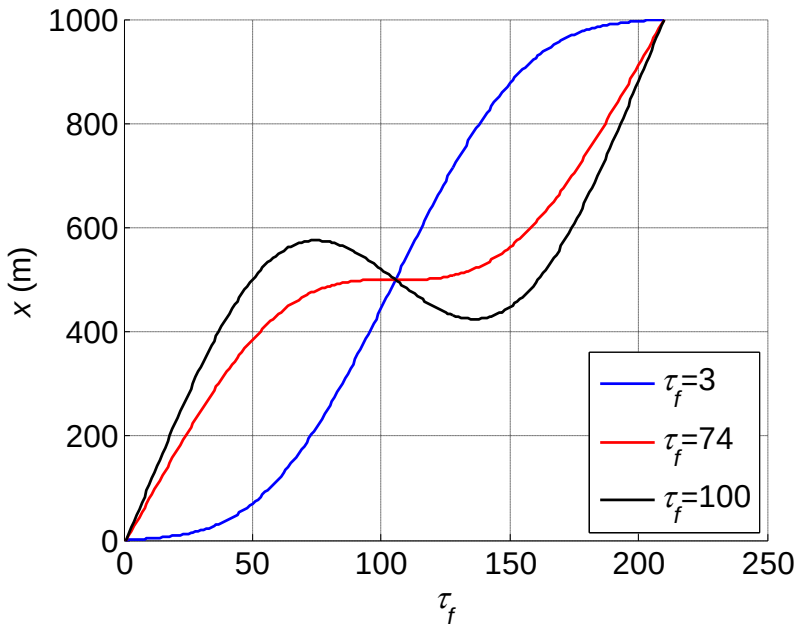

Figure 3-2. Course Reversal Caused by τ_f

This example demonstrates that τ_f has a critical effect on the path, that $s' = 0$ can occur at trajectory-specific values of τ_f, and that it can cause physically infeasible trajectories such as an instantaneous course reversal. Eqs. (3.22), (3.25), and (3.26) for ξ, l_z, and μ are indeterminate at the course reversal. Since τ_f is a free variable and therefore adjusted by the NLP algorithm, the user cannot force it to a particular value. The trajectory evaluation must therefore detect this infeasibility using appropriate constraints, which will be nonsmooth and possibly discontinuous at the critical value of τ_f. The existence and location of maxima and minima of $\mathbf{r}$ are obviously also dependent on the free variables $\mathbf{r}_0'''$ and $\mathbf{r}_f'''$ so the constraints may also be nonsmooth with respect to these variables. Hence the Jacobian matrix of the constraint vector will not, in general, be analytic over the search space.

The controls produced by the Euler-angle model using Eqs. (3.25) and (3.26) will be invalid over the course reversal because $l_z = 1g$ and $\mu = 0$ at the bracketing nodes. A segment-based expression using finite differences can be used for l_z (Eq. (3.45) below), and a finite difference expression using ξ may be used to approximate μ, but in its standard form the Euler-angle model does not generate accurate controls for the course reversal.

3.4.5 Zero Spatial Parametric Speed

A zero of s' can also arise as a consequence of other optimization variables. For example, Figure 3-3 shows how variation in x''' at the initial point can affect the spatial parameterization and cause a course reversal (assuming $y' = z' = 0$). For this example $\tau_f = 100$, $x_0 = -1000$, $x_f = 1000$, and $x_0' = v = 23$.

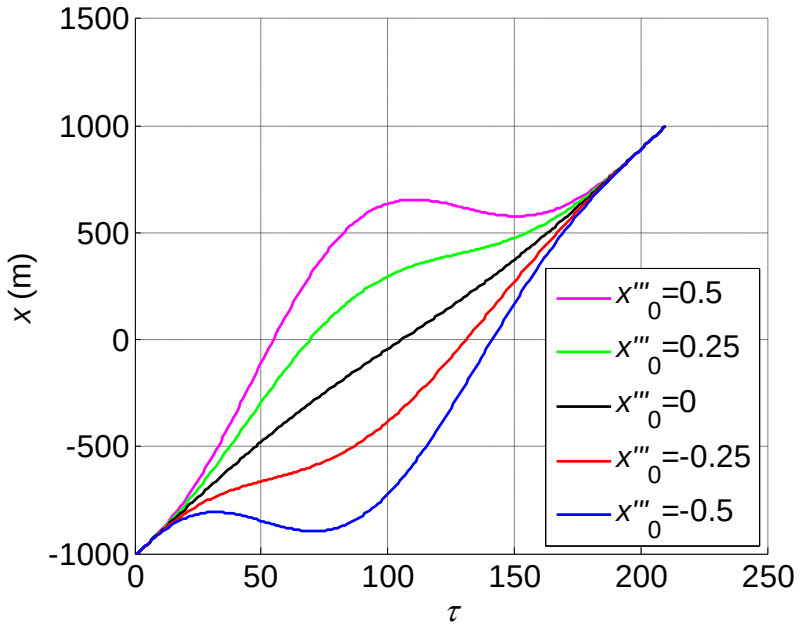

Figure 3-3. Zero Spatial Parametric Speed Caused by x_0'''

If $s' = 0$ then $x' = 0$ and $y' = 0$ and Eq. (3.22) is indeterminate. It is therefore necessary to handle $s' = 0$ algorithmically.

In general no closed form analytic solution of $s' = 0$ exists, and iterative root finding (e.g. by Brent's method[21, 108] which is used in Chapter 5) would be computationally expensive. However, if x', y', and z' are parameterized by functions with known finite maximum numbers b_x, b_y, b_z of real roots (e.g. global polynomials), then there can exist no more than $b = \min(b_x, b_y, b_z)$ zeros of s'. Three possible actions to take at any node at which $s' = 0$ are to: 1) skip the node entirely; or 2) add a small arbitrary quantity to s'; or 3) move the node by iterating $\tau_j \leftarrow \tau_j + \zeta\delta\tau$ where ζ is an arbitrary number such that $-1 \le \zeta \le 1$, until $s' > 0$. The first approach effectively doubles the segment length between the preceding and succeeding nodes which reduces the confidence level of trajectory feasibility. The second approach can only be implemented by adding an arbitrary quantity to x', y', and/or z' which introduces an unnecessary error and may make the inverse dynamics ill-conditioned (e.g. adding machine precision (eps) to x' only, compared to adding it to y' only, causes ξ to change by $\pi/2$). The third approach adds no more than b nodes per trajectory to the computational load, and does not reduce confidence level or introduce an unnecessary error (except into the quadrature of the objective which is small due to the small displacement of only one of N nodes); this approach has been adopted in this work.

If $\delta s_j = 0$ then from Eq. (3.20) $\delta t_j = 0$. However, provided that $s' \neq 0$ and $v \neq 0$ then for this case δt may instead be expressed as

$$\delta t = \frac{s'}{v}\delta\tau \tag{3.33}$$

3.4.6 Airspeed ≤ 0

Assumption 1, $v > 0$, is required for aerodynamic lift, to avoid singularities in Eq. (3.20), and to avoid errors in Eq. (3.25) and any constraints that depend on $v > 0$. As in the case of $s' = 0$, in general finding the roots of $v = 0$ would be computationally expensive. However, if $v_j \le 0$ at any node j, setting $v_j = \zeta$ where ζ is a small arbitrary value such that $0 < \zeta << v_s$ ensures that $v_j > 0$ and $v_j + v_{j-1} > 0$ and that the stall speed constraint will be violated.

If airspeed is parameterized using Bernstein basis functions then placing a lower bound of zero plus a small tolerance on each coefficient ensures that $v > 0$, which might help convergence since it excludes any parameterization with $v_j \leq 0$ from the search space.

3.4.7 Constraints

Section 3.4.4 showed that constraints may be nonsmooth; in this section it is shown that they may be discontinuous.

For a wind-frame point-mass aircraft model, a basis set of inequality path constraints is normal load factor, thrust (maximum and minimum), rate of change of bank angle (in each direction of rotation), never-exceed speed, and stall speed: $\{l_z, T_{max}, T_{min}, p_{max}, p_{min}, v_{ne}, v_s\}$. Curvature is not, in itself, a valid constraint because the load on the aircraft is a function of curvature and gravity, and the gravity component is dependent on orientation.

Below manoeuvring speed the positive-g load factor is limited by the angle of attack stall limit α_s, and at higher airspeeds by a structural limit (l_{struct}). Figure 3-4 shows an example of the positive-g part of an "n-V" diagram.

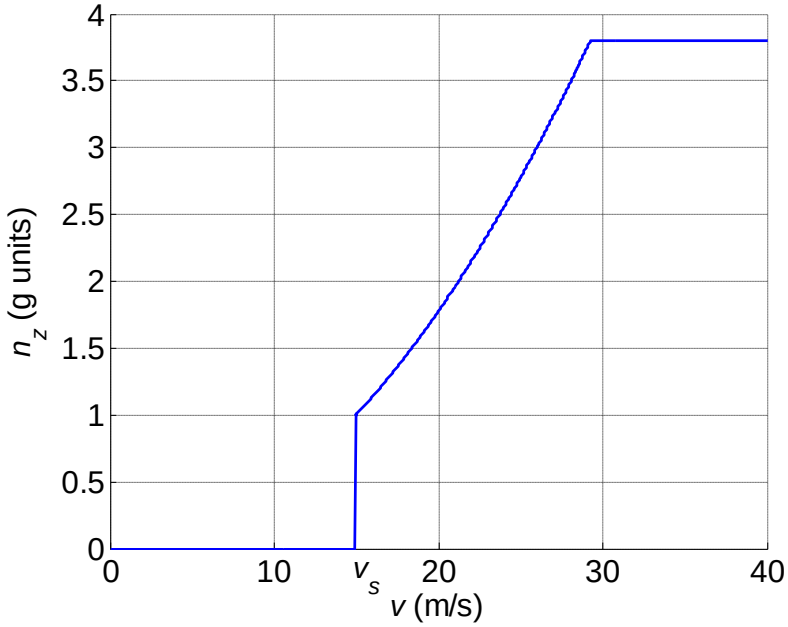

Figure 3-4. Example n-V Diagram

The discontinuity at $v = v_s$ will cause a concomitant discontinuity in the load factor constraint for some values of l_z and v.

The positive-g load factor limit may be written

$$l^+ = \begin{cases} 0, & v < v_s \\ h(v), & v_s \le v \le v_a \\ l_{struct}, & v_a < v \end{cases} \tag{3.34}$$

In this work it is assumed that α_s, mass (M), lift curve slope ($C_{L\alpha}$), and air density (ρ) are constant, and that $h(v)$ may be expressed as

$$h(v) = \frac{\rho v^2}{2M}\left(C_{L\alpha}\alpha_s + b\right) \tag{3.35}$$

(Alternative expressions for $h(v)$ can be used. In this work the constant b was derived from the requirement that $h(v)$ must pass through the three points $\{0, 0\}$, $\{v_s, g\}$, and $\{v_a, l_{struct}\}$.)

Given l_z (e.g. from Eq. (3.25)), in this work thrust and drag have been evaluated using (in conventional notation)

$$C_L = \frac{2l_z M}{\rho v^2 S} \tag{3.36}$$

$$C_D = C_{D0} + \frac{\left(C_L - C_{L\min D}\right)^2}{\pi \varepsilon AR} \tag{3.37}$$

$$D = \tfrac{1}{2}\rho v^2 S C_D \tag{3.38}$$

$$T = \left(\lambda v' + g\sin\gamma\right)M + D \tag{3.39}$$

Thrust T in Eqs. (3.8), (3.39), and (3.50) is the thrust component in the wind frame x-axis. Thrust limits are in general dependent on airspeed, air density, and a throttle response delay function, and drag is dependent on α. Explicit incorporation of α into the aircraft model would lead to a more complicated model which may not satisfy

Eq. (2.27) hence iteration would be required to invert the state equations (e.g. Kato and Sugiura[74]). Instead in this work it is assumed that T_{max}, and T_{min} take account of the effects of α, and simple scalar bounds T_{max} and T_{min} have been used. Similarly v_s, v_{ne}, and p_{max} have been specified as simple bounds. For example the effect on maximum roll rate of the torque reaction due to the rotation of the propeller for a typical single-engine, propeller-driven aircraft has been excluded, and $p_{min} := -p_{max}$. Higher fidelity expressions could be substituted for Eqs. (3.36)-(3.39), for the constraint bounds, or a higher fidelity aircraft model could be used, without invalidating the inverse dynamics method provided that Eq. (2.27) remains satisfied and additional iteration is not required.

The constraint vector may be expressed as

$$\mathbf{c} = \begin{pmatrix} \max\limits_{j\in[1,..N]} \left(l_{z\,j} - l^{+} \right) \\ \max\limits_{j\in[1,..N]} \left(T_j - T_{\max} \right) \\ \max\limits_{j\in[1,..N]} \left(T_{\min} - T_j \right) \\ \max\limits_{j\in[1,..N]} \left(p_j - p_{\max} \right) \\ \max\limits_{j\in[1,..N]} \left(-p_{\max} - p_j \right) \\ \max\limits_{j\in[1,..N]} \left(v_s - v_j \right) \\ \max\limits_{j\in[1,..N]} \left(v_j - v_{ne} \right) \end{pmatrix} \tag{3.40}$$

An alternative, more efficient, formulation is introduced in Section 3.5.

3.4.8 Convexity and Multimodality

This section provides examples of the multimodality of the method, which is a critical factor in numerical optimization (the performance of four different optimization algorithms is described in Chapter 6).

Von Stryk and Bulirsch[130] reported that they and others had found that direct methods resulted in multimodal problems. Boyd[18] noted that algorithms for solving algebraic equations arising as a result of discretization of differential equations have multiple

solutions. Sections 3.4.4 and 3.4.7 above showed that constraints will in general be nonsmooth and possibly discontinuous functions of χ; therefore discontinuities may arise in the gradients of the constraints. Figures 3-5, 3-6, and 3-7 plot final flight time, the l_z constraint violation (c_1), and the T_{max} constraint violation (c_2), respectively against the optimization parameter τ_f, for one example trajectory. They show that, even though in this case the objective is smooth and convex, the constraints are nonsmooth, non-convex and multimodal.

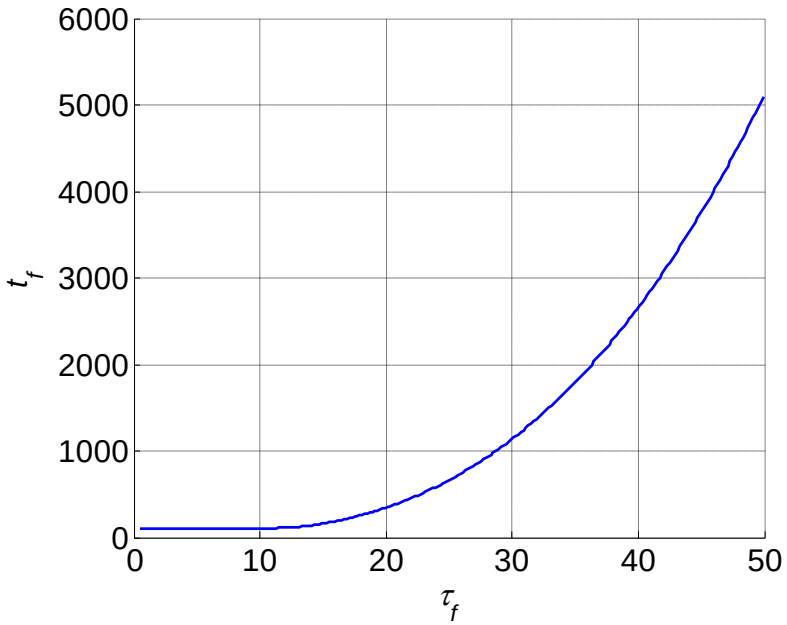

Figure 3-5. Final Flight Time as a Function of τ_f

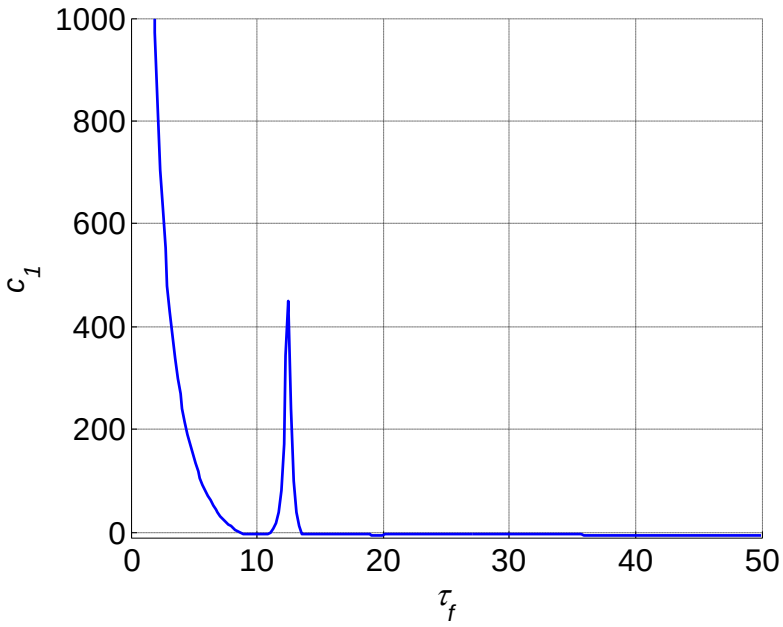

Figure 3-6. Normal Load Factor Constraint Violation as a Function of τ_f

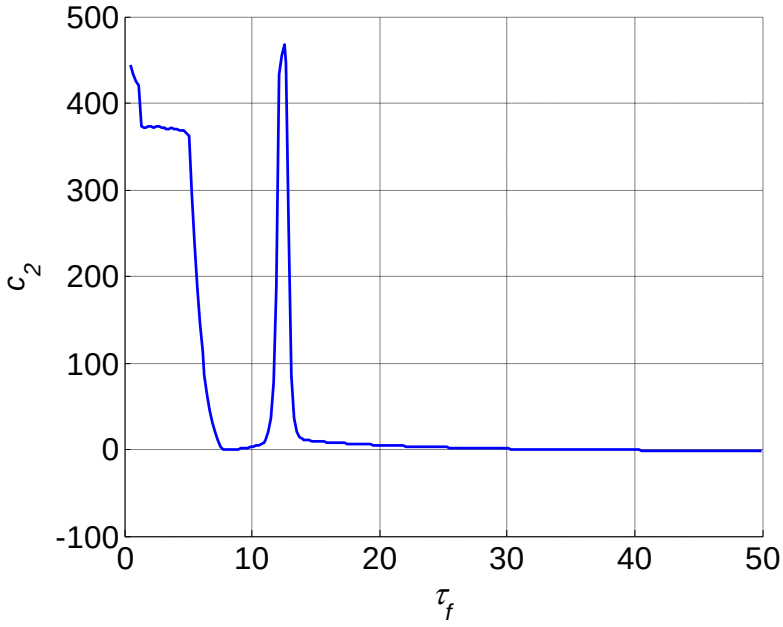

Figure 3-7. Maximum Thrust Constraint Violation as a Function of τ_f

The convergence analyses for the SNOPT, Nelder-Mead and Hooke-Jeeves algorithms rely on smooth objectives and constraints for convergence to a local minimum: in consequence there is no guarantee that these algorithms will converge, or that if they do converge they will do so to a KKT point. Further, when the constraints are multi-modal, even if the algorithm converges to a local minimum, it may not be a global minimum or even a feasible solution. For convergence to a feasible global minimum the initial guess must lie in a basin of attraction around the desired minimum. This criterion is satisfied if a convex connected region exists which includes both the initial guess and the desired minimum. If such a region exists except that it contains a finite number of stationary points other than the desired minimum, then convergence of a gradient algorithm is dependent on the algorithm implementation, and a derivative-free algorithm may step over the stationary points and converge to the desired minimum but convergence is not guaranteed.

These observations reinforce the case for using a global optimization routine such as DE or SA, at least to find a basin of attraction to a feasible global minimum, if one exists.

3.4.9 Segment and Node Variables

This section examines the combination of instantaneous analytic variables together with finite differences approximations. It is shown that both types of expression are required in the method.

3.4.9.1 Evaluation of δt and λ

Eqs. (3.20) and (3.21) define δt and λ as segment variables, i.e. they are defined as functions of values at both ends of a segment. It is also possible to define δt and λ as functions of instantaneous values at a single node

$$\lambda_j = \frac{v_j}{s'_j} \qquad \text{and} \quad \delta t_j = \frac{\delta\tau}{\lambda_j} \tag{3.41}$$

Since δt is the time interval between two consecutive nodes it is clearly a segment variable. However, λ can be considered as a node variable used to transform between

instantaneous values of derivatives with respect to t and instantaneous derivatives with respect to τ, and as a segment variable defining the relationship between δt and $\delta\tau$.

Three possible pairs of expressions for δt and λ are Eq. (3.41) above, the original expressions

$$\delta t_j = \frac{\delta s_j}{\left(v_j + v_{j-1}\right)/2}\delta\tau \quad \text{and} \quad \lambda_j = \frac{\delta\tau}{\delta t_j} \tag{3.42}$$

and, defining δt as a segment variable and λ as a node variable

$$\delta t_j = \frac{\delta s_j}{\left(v_j + v_{j-1}\right)/2}\delta\tau \quad \text{and} \quad \lambda_j = \frac{v_j}{s'_j} \tag{3.43}$$

A comparison of the effects of these variants on the accuracy of control evaluation was made for $N \in \{33, 65, 129, 257, 513, 1025, 2049\}$, using the quaternion-based model of Chapter 4 with controls derived using Eq. (4.47) for over 3000 trajectories.

The condition $\delta s_j = 0$ (Section 3.4.5) did not occur during any of these experiments and consequently did not affect the results. Eq. (3.43) achieved the lowest tracking errors for approximately 45% of tests with tracking errors (as a percentage of path length) of 0.4 % reducing to 10^{-4} % as N was increased. The original version Eq. (3.42) produced results within 3 significant digits of those of Eq. (3.43). Tracking errors produced by Eq. (3.41) were, at best, 9 times worse than the other expressions, and for high N this factor increased to over 500. Since Eq. (3.43) is no more computationally expensive than Eq. (3.42) and it is marginally more accurate, it is the preferred version.

Differentiating the expression for λ in Eq. (3.43) leads to

$$\lambda' = \frac{v' - \lambda s''}{s'} \tag{3.44}$$

Replacing λ' with a simple finite difference introduced a relative error of less than 10^{-6}, but because Eq. (3.44) is only marginally computationally slower, Eq. (3.44) was retained.

3.4.9.2 Evaluation of Path Constraints

As noted by Hull[65], path constraints can be incorporated into shooting and collocation methods but in general are only satisfied at discrete points in the trajectory and not across inter-node segments. In those methods constraint satisfaction may be improved by increasing the number of nodes, and in practice if the segment duration is sufficiently short an infeasible trajectory will violate control constraints. However, the course reversal case of Section 3.4.4 shows that this is not necessarily true for the inverse dynamics method. The load factors given by Eq. (3.25) at j - 1 and at j will be 1g, but it is obvious that an aircraft would have had to generate a higher load factor during the segment to carry out the turn: the segment load factor would have to exceed the values at the nodes. Therefore evaluating path constraints at a node using only instantaneous values at that node is insufficient to ensure feasibility.

The inverse dynamics method admits the evaluation of a normal load factor constraint across segments: linear interpolation between the nodes gives the following expression for $\mathbf{l}_z$ across each segment

$$\mathbf{l}_z^{seg}\,|_{j-1} = 0.5\left(\left(v_j + v_{j-1}\right)\omega^{seg}\hat{\boldsymbol{\varepsilon}}_n^{seg} - \left(\mathbf{g}_{z\,j} + \mathbf{g}_{z\,j-1}\right)\right) \tag{3.45}$$

where

$$\hat{\mathbf{x}}_W = \frac{\mathbf{r}'}{s'} \tag{3.46}$$

$$\omega^{seg} = \frac{\arccos\left(\hat{\mathbf{x}}_{W\,j} \cdot \hat{\mathbf{x}}_{W\,j-1}\right)}{\delta t_j} \tag{3.47}$$

$$\hat{\boldsymbol{\varepsilon}}_n^{seg} = \begin{cases} \dfrac{\hat{\mathbf{x}}_{W\,j} - \hat{\mathbf{x}}_{W\,j-1}}{\left\|\hat{\mathbf{x}}_{W\,j} - \hat{\mathbf{x}}_{W\,j-1}\right\|}, & \text{if } \omega^{seg} \neq 0 \\ \left(0,0,0,\right)^T & \text{otherwise} \end{cases} \tag{3.48}$$

$$\mathbf{g}_z = \frac{g}{\dot{s}^2}\left(-\dot{x}\dot{z}, -\dot{y}\dot{z}, \dot{x}^2 + \dot{y}^2\right) \tag{3.49}$$

Eq. (3.46) determines the wind frame *x*-axis. Eq. (3.47) determines the angular velocity across the segment, Eq. (3.48) determines the direction of the normal acceleration vector, and $\mathbf{g}_z$ is the gravity vector component parallel to the wind frame *z*-axis.

Figure 3-8 shows an example of the difference between computed values of l_z and l_z^{seg} for the level course reversal trajectory described above.

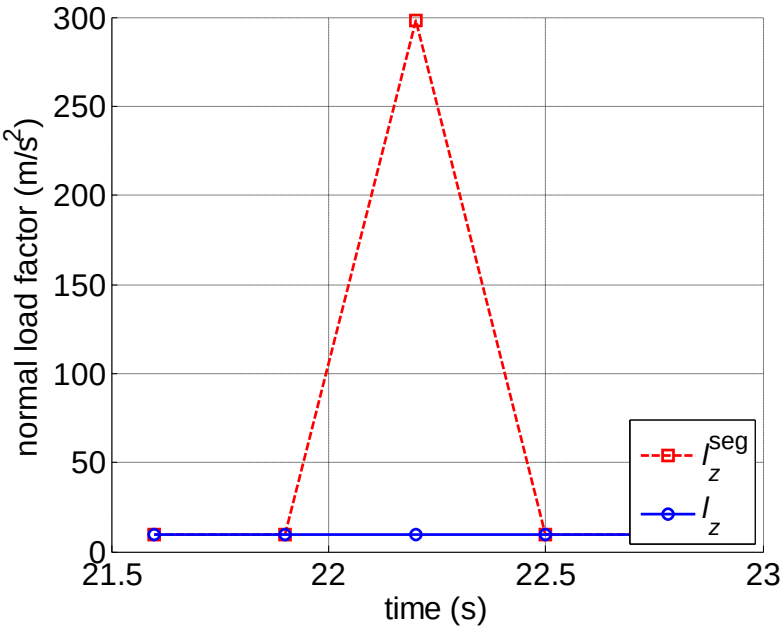

Figure 3-8. Comparison of Node-Based and Segment-Based Load Factors

The main source of inaccuracy in Eq. (3.45) is Eq. (3.48): the direction of the finite difference should be perpendicular to $\hat{\mathbf{x}}_W$, but as the angular difference increases the difference vector direction rotates until it is parallel to $\hat{\mathbf{x}}_W$ when the angle equals 180°. Empirically, it has not been found necessary to use a more accurate expression than Eq. (3.45).

For thrust, instead of using Eq. (3.39) a segment-based value of *T* may be used

$$T_{j-1} = \left(\frac{v_j - v_{j-1}}{\delta t_j} + g \sin \gamma_{j-1} \right) M + D_{j-1} \tag{3.50}$$

3.4.9.3 Evaluation of Euler-Angle Orientation

Eq. (3.23) uses the segment variable δs_j whereas all other variables used to evaluate the wind frame orientation (Eqs. (3.22) and (3.26)) are node-based. Since the method relies on aligning the aircraft wind axes with the flight path at each node, accuracy is improved if Eq. (3.23) is replaced by

$$\gamma_j = \arcsin\left(\frac{-z'_j}{s'_j}\right) \tag{3.51}$$

This change also removes the singularity at $\delta s_j = 0$; the singularity at $s' = 0$ is addressed in Section 3.4.5 above.

3.4.9.4 Evaluation of Tangential Acceleration

It is more accurate to evaluate tangential acceleration (acceleration in the direction of the wind frame *x*-axis) as a segment variable

$$a_{x\,j-1} = \frac{v_j - v_{j-1}}{\delta t_j} \tag{3.52}$$

3.4.10 Further Observations

This section describes some additional observations on:

- The connection between boundary values of the third derivative of position and the rate of change of bank angle.
- Boundary values of the speed factor λ.
- The application of numerical quadrature to evaluate the trajectory flight time.
- Initial guesses for τ_f.
- Penalty functions and weights.
- Angle of attack evaluation.
- The interpolation of generated controls.

3.4.10.1 Bank Rate and the Third Derivative

A consequence of using $\mathbf{r}'''$ as part of the optimization vector is that p at the boundary points is not under user control (since p depends on $\mathbf{r}'''$), i.e. the bank rate at the boundaries is varied by the NLP algorithm. If it is necessary to satisfy specific boundary conditions on p then either the degree of the spatial polynomials must be increased or a reduced NLP dimension must be accepted (which may increase the rate of convergence but reduce optimality). For example, Yakimenko's choice of Eq. (3.14) reduces the NLP dimension by three and ensures that $p_f = 0$.

3.4.10.2 Speed Factors λ_0 and λ_f

For the minimum-time problem, the objective function is

$$t_f = \int_0^{\tau_f} \frac{1}{\lambda} d\tau \tag{3.53}$$

Hence smaller values of λ lead to smaller values of τ_f for a given final time, which has been shown in Sections 3.4.3 and 3.4.4 to be desirable, and rather than using Eq. (3.17) the following has been used to set λ at the boundary points

$$\begin{aligned} \lambda_0 &:= 1, \quad \text{and} \quad \lambda_f := 1 \\ \lambda_0' &:= 0, \quad \text{and} \quad \lambda_f' := 0 \end{aligned} \tag{3.54}$$

Tests using SNOPT, DE and Hooke-Jeeves on a number of pseudo-random trajectories (see Section 6.2.3 for the test trajectory definitions) confirmed that Eq. (3.54) required fewer trajectory evaluations and produced lower values of t_f than Eq. (3.17); Eq. (3.54) has therefore been used in this work.

3.4.10.3 Objective Quadrature

The objective function in this work was to minimize the trajectory flight time

$$t_f = \int_0^{t_f} dt = \int_0^{\tau_f} \frac{s'}{v} d\tau \approx a(\tau_f) \sum_{j=1}^{N} w_j \frac{s_j'}{v_j} \tag{3.55}$$

Computationally the quadrature requires only element-wise vector division, matrix-vector multiplication and summation once per trajectory. The factor *a* transforms the sum to the interval $[0, \tau_f]$. The quadrature weights **w** and factor *a* depend only on *N* and the node distribution and can therefore be evaluated outside the NLP algorithm. For most experiments Chebyshev-Gauss-Lobatto nodes were used with Clenshaw-Curtis quadrature[129]; for uniform node spacing the extended trapezoidal or Simpson's rules were used[108].

It is possible to use t_f^2 as the objective to try to avoid the NLP algorithm converging towards $t_f = -\infty$. However, if this approach is used with the virtual arc then unless additional conditions are applied the method may generate a solution with $\tau_f < 0$, which is physically meaningless.

3.4.10.4 Initial Guess

Eq. (3.29) leads to large values of τ_f, for example for a 7 km Euclidean distance from $\mathbf{r}_0$ to $\mathbf{r}_f$ it gives $\tau_f = 7\times10^3$ yet $t_f = 280$ s for a straight path at 25 m/s. Chapter 6 Figure 6-1 shows values of τ_f for 1000 optimized trajectories based on pseudo-random boundary conditions: from this data an initial guess of $\tau_f = 10$ can be seen to be closer to the optimal value than that generated by Eq. (3.29). In Section 2.5.4 the use of a global optimization algorithm to produce an initial guess for a quasi-Newton algorithm was suggested; this approach is discussed in Section 6.4.5.

3.4.10.5 Penalty Function and Penalty Weights

For experiments using the Nelder-Mead and Hooke-Jeeves NLP algorithms the constraint vector must be transformed to a penalty function and added to the objective. The squared two-norm l_2^2 was used as a penalty function for most experiments because of its smoothness. A number of ad-hoc trials of the exact infinity-norm penalty function were carried out, with no consistent pattern distinguishing its performance from that of the squared two-norm. However, the data sample was very small. For the l_2^2 penalty function

$$J = t_f + \rho P\left(\mathbf{c}^+\right) \tag{3.56}$$

$$c_i^+ = \max(0, c_i), \quad i = \{1,...7\} \tag{3.57}$$

where P is given by Eq. (2.47) and c_i by Eq.(3.40).

Unity penalty weights have been used in this work, because the individual constraints were of similar magnitude ($T_{max} \approx 30$, $v_{ne} \approx 40$, $l_{struct} \approx 40$). Although $p_{max} \approx 7$, this constraint was found to be rarely exceeded in isolation.

3.4.10.6 Evaluation of Angle of Attack

Angle of attack α may be evaluated from the segment load factor using a variety of expressions according to the desired fidelity. In this work the following expression was used

$$\alpha = \frac{1}{C_{L\alpha}}\left(\frac{2l_z^{seg}}{M\rho v^2 S} - C_{L0}\right) \tag{3.58}$$

3.4.10.7 Interpolation of Controls for Flight Controllers

Clearly the output values of the controls and states at each node may be interpolated to provide inputs to a flight control system. However, λ varies with time so the time-based node distribution is not the same as the τ-based node distribution, and high-degree global interpolation of **u** (or **r** or **x**) to create input functions for a flight controller may introduce the Runge phenomenon. This is less likely, but cannot be guaranteed not to occur, if τ is based on a Chebyshev-Gauss or Legendre-Gauss distribution (with or without end points).

3.5 Constraint Accuracy

This section describes the use of local low-degree interpolation to improve the accuracy with which constraints are evaluated. It is shown that using local quadratic interpolation improves the trade-off between accuracy and computational speed.

The optimality and feasibility of generated trajectories depend on the accuracy with which the extrema of the constraint violations are evaluated. In the literature each constraint is evaluated at each discretization node and the violation for that constraint over the trajectory is defined as the maximum positive value at any node (Eq. (3.40)).

This approach does not allow for a limiting value occurring between nodes: interpolation is needed to capture this possibility. It is equivalent to using piecewise linear interpolation and it is well-known that quadratic or cubic interpolation reduces interpolation error compared to piecewise linear interpolation. Local quadratic or cubic interpolation requires only the roots of linear or quadratic equations, at small computational cost, and these low degree interpolants are not dependent on any particular node distribution. Hence the accuracy of constraint evaluation can be improved by using local quadratic or local cubic interpolation of constraints instead of relying on the worst-case node value.

Figure 3-9 shows an example of four node values of an arbitrary variable, with quadratic interpolation over the last three of those values and cubic interpolation over all four values: the maxima can be seen to be different in each case. There is no information obtainable from that graph to determine the most accurate maximum. This section quantifies the effect of these three approaches to determine which of them has the shortest computation time for a given accuracy.

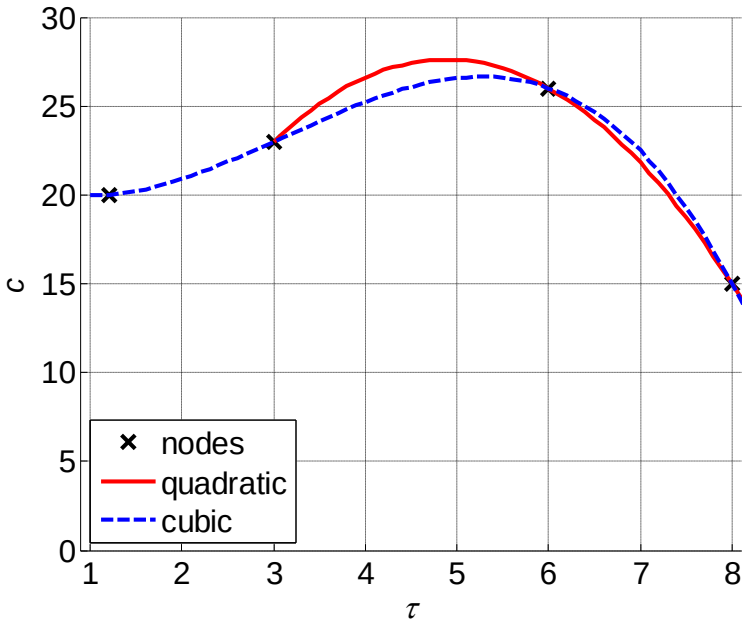

Figure 3-9. Example of Local Interpolations

A reference is required to quantify the accuracy of each approach. This reference was obtained by interpolating by an degree N-1 Chebyshev interpolant $f(\tau)$, and evaluating the global maximum of that function over $[0, \tau_f]$. It is not guaranteed *a priori* that $f(\tau)$ has a stationary point in $[0, \tau_f]$, so it was necessary to find the real roots of $f'(\tau)$ on $[0, \tau_f]$ and evaluate $f(\tau)$ at those roots and at the boundary points to establish the maximum. Boyd[20] described the stable and robust Chebyshev-Frobenius matrix method for rootfinding and gave assessments of the computational load of that method compared to other methods[19].

3.5.1 Method

The aircraft data and test database of feasible spatial paths described in 5.3.1 were used to provide test trajectories, and the maximum and minimum feasible airspeeds (v_{max} and v_{min}) were evaluated as described in Section 5.2.

Using l_z as the constrained variable, the method used to evaluate constraint accuracy as N varied was, for each trajectory in the database and each N:

1. Parameterize airspeed by a degree 10 Bernstein polynomial such that airspeed and acceleration boundary conditions were met; set the other seven airspeed coefficients to alternate between $0.85v_{min}$ and $1.15v_{max}$ through the trajectory to ensure significant variation of l_z.
2. Discretize the trajectory in N Chebyshev-Gauss-Lobatto nodes.
3. Construct the degree N - 1 Chebyshev interpolant $f(\tau)$ to l_z using the chebfun operators developed by Battles and Trefethen[6] to calculate the coefficients of $f(\tau)$ and the algorithm in Boyd[20] to calculate the coefficients of $f'(\tau)$.
4. Apply the Chebyshev-Frobenius matrix method to find the roots of $f'(\tau)$ and hence the maximum of $f(\tau)$.
5. Evaluate l_z at each node and record the maximum node value.
6. Evaluate, using the local quadratic and local cubic interpolations described in Sections 3.5.1.1 and 3.5.1.2 below, the corresponding maxima.

The upper bound on N was set to 1025 by doubling N until the difference between the maxima of $f(\tau)$ over all trajectories changed by less than 0.1 m/s^2 for successive values of N. For each trajectory the maximum of $f(\tau)$ for N = 1025 was used as the reference. For each N over all trajectories the errors between the reference and each of the maximum values obtained in steps 4, 5, and 6 above were recorded.

3.5.1.1 Local Quadratic Interpolation

Local quadratic interpolation was applied at $j = \{3,\ldots N\}$ over the interval $[\tau_{j-2}, \tau_j]$ (i.e. nodes $\{j - 2, j - 1, j\}$) if and only if

$$c_{j-1} > c_{j-2} \cap c_{j-1} > c_j \tag{3.59}$$

where c is the variable of which the maximum is required and the constraint index i is omitted for clarity.

The purpose of condition (3.59) is to accelerate the trajectory evaluation while ensuring that every inter-node segment that may contain the maximum is interpolated. Eq. (3.59) is a sufficient, but not a necessary, condition for a maximum to exist in the interval $[\tau_{j-2}, \tau_j]$; if it is true for one interval it cannot be true for the adjacent intervals. Hence it

imposes an upper bound of (N - 2)/3 on the number of quadratic interpolations that will be performed. It is possible that a maximum exists on an interval for which c_{j-1} exceeds one end node value but not the other. Consider $c_{j-1} > c_{j-2}$ but $c_{j-1} < c_j$ and assume that a maximum exists in $[\tau_{j-1}, \tau_j]$. Eq. (3.59) evaluates to false over $[\tau_{j-2}, \tau_j]$ but if the maximum in $[\tau_{j-1}, \tau_j]$ is a maximum of the whole function then $c_{j+1} < c_j$ and Eq. (3.59) will be true for $[\tau_{j-1}, \tau_{j+1}]$, causing interpolation of that interval.

Only 3 divisions, 16 multiplications, 18 additions/subtractions with 2 conditions and 1 *max* function over 2 variables were required to evaluate the 3 quadratic coefficients and find the maximum over the interval.

3.5.1.2 Local Cubic Interpolation

Local cubic interpolation was applied at $j = \{4,\ldots N\}$ over $\{j - 3, j - 2, j - 1, j\}$. Newton interpolation was used, evaluating the coefficients of the quadratic derivative from the finite divided differences, finding the root(s) of the quadratic and evaluating the cubic at the in-range root(s). This required 1 square root, 8 divisions, 25 multiplications and 48 additions/subtractions, with 5 conditions and 1 *max* function over 6 variables. Unfortunately, no simple condition corresponding to Eq. (3.59) was found for the cubic case.

3.5.2 Results and Analysis

Figures 3-10 and 3-11 show the maximum interpolation errors for node values and for high-degree interpolation over the test database using a constraint tolerance of 0.5 m⁄s^2 (since load factor limits are typically expressed with a tolerance of ±0.05g, equivalent to ~1.3% tolerance on a typical limit of 3.8g). Chebyshev-Gauss-Lobatto node distributions were used. Figure 3-11 shows that reliance on node values would require N > ~390 while degree N - 1 Chebyshev interpolation would require only N > ~133, an improvement factor of ~3. It can be seen that this factor is reasonably insensitive to the chosen tolerance: the range of the factor is ~[2,4.5] over the domain [0.1,1].

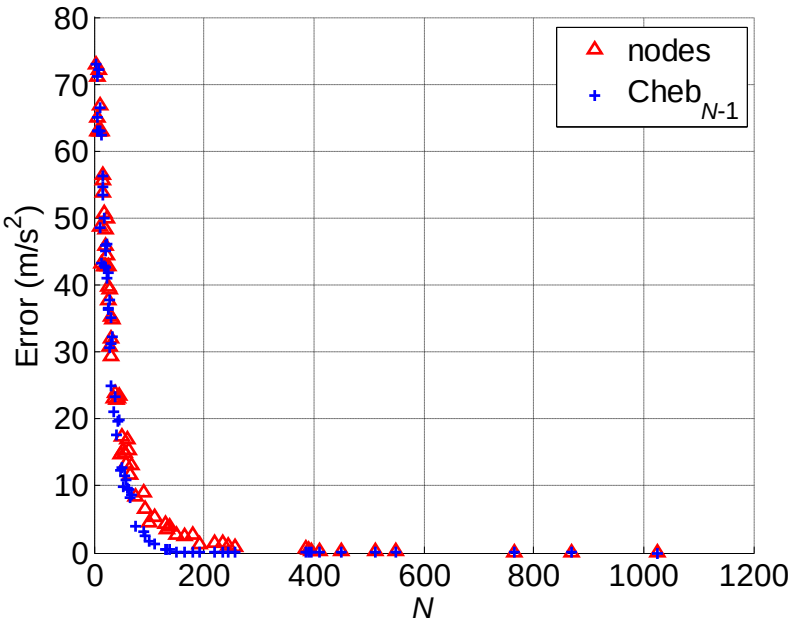

Figure 3-10. Error in Maxima: Node Values and Chebyshev Interpolation

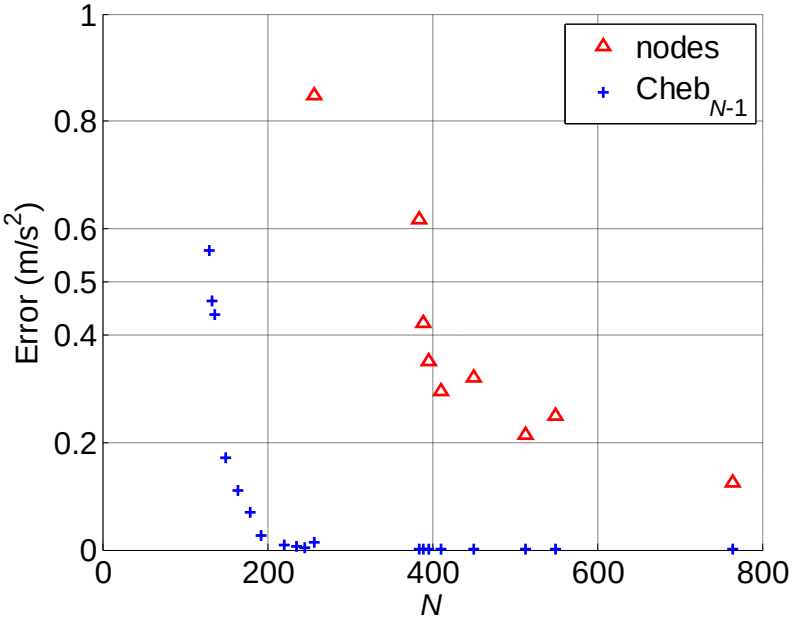

Figure 3-11. Expansion of Figure 3-10

The computational cost of the rootfinding for the Chebyshev interpolation is too high for practical use in the inverse dynamics method: between $10n^3$ and $12n^3$ (n = degree = N - 1) floating-point operations (flops)[19], which for N = 133 results in at least 2.3×10^7 flops. Although the cost of a 13-N/tredecic rootfinder[19] is only $22000n+42n^2$ this would still require 3.6×10^6 flops. Evaluation of a single node, without the degree N - 1 Chebyshev interpolation and rootfinding, was found to require approximately 600-800 flops, hence incorporating the Chebyshev interpolation and rootfinding, even for a single constraint, would incur significantly larger computational cost than increasing N to obtain equivalent accuracy and is not viable for real-time application.

Figures 3-12 and 3-13 show the errors using local quadratic and cubic interpolation compared to node values and degree N - 1 Chebyshev interpolation, plotted using expanded scales. The minimum N values required to achieve a maximum error less than 0.5 m⁄s^2 were found to be 210 for quadratic and 165 for cubic interpolation. Quadratic or cubic interpolation would be required for each constraint or constrained variable. For 8 constrained variables (the 7 variables of Eq. (3.40) plus a negative load factor limit for the negative-g trajectories of Chapter 7), and d_v = 8, the quadratic interpolation would add approximately 20% (flops) to each node evaluation, and the cubic would add approximately 150%. Hence, with the Chebyshev-Gauss-Lobatto node distribution, the net effect of quadratic interpolation would be to increase computational speed by 35% (1-(1.2×210⁄390)) compared to reliance on node values.

For comparison, the tests were repeated with uniformly-spaced nodes. The minimum N values required were found to be 345, 175, and 145 for node values, quadratic and cubic interpolation respectively, indicating that quadratic interpolation can increase computational speed by 40% (1-(1.2×175⁄345)) with uniform node spacing.

Hence using local quadratic interpolation for evaluating constraints can provide a 35%-40% increase in computational speed by allowing the use of fewer discretization nodes while maintaining the accuracy of evaluated constraints, compared to the previously-used node values.

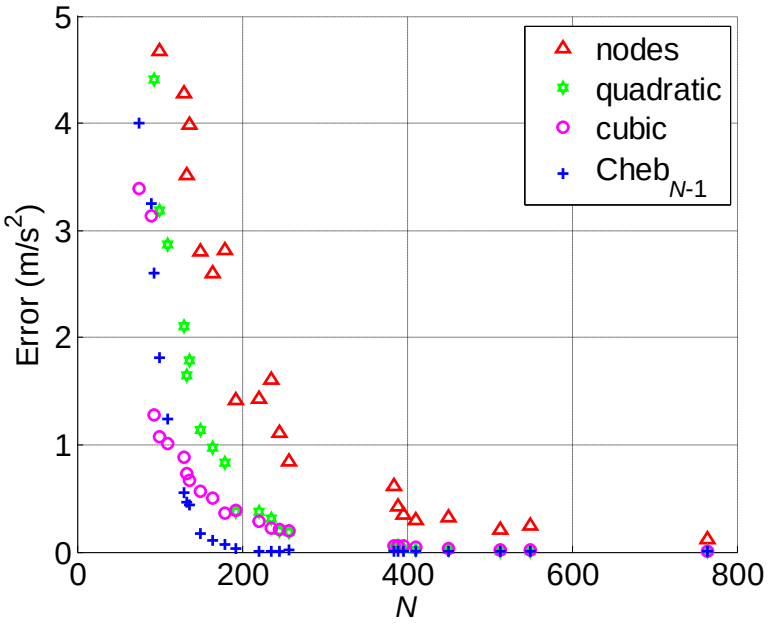

Figure 3-12. Error in Maxima: Quadratic and Cubic Interpolation

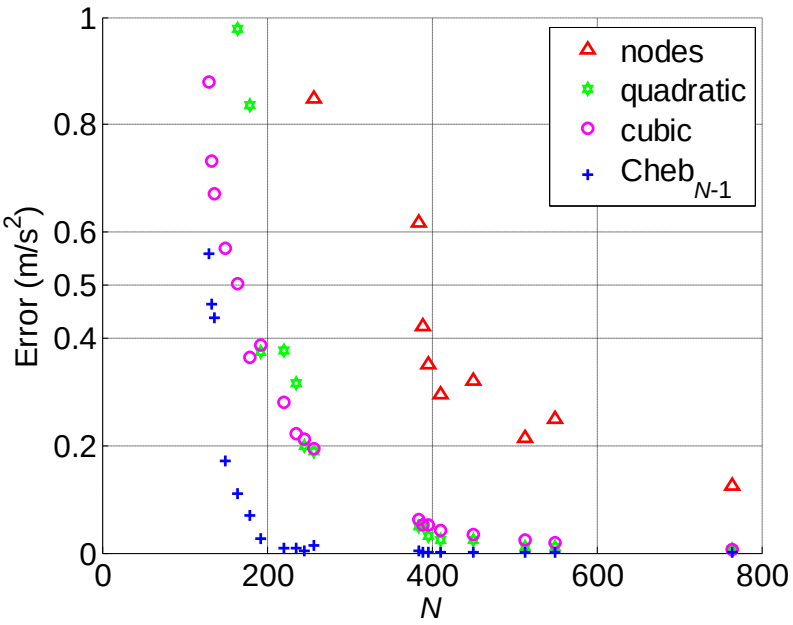

Figure 3-13. Expansion of Figure 3-12

The minimum values of N are only valid for this particular aircraft, tolerance, and for trajectories of durations similar to those in the test database (which ranged from 15 s to 1153 s, with 85% of flight times < 200 s. However, it is reasonable to conjecture, since Figure 6-1 indicates that τ_f is not sensitive to t_f and the interpolation is with respect to τ, that these values of N may apply to a wider range of trajectories.

4 QUATERNION-BASED POINT-MASS AIRCRAFT MODEL

4.1 Introduction

This chapter describes a unit-quaternion-based inverse dynamics model that was conceived by the author to overcome the singularities of the Euler-angle model, to improve the accuracy of the controls, to further the investigation of aerobatic and negative-g trajectories, and to introduce the possibility of parameterizing orientation. The model was stimulated by the discussions on quaternions in the books by Stengel[122] and Stevens and Lewis[123].

Unit quaternions can be used to represent all orientations without singularities, and can be smoothly parameterized and interpolated. The quaternion-based inverse dynamics model also has the advantage over an Euler-angle model of producing angular velocity and/or quaternion attitude control vectors. Kaminer et al.[72, 73] described path following and coordinated control of multiple unmanned aircraft by trajectory-following flight controllers using pitch and yaw rates as the control vector. Knoebel et al.[79] described a flight controller for an unmanned aircraft using quaternion feedback. The quaternion model is also well suited to the imposition of path constraints on wind-axis angular velocity. It is not restricted only to the inverse dynamics method, it could be applied in pseudospectral or other methods.

Sections 4.2, 4.3, 4.4, and 4.5 describe the original derivation of the model, and corresponding test results are given in Section 4.6 which show that the model is computationally efficient. Section 4.7 shows the controls generated for two trajectory examples. Further research into the accuracy of the controls was carried out after the original model derivation: Section 4.8 describes this further research and the use of quaternion calculus to generate more accurate controls. Section 4.9 describes a minor change to the algorithm that improves computational efficiency. Section 4.10 notes that the quaternion model state equations are linear in the controls. Section 4.11 summarizes, for convenience, the revised algorithm and model, bringing together the improvements of Chapters 3 and 4.

4.2 Unit Quaternion-Based Point-Mass Model State Equations

A unit quaternion may be written as

$$\mathbf{e} = \left(e_0, e_1, e_2, e_3\right)^T \tag{4.1}$$

where e_0 is the scalar component and

$$\left\|\mathbf{e}\right\| = \left(e_0^2 + e_1^2 + e_2^2 + e_3^2\right)^{1/2} = 1 \tag{4.2}$$

The orthogonal transformation matrix between flat-Earth axes and wind axes is well known and can be expressed in both Euler-angle and quaternion forms[103, 122, 123]. Defining a state vector $(x, y, z, v, \mathbf{e}^T)^T$ and control vector $(a_x, p, q, r)^T$, equating corresponding elements of the two forms of the transformation matrix, and incorporating the quaternion kinematic equation $\dot{\mathbf{e}}=\boldsymbol{\Omega}\mathbf{e}$ gives the following system of state equations

$$\dot{x} = v\left(e_0^2 + e_1^2 - e_2^2 - e_3^2\right) \tag{4.3}$$

$$\dot{y} = 2v\left(e_1 e_2 + e_0 e_3\right) \tag{4.4}$$

$$\dot{z} = 2v\left(e_1 e_3 - e_0 e_2\right) \tag{4.5}$$

$$\dot{v} = a_x \tag{4.6}$$

$$\dot{\mathbf{e}} = \boldsymbol{\Omega}\mathbf{e} \tag{4.7}$$

where

$$\boldsymbol{\Omega} = \frac{1}{2}\begin{pmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{pmatrix} \tag{4.8}$$

4.3 Model Inverse Dynamics

Let *x*, *y*, and *z* be parameterized by functions that are at least C^3 continuous, then from Eqs. (3.16)

$$\dot{s} = \left(\dot{x}^2 + \dot{y}^2 + \dot{z}^2 \right)^{\frac{1}{2}} \tag{4.9}$$

$$\ddot{s} = \frac{1}{\dot{s}} \left(\dot{x}\ddot{x} + \dot{y}\ddot{y} + \dot{z}\ddot{z} \right) \tag{4.10}$$

Tangential acceleration a_x is given by

$$a_x = \frac{T - D}{M} + \frac{g\dot{z}}{\dot{s}} \tag{4.11}$$

Using Frenet's formulae[4] leads to

$$\hat{\boldsymbol{\varepsilon}}_t = \frac{\dot{\mathbf{r}}}{\dot{s}}; \quad \hat{\boldsymbol{\varepsilon}}_n := \rho \frac{d\hat{\boldsymbol{\varepsilon}}_t}{ds} \tag{4.12}$$

and

$$\kappa = \frac{1}{\rho} = \frac{\left(s'^2 \left(\ddot{\mathbf{r}} \cdot \ddot{\mathbf{r}} \right) - \left(\dot{\mathbf{r}} \cdot \ddot{\mathbf{r}} \right)^2 \right)^{\frac{1}{2}}}{\dot{s}^3} \tag{4.13}$$

Differentiating Eq. (4.12) and cancelling $\hat{\boldsymbol{\varepsilon}}_t$

$$\ddot{\mathbf{r}} = \ddot{\mathbf{s}}\hat{\boldsymbol{\varepsilon}}_t + \frac{\dot{s}^2}{\rho}\hat{\boldsymbol{\varepsilon}}_n \tag{4.14}$$

$$\hat{\boldsymbol{\varepsilon}}_n = \frac{\rho}{\dot{s}^2} \left(\dot{s}\ddot{\mathbf{r}} - \ddot{s}\dot{\mathbf{r}} \right) \tag{4.15}$$

$$\mathbf{a}_n = \frac{\dot{s}^2}{\rho}\hat{\boldsymbol{\varepsilon}}_n = \ddot{\mathbf{r}} - \ddot{s}\frac{\dot{\mathbf{r}}}{\dot{s}} \tag{4.16}$$

The velocity frame *V* is the flat-Earth frame rotated through ξ and γ only, hence

$$\hat{\mathbf{x}}_V = \frac{\dot{\mathbf{r}}}{\dot{s}} \tag{4.17}$$

$$\hat{\mathbf{y}}_V = \left(\frac{-\dot{y}}{\sqrt{\left(\dot{x}^2 + \dot{y}^2\right)}}, \frac{\dot{x}}{\sqrt{\left(\dot{x}^2 + \dot{y}^2\right)}}, 0 \right)^T \tag{4.18}$$

$$\begin{aligned} \hat{\mathbf{z}}_V &= \hat{\mathbf{x}}_V \times \hat{\mathbf{y}}_V \\ &= \left(\frac{-\dot{x}\dot{z}}{\dot{s}\sqrt{\left(\dot{x}^2 + \dot{y}^2\right)}}, \frac{-\dot{y}\dot{z}}{\dot{s}\sqrt{\left(\dot{x}^2 + \dot{y}^2\right)}}, \frac{\left(\dot{x}^2 + \dot{y}^2\right)}{\dot{s}\sqrt{\left(\dot{x}^2 + \dot{y}^2\right)}} \right)^T \end{aligned} \tag{4.19}$$

Clearly $\hat{\mathbf{x}}_W = \hat{\mathbf{x}}_V$. From Eq. (4.16) and $\hat{\mathbf{x}}_W = \hat{\boldsymbol{\varepsilon}}_t$ and $\hat{\mathbf{z}}_V \perp \hat{\mathbf{x}}_V$ and $\hat{\boldsymbol{\varepsilon}}_n \perp \hat{\boldsymbol{\varepsilon}}_t$ then

$$\hat{\mathbf{z}}_V \times \hat{\boldsymbol{\varepsilon}}_n = k\hat{\mathbf{x}}_W, \quad \text{where } k \in \mathbb{R}, k \neq 0 \tag{4.20}$$

Thus the plane containing $\hat{\mathbf{z}}_V$ and $\mathbf{a}_n$ is perpendicular to $\hat{\mathbf{x}}_W$ and the gravity component in that plane is given by $\left(\mathbf{g} \cdot \hat{\mathbf{z}}_V\right)\hat{\mathbf{z}}_V$. From Eq. (4.19)

$$\left(\mathbf{g} \cdot \hat{\mathbf{z}}_V\right)\hat{\mathbf{z}}_v = \frac{g}{\dot{s}^2}\left(-\dot{x}\dot{z}, -\dot{y}\dot{z}, \dot{x}^2 + \dot{y}^2\right) \tag{4.21}$$

therefore

$$\begin{aligned} \mathbf{l}_z &= \mathbf{a}_n - \left(\mathbf{g} \cdot \hat{\mathbf{z}}_V\right)\hat{\mathbf{z}}_v \\ &= \ddot{\mathbf{r}} - \frac{\ddot{s}\dot{\mathbf{r}}}{\dot{s}} - \frac{g}{\dot{s}^2}\left(-\dot{x}\dot{z}, -\dot{y}\dot{z}, \dot{x}^2 + \dot{y}^2\right) \end{aligned} \tag{4.22}$$

and applying Assumption 2

$$\hat{\mathbf{z}}_W = -\frac{\mathbf{l}_z}{\|\mathbf{l}_z\|} \tag{4.23}$$

$$\hat{\mathbf{y}}_W = \hat{\mathbf{z}}_W \times \hat{\mathbf{x}}_W \tag{4.24}$$

The relationship between the angular velocity of a frame and the transformation matrix is given by the strapdown (Poisson kinematic) equation

$$\tilde{\boldsymbol{\omega}} = \mathbf{H}_E^W \left(\dot{\mathbf{H}}_E^W \right)^T \tag{4.25}$$

which can be derived from the transport theorem[4, 123], and where the angular velocity cross-product equivalent matrix $\tilde{\boldsymbol{\omega}}$ is

$$\tilde{\boldsymbol{\omega}} = \begin{pmatrix} 0 & -r & q \\ r & 0 & p \\ -q & p & 0 \end{pmatrix} \tag{4.26}$$

and

$$\mathbf{H}_E^W = \begin{pmatrix} \hat{\mathbf{x}}_W^T \\ \hat{\mathbf{y}}_W^T \\ \hat{\mathbf{z}}_W^T \end{pmatrix} \tag{4.27}$$

Equating elements on each side of Eq. (4.25) gives a set of control expressions

$$\begin{aligned} p &= -\hat{\mathbf{y}}_W \cdot \dot{\hat{\mathbf{z}}}_W, \quad \text{or} \quad & p &= \hat{\mathbf{z}}_W \cdot \dot{\hat{\mathbf{y}}}_W \\ q &= -\hat{\mathbf{z}}_W \cdot \dot{\hat{\mathbf{x}}}_W, \quad \text{or} \quad & q &= \hat{\mathbf{x}}_W \cdot \dot{\hat{\mathbf{z}}}_W \\ r &= \hat{\mathbf{y}}_W \cdot \dot{\hat{\mathbf{x}}}_W, \quad \text{or} \quad & r &= -\hat{\mathbf{x}}_W \cdot \dot{\hat{\mathbf{y}}}_W \end{aligned} \tag{4.28}$$

Eq. (4.28) uses only node-based values to evaluate the controls and p depends on $\mathbf{r}'''$ whilst a_x, q, r, and the Euler-angle model controls only require the first and second derivatives of $\mathbf{r}$. In Section 4.8 segment-based expressions are derived which use interpolated angular velocity instead of $\mathbf{r}'''$ for improved accuracy.

4.4 Differential Flatness

To confirm that the unit quaternion system of Eqs. (4.3)-(4.7) is differentially flat, $\mathbf{r}$ is defined as the flat output. Eq. (4.28) shows that p, q, and r are real analytic functions of $\hat{\mathbf{x}}_W$, $\hat{\mathbf{y}}_W$, and $\hat{\mathbf{z}}_W$; Eqs. (4.17), (4.22), (4.23), and (4.24) express $\hat{\mathbf{x}}_W$, $\hat{\mathbf{y}}_W$, and $\hat{\mathbf{z}}_W$ as real analytic functions of $\mathbf{r}$ and its derivatives; a_x is obviously a function of $\mathbf{r}$ and its derivatives, hence the controls are real analytic functions of a flat output $\mathbf{r}$.

To show that $\mathbf{e}$ can be expressed as a real analytic function of $\mathbf{r}$ and its derivatives, let

$$\mathbf{H}_E^W = \left[h_{ij} \right], \quad i = \{1,2,3\},\ j = \{1,2,3\} \tag{4.29}$$

and re-arrange to give the standard result[123]

$$\begin{aligned}
e_0^2 &= \left(1 + h_{11} + h_{22} + h_{33}\right)/4 & e_1^2 &= \left(1 + h_{11} - h_{22} - h_{33}\right)/4 \\
e_2^2 &= \left(1 - h_{11} + h_{22} - h_{33}\right)/4 & e_3^2 &= \left(1 - h_{11} - h_{22} + h_{33}\right)/4 \\
e_0 e_1 &= \left(h_{23} - h_{32}\right)/4 & e_1 e_2 &= \left(h_{12} + h_{21}\right)/4 \\
e_0 e_2 &= \left(h_{31} - h_{13}\right)/4 & e_2 e_3 &= \left(h_{23} + h_{32}\right)/4 \\
e_0 e_3 &= \left(h_{12} - h_{21}\right)/4 & e_1 e_3 &= \left(h_{13} + h_{31}\right)/4
\end{aligned} \tag{4.30}$$

Therefore **e** is a real analytic function of the elements of $\mathbf{H}_E^W$, and of $\hat{\mathbf{x}}_W$, $\hat{\mathbf{y}}_W$, and $\hat{\mathbf{z}}_W$ and, subject to Assumptions 2 and 3, the model is differentially flat.

4.5 Computation of Time Derivative of Wind Frame *z*-Axis

This section describes a computational algorithm for evaluating the time derivative of the wind frame *z*-axis, which is required to evaluate the controls using Eq. (4.28). This computation is superseded by the control expressions derived in Section 4.8 but is included here for completeness.

The time derivatives $\dot{\hat{\mathbf{x}}}_W$, $\dot{\hat{\mathbf{y}}}_W$, and $\dot{\hat{\mathbf{z}}}_W$ required by Eq. (4.28) can be obtained by analytically differentiating Eqs. (4.17), (4.24), and (4.23) but for $\dot{\hat{\mathbf{y}}}_W$, and $\dot{\hat{\mathbf{z}}}_W$ the equations become computationally expensive to implement; $\dot{\hat{\mathbf{y}}}_W$ can be avoided by using the first expressions in Eq. (4.28) and $\dot{\hat{\mathbf{z}}}_W$ may be evaluated as follows. Define **l** as a vector and **z** as the unit vector parallel to **l**, $d\mathbf{l}$ to be the total differential of **l**, $d\mathbf{l}_\mathrm{t}$ to be the component of $d\mathbf{l}$ parallel to **l**, $d\mathbf{z}$ to be the total differential of **z**, $d\mathbf{z}_\mathrm{t}$ to be the component of $d\mathbf{z}$ parallel to **z**, and $d\mathbf{z}_\mathrm{n}$ to be the component of $d\mathbf{z}$ perpendicular to **z**. Then

$$
\begin{aligned}
d\mathbf{z} &= \frac{d\mathbf{l}}{\|\mathbf{l}\|}, \quad d\mathbf{l}_t = \frac{d\mathbf{l}\cdot\mathbf{l}}{\|\mathbf{l}\|}\mathbf{z} = \left(d\mathbf{l}\cdot\mathbf{z}\right)\mathbf{z} \\
d\mathbf{z}_t &= \frac{\|d\mathbf{l}_t\|}{\|\mathbf{l}\|} = \frac{\left(d\mathbf{l}\cdot\mathbf{z}\right)\mathbf{z}}{\|\mathbf{l}\|} \\
d\mathbf{z}_n &= d\mathbf{z} - d\mathbf{z}_t = \frac{1}{\|\mathbf{l}\|}\left(d\mathbf{l} - \left(d\mathbf{l}\cdot\mathbf{z}\right)\mathbf{z}\right)
\end{aligned}
\tag{4.31}
$$

Therefore, substituting $\mathbf{l}_z$ for $\mathbf{l}$ and $\hat{\mathbf{z}}_W = -\mathbf{z}$

$$
\dot{\hat{\mathbf{z}}}_W = -\frac{1}{\|\mathbf{l}_z\|}\left(\dot{\mathbf{l}}_z - \left(\dot{\mathbf{l}}_z\cdot\hat{\mathbf{z}}_W\right)\hat{\mathbf{z}}_W\right) \tag{4.32}
$$

where $\mathbf{l}_z$ is obtained by differentiating Eq. (4.22). (As noted above, evaluating the controls using the expressions in Section 4.8 obviates the need for $\dot{\hat{\mathbf{x}}}_W$, $\dot{\hat{\mathbf{y}}}_W$, and $\dot{\hat{\mathbf{z}}}_W$ but Eq. (4.32) is included here for completeness.)

4.6 Computational Load

Computational speed is a key attribute of the inverse dynamics method. Hence the speed with which the quaternion model can be evaluated, compared to that of the Euler-angle model, is of interest. In this section numerical comparisons of the computational speeds of the Euler-angle and quaternion models are presented.

To assess computational performance, each model was applied to 19 different trajectories for which the parameterization function types were as follows: three linear, four vertical helices, two vertical loops, three horizontal helices, one quadratic polynomial, five quintic polynomials and one degree 7 polynomial. Each trajectory was run in Matlab on a 32-bit PC using the quaternion model then using the Euler-angle model, for $N = 2^n + 1$; $\forall n \in \{17,16,...7\}$, and the ratio of CPU time consumed by the quaternion model to CPU time consumed by the Euler-angle model was recorded. The measurements were then repeated but running the Euler-angle model first. For $N \le 2^{13}+1$ each trajectory was repeated $2^{14}/(N\text{-}1)$ times to reduce any effects from the Matlab CPU timer quantization. The times include the processing necessary to extract $\mathbf{r}$ and its derivatives from the parameterization functions at each node, but exclude

trajectory parameterization setup, constraint evaluations, and transforming controls to physical variables. Controls were evaluated using Eq. (4.28), i.e. excluding evaluation of Eqs. (4.62)-(4.65) below but including Eq. (4.32).

As tested the Euler-angle inverse dynamics routine performed, at each node: 1 square root, 3 inverse trigonometric functions, 1 cosine, 22 multiplications, 3 divisions and 15 additions/subtractions (no functions were evaluated more than once) compared to: 1 square root, 52 multiplications, 20 divisions and 44 additions/subtractions for the quaternion inverse dynamics routine; in both cases assignments and simple conditions were also performed. CPU times depend not only on the hardware and the model but on the efficiency with which the compiler and run-time environment implement the square root and trigonometric functions and on memory management; for the Matlab environment Table 1 shows the mean, standard deviation and range of the CPU times from which it can be seen that for this version of the quaternion model mean CPU time was comparable with that of the Euler-angle model.

N	**Mean**	**Std Dev**	**Max**	**Min**
131073	0.42	0.01	0.45	0.38
65537	0.85	0.04	0.90	0.8
32769	0.95	0.02	1.00	0.91
16385	1.01	0.05	1.10	0.91
8193	1.01	0.05	1.15	0.93
4097	1.02	0.04	1.13	0.94
2049	1.03	0.06	1.14	0.94
1025	1.03	0.04	1.13	0.93
513	1.06	0.06	1.17	0.96
257	1.03	0.05	1.13	0.96
129	1.03	0.05	1.12	0.93
65	1.04	0.05	1.17	0.94
Overall	0.96	0.18	1.17	0.38

Table 4-1 Ratio of Quaternion CPU Time to Euler-Angle CPU Time (using 32-bit hardware and software)

4.7 Example Results

This section describes two trajectory examples, one of which includes vertical flight, to demonstrate the quaternion model. For these examples aircraft data representative of a small unmanned aircraft were used: mass 11 kg, maximum thrust 20 N, $15 \le v \le 60$ m/s and $l_z \le 3g$.

Figures 4-1 and 4-2 show a vertical loop trajectory and the control vector produced using Eq. (4.28). The trajectory was defined as a loop through 360° with constant normal acceleration and constant airspeed, starting and ending in level flight heading 45°, and discretized into 129 uniformly-spaced nodes. The control values are clearly feasible. Figure 4-2 shows the error deviation of the computed values of the control vector to be comparable to machine precision (a_x, p, and r should be zero and q should be constant). Repeating the calculation with varying N from $2^{17}+1$ to 2^4+1 (equivalent to δt = 1.1 s) confirmed that the error variation was comparable to machine precision over this range. For comparison, controls were also evaluated using finite differences (Eq. (4.36) below) which produced, because the curvature was constant, a constant q error which reduced as δt was reduced; for N = 129 (δt = 0.15 s) the q error was ~10^{-4} rad/sec and the maximum position error was 0.2 m.

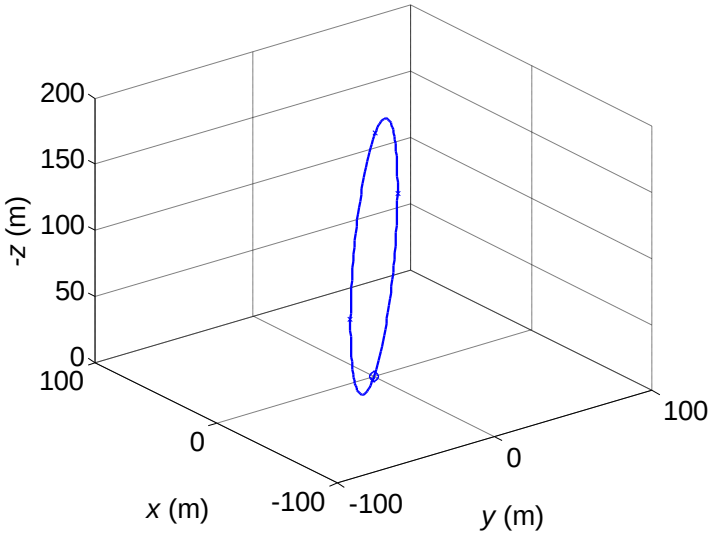

Figure 4-1. Vertical Loop Trajectory

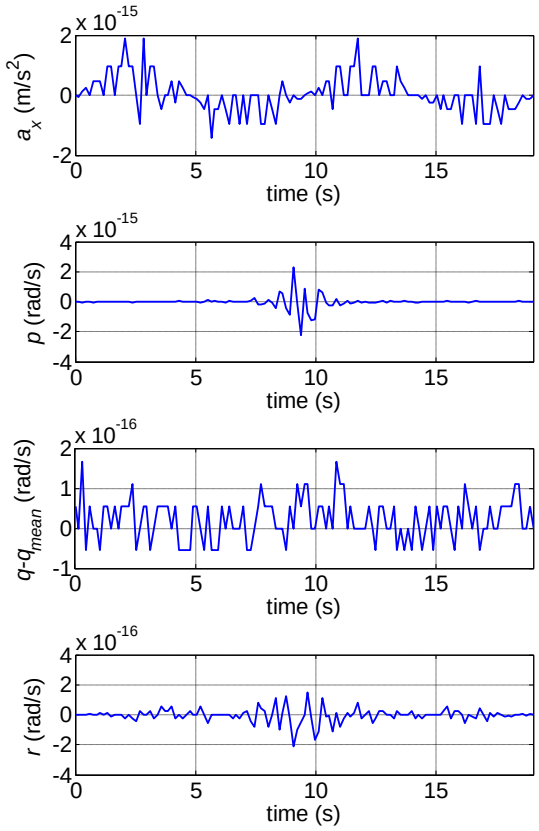

Figure 4-2. Control Vector for Vertical Loop Trajectory

Figure 4-3 shows a constant airspeed vertical helix trajectory In this case the angular velocities should be constant: Figure 4-4 shows that the errors in the node-based controls are of the order of machine precision.

These examples demonstrate that, for these trajectories, Eq. (4.28) produces accurate and feasible control vectors.

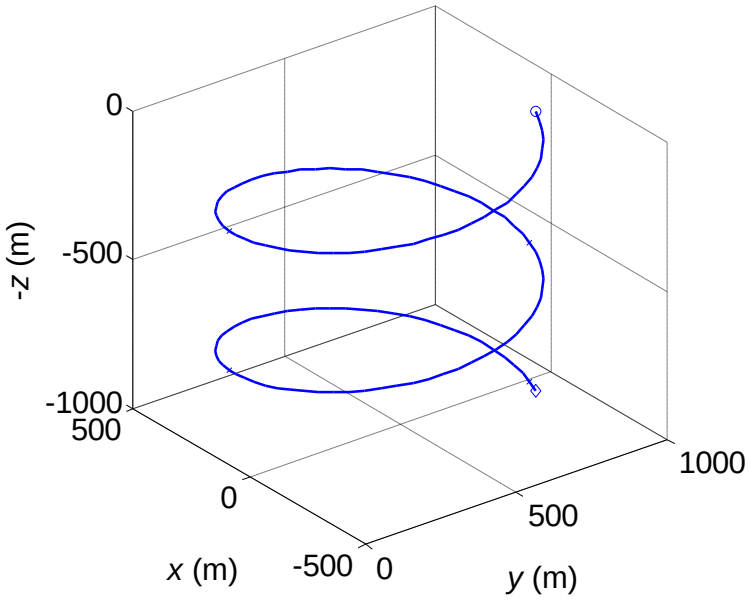

Figure 4-3. Vertical Helix Trajectory

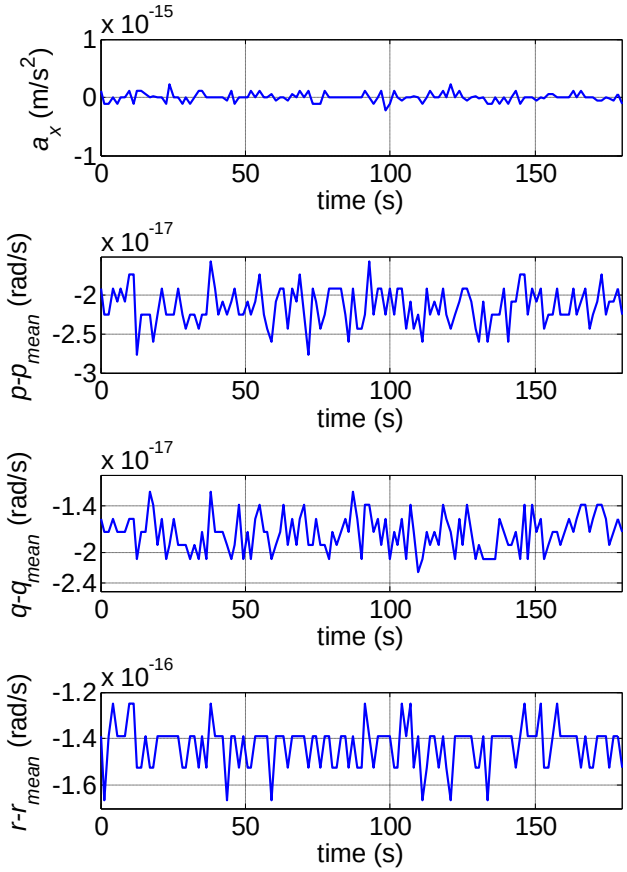

Figure 4-4. Control Vector for Vertical Helix Trajectory

4.8 Control Expressions

In this section more accurate control expressions are derived to replace those derived in Section 4.3. The revised controls use quaternion calculus and therefore require accurate expressions for orientation and the first derivative of orientation. This section is divided into sub-sections covering: the derivation of alternative angular velocity expressions (Section 4.8.1); the evaluation of orientation, including a comparison of two common algorithms, (Section 4.8.2); the evaluation of the first derivative of orientation (Section 4.8.3); a numerical comparison of the candidate control expressions (Section 4.8.4); and a brief conclusion (Section 4.8.5). It is shown that the most accurate of the alternative expressions produces accurate controls even for trajectories such as the course reversal case of Section 3.4.4.

4.8.1 Derivation of Alternative Angular Velocity Expressions

Using

$$\boldsymbol{\omega} := (p, q, r)^T := (\omega_1, \omega_2, \omega_3)^T \tag{4.33}$$

the first set of expressions of Eq. (4.28) becomes

$$\boldsymbol{\omega} = \begin{pmatrix} -\hat{\mathbf{y}}_W \cdot \dot{\hat{\mathbf{z}}}_W \\ -\hat{\mathbf{z}}_W \cdot \dot{\hat{\mathbf{x}}}_W \\ \hat{\mathbf{y}}_W \cdot \dot{\hat{\mathbf{x}}}_W \end{pmatrix} \tag{4.34}$$

Eq. (4.34) uses only instantaneous analytic derivatives at nodes, thereby avoiding the inherent ill-conditioning of numerical differentiation. However, error arises in controls derived from analytic derivatives because in general for a nonlinear function $f(t)$

$$f(t_b) - f(t_a) \neq f'(t_a)(t_b - t_a), \quad t \in [t_a, t_b] \tag{4.35}$$

and for a nonsmooth function the error does not necessarily reduce to zero as $t_b - t_a \to 0$.

An alternative is two-point finite differences

$$\boldsymbol{\omega}_{j-1} = \begin{pmatrix} -\hat{\mathbf{y}}_W \cdot \left(\hat{\mathbf{z}}_{W\,j} - \hat{\mathbf{z}}_{W\,j-1}\right) / \delta t_j \\ -\hat{\mathbf{z}}_W \cdot \left(\hat{\mathbf{x}}_{W\,j} - \hat{\mathbf{x}}_{W\,j-1}\right) / \delta t_j \\ \hat{\mathbf{y}}_W \cdot \left(\hat{\mathbf{x}}_{W\,j} - \hat{\mathbf{x}}_{W\,j-1}\right) / \delta t_j \end{pmatrix} \tag{4.36}$$

If controls are calculated using Eq. (4.36) instead of (4.34), reductions of 27 multiplications, 6 divisions, and 15 additions/subtractions are possible. However, for large rotations significant error arises in the direction of the finite difference approximations (the derivative of a unit vector is orthogonal to the unit vector, but the direction of the finite difference is not: e.g. it is parallel to the unit vector for a 180° change).

Control accuracy can be improved by rewriting Eq. (4.7) as

$$\dot{\mathbf{e}} = \mathbf{E}\boldsymbol{\omega} \tag{4.37}$$

where

$$\mathbf{E} = \frac{1}{2}\begin{pmatrix} -e_1 & -e_2 & -e_3 \\ e_0 & -e_3 & e_2 \\ e_3 & e_0 & -e_1 \\ -e_2 & e_1 & e_0 \end{pmatrix} \tag{4.38}$$

The structure of **E** enables the exact solution of the over-determined system Eq. (4.37) to be derived by the least squares method, without recourse to the cost of the Moore-Penrose pseudoinverse. Define the residual

$$R := \left\| \dot{\mathbf{e}} - \mathbf{E}\boldsymbol{\omega} \right\|^2 \tag{4.39}$$

then

$$\frac{\partial R}{\partial \omega_n} = \sum_{i=1}^{4} 2\left(\mathbf{E}_{i1}\omega_1 + \mathbf{E}_{i2}\omega_2 + \mathbf{E}_{i3}\omega_3 - \dot{e}_i\right)\mathbf{E}_{in}, \quad n = 1,2,3 \tag{4.40}$$

and the least squares solution is given by

$$\frac{\partial R}{\partial \omega_n} = 0, \quad n = 1,2,3 \tag{4.41}$$

hence

$$\boldsymbol{\omega} = 2\begin{pmatrix} -\dot{e}_0 e_1 + \dot{e}_1 e_0 + \dot{e}_2 e_3 - \dot{e}_3 e_2 \\ -\dot{e}_0 e_2 - \dot{e}_1 e_3 + \dot{e}_2 e_0 + \dot{e}_3 e_1 \\ -\dot{e}_0 e_3 + \dot{e}_1 e_2 - \dot{e}_2 e_1 + \dot{e}_3 e_0 \end{pmatrix} \tag{4.42}$$

is the exact solution of Eq. (4.37). Alternative derivations may be found in Shuster[121].

4.8.2 Evaluation of Orientation

Evaluation of Eq. (4.42) requires orientation **e** which may be derived from the rotation matrix of Eq. (4.27). A number of algorithms have been proposed for this purpose e.g. Grubin[57, 58], Reynolds[111], Shoemake[120], and Shepperd[119]. Grubin's algorithm has a singularity; Reynolds' algorithm is elegant and efficient, but uses vector dot and cross products which can become zero or indeterminate and the conditions which must be evaluated to handle these cases are computationally expensive. The two most widely used algorithms are those of Shoemake and Shepperd.

Shoemake's algorithm is efficient but it relies on a tolerance which determines how the matrix elements are combined. Shepperd's algorithm does not use a tolerance: it automatically uses the combination that minimizes round-off error. Figure 4-5 shows the peak error (error was defined as 1 - $|\mathbf{e}_a \cdot \mathbf{e}_b|$) arising from the Shoemake algorithm as a function of the tolerance. These results were obtained by implementing $\mathbf{e}_b = \mathbf{f}_2\left(\mathbf{f}_1\left(\mathbf{e}_a\right)\right)$, using a standard algorithm[120, 123] for $\mathbf{f}_1 : \mathbf{e}_a \mapsto \mathbf{H}_E^W$, and the Shoemake algorithm for $\mathbf{f}_2 : \mathbf{H}_E^W \mapsto \mathbf{e}_b$ where the domain of $\mathbf{e}_a$ covered the set of 3D orientations at intervals ≤1° in each dimension.

The minimum peak error of the Shoemake algorithm was found to be 6×10^{-6} at a tolerance of 1×10^{-11}. The error was high at very small tolerances due to round-off error, and increased again at high tolerances because this forced at least one element of the quaternion to be zero. The peak error of the Shepperd algorithm, over the same domain,

was found to be 9×10^{-15}. Although the Shepperd algorithm was found to be 50-80% slower, it has been used to evaluate **e** because of its accuracy.

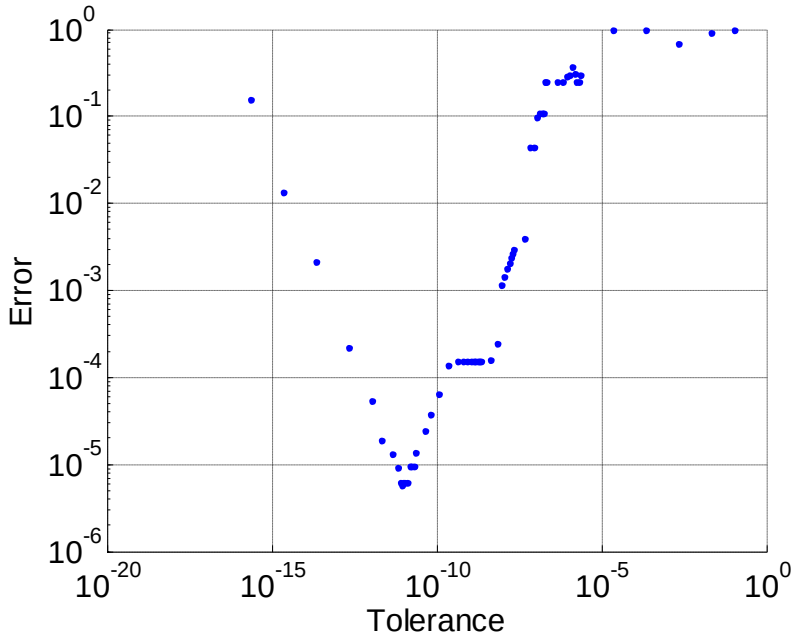

Figure 4-5. Peak Error as a Function of Tolerance for the Shoemake Algorithm

4.8.3 Evaluation of the First Derivative of Orientation

It remains to choose expressions for $\dot{\mathbf{e}}$ for substitution into Eq. (4.42). A compact analytic expression for the first derivative of a unit quaternion was described by Kim et al.[76], but this approach would suffer from discretization error in the same way as Eq. (4.34). Writing $\dot{\mathbf{e}}$ as a finite difference gives

$$\dot{\mathbf{e}}_{j-1} = \frac{\mathbf{e}_j - \mathbf{e}_{j-1}}{\delta t_j} \tag{4.43}$$

However, $\dot{\mathbf{e}}$ is the velocity of the unit quaternion on the surface of the 3-sphere S^3, whereas $\mathbf{e}_j - \mathbf{e}_{j-1}$ is the straight line between $\mathbf{e}_{j-1}$ and $\mathbf{e}_j$ and is therefore inside S^3 except at the ends. Therefore although Equation (4.43) generates smaller errors than Eq. (4.36), (because the denominator has a single source of direction error whereas Eq. (4.36)

introduces three direction errors separately), errors may be significant for large rotations.

This can be readily overcome by interpolating between $\mathbf{e}_{j-1}$ and $\mathbf{e}_j$ such that the interpolant lies on a great circle on the surface of S^3. Spherical linear interpolation (slerp) satisfies this requirement and may be defined[120] as

$$\text{slerp}(t;\mathbf{e}_a,\mathbf{e}_b) := \frac{\mathbf{e}_a \sin\left((1-t)\theta\right) + \mathbf{e}_b \sin(t\theta)}{\sin\theta}, \quad t \in [0,1] \tag{4.44}$$

where

$$\theta = \arccos(\mathbf{e}_a \cdot \mathbf{e}_b) \tag{4.45}$$

The singularities in Eq. (4.44) when $\sin\theta = 0$ may be handled by selecting the sign of $\mathbf{e}_j$ such that $\mathbf{e}_j \cdot \mathbf{e}_{j-1} \geq 0$ and setting $\mathbf{e}_\Delta = \mathbf{e}_j$ if $\theta = 0$.

Slerp can be applied to the evaluation of $\dot{\mathbf{e}}$ by defining

$$\mathbf{e}_\Delta := \text{slerp}(\Delta;\mathbf{e}_{j-1},\mathbf{e}_j) \tag{4.46}$$

It is only necessary to evaluate Eq. (4.46) at a single Δ and to express $\dot{\mathbf{e}}_{j-1}$ as

$$\dot{\mathbf{e}}_{j-1} = \frac{\mathbf{e}_\Delta - \mathbf{e}_{j-1}}{\Delta \cdot \delta t_j} \tag{4.47}$$

The interpolating variable Δ is chosen such that $\mathbf{e}_\Delta - \mathbf{e}_{j-1}$ is small enough, even for large θ, to avoid discretization error and large enough, even for small θ, to avoid round-off error. An empirical value of $\Delta = 0.001$ was used.

The quaternion exponential map[77] is defined by

$$\begin{aligned} \exp\left(0,\sigma(a,b,c)\right) &:= \sum_{i=1}^{\infty} \frac{1}{i!}\left(0,\sigma(a,b,c)\right)^i \\ &= \left(\cos\sigma, \sin\sigma(a,b,c)\right) \end{aligned} \tag{4.48}$$

where (a, b, c) is a unit vector, σ is an arbitrary scalar, and the transposition operator has been omitted for clarity.

The quaternion logarithm is defined as the inverse of the exponential and can be used in an alternative (equivalent) definition of slerp, which leads to

$$\dot{\mathbf{e}}_{j-1} = \frac{\ln(\mathbf{e}_j * \mathbf{e}^*_{j-1}) * \mathbf{e}_{j-1}}{\delta t_j} \tag{4.49}$$

Eq. (4.49) is equivalent to the limit of Eq. (4.47) as $\Delta \to 0$, and is therefore mathematically preferable, but computationally Eq. (4.47) requires fewer operations (approximately 25 fewer flops) than Eq. (4.49).

4.8.4 Numerical Comparison of Control Expressions

The five candidate angular velocity control expressions described in this chapter are:

- Eq. (4.34).
- Eq. (4.36).
- Eq. (4.42) with (4.43).
- Eq. (4.42) with (4.47).
- Eq. (4.42) with (4.49).

A numerical comparison of these expression was carried out in two stages: since Eqs. (4.47) and (4.49) should give the most accurate results, these two expressions were compared against each other, then the worst of the two was compared against the remaining expressions.

4.8.4.1 Test Setup

A test database of trajectories was created by manually generating arbitrary boundary conditions. Pairs of horizontal positions in the range [-2300,2300] m and distributed across the four horizontal quadrants were generated, with vertical displacements in the range [-200,500] m, five values of heading (a value in each quadrant, and zero), three values of flight path angle and three values of bank angle all in the range [-10°,5°], and

two values of load factor (9.81 and 15 m/s^2). Airspeed was held constant to eliminate any errors arising from a_x. These boundary conditions were combined with an optimization vector of $(60,0,0,0,0,0,0)^T$, resulting in 2160 different trajectories. The tests were carried out using uniformly-spaced nodes with N = 129. The Matlab *ode45* (Runge-Kutta 4/5) function was used to numerically integrate the state equations, and error was defined as the two-norm of the difference between demanded position and the *ode45* position at each node.

4.8.4.2 Results

Figure 4-6 shows the peak differences in errors for Eqs. (4.47) and (4.49). Eq. (4.49) was found to produce tracking errors of less than approximately 0.02% of path length, (Using N = 257 and Eq. (4.49), the tracking error reduced to 0.005% of path length.)

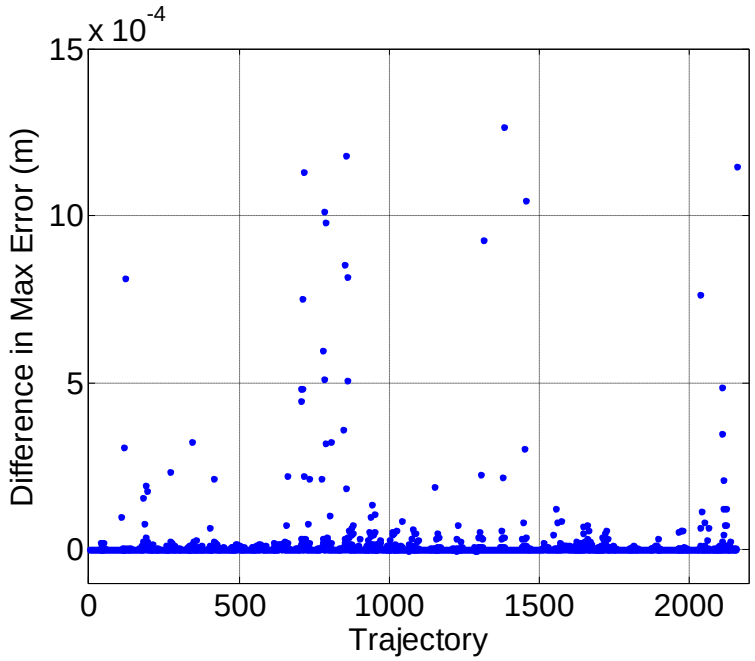

Figure 4-6. Difference Between Maximum Errors, Eq. (4.47) - Eq. (4.49) (Positive values mean Eq. (4.47) has larger error)

The percentages of trajectories for which each expression resulted in the lowest maximum position errors were recorded. Eq. (4.49) was more accurate than Eq. (4.47) for 83% of the trajectories, and Eq. (4.49) position errors were never more than approximately 3×10^{-7}% of path length (equivalent to 10^{-6} m) worse than the Eq. (4.47)

position errors. Although less accurate, Eq. (4.47) peak errors were within 0.3% (1.5×10^{-3} m) of the Eq. (4.49) errors, confirming that Eq. (4.49) is slightly more accurate than Eq. (4.47), but that both produce good tracking results. Corroborating results were obtained using Chebyshev-Gauss-Lobatto node distribution and higher N: as N increased the accuracy of both expressions increased, and the difference consequently reduced.

As the slightly less accurate of the two, Eq. (4.47) was used as the basis for comparison with the other three expressions: Eq. (4.47) was the most accurate for approximately 97% of trajectories, and Eq. (4.43) for the remaining 3% of trajectories. Further, Eq. (4.47) position errors were never more than approximately 0.01 m worse than the Eq. (4.43) position errors, but Eq. (4.43) position errors were often significantly worse than those from Eq. (4.47), as shown in Figure 4-7. Eqs. (4.34) and (4.36) were less accurate than Eq. (4.47) for all trajectories, Eq. (4.34) producing errors over 6000 m worse than Eq. (4.47) and Eq. (4.36) producing errors over 3700 m worse than Eq. (4.47). Results for a range of values of N and for Chebyshev-Gauss-Lobatto node spacing corroborated these findings.

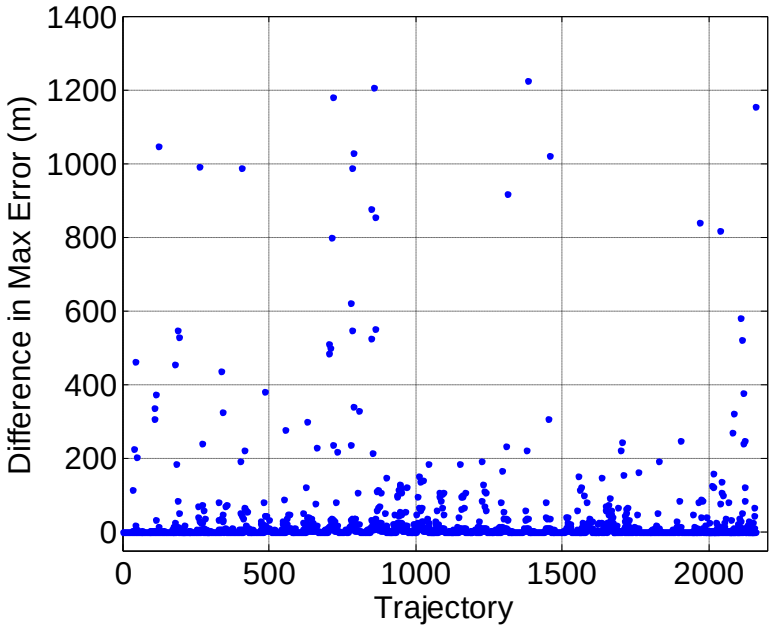

Figure 4-7. Difference Between Maximum Errors, Eq. (4.43) - Eq. (4.47) (Positive values mean Eq. (4.43) has larger error)

Eqs. (4.34), (4.36), (4.42) with (4.43), and (4.42) with (4.47) were evaluated for a course reversal trajectory (Section 3.4.4) in which at every node the aircraft was in straight and level unaccelerated flight and the path had two 180° course reversals, each course reversal taking place entirely between two consecutive nodes. Uniformly-spaced nodes with $N = 210$ were used. For all four control expressions, the errors in the evaluated bank and pitch rates were less than 10^{-6} (to be expected since they should be zero). Figure 4-8 shows the yaw rates over a course reversal. The time δt between nodes 154 and 155 was 0.1262 s: a 180° track change over that interval requires a yaw rate of approximately 1426 °/s. The relative error in the yaw rate generated by the combination of Eqs. (4.42) with (4.47) was less than 10^{-6}, showing that the controls can accurately track this trajectory despite its singularity due to $s' = 0$.

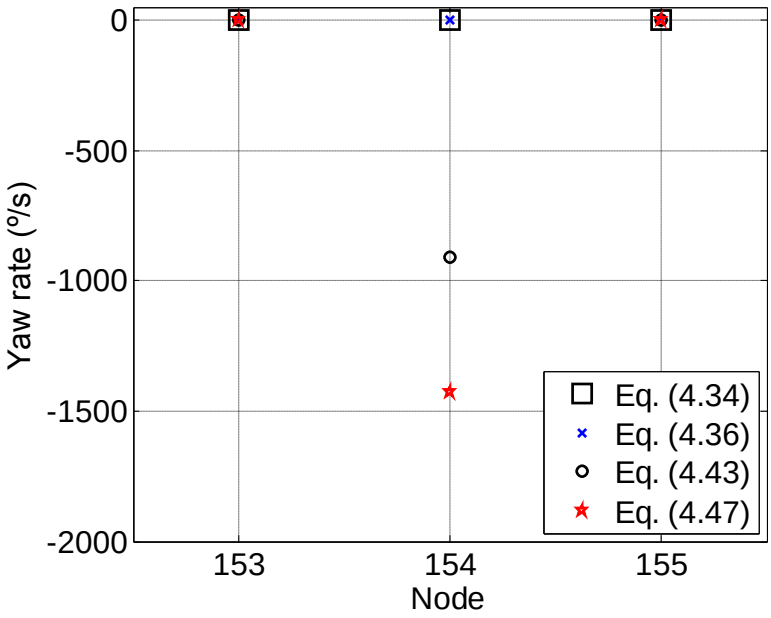

Figure 4-8. Yaw Rates for Course Reversal

4.8.5 Conclusion

For mathematical accuracy, the Shepperd algorithm to evaluate **e**, Eq. (4.49) for **ė**, and Eq. (4.42) to evaluate **u** should be chosen. This analysis assumes that there are no disturbances, noise or uncertainty, and that the model accurately represents the aircraft.

In practice the overall system relies on feedback to compensate for these unwanted effects, so the computational cost of the higher accuracy of Eq. (4.49) over Eq. (4.47) may not be justified. This design decision depends on the particular aircraft and mission requirements.

4.9 Computation of Normal Load Factor

A more computationally efficient node-based expression for $\mathbf{l}_z$ than Eq. (4.22) is to resolve the total force per unit mass on the model into tangential and normal components:

$$\mathbf{f}_M = \ddot{\mathbf{r}} - \mathbf{g} \tag{4.50}$$

$$\mathbf{f}_x = \left(\mathbf{f}_M \cdot \hat{\mathbf{x}}_W\right)\hat{\mathbf{x}}_W \tag{4.51}$$

$$\mathbf{l}_z = \mathbf{f}_M - \mathbf{f}_x \tag{4.52}$$

The node-based value of Eq. (4.52) is required to evaluate $\hat{\mathbf{z}}_W$ to ensure that the model is aligned with the parameterized spatial trajectory at each node; accurate magnitude is not critical for this purpose because $\hat{\mathbf{z}}_W$ is a unit vector. The segment-based value of Eq. (3.45) is used for constraint evaluation, for which magnitude is essential.

4.10 Singular Arc

The quaternion model state equations (4.3)-(4.8) are linear in **u**. For the minimum time problem this leads to a singular arc because $\partial f/\partial \mathbf{u} = 0$, the Hamiltonian becomes $H = 1 + \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{f}$, therefore $\partial H/\partial \mathbf{u} = 0$ for all admissible **u**. The inverse dynamics method handles the singular arc without problems, but this property of the model may limit its use in methods that have difficulty handling singular arcs.

4.11 Revised Trajectory Evaluation Algorithm

For convenience, the expressions required at each node, using the quaternion-based model, are listed below. The list assumes that **r,** v, their derivatives with respect to τ, and s' and s'' have been evaluated at the node and that $s' = 0$, $\delta s_j = 0$, $l_z = 0$ and $v \le 0$ are

handled as already described. The subscript *j* is omitted in the equations below except when both *j* and *j* - 1 appear.

$$\delta s_j = \left\| \mathbf{r}_j - \mathbf{r}_{j-1} \right\| \tag{4.53}$$

$$\delta t_j = \begin{cases} \dfrac{\delta s_j}{\left(v_j + v_{j-1}\right)/2} \delta\tau_j, & \text{if } \delta s_j \neq 0 \\ \dfrac{s'_j}{v_j} \delta\tau_j, & \text{if } \delta s_j = 0 \end{cases} \tag{4.54}$$

$$\lambda = \frac{v}{s'} \tag{4.55}$$

$$\lambda' = \frac{v' - \lambda s''}{s'} \tag{4.56}$$

$$a_{x\,j-1} = \frac{v_j - v_{j-1}}{\delta t_j} \tag{4.57}$$

$$\hat{\mathbf{x}}_W = \frac{\mathbf{r}'}{s'} \tag{4.58}$$

$$\ddot{\mathbf{r}} = \lambda\left(\lambda \mathbf{r}'' + \lambda' \mathbf{r}'\right) \tag{4.59}$$

$$\mathbf{l}_z = \left(\ddot{\mathbf{r}} - \mathbf{g}\right) - \left(\left(\ddot{\mathbf{r}} - \mathbf{g}\right) \cdot \hat{\mathbf{x}}_W\right) \hat{\mathbf{x}}_W \tag{4.60}$$

$$\hat{\mathbf{z}}_W = -\frac{\mathbf{l}_z}{\left\| \mathbf{l}_z \right\|} \tag{4.61}$$

$$\mathbf{e} = \text{Shepperd}\left(\hat{\mathbf{x}}_W, \hat{\mathbf{z}}_W \times \hat{\mathbf{x}}_W, \hat{\mathbf{z}}_W\right) \tag{4.62}$$

$$\mathbf{e}_j \leftarrow \begin{cases} \mathbf{e}_j, & \text{if } \mathbf{e}_j \cdot \mathbf{e}_{j-1} \geq 0 \\ -\mathbf{e}_j, & \text{otherwise} \end{cases} \tag{4.63}$$

$$\dot{\mathbf{e}}_{j-1} = \frac{\ln(\mathbf{e}_j * \mathbf{e}^*_{j-1}) * \mathbf{e}_{j-1}}{\delta t_j} \tag{4.64}$$

$$\boldsymbol{\omega} = 2\begin{pmatrix} -\dot{e}_0 e_1 + \dot{e}_1 e_0 + \dot{e}_2 e_3 - \dot{e}_3 e_2 \\ -\dot{e}_0 e_2 - \dot{e}_1 e_3 + \dot{e}_2 e_0 + \dot{e}_3 e_1 \\ -\dot{e}_0 e_3 + \dot{e}_1 e_2 - \dot{e}_2 e_1 + \dot{e}_3 e_0 \end{pmatrix} \tag{4.65}$$

$$\begin{gathered} \mathbf{g}_z = \frac{g}{s'^2}\left(-x'z', -y'z', x'^2 + y'^2\right) \\ \zeta = \arccos\left(\hat{\mathbf{x}}_{W\,j} \cdot \hat{\mathbf{x}}_{W\,j-1}\right) \\ \mathbf{l}_z^{seg}\,|_{j-1} = 0.5\left(\left(v_j + v_{j-1}\right)\left(\frac{\zeta/2}{\sin\zeta/2}\right)\frac{1}{\delta t_j}\left(\hat{\mathbf{x}}_{W\,j} - \hat{\mathbf{x}}_{W\,j-1}\right) - \left(\mathbf{g}_{z\,j} + \mathbf{g}_{z\,j-1}\right)\right) \end{gathered} \tag{4.66}$$

$$T = \left(a_x - \frac{gz'}{s'}\right)M + D \tag{4.67}$$

The three expressions in Eq. (4.66) implement Eq. (3.45).

The constraints may be evaluated using quadratic interpolation. Define a function *m* that applies local quadratic interpolation to find the maximum of a curve passing through *N* node values then

$$\mathbf{c} = \begin{pmatrix} m\left(l_{z\,j}^{seg} - l_j^+, \quad \forall j \in [1, N]\right) \\ m\left(T_j - T_{\max}, \quad \forall j \in [1, N]\right) \\ m\left(T_{\min} - T_j, \quad \forall j \in [1, N]\right) \\ m\left(p_j - p_{\max}, \quad \forall j \in [1, N]\right) \\ m\left(-p_{\max} - p_j, \quad \forall j \in [1, N]\right) \\ m\left(v_s - v_j, \quad \forall j \in [1, N]\right) \\ m\left(v_j - v_{ne}, \quad \forall j \in [1, N]\right) \end{pmatrix} \tag{4.68}$$

The final flight time is evaluated by

$$t_f = a(\tau_f)\sum_{j=1}^{N} w_j \frac{s'_j}{v_j} \tag{4.69}$$

where a(τ_f) and the weights **w** are determined using standard formulae dependent only on the node distribution and *N*.

If a penalty function is required because of the particular NLP algorithm being used, the penalized objective may be evaluated by

$$J = t_f + \rho P(\mathbf{c}^+) \tag{4.70}$$

where

$$c_i^+ = \max(0, c_i), \quad i = \{1, \ldots 7\} \tag{4.71}$$

The penalty function used in this work was the squared two-norm

$$P(\mathbf{c}^+) = \sum_{i=1}^{7} k_i \left\| c_i^+ \right\|_2^2 \tag{4.72}$$

5 AIRSPEED PARAMETERIZATION

5.1 Introduction

The optimality of a solution to a minimum-time aircraft trajectory generation problem depends on the closeness of the generated airspeed to the maximum airspeed that satisfies all path and boundary constraints. Hence the accuracy and computational speed of airspeed determination is a critical part of the method. Low-degree polynomial parameterization reduces optimality, but high-degree parameterization increases the dimension of the optimization problem. No structured approach to choosing the most appropriate airspeed parameterization was found in the literature. This chapter describes a new computational approach to estimate maximum feasible airspeed without airspeed parameterization or optimization. Results obtained with this approach are then used to measure the effects of the degree and form of polynomial airspeed parameterization on the robustness, optimality and computational speed of optimization in the inverse dynamics method. The effects of Chebyshev, barycentric Lagrange, Bernstein and power series polynomial basis functions are compared.

The computation times taken by direct evaluation of maximum feasible airspeed are also compared to the times taken to optimize parameterized airspeed, as a guide to whether direct evaluation of maximum airspeed is a practicable alternative to airspeed optimization.

5.2 Direct Evaluation of Maximum Feasible Airspeed

The airspeed profile of a minimum-time trajectory is the maximum airspeed that satisfies all constraints. In this section instead of parameterizing airspeed and optimizing the parameters (airspeed coefficients), the maximum feasible airspeed (v_{max}) and corresponding minimum feasible final flight times (t_{Ref}) are derived directly from the spatial path without optimization.

Section 3.4.7 describes a set of aircraft constraints $\{l_z, T_{max}, T_{min}, p_{max}, p_{min}, v_{ne}, v_s\}$. From this set the maximum feasible airspeed v_{max} may be written

$$v_{max} = \min\left(v_{ne}, v_l, v_{Tmax}, v_{Tmin}, v_p\right) \tag{5.1}$$

where v_l, v_{Tmax}, v_{Tmin}, and v_p are the maximum airspeeds that satisfy their respective constraints. Clearly if $v_{max} < v_s$ then there is no feasible airspeed for that spatial path.

Each trajectory is discretized and each element of Eq. (5.1) is evaluated at each node as described below, assuming that N is sufficiently large that acceleration between nodes may be assumed constant and the Euclidean distance between nodes is a good approximation to the corresponding arc length.

The next four sections describe the evaluation of v_l, v_{Tmax}, v_{Tmin} and v_p at each node. All the expressions apply to values at node j unless subscripted to apply at node j-1.

5.2.1 Normal Load Factor

The load factor magnitude at each node may be expressed as an explicit function of v

$$l_z = \left\| v^2 \kappa \hat{\boldsymbol{\varepsilon}}_n - \mathbf{g}_z \right\| \tag{5.2}$$

where $\mathbf{g}_z$ is given by Eq. (3.49), κ by Eq. (4.13) and $\hat{\boldsymbol{\varepsilon}}_n$ by Eq. (4.15). Eq. (5.2) is equivalent to Eq. (4.22). The expression

$$v_l = \max\left(v \mid l^+ - l_z\right) \tag{5.3}$$

can be solved using any suitable univariate zero-finding algorithm. Brent's method[21, 108] is robust, stable and efficient, with superlinear and guaranteed convergence. To ensure convergence, it is only necessary to bracket the zero by subdividing the speed range at $v = v_a$ and checking the sign of $l^+ - l_z$ at $v = v_s$, $v = v_a$, and $v = v_{ne}$.

5.2.2 Maximum Thrust

The load-factor-limited airspeed v_l at any node $j \in \{1,\ldots N\}$ may be evaluated solely on data at j, but thrust-limited airspeeds v_{Tmax} and v_{Tmin} depend on values at multiple nodes. Given an initial airspeed v_0 and a drag polar $D(v, l_z, M)$, (Eqs. (3.36)-(3.38)) the

following algorithm may be used to evaluate the maximum airspeed (v_{Tmax}) that satisfies the upper thrust limit (T_{max}).

1. Set $v_{max} = v_l$, $\forall j \in \{1,...N\}$.
2. Set $v_{Tmax\,1} = v_0$ and $v_{Tmax\,N} = v_f$.
3. Iterate for $j \in \{2,\ldots N - 1\}$:

   Evaluate drag D_1 *at* $v_{max\,j-1}$ and D_2 at v_{ne}

$$D_1 = D\left(v_{\max j-1}, l_z\left(v_{\max j-1}\right), M\right) \tag{5.4}$$

$$D_2 = D\left(v_{ne}, l_z\left(v_{ne}\right), M\right) \tag{5.5}$$

   Evaluate the mean gravitational component

$$g_c = 0.5\left(\left.\frac{gz'}{s'}\right|_{j-1} + \left.\frac{gz'}{s'}\right|_{j}\right) \tag{5.6}$$

   Evaluate the available acceleration a_A over $[\tau_{j-1}, \tau_j]$

$$a_A = \left(T_{\max} - 0.5\left(D_1 + D_2\right)\right)/M + g_c \tag{5.7}$$

   Evaluate the acceleration a_2 required to reach v_{ne} over $[\tau_{j-1}, \tau_j]$

$$a_2 = \left(v_{ne} - v_{max\,j-1}\right)\left(v_{ne} + v_{max\,j-1}\right)/2\delta s_j \tag{5.8}$$

   If $a_A \geq a_2$ then $v_{Tmax\,j} = v_{ne}$; else

      Evaluate the acceleration a_1 required to reach v_s

$$a_1 = \left(v_s - v_{max\,j-1}\right)\left(v_s + v_{max\,j-1}\right)/2\delta s_j \tag{5.9}$$

      If $a_A < a_1$ then trajectory is infeasible, stop.

      Else apply Brent's method to solve

$$\frac{\left(T_{max} - 0.5\left(D\left(v_{Tmax\,j}, l_z, M\right) + D_1\right)\right)}{M} + \frac{gz'}{s'} + \frac{\left(v_{Tmax\,j} - v_{max\,j-1}\right)\left(v_{Tmax\,j} + v_{max\,j-1}\right)}{2\delta s_j} = 0 \tag{5.10}$$

for $v_{Tmax\,j}$.

4. Loop to step 3.

5.2.3 Minimum Thrust

Clearly at each node the aircraft must be flying sufficiently slowly so that if maximum deceleration was applied (which would occur at minimum thrust) the deceleration would be sufficient to reduce the airspeeds at all later nodes to within the maximum airspeeds imposed by load factor, maximum thrust, bank rate and v_{ne} at those later nodes. Therefore maximum feasible airspeed is also constrained by the deceleration achievable at minimum thrust, to ensure that the aircraft can decelerate so as to not exceed $\min\left(v_l, v_{Tmax}, v_{ne}\right)$ at any later node. Defining v_{Tmin} as the maximum airspeed that ensures that at minimum thrust the aircraft can decelerate sufficiently to satisfy this requirement, the following algorithm may be used to evaluate v_{Tmin} without requiring iterative zero-finding.

1. Set $v_{Tmin\,N} = v_f$ and $v_{lim} = v_f$
2. $\forall j \in \{1,\ldots N\}$ set $v_{1\,j} = \min\left(v_l, v_{Tmax}, v_{ne}\right)$
3. Iterate backwards for $j \in \{N - 1,\ldots 1\}$:

   Evaluate drag $D\left(v_{1\,j}, l_z\left(v_{1\,j}\right), M\right)$, acceleration $a_1 = \left(T_{min} - D\right)/M + gz'/s'$, the time δt taken to cover the Euclidean distance δs between j and j+1 starting at speed v_1 with acceleration a_1

$$\delta t = \begin{cases} \dfrac{\delta s}{v_1}, & \text{if } a_1 = 0 \\ \dfrac{-v_1 + \sqrt{v_1^2 + 2a_1\delta s}}{a_1}, & \text{if } (a_1 \neq 0) \bigcap \left(a_1 > \dfrac{-v_1^2}{2\delta s} \right) \\ \dfrac{2\delta s}{v_1}, & \text{otherwise} \end{cases} \tag{5.11}$$

Evaluate the minimum achievable speed v_2 at j+1 given by $v_2 = v_{1\,j} + a_1 \delta t$

If $v_2 > v_{lim}$ then

If $j > 1$ then

Evaluate drag $D\left(v_{lim}, l_z\left(v_{lim}\right), M\right)$

Evaluate acceleration $a_2 = \left(T_{min} - D\right)/M + gz'/s'$

Evaluate limiting airspeed $v_{Tmin\,j} = \sqrt{v_2^2 - 2a_2\delta s}$

Reset $v_{lim} = v_{Tmin\,j}$

Else (i.e. $j = 1$) trajectory is infeasible, stop.

Else (i.e $v_2 \leq v_{lim}$) continue iteration

4. Loop to step 3.

5.2.4 Bank Rate

Bank rate p is a function of $\left\{\dot{\mathbf{r}}, \mathbf{l}_z, \dot{\mathbf{l}}_z\right\}$ and depends on values at multiple nodes. It is not necessary to solve p_{max} - p = 0 since the primary purpose of directly evaluating v_{max} for the reference paths is to create reference maximum airspeeds and final flight times and the secondary purpose is to measure computation times. It is only necessary to categorize as infeasible any trajectories for which $p > p_{max}$ when $v = \min(v_{ne}, v_l, v_{Tmax}, v_{Tmin})$. Since v_p has not been evaluated in this work the computation times will be shorter than evaluation of v_{max} including v_p.

Load factor and thrust constraints tend to be more restrictive than the bank rate for small aircraft, so the p_{max} condition should not reduce the number of feasible trajectories in the

database significantly: for the test database used in this work it only removed approximately 0.05% of otherwise feasible trajectories.

5.2.5 Evaluation of Minimum Feasible Airspeed

Parenthetically, minimum feasible airspeed v_{min} may be readily evaluated without iterative zero-finding, since it is constrained by v_s, T_{max} and T_{min} only (assuming $v_{max} \geq v_s$).

To evaluate the minimum thrust effect, set $v_{min\,1} = v_0$, $v_{min\,N} = v_f$ and iterate for $j \in \{2,\ldots N - 1\}$: evaluate drag $D\left(v_{min\,j-1}, l_z\left(v_{min\,j-1}\right), M\right)$, acceleration $a = \left(T_{min} - D\right)/M + gz'/s'$, and the time δt taken to cover the Euclidean distance δs between j - 1 and j with acceleration a using Eq. (5.11) then set $v_{min\,j} = \max\left(v_s, v_{min\,j-1} + a\delta t\right)$ and continue the iteration.

Some manoeuvres, such as a sustained steep climb, may cause deceleration even at maximum thrust; hence the aircraft must start such a manoeuvre at a sufficiently high airspeed to ensure that the airspeed remains above v_s throughout. We define the minimum airspeed that satisfies this requirement as v_B. After the preliminary evaluation of v_{min} $\forall j \in \{1\ldots N\}$ using the preceding paragraph, v_B and hence v_{min} can be evaluated as follows:

1. Set $v_{lim} = v_f$
2. Iterate backwards for $j \in \{N - 1,\ldots 1\}$:

   Evaluate drag $D\left(v_{min\,j}, l_z\left(v_{min\,j}\right), M\right)$,

   Evaluate acceleration $a = \left(T_{max} - D\right)/M + gz'/s'$

   Evaluate the time δt taken to cover the Euclidean distance δs between j and j+1 starting at speed v_1 with acceleration a using Eq. (5.11)

   Evaluate the maximum achievable speed v_2 given by $v_2 = v_{min\,j} + a\delta t$

   If $v_2 < v_{lim}$ then

If $j > 1$ then

Evaluate drag $D\left(v_{lim}, l_z\left(v_{lim}\right), M\right)$

Evaluate acceleration $a = \left(T_{max} - D\right)/M + gz'/s'$

Evaluate $v_{B\,j} = \sqrt{v_2^2 - 2a\delta s}$

Reset $v_{lim} = v_{B\,j}$ and continue iteration

Else (i.e. j=1) trajectory is infeasible, stop.

Else (i.e $v_2 \geq v_{lim}$) set $v_{lim} = v_{min\,j}$ and continue iteration

3. Loop to step 2.
4. Set $v_{min\,j} = \max\left(v_{min\,j}, v_{B\,j}\right), \forall j \in \{2, ...N-1\}$.

The range of feasible arrival times at the final point is defined by v_{max} and v_{min}: a feasible airspeed profile for any in-range desired final flight time may be obtained by linear interpolation, for example to achieve a rendezvous.

5.2.6 Computation Times – Direct Evaluation

To obtain results for a large sample of boundary conditions, a test database of feasible spatial paths was created (Section 5.3.1), and the maximum feasible airspeed v_{max} and corresponding final flight time t_{Ref} for each spatial path in the test database were evaluated.

The computation time taken to directly evaluate v_{max} will depend on the number of iterations required by the zero-finding algorithm, on the values of the conditional expressions, and on the hardware and software environment. As a guide the mean time taken to compute v_{max} per trajectory, averaged over a test database of the feasible spatial paths (Section 5.3.1), with N = 257, was 0.14 s.

The mean times taken to evaluate v_{max} per node, over all trajectories and over only feasible trajectories (i.e. trajectories for which $v_{max} \geq v_s \; \forall \tau \in [0, \tau_f]$), for various N are shown in Table 5-1.

N	**All trajectories: mean time per node (s)**	**Feasible trajectories: mean time per node (s)**
257	2.41×10^{-4}	5.47×10^{-4}
390	2.39×10^{-4}	5.47×10^{-4}
513	2.41×10^{-4}	5.46×10^{-4}
764	2.41×10^{-4}	5.45×10^{-4}

Table 5-1. Computation Times – Direct Evaluation (s)

The direct evaluation algorithm detects infeasibility as it iterates and will terminate after infeasibility is detected, so the mean time for an infeasible trajectory will, over a sufficiently large sample, tend to be significantly less than that for a feasible trajectory. This effect is clear in Table 5-1.

In Section 5.3.3.1 these times are compared to the computation times taken by airspeed optimization.

5.3 Airspeed Optimization

This section quantifies the effects of low-degree polynomial airspeed parameterization on the optimality, robustness and computational speed of the inverse dynamics method.

Airspeed was parameterized by polynomials of degree d_v, and the $d_v + 1$ coefficients were used as the vector of optimization parameters that was input to the NLP algorithms. The NLP problem was to minimize t_f subject to

$$c_i^+ \le 0, \quad i = 1,2 \tag{5.12}$$

where

$$c_i^+ = \max_{j\in\{1,\dots N\}} \left(0, c_i\right), \quad i = 1,2 \tag{5.13}$$

$$\begin{aligned} c_{1j} &= v_j - v_{max\,j}, && \forall j \in \{1,\dots N\} \\ c_{2j} &= v_s - v_j, && \forall j \in \{1,\dots N\} \end{aligned} \tag{5.14}$$

Choosing the constraint set $\{v_{max}, v_s\}$ in Eq. (5.14) removed the effects of approximations in the evaluation of v_{max} from the optimization results, and obviated the need for any non-unity penalty weights.

The Sequential Nelder-Mead (SNM) algorithm was chosen as the primary NLP algorithm for the airspeed optimization tests because derivative-free algorithms were expected to be more robust than SNOPT, and the SNM algorithm was expected to be less dependent on the NLP dimension than the Sequential Hooke-Jeeves (SHJ) algorithm (Section 2.5.2). The SHJ algorithm was used in one experiment set, with Chebyshev parameterization and $d_v \in \{4,...16\}$, to confirm this expectation. The squared two-norm penalty function was used with a constraint tolerance of 0.1 m/s, which introduces the possibility that the NLP algorithm may produce a final flight time less than t_{Ref} .

Since the SNM and SHJ algorithms are local algorithms, it was necessary to use an initial airspeed guess as close as possible to the desired solution (v_{max}): the minimax interpolant to v_{max} would satisfy this requirement but computing an exact minimax interpolant is computationally expensive. Truncated Chebyshev interpolants[18] are close to the minimax interpolant and are readily computed. Hence the degree $N - 1$ Chebyshev interpolant to the v_{max} profile for each spatial path in the test database was generated, using the Chebyshev-Gauss-Lobatto node distribution to avoid ill-conditioning[50], then truncated to degree d_v to form the initial guess.

Figure 5-1 shows an example of v_{max} and the corresponding truncated Chebyshev interpolants for $d_v = \{4,8,16\}$. The nonsmooth points in v_{max} arise where the active constraint changes, e.g. from l_z to v_{ne}. The constraint values used in this example were chosen by trial and error to produce multiple switching points. As expected, the interpolation error reduces as d_v increases.

A cubic is the minimum degree polynomial that can be guaranteed to meet the four initial and final boundary values of airspeed and tangential acceleration: a quartic is the lowest degree that has a degree of freedom available for optimization. The inverse dynamics method uses low-degree parameterization: hence $d_v \in [4,16]$ was chosen for this work.

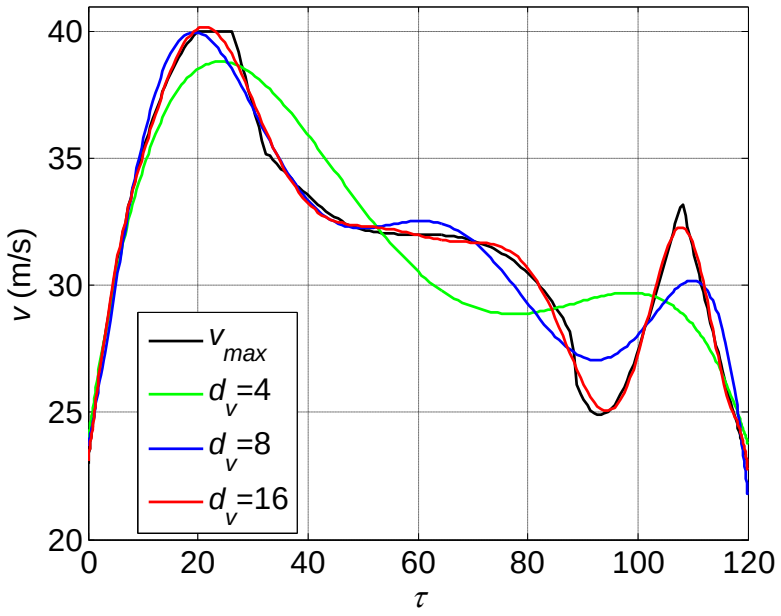

Figure 5-1. Example of v_{max} and Chebyshev Interpolants

Since any choice of polynomial basis functions can be used to exactly represent any polynomial, the optimal solution is not dependent on the choice of the basis functions (except in so far as the choice causes the NLP algorithm not to converge to an optimal solution) but only the degree of the function itself. However, the relative magnitudes of the coefficients of the function and of its derivatives do vary with choice of basis functions i.e. the function gradient is dependent on the basis functions, so the convergence of an NLP algorithm will depend on the choice of basis functions.

To compare different forms of parameterization, the truncated Chebyshev interpolants were transformed to the Bernstein form[46], the barycentric Lagrange form[11], and power series form. These forms were chosen because the Bernstein basis is optimally stable (Farouki and Goodman[45]), the Lagrange form retains node values as coefficients and the power form is widely used. The Chebyshev-Gauss-Lobatto node distribution was used for all experiments to avoid any ill-conditioning.

To retain the closeness of the initial guesses to v_{max}, basis conversions were carried out after truncation of the degree N -1 Chebyshev interpolation. If the conversion relies on inverting a Van Der Monde matrix (for example for the Bernstein and power series

forms), the conversion may be ill-conditioned. The accuracy of the conversion was checked and it was found that the errors introduced by transforming the initial guesses from Chebyshev to Lagrange, Bernstein and power series form were less than 10^{-13}, 10^{-4}, and 2×10^{-5} respectively.

For Chebyshev parameterization the optimization vector is the set of d_v+1 Chebyshev coefficients, each of which has global effect, and the coefficients tend to decrease to zero for smooth functions as the degree of the associated term increases.

For the Bernstein form the optimization vector is the set of d_v+1 control point values. This form is particularly well suited to Hermite interpolation at the boundary points because the $k+1$ coefficients at each end of the optimization vector are directly related to v and its first k derivatives at the boundary points; the NLP dimension can therefore easily be reduced by four by fixing the first two and last two coefficients using airspeed and acceleration at the boundary points (this was not used in this chapter, but was used in Chapter 6, see Section 6.2.4). The control points form a convex hull of the curve, so each parameter affects the whole curve, but its effect is most concentrated in its own locality.

For Lagrange form parameterization the vector is the set of values of airspeed at $d_v + 1$ points: each coefficient therefore has most effect in a small locality.

Algorithms for manipulating Chebyshev polynomials are well-known[18, 20, 108]. The chebfun operator introduced by Battles and Trefethen[6] was used to generate the Chebyshev coefficients, from which the Clenshaw recurrence[108] was used to evaluate the degree d_v interpolants. The algorithms described by Farouki and Rajan[46] were used to generate Bernstein coefficients and evaluate the polynomials, and the algorithms described by Berrut and Trefethen[11] were used with the barycentric Lagrange parameterizations.

5.3.1 Aircraft Data and Test Database

The aircraft data were: M = 11 kg, $l_{zstruct}$ = 27 m/s^2, $0 \leq T \leq 30$ N, v_s = 15 m/s, v_{ne} = 40 m/s, p_{max} = 400 °/s.

To provide test spatial paths, a series of ranges of arbitrary (manually generated) boundary values defining {**r**, **r**', **r**'', **r**'''} were chosen with path lengths covering approximately 500 m to 30 km, with boundary airspeeds in the range [18, 35] m/s, tangential accelerations in the range [-1, 2] m/s^2, load factors in the range [0.5g, 2g], bank angles in the range [-40 40] degrees, flight path angles in the range [-6, 10] degrees, and τ_f values in the range [20, 120]. These ranges were chosen as reasonably representative for the aircraft data being used. Combinations of these values were used to define test spatial paths; each path was then evaluated as described in Section 5.2 resulting in a set of over 49,000 feasible spatial paths (i.e. $v_{max} \geq v_s$, $\forall \tau \in [0, \tau_f]$) and their associated v_{max} profiles; these profiles were used as the starting points for the initial guesses for the optimizations and as the test set for the evaluation of constraint accuracy in Section 3.5.

5.3.2 NLP Settings

Section 6.2.6.2 describes the SNM and SHJ algorithms and settings used in Chapter 6; in this chapter the same settings were used except for the SNM settings shown in Table 5-2.

Variable	Description	Value
σ_{max}	Maximum trajectory evaluations	150000
$\Delta_{f\,0}$	Initial objective range	128
$\Delta_{f\,thresh}$	Objective threshold	1
$\Delta_{f\,min}$	Minimum objective range	10^{-4}
Δ_{χ}	Maximum ∞-norm of χ_w-χ_b	1
$\chi_{\Delta\,0}$	Initial step vector	0.05χ

Table 5-2. NLP Settings

It was found that, for all trajectories, with SNM and the power series form when $d_v \geq 10$ the initial simplex exceeded the maximum objective value; the limit was raised to 10^{20} and the minimum initial simplex step size was reduced from 2.5×10^{-4} to 2.5×10^{-17} to overcome this.

5.3.3 Results and Analysis

5.3.3.1 Computation Times

Mean computation time to optimize airspeed per trajectory, using the Chebyshev form of airspeed parameterization, over successful optimizations and with $N = 257$, was 0.044 s. Comparing this value with the time taken to estimate v_{max} given in Section 5.2.6 shows that airspeed optimization using the SNM/Chebyshev combination was approximately 3 times faster than the algorithm of Section 5.2, although since spatial optimization was not performed this finding does not necessarily generalize to combined spatial and airspeed optimization. Further, to use direct evaluation of v_{max} as a replacement for parameterized airspeed, v_p would have to be evaluated instead of being imposed indirectly (Section 5.2.4). Hence direct evaluation is not a promising approach but may warrant future research.

5.3.3.2 Optimization Convergence, Optimality and Speed

Loss of optimality for each trajectory was defined as

$$\frac{t_f - t_{Ref}}{t_{Ref}} \tag{5.15}$$

Robustness was defined as the percentage of test cases for which the NLP algorithm satisfied the termination criteria within the constraint tolerance and within the allowed total number of trajectory evaluations (1.5×10^4).

Computational speed was measured by the number of trajectory evaluations (σ) invoked by the NLP algorithm.

The results are shown graphically in Figures 5-2 to 5-16 and in tabular form in Tables 5-3 to 5-7.

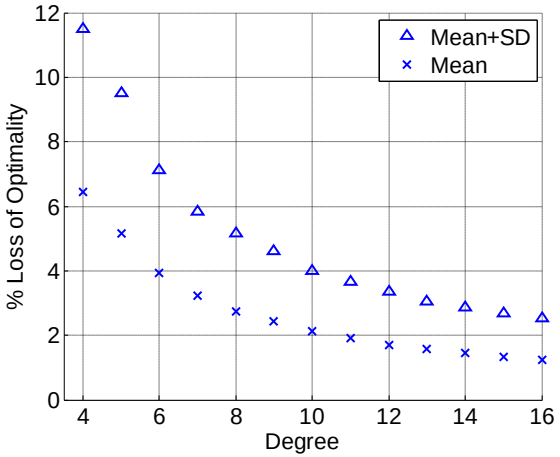

Figure 5-2. Loss of Optimality, SNM with Chebyshev Parameterization

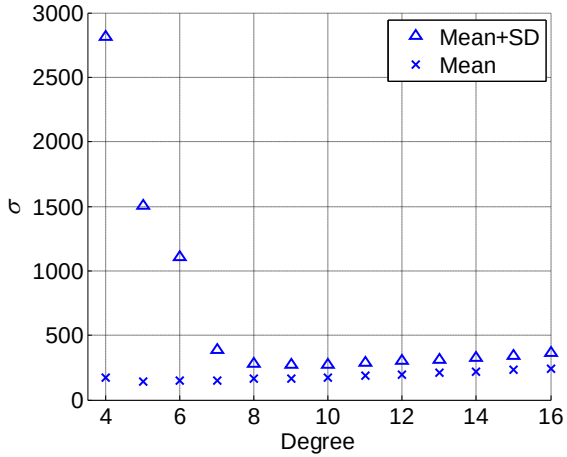

Figure 5-3. Trajectory Evaluations, SNM with Chebyshev Parameterization

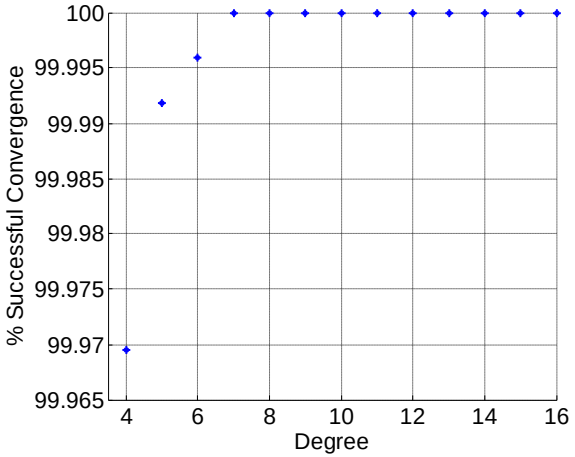

Figure 5-4. Robustness, SNM with Chebyshev Parameterization

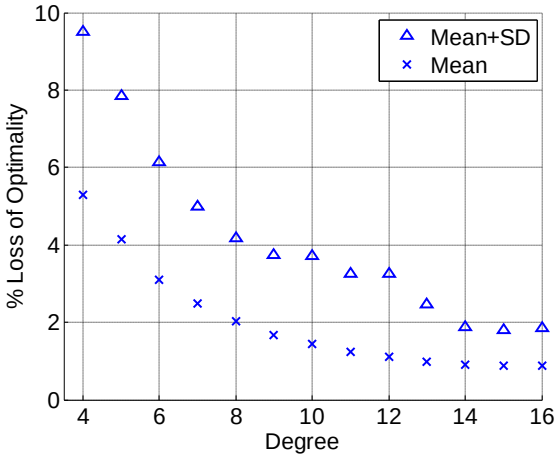

Figure 5-5. Loss of Optimality, SNM with Bernstein Parameterization

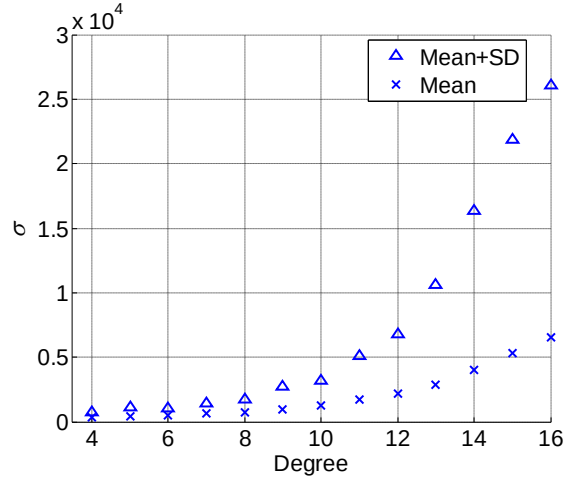

Figure 5-6. Trajectory Evaluations, SNM with Bernstein Parameterization

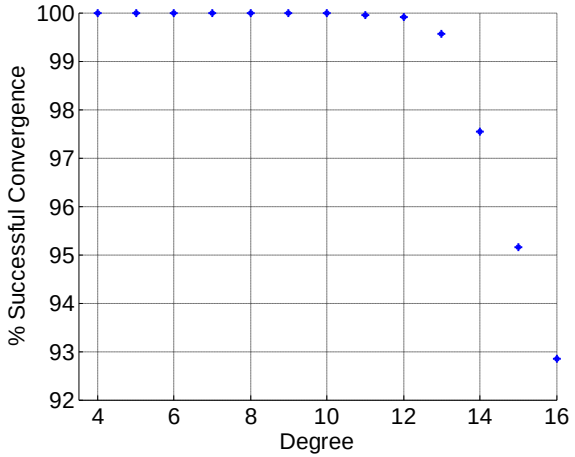

Figure 5-7. Robustness, SNM with Bernstein Parameterization

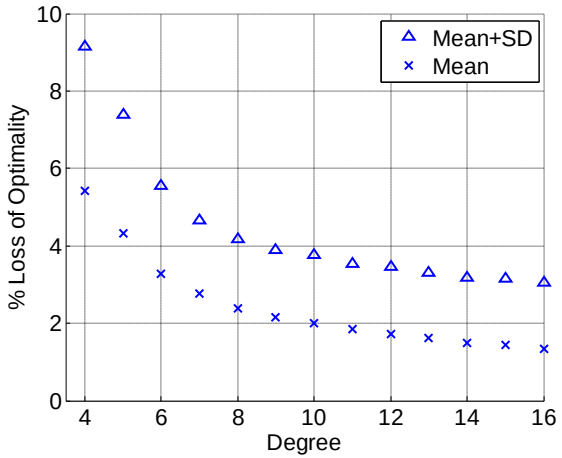

Figure 5-8. Loss of Optimality, SNM with Lagrange Parameterization

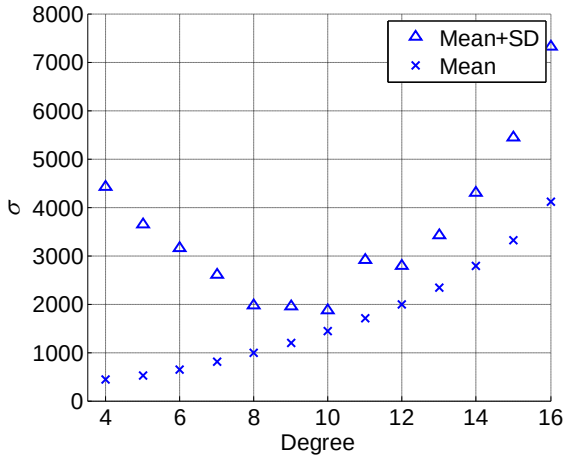

Figure 5-9. Trajectory Evaluations, SNM with Lagrange Parameterization

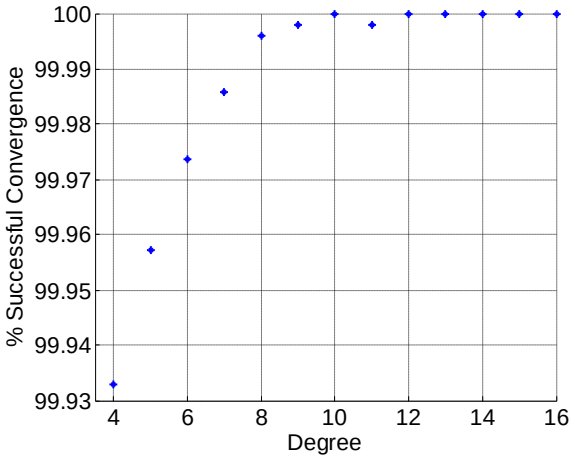

Figure 5-10. Robustness, SNM with Lagrange Parameterization

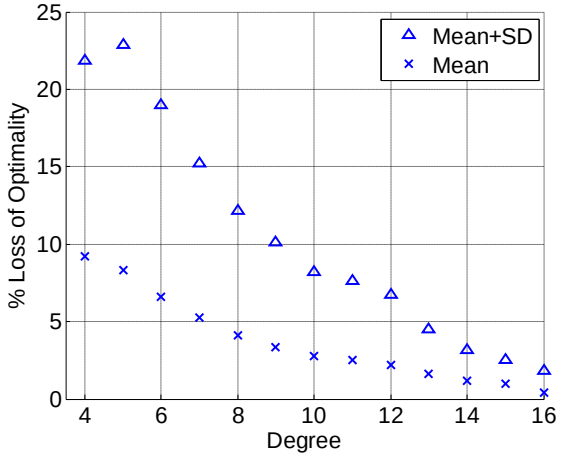

Figure 5-11. Loss of Optimality, SNM with Power Series Parameterization

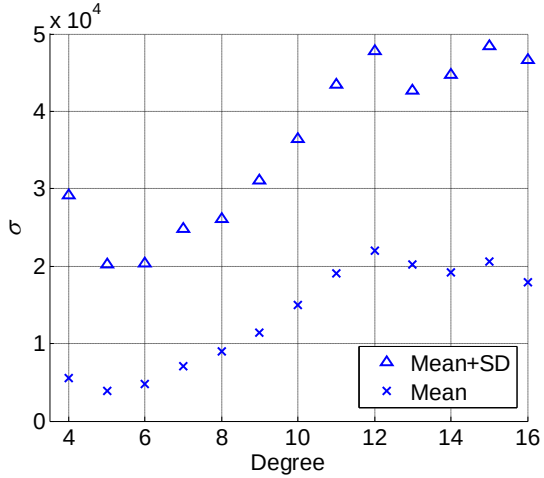

Figure 5-12. Trajectory Evaluations, SNM with Power Series Parameterization

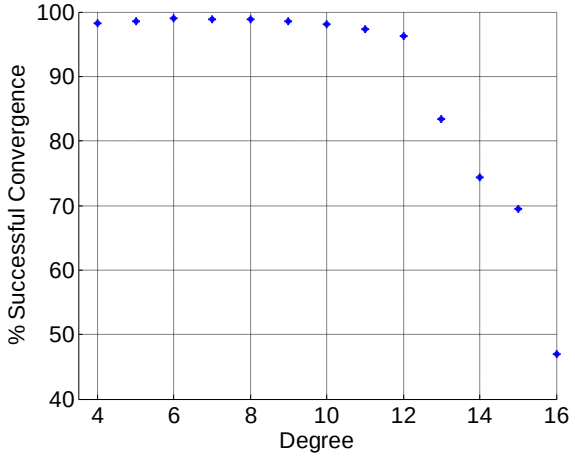

Figure 5-13. Robustness, SNM with Power Series Parameterization

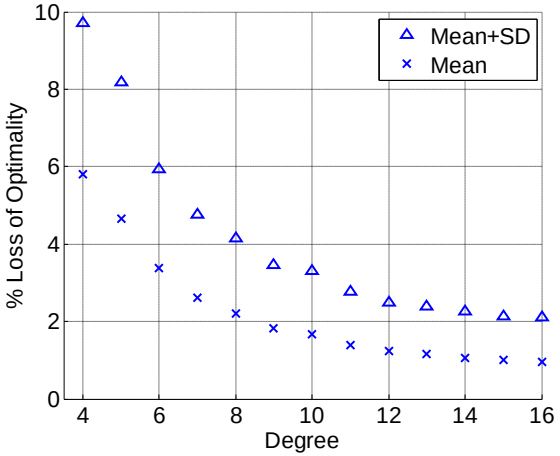

Figure 5-14. Loss of Optimality, SHJ with Chebyshev Parameterization

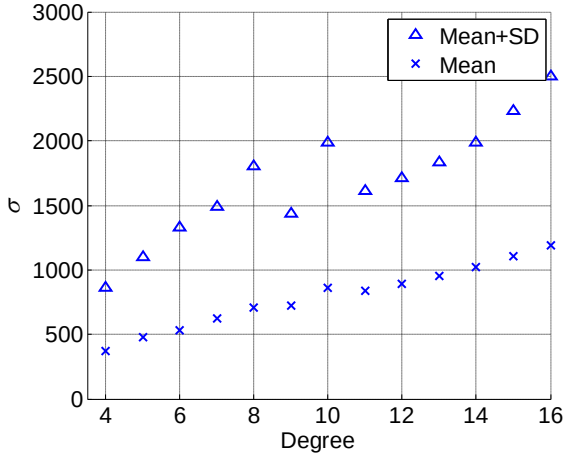

Figure 5-15. Trajectory Evaluations, SHJ with Chebyshev Parameterization

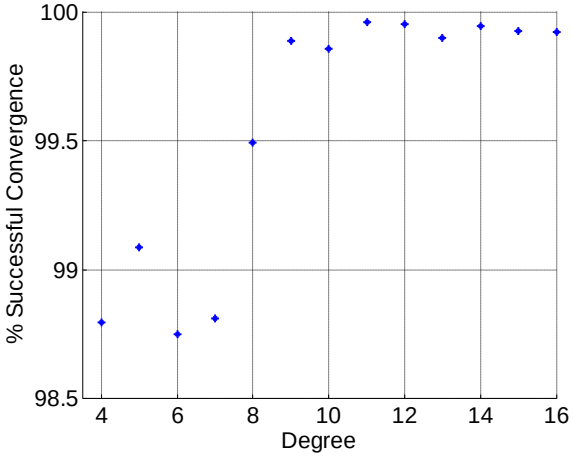

Figure 5-16. Robustness, SHJ with Chebyshev Parameterization

Combination	Degree												
	4	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
SNM and Chebyshev	6.4	5.2	3.9	3.2	2.7	2.4	2.1	1.9	1.7	1.6	1.4	1.3	1.2
SNM and Bernstein	5.3	4.1	3.1	2.5	2.0	1.7	1.4	1.2	1.1	1.0	0.9	0.9	0.9
SNM and Lagrange	5.4	4.3	3.3	2.8	2.4	2.2	2.0	1.8	1.7	1.6	1.5	1.4	1.4
SNM and power series	9.2	8.3	6.6	5.2	4.1	3.3	2.7	2.5	2.2	1.6	1.2	1.0	0.4
SHJ and Chebyshev	5.8	4.7	3.4	2.6	2.2	1.8	1.7	1.4	1.2	1.2	1.1	1.0	1.0

Table 5-3. Mean Loss of Optimality (%)

Combination	Degree												
	4	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
SNM and Chebyshev	5.1	4.3	3.2	2.6	2.4	2.2	1.9	1.7	1.6	1.5	1.4	1.4	1.3
SNM and Bernstein	4.2	3.7	3.0	2.5	2.2	2.1	2.3	2.0	2.1	1.5	1.0	0.9	1.0
SNM and Lagrange	3.7	3.0	2.3	1.9	1.8	1.7	1.8	1.7	1.7	1.7	1.7	1.7	1.7
SNM and power series	12.6	14.6	12.4	9.9	8.1	6.8	5.5	5.1	4.5	2.9	2.0	1.6	1.4
SHJ and Chebyshev	3.9	3.5	2.6	2.1	1.9	1.6	1.6	1.4	1.3	1.2	1.2	1.1	1.1

Table 5-4. Standard Deviation of Loss of Optimality (%)

Combination	Degree												
	4	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
SNM and Chebyshev	175	145	147	153	162	167	175	186	197	208	219	231	243
SNM and Bernstein	365	445	513	618	765	987	1296	1686	2176	2868	3997	5311	6579
SNM and Lagrange	447	517	649	802	980	1194	1434	1702	1996	2344	2778	3325	4107
SNM and power series	5586	3835	4791	7127	9014	11362	14987	19061	21942	20177	19167	20627	17933
SHJ and Chebyshev	374	479	533	627	704	726	858	840	892	954	1021	1109	1194

Table 5-5. Mean Trajectory Evaluations

Combination	Degree												
	4	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
SNM and Chebyshev	2641	1356	961	234	115	101	96	101	107	105	105	111	118
SNM and Bernstein	361	697	534	817	966	1751	1845	3408	4614	7716	12379	16529	19472
SNM and Lagrange	3969	3122	2511	1805	994	752	434	1199	793	1067	1526	2107	3218
SNM and power series	23619	16386	15496	17719	17093	19707	21379	24390	25787	22520	25590	27760	28650
SHJ and Chebyshev	487	619	799	861	1098	709	1127	774	822	878	965	1120	1306

Table 5-6. Standard Deviation of Trajectory Evaluations

Combination	Degree												
	4	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
SNM and Chebyshev	30	8	4	0	0	0	0	0	0	0	0	0	0
SNM and Bernstein	0	2	6	2	2	20	16	43	100	445	2468	4840	7145
SNM and Lagrange	67	43	26	14	4	2	0	2	0	0	0	0	0
SNM and power series	1787	1419	955	1216	1106	1458	1895	2604	3812	16588	25711	30568	52984
SHJ and Chebyshev	1203	911	1248	1187	506	112	144	41	47	100	57	73	79

Table 5-7. Unsuccessful Cases per 100,000 Trajectories

Confidence intervals for each mean value in Table 5-5 were calculated using a 90% confidence level; the large sample size (49192) led to the half-width of the confidence interval of each population mean being ~0.007× the corresponding sample standard deviation. The confidence intervals for the NLP/parameterization form combinations were non-overlapping, except for the SNM/Chebyshev combination when $d_v \in \{5,6,7\}$, the SNM/power combination when $d_v \in \{11,14\}$, and the $d_v = 11$ values for SNM/Lagrange and SNM/Bernstein.

The Chebyshev, Bernstein, and Lagrange forms, and the SHJ-Chebyshev combination, all produced similar optimality results, with mean and standard deviation at $d_v = 10$ of approximately 2% and 4%, and decreasing with higher d_v. The power series form performance was worse than the other forms: for $d_v \leq 9$ mean optimality was approximately 1-4% worse than that of the other forms, and the standard deviation was larger; overall the power series form lost of the order of twice as much optimality as any of the other forms.

The Chebyshev form with SNM resulted in the lowest mean σ, which was low for all d_v with the slowest ($d_v = 16$) only 67% slower than the fastest ($d_v = 5$). The optimization was robust: even the worst case ($d_v = 4$) was unsuccessful in only 0.03% of cases. The standard deviation of σ was also low except for $d_v \leq 6$. The combination performed well on all measures compared to the other combinations.

The Bernstein form was, on average, the next fastest of the four SNM combinations for $d_v \leq 11$, and mean plus standard deviation of σ (which covers approximately 85% of the distribution) was lower for the Bernstein form than for the Chebyshev form for $d_v \leq 6$. The mean robustness (≔ success rate) of the Bernstein form was higher than that for the Chebyshev form over $d_v \in [4,8]$, although the robustness of both forms was greater than 99.97% over that range. However, the Bernstein form became much slower than the Chebyshev form as d_v increased above 6, being at least an order of magnitude slower for $d_v \geq 9$, and its robustness reduced rapidly as d_v increased above 10.

The Lagrange form was slower than both Bernstein and Chebyshev forms up to approximately $d_v = 9$. Although it was not as fast or robust as the Chebyshev form for

any d_v, like the Chebyshev form its robustness increased as d_v increased, and for $d_v \geq 9$ it was successful on all but 0.002% of cases, comparable with the Chebyshev form.

The power series form was slow for all d_v: its mean σ was approximately an order of magnitude slower than for the Bernstein form for $d_v \leq 12$ and still approximately a factor of 3 slower at $d_v = 16$. The need to raise the absolute objective limit for the Nelder-Mead initial simplex and reduce the initial step size (Section 5.3.2) is attributed to ill-conditioning. Even after these changes its best robustness was approximately 99% at $d_v = 6$, but was below 90% for $d_v \geq 13$.

With the SNM algorithm the Chebyshev and the Lagrange forms showed clear minima, for $d_v \in [8,10]$, in the mean plus standard deviation results; these minima were not present in the mean data. The corresponding Bernstein values showed a minimum at $d_v = 6$, but the value was close to a best fit curve; the power series data shows a minimum for $d_v \in [5,6]$. The SHJ results did not exhibit these minima, although the data around $d_v \in [8,10]$ do not appear to fit on a smooth curve through the other data points.

For the SHJ algorithm, σ was, as expected, more dependent on N than the SNM algorithm was, and SHJ was less robust than SNM over the whole d_v range, although it still achieved better than 98.5% robustness.

5.4 Discussion

Direct evaluation of maximum feasible airspeed can be accomplished but was found to be approximately 3 times slower than the best tested NLP/parameterization combination. However, the initial guess is critical to optimization performance and optimization was set up with initial guesses close to the known solutions; in practical use this is a reasonable (but perhaps optimistic) assumption since the previous solution may be a good approximation to the next solution, subject to the comments in Section 3.4.8.

The accuracy of parameterization as an approximation to v_{max} is low at low d_v, and there are fewer degrees of freedom for the NLP algorithm to adjust which limits achievable

optimality and affects computational speed. At high d_v the accuracy of the approximation is high so the achievable optimality improves, but the NLP dimension is also high: how these two factors affect convergence at high d_v is dependent on how the optimization parameters (the coefficients) control the shape of the approximation, i.e. how the next iterate depends on the coefficients and how these are used by the NLP algorithm. Hence both the form of airspeed parameterization and the choice of NLP algorithm affect robustness and computational speed.

For the Chebyshev form the coefficients have global influence but the high-degree terms have small coefficients, which helps reduce ill-conditioning at high d_v and enables the Chebyshev form to be robust and relatively fast. The Bernstein form has global coefficients but their influence is concentrated in neighbourhoods, and they form a convex hull of the curve. Despite its numerical stability as a polynomial basis, the Bernstein form lost robustness as d_v increased which may be due to ill-conditioning with respect to τ_f at high d_v, due to the Van Der Monde matrix implicit in the relationship between the coefficients and the values of the curve at the $d_v + 1$ control points.

The Lagrange coefficients do not have the diminishing property of the Chebyshev coefficients, but they are values of the curve at $d_v + 1$ points and are therefore more strongly localized than the Bernstein coefficients: the effect of each coefficient is concentrated in a small neighbourhood of the curve which may have contributed to the robustness at high d_v and the relative speed compared to the Bernstein form (slower at low d_v, faster at high d_v).

The degree and form of the airspeed parameterization clearly affected the convergence of the optimization, but the effects were dependent on the NLP algorithm. With the SNM algorithm, measured by mean plus standard deviation of σ the Bernstein form was the fastest and most robust for $d_v \leq 6$, but for higher d_v the Chebyshev form was the fastest and most robust. As the degree increased the Chebyshev form maintained its computational speed while the Bernstein form slowed, such that for $d_v \geq 9$ the Bernstein form was more than an order of magnitude slower than the Chebyshev form. Measured by mean trajectory evaluations, the Chebyshev form was the fastest, in some cases by an order of magnitude, for all d_v.

Both the Chebyshev and Bernstein forms were at least 99.97% successful for $d_v \in [4,8]$, but the Bernstein form is not suitable for use above degree 8 due to its poor robustness. The barycentric Lagrange form was approximately as robust as the Chebyshev form for $d_v \geq 9$, but it was not sufficiently robust for low degrees, and was slower than one or both of the Chebyshev or Bernstein forms for all d_v.

The power series form was slower, less optimal and less robust than the other three forms, on all measures.

Except for the power series form, the form of the parameterization did not significantly affect the loss of optimality, the range $d_v \in [8,10]$ resulting in about 2-5% loss of optimality together with low mean plus standard deviation of trajectory evaluations and high robustness. If ~3% mean loss of optimality due to the airspeed parameterization is tolerated, then there is no need to use polynomials of degree > 8 in order to obtain a good balance between optimality, robustness, and computational speed, provided that the power series form is not chosen. The choice should be either the Chebyshev or Bernstein forms depending on which measure is most important to the user.

In this chapter the spatial path was excluded from the optimization in order to isolate the effects of airspeed parameterization: in operation both spatial and airspeed parameters would normally be included in the optimization vector. The increase in degrees of freedom with spatial parameterization included in the optimization vector should reduce the time spent by the NLP algorithm in attempting to optimize infeasible trajectories, and is therefore likely to reduce the most advantageous value of d_v. Chapter 6 describes research into the performance of combined spatial and airspeed optimization.

6 OPTIMIZATION

6.1 Introduction

The overall performance of the inverse dynamics method may be measured by the optimality of its solutions and the robustness and computational speed with which the solutions are generated: these are dependent on the formulation of the problem and the performance of the NLP algorithm. In Chapter 5 it was found that for the minimum-time problem, using the SNM algorithm with spatial parameterization excluded from the optimization, optimality, robustness, and computational speed were dependent on the degree of the airspeed parameterization. In the work described in this chapter the minimum-time problem was formulated to include both spatial and airspeed optimization, and the performance of the method was investigated with each of the four NLP algorithms selected in Chapter 2.

The objectives of the work described in this chapter were to:

1. Confirm that achievable optimality is dependent on d_v and improves as d_v increases, when spatial parameterization is included in the optimization vector.
2. Investigate whether the theoretical optimality advantages of SNOPT over the derivative-free SNM and SHJ algorithms (Section 2.5) are achieved in practice with the inverse dynamics method, and compare SNOPT optimality with DE optimality.
3. Investigate the effects of the choice of NLP algorithm on robustness and computational speed, especially the performance of DE because of the lack of previous empirical results for DE with the inverse dynamics method and the absence of convergence proofs.
4. Measure achieved computation times to assess whether the inverse dynamics method would be computationally fast enough for near-real-time application if used with the tested NLP algorithms on the test hardware and software.
5. Identify candidate optimization approaches for near-real-time application.

6.2 Method

The input test space was:

- The four NLP algorithms DE, SNM, SHJ, and SNOPT.
- Airspeed parameterization by Bernstein form polynomials with $d_v \in \{3,...8\}$.

Bernstein form polynomials were selected for the airspeed parameterization because of their robustness and computational speed compared to Chebyshev, barycentric Lagrange and power series forms, and their suitability for Hermite interpolation at the boundary points. The set {3,...8} for the degree of airspeed parameterization was selected because:

- A cubic removes airspeed from the optimization vector and is the lowest degree polynomial which can be guaranteed to satisfy boundary conditions on airspeed and tangential acceleration.
- The findings of Chapter 5 indicate that it is not necessary to use higher than degree 8 for airspeed parameterization.

The NLP problem was defined as minimizing the flight time (t_f), subject to the constraint set defined by Eq. (4.68). Optimality was measured by:

- $\alpha_0 \coloneqq$ the percentage of all test cases for which $\eta < \eta_{max}$ and t_f was the lowest of the t_f values produced by any of the NLP algorithms for that test case
- $\alpha_2 \coloneqq$ the percentage of all test cases for which the algorithm achieved within 2% of the corresponding lowest t_f (subject to $\eta < \eta_{max}$)
- $\alpha_5 \coloneqq$ the percentage of all test cases for which the algorithm achieved within 5% of the corresponding lowest t_f (subject to $\eta < \eta_{max}$).

These are all relative measures: they quantify the optimality of each NLP algorithm and each parameterization degree relative to the other algorithms and degrees, not against true optimality.

Robustness was measured by the ratio (β) of successful test cases to total test cases. Success was defined as satisfying the NLP termination criteria without triggering

termination by the maximum trajectory evaluations limit (σ_{max}) specified for each algorithm, and satisfying $\eta < \eta_{max}$.

The computational speed was measured by the number (σ) of trajectory evaluations invoked by the NLP algorithm. For SNOPT σ included all trajectory evaluations that SNOPT invoked to evaluate gradient approximations. The computational loads of the NLP algorithms themselves are discussed in Section 6.3.4.1 below.

A database of pairs of pseudo-random boundary conditions ("test cases") was created; simple checks were applied to reduce the number of infeasible test cases in the database.

Initial guess expressions were created to be used by SNM, SHJ, and SNOPT. DE was used to guide the choice of an initial guess for τ_f. Subsets of the test database were used to guide the choice of values for the NLP algorithm settings.

A series of experiments was carried out in which a combination of an NLP algorithm with each degree of airspeed parameterization, together with an initial guess expression when required, was applied to 1000 test cases. Optimality, robustness, trajectory evaluation counts, and average computational times were recorded.

To assist the analysis of the results the performance profile graphs introduced by Dolan and Moré[27, 28] were used to analyze the t_f and σ results, based on the Matlab implementation described by Higham[62]. The profiles summarize the relative performance of each test combination by plotting, against ϕ as a percentage, the cumulative probability that the result (t_f or σ) produced by a combination was less than or equal to ϕ times the best (i.e. least and successful) result obtained for the same test case from any of the combinations for which the results were plotted. Since the profiles compare each combination and test case against the other combinations for the same test case, the profiles show only relative performance: addition, removal or changes to the results of any combination changes the displayed values for all the combinations in the experiment. Tabular results are also given.

6.2.1 Hardware and Software Environment

The hardware and packaged software environment used in the experiments described in this Chapter was the same as that described in Section 3.3. Controls were evaluated using Eq. (4.47) rather than Eq. (4.64), for computational speed.

6.2.2 Aircraft Data

The aircraft data were: M = 11 kg, $l_{struct} = 3.8g$, $0 \leq T \leq 25$ N, v_s = 15 m/s, v_{ne} = 40 m/s, p_{max} = 200 °/s.

6.2.3 Test Database and Settings

A database of 2000 unique pairs of boundary conditions was created with pseudo-random values that satisfied the following bounds $\forall i \in \{[1,2000] \cap \mathbb{Z}\}$

$$x_0 = y_0 = z_0 = 0 \tag{6.1}$$

$$x_f \in \begin{cases} [-10000, 10000], & \text{if } i \leq 1000 \\ [-5000, 5000], & \text{otherwise} \end{cases} \tag{6.2}$$

$$y_f \in \begin{cases} [-10000, 10000], & \text{if } i \leq 1000 \\ [-5000, 5000], & \text{otherwise} \end{cases} \tag{6.3}$$

$$z_f \in \begin{cases} [-1000, 1000], & \text{if } i \leq 1000 \\ [-500, 500], & \text{otherwise} \end{cases} \tag{6.4}$$

$$v_{0,f} \in [1.05v_s, 0.95v_{ne}] \tag{6.5}$$

$$l_{z\,0,f} \in [0.5g, 2.1g] \tag{6.6}$$

$$\xi_{0,f} \in [0, 2\pi] \tag{6.7}$$

$$\gamma_{0,f} \in [-10, 10]\left(\pi/180\right) \tag{6.8}$$

$$\mu_{0,f} \in [-60, 60]\left(\pi/180\right) \tag{6.9}$$

The following conditions were also applied and any boundary value pair that violated any condition was pseudo-randomly re-generated.

$$\dot{v}_{0,f} \in \left[1.5(v_s - v_{0,f}), 1.5(v_{ne} - v_{0,f})\right] \tag{6.10}$$

$$\frac{2\max\left(\left|x_f\right| + \left|y_f\right| + \left|z_f\right|, \dfrac{\left|z_f\right|}{\sin 7°}\right)}{v_0 + v_f} \geq 10 \tag{6.11}$$

$$T_{0,f} \in \left[T_{\min}, T_{\max}\right] \tag{6.12}$$

$$l_{z\,0,f} \in \left[0, l^+\right] \tag{6.13}$$

The two ranges of final position in Eqs. (6.2)-(6.4) were used to spread the path length over a larger range than would have been obtained if only one range had been used. Eq. (6.10) was used to exclude boundary accelerations that would be likely to be infeasible for all optimization vectors; the generated range of $\dot{v}$ was observed to be (-3.49, 3.25). Eq. (6.11) imposed an approximate minimum flight time of 10 s, the 7° value being approximately the maximum flight path angle that the aircraft could sustain at maximum thrust. Eq. (6.12) ensured that the thrust required at the boundary points satisfied the aircraft limits. No duplicate boundary condition pairs were allowed, hence each test case was unique in the database.

Except where stated a subset of 1000 test cases, $i \in \{501,...1500\}$, was used together with Chebyshev-Gauss-Lobatto node distribution, $N = 210$, and local quadratic interpolation for constraint evaluation (from the results in Section 3.5.2). Clenshaw-Curtis quadrature was used to evaluate t_f.

6.2.4 Optimization Vector

Degree 7 power series polynomials were used for the spatial parameterization, with the third derivatives at both boundary points included in the optimization vector $\boldsymbol{\chi}$ in order to exploit the maximum available degrees of freedom.

The first two and the last two coefficients C_0, C_1, $C_{dv\text{-}1}$, and C_{dv} of the Bernstein form airspeed parameterization were derived directly from the boundary conditions by

$$C_0 = v_0, \quad C_1 = v_0 + \frac{\dot{v}_0}{d_v}, \quad C_{d_v-1} = v_f - \frac{\dot{v}_f}{d_v}, \quad C_{d_v} = v_f \tag{6.14}$$

This construction ensured that the airspeed parameterization satisfied the boundary conditions without any need for airspeed constraint checking at the first and the last nodes. When $d_v = 3$ airspeed was determined entirely by the boundary conditions and there was no airspeed optimization; for $d_v \geq 4$ the remaining airspeed coefficients were included in $\boldsymbol{\chi}$.

Hence the optimization vector $\boldsymbol{\chi} \in \mathbb{R}^D$ was (omitting transposition operators for clarity)

$$\boldsymbol{\chi} = \begin{cases} \left(\tau_f, \mathbf{r}_0''', \mathbf{r}_f'''\right), & \text{if } d_v = 3 \\ \left(\tau_f, \mathbf{r}_0''', \mathbf{r}_f''', C_2, \ldots C_{d_v-2}\right), & \text{if } d_v \in \{4, \ldots 8\} \end{cases} \tag{6.15}$$

and D (the NLP dimension) $\in [7,12]$ accordingly.

6.2.5 Initial Guess

In Chapter 3 it was found that the inverse dynamics method produced multimodal NLP problems. Hence for gradient or derivative-free NLP algorithms successful convergence is critically dependent on the initial guess being in a suitable basin of attraction to a feasible local minimum, and optimality is dependent on the closeness of the initial guess to a feasible global minimum.

It was shown in Section 3.4.3 that an initial guess of τ_f greater than but close to 2 would keep the ill-conditioning of the spatial interpolation close to its minimum. In order to

investigate suitable initial guesses for τ_f, DE was run against the test database with N = 210, uniformly-spaced nodes, and linear constraint interpolation. Figure 6-1 shows the resulting pseudo-optimal values of τ_f. The tests were also run with Chebyshev-Gauss-Lobatto node distribution and with quadratic constraint interpolation: consistent results were obtained and τ_f = 10 was chosen for the initial guess because it was near, but within, the lower limit of the range of pseudo-optimal values.

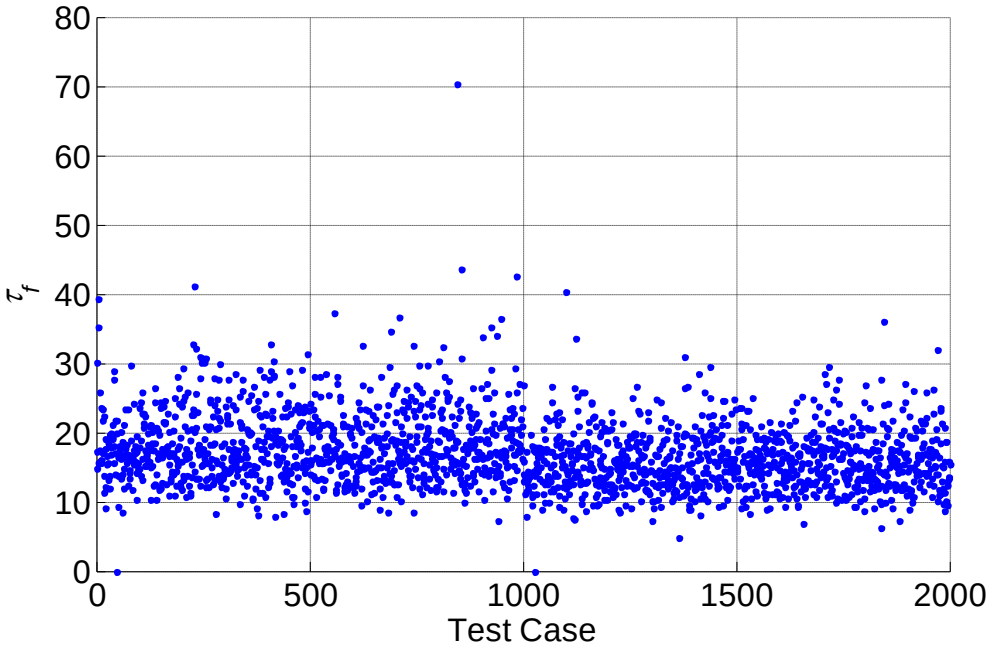

Figure 6-1. Pseudo-Optimal Values of τ_f for 2000 Test Cases

For the 6 spatial coefficients the simple and obvious initial guess $\mathbf{r}_0''' = \mathbf{r}_f''' = (0,0,0)^T$ was used.

Eq. (6.14) obviates any need for an initial guess of the airspeed coefficients for d_v = 3. An obvious initial guess for d_v > 3 that satisfies the boundary conditions is to elevate to the required degree the cubic given by substituting d_v = 3 into Eq. (6.14); this is readily carried out using standard relationships[46]. This approach was used for all experiments.

6.2.6 NLP Algorithm Settings

The DE, SNM, and SHJ algorithms were implemented from the references given in Section 2.5. The SNOPT version 7.2 package was run as a *mexw64* file under an academic licence.

DE was implemented as a Matlab script and was called directly by a top-level Matlab script; the Nelder-Mead and Hooke-Jeeves algorithms were implemented as Matlab functions and called from an intermediate script. SNOPT was called as a *mexw64* function from an intermediate script that was adapted from the Matlab script provided with SNOPT. The trajectory evaluation algorithm was implemented as a Matlab function, with no sub-functions except a private quadratic interpolation function. It was adapted to evaluate the penalty function, if required, and to match the objective and constraint formats required by the NLP algorithms, but was otherwise identical for each NLP algorithm.

Settings for each of the four NLP algorithms were chosen to balance two objectives: to allow valid comparisons between the algorithms, and to allow each algorithm to obtain results representative of the potential of each algorithm. Since finding an optimal solution to this problem is itself a multi-objective high-dimension optimization problem, the settings were chosen heuristically, guided by small numbers of experiments.

6.2.6.1 DE

Price et al.[110] described four major versions of DE, and many variations. The version used in this work was DE/Rand/1/Bin with upper and lower bounds on elements of χ and Lampinen's constraint handling[85] (Section 2.5.3). The initial pseudo-random population was generated with a uniform distribution; subsequently the target, base, and pairs of mutation vectors were all distinct and obtained by pseudo-random permutation of the population (Matlab's *randperm* function, with offsets of 1, 3 and 7 from the target for the base and mutation vectors). The scale factor F and the crossover probability C_r were set to 0.9 after a small number of tests. Although Price suggests that the population N_p should be up to $5D$, it was found that with $N_p = 15$ DE was robust and the

solutions were closer to optimal than those achieved with the other NLP algorithms for the tested range of $D \in [7,12]$; i.e. it was not necessary to use higher values of N_p.

DE does not require an initial guess, but bounds are required by Matlab's random number generator *rand* which was used to populate the first generation. The initialization bounds were

$$\tau_f \in \left[3, \left(\frac{6}{v_0 + v_f} \right) \max\left(\left| x_f \right| + \left| y_f \right| + \left| z_f \right|, \frac{\left| z_f \right|}{\sin 7^\circ} \right) \right] \tag{6.16}$$

$$\mathbf{r}'''_{0,f} \in [-10,10] \tag{6.17}$$

$$C_i \in [0, 2v_{ne}], \quad \forall i \in \{2, \ldots d_v - 2\} \tag{6.18}$$

The expression in the maximum function in Eq. (6.16) is three times the expression in Eq. (6.11).

Only a single bound was applied after the first generation: $\tau_f \geq 3$ to ensure that no negative final times could be generated. Hence DE was not limited only to solutions within the initialization bounds, although it may be slow to find solutions significantly outside those bounds.

In principle, DE does not require any constraint violation tolerance, but to avoid round-off error affecting feasibility a tolerance of 2^{-42} ($1024 \times eps$) was incorporated in each constraint.

Termination criteria for DE, SNM, and SHJ were chosen to ensure that the solution was close to optimality by requiring that the difference between defined objective (t_f) values, and the difference between the corresponding $\boldsymbol{\chi}$ vectors, were both within tolerances Δ_f and Δ_χ respectively. For DE, the termination criterion was

$$\left(\sigma_g \geq \sigma_{\max} / N_p \right) \cup \left(\left(N_F = N_p \right) \cap \left(f_w - f_b \leq \Delta_f \right) \cap \left(\left\| \boldsymbol{\chi}_w - \boldsymbol{\chi}_b \right\|_\infty \leq \Delta_\chi \right) \right) \tag{6.19}$$

where f_w is the worst (highest) objective value in the current generation, f_b is the best (lowest) objective value, $\boldsymbol{\chi}_w$ and $\boldsymbol{\chi}_b$ are the corresponding optimization vectors, and N_F is the number of members of the latest generation that are feasible. The first term implemented an overriding limit on σ, which was expressed as σ_{max}/N_p generations because the termination criteria were only applied at the completion of each generation. The term containing N_F required that all members of the population were feasible.

The σ_g count (but not the overall σ count) was reset to 1 at the generation which generated the first feasible solution. This was tried because during the settings testing it appeared that once it had one feasible solution, DE rapidly found feasible solutions for the remaining N_p - 1 population members, and that this reset would therefore significantly improve robustness with only a small increase in σ. However, it was found that the effect was small: it added one success to each of $d_v \in \{3,...6\}$, two to $d_v = 7$, and zero to $d_v = 8$. The results below include these successes.

The settings used for DE are shown in Table 6-1.

Variable	Description	DE Value
σ_{max}	Maximum trajectory evaluations	45000
Δ_f	Maximum spread of objective values over population	0.1
Δ_χ	Maximum ∞-norm of $\boldsymbol{\chi}_w$-$\boldsymbol{\chi}_b$	1
F	Mutation scale factor	0.9
C_r	Crossover probability	0.9
N_p	Population size	15

Table 6-1. DE Settings

6.2.6.2 SNM and SHJ

For SNM and SHJ Eqs. (4.70)-(4.72) were used for the penalized objective, constraints and squared two-norm penalty function. Unity penalty weights (k) were used (Section 3.4.10.5).

The sequential outer loop used with the Nelder-Mead and Hooke-Jeeves algorithms was based on that described by Griffin and Kolda[56], slightly simplified because use of the squared two-norm penalty function obviated the need for a smoothing parameter. The outer loop (excluding housekeeping operations) was

Until Finished
    Hooke-Jeeves (or Nelder-Mead)
    if ($\eta < \eta_{max}$) $\cap$ ($\Delta_f \leq \Delta_{f\,thresh}$)
        Finished = true, successful
    else if (inner loop failed) $\cup$ ($\sigma > \sigma_{max}$)
        Finished = true, unsuccessful
    else
        $\Delta_f \leftarrow \max(\Delta_f / \zeta_f, \Delta_{f\,min})$
        if $\eta \geq \eta_{max}$
            $\rho \leftarrow \min(\rho\zeta_p, \rho_{max})$
            stagnation check
        end
    end
end

The value of Δ_f used by the Nelder-Mead and Hooke-Jeeves algorithms was initialized to $\Delta_{f\,0}$, and at each outer iteration it was divided by a factor ζ_f subject to a lower limit of $\Delta_{f\,min}$. Outer loop successful termination was inhibited until Δ_f reached a threshold value $\Delta_{f\,thresh}$ (unless σ_{max} was reached). The penalty parameter ρ was initialized to ρ_0 and at each outer iteration it was multiplied by a factor ζ_p subject to an upper bound of ρ_{max}.

The stagnation check was included to cause the outer loop to exit if ρ and η were unchanged compared to the previous loop, and the number of trajectory evaluations in the last two inner loops were equal. This check was added after it was found that the Nelder-Mead algorithm could reach a stagnation condition in which it looped without changing the simplex under some conditions.

The *fminsearch* function in Matlab (as described in the Matlab software help file) is an implementation of the Nelder-Mead algorithm based on Lagarias et al.[84]. To form the initial simplex it perturbs each element of the initial guess by a factor of 5%, or by 2.5×10^{-4} if the element is zero. It was found that a specified initial step length ($\boldsymbol{\chi}_{\Delta 0}$) in each dimension was more effective for the SNM algorithm. The same $\boldsymbol{\chi}_{\Delta 0}$ was used to set the initial step length for the Nelder-Mead and Hooke-Jeeves algorithms.

The termination criteria for the Nelder-Mead algorithm used tests on the objective and $\boldsymbol{\chi}$ values in the simplex, analogous to Eq. (6.19)

$$\left(\sigma \geq \sigma_{\max}\right) \cup \left(\left(\left| f_{\max} - f_{\min} \right| \leq \Delta_f \right) \cap \left(\left\| \boldsymbol{\chi}_{\max} - \boldsymbol{\chi}_{\min} \right\|_{\infty} \leq \Delta_{\chi} \right)\right) \tag{6.20}$$

A maximum absolute value of the objectives in the initial simplex was also imposed to prevent divergence to $\pm\infty$.

The core Hooke-Jeeves algorithm was terminated when a base point exploration failed, the step size had been reduced h_{max} times, and the most recent base and explore point objective values were within Δ_f of each other

$$\left(\sigma \geq \sigma_{\max}\right) \cup \left(\left(f_{base} - f_{\exp} \leq \Delta_f \right) \cap b \cap \left(h \geq h_{\max}\right)\right) \tag{6.21}$$

Upper and lower bounds on $\boldsymbol{\chi}$ are readily applied in the Hooke-Jeeves algorithm: only $\tau_f \geq 3$ was used, matching DE. The Nelder-Mead algorithm is not so well suited to maintaining parameter bounds, hence a limit of 10^{10} on the absolute function value was imposed instead.

The Hooke-Jeeves algorithm cached the most recent $4D$ trajectory evaluation results to reduce duplicated trajectory evaluations (Kelley[75]). Over a small number of setup tests this was observed to reduce σ by 5-10%.

The settings used for the SNM and SHJ algorithms are shown in Table 6-2. The core Nelder-Mead algorithm also requires a number of other settings. In this work Price's parameters τ and h were set to 10^{-15} and 1 respectively and all other settings were as specified in Lagarias et al.[84] and Price et al.[109].

The settings in Tables 6-1 and 6-2 and the associated termination criteria were chosen after a small number of tests. It is probable that improvements in the performance of each NLP algorithm could be obtained by modifying the settings and criteria. For example the SHJ termination criteria Eq. (6.21) includes a limit on the number of step size reductions as the means of terminating when the final step size is sufficiently small: 0.002 from the settings in Table 6-2. If h_{max} was increased, SHJ optimality (which was the worst of the four algorithms, as described in Section 6.3.3 below) may be improved at the expense of computational speed and/or robustness, although the SHJ condition is already more onerous than that of SNM.

Variable	Description	Value
σ_{max}	Maximum trajectory evaluations	45000
η_{max}	Maximum constraint violation	0.1
ρ_0	Initial penalty parameter	1
ρ_{max}	Maximum penalty parameter	2^{28}
$\Delta_{f\,0}$	Initial objective range	1
$\Delta_{f\,thresh}$	Objective threshold	1
$\Delta_{f\,min}$	Minimum objective range	0.01
Δ_{χ}	Maximum ∞-norm of χ_w-χ_b (Nelder-Mead)	0.5
h_{max}	Maximum step reductions (Hooke-Jeeves)	3
$\chi_{\Delta\,0}$	Initial step vector	(2,...)
ζ_f	Δ_f reduction factor	2
ζ_p	ρ multiplication factor	2
ζ_h	h reduction factor (Hooke-Jeeves)	10
	Maximum absolute objective (Nelder-Mead)	10^{10}

Table 6-2. SNM and SHJ Settings

6.2.6.3 SNOPT

For SNOPT "Derivative Option 0" was set because analytic derivatives were not available. The maximum permitted constraint violation was imposed by setting the upper bound for each constraint equal to η_{max}. It was found that, for any value of $\eta_{max} \in \{-0.1,0,0.05,0.1\}$, SNOPT was less than 6% successful if success was defined as

$$\eta \leq \eta_{\max} \tag{6.22}$$

but that if success was instead defined (for the purpose of analysis) as

$$\eta \leq \eta_{\max} + \eta_{tol} \tag{6.23}$$

with η_{tol} = 0.1 in Eq. (6.23), then success rates of the order of 60-70% were obtained. Hence η_{max} = 0 was specified in the input to SNOPT, and η_{tol} = 0.1 was applied during data analysis to make the effective SNOPT constraint violation tolerance equal to the SNM and SHJ constraint violation tolerance.

SNOPT requires bounds on the optimization variables: using $\pm\infty$ was not effective, the DE values in Eqs. (6.16)-(6.18) were tried but better results were obtained with the following bounds, which were therefore used in the comparative experiments

$$\tau_f \in [3, \infty] \tag{6.24}$$

$$\mathbf{r}'''_{0,f} \in [-2000, 2000] \tag{6.25}$$

$$C_i \in [-200, 200], \quad \forall i \in \{2, \ldots d_v - 2\} \tag{6.26}$$

Default SNOPT values were used for all other settings including iteration count limits.

6.3 Results and Analysis

6.3.1 Robustness

Table 6-3, and Figures 6-2 and 6-3, show the percentages of test cases for which each NLP algorithm succeeded (β) for each d_v.

NLP	Degree of Airspeed Parameterization					
	3	**4**	**5**	**6**	**7**	**8**
DE	93.9	99.1	99.7	99.7	99.9	99.8
SNM	82.3	91.8	94.2	94.9	95.1	95.7
SHJ	80.7	91.0	92.9	94.4	95.4	95.7
SNOPT	65.7	68.1	67.6	66.0	64.2	65.7

Table 6-3. Percentage Success Rates

Confidence intervals for each proportion in Table 6-3 were calculated using a 90% confidence level. With airspeed included in the optimization, for DE the half-width was less than 0.5%, for SNM and SHJ it was less than 1.5%, and for SNOPT it was approximately 2.5%.

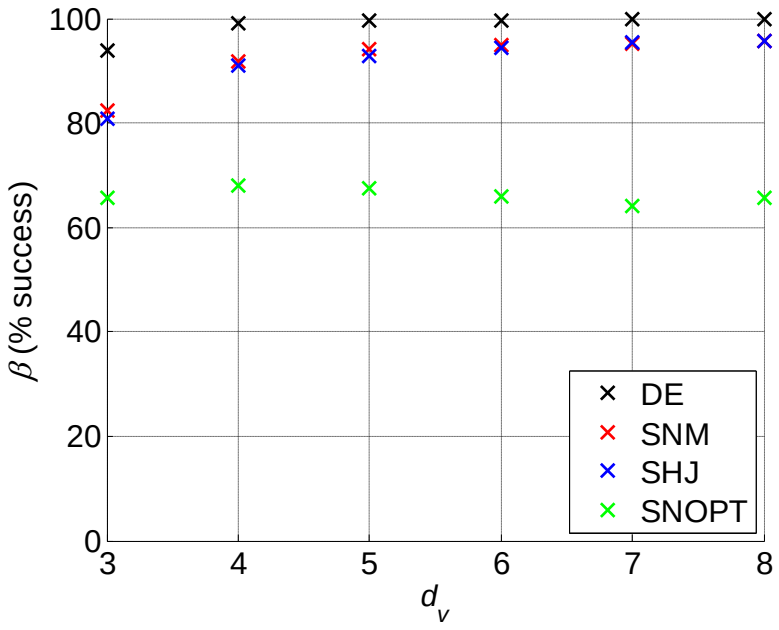

Figure 6-2. Percentage Success Rates vs d_v

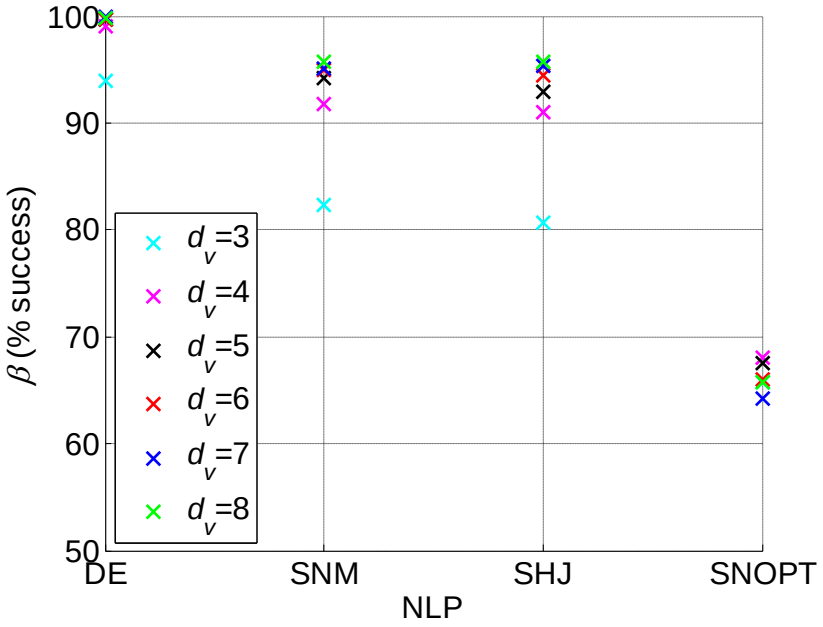

Figure 6-3. Percentage Success Rates vs NLP

DE, SNM, and SHJ showed reduced robustness when airspeed was excluded from the optimization even though this had the lowest NLP dimension. Although all the test cases in the database were solved by DE for $d_v \geq 5$ and therefore had feasible solutions, at zero constraint tolerance 4 of the test cases were not solved by any of the four algorithms for $d_v = 4$, and 43 of the test cases were not solved by any of the four algorithms for $d_v = 3$, despite multiple retries. Therefore including airspeed in the optimization enabled DE, SNM, and SHJ to converge successfully for a wider range of boundary conditions. SNOPT's mean robustness varied by less than 4% over $d_v \in \{3,...8\}$, which is within the 90% confidence interval: hence the data do not support the same conclusion for SNOPT.

For $d_v \geq 4$ the results were more sensitive to the choice of NLP algorithm than to d_v (and therefore to the NLP dimension $D = d_v + 4$). DE's robustness peaked at $d_v = 7$, although it was within the confidence interval for $d_v \geq 4$ so the data only weakly support the presence of a peak. The robustness of both SNM and SHJ increased with d_v, but by less than 5% over $d_v \in \{4,...8\}$. Hence the inclusion or exclusion of airspeed from the

optimization had a more significant effect on robustness than the variation of NLP dimension for DE, SNM, and SHJ.

SNOPT was less robust than the other three algorithms, reaching only 66-68%, whereas for $d_v \geq 4$ SNM and SHJ achieved 91-96% and DE achieved over 99% successful test cases.

6.3.2 The Effects of d_v on Optimality

Figure 6-4 shows the relative optimality of each of the four NLP algorithms across the range of $d_v \in \{3,...8\}$. It is clear from the spacing of the curves that for each of the four algorithms optimality improved with increasing d_v.

The relatively poor robustness of SNOPT limited the number of test cases for which it was possible for SNOPT to be the closest to optimal (i.e. it places an upper bound on α_0 measured over all test cases). To remove the limiting effects of robustness the data of Figure 6-4 was divided by β and re-plotted as Figure 6-5.

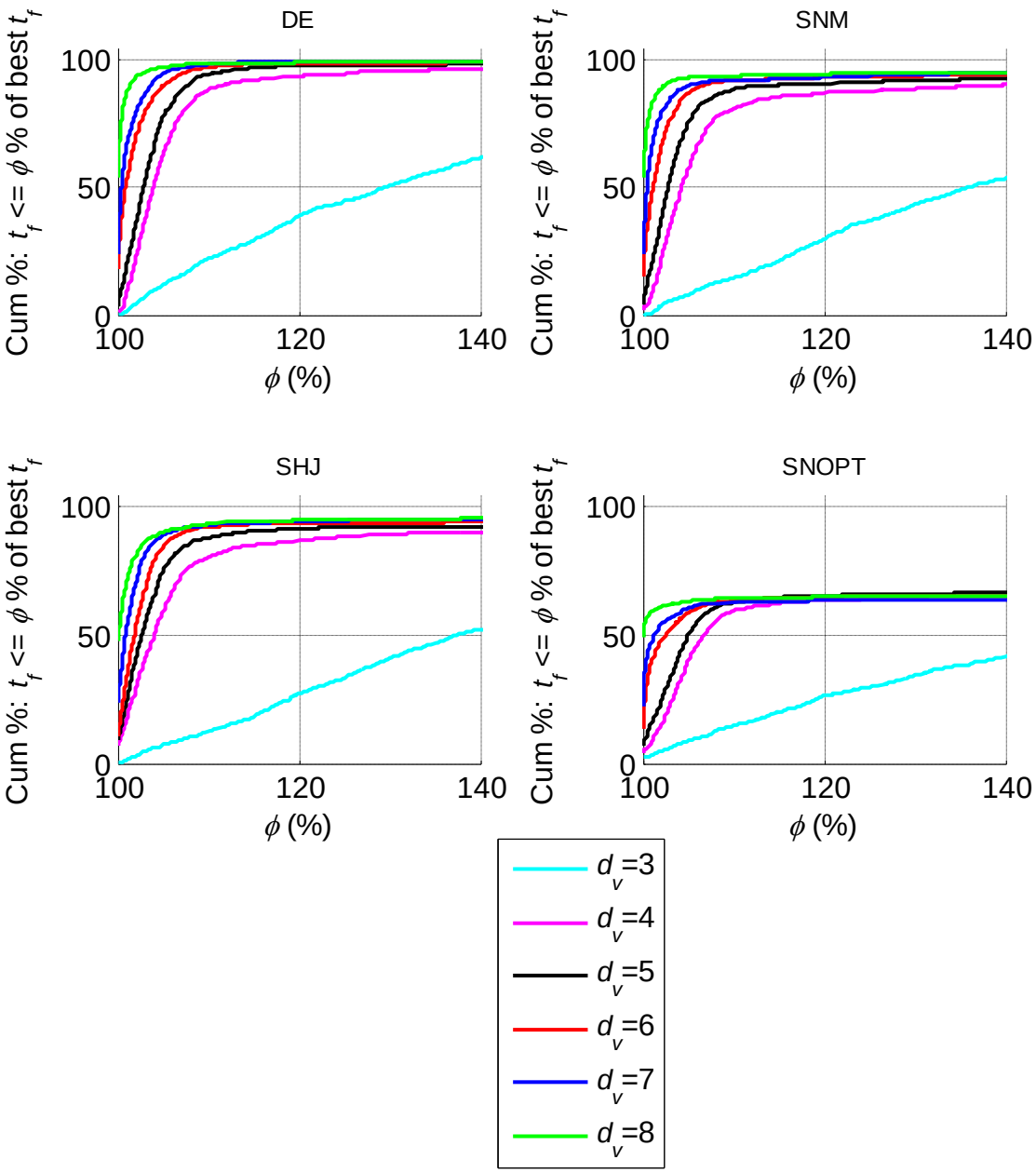

Figure 6-4. Optimality Profiles for Each NLP, vs d_v

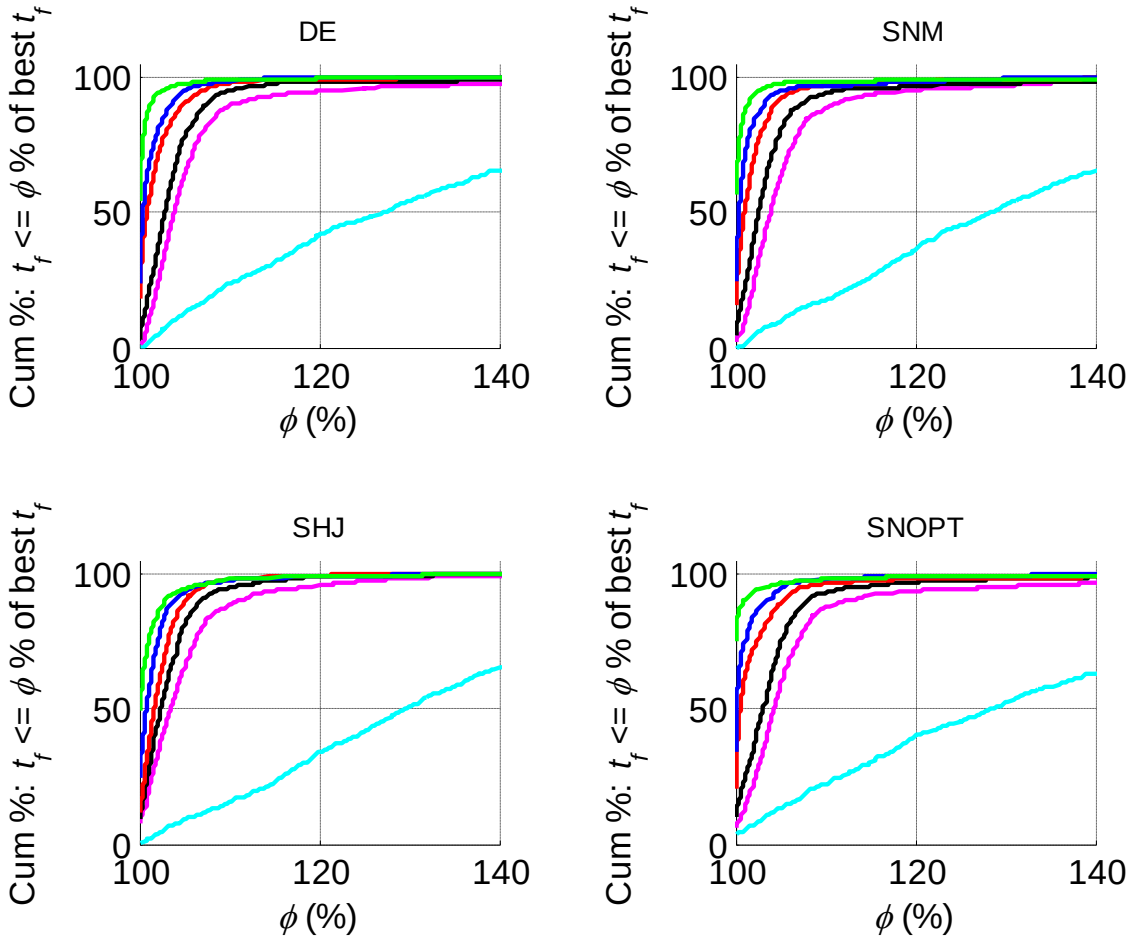

Figure 6-5. Scaled Optimality Profiles for Each NLP, vs d_v
(Using the same colour coding as Figure 6-4)

From Figure 6-5 it can be seen that a tolerance of approximately 28% loss of optimality would be required to cover 50% of the successful test cases for $d_v = 3$, whereas with airspeed included in the optimization only 4% tolerance would be required to cover 50% of successful test cases. Hence not only was α_0 better when airspeed was included in the optimization but the dispersion of optimality was also better (i.e. more results were within a given tolerance of the lowest t_f).

Parenthetically, Figure 6-5 appears to show that optimality was approximately a linear function of tolerance for $d_v = 3$. This is an artefact of the ϕ-axis scales: if the graphs had

been displayed with the ϕ-axis extended out to ϕ = 3000% then the d_v = 3 curves would be seen to have the same pattern as the other curves.

To relate these results to the results presented in Chapter 5 for the Bernstein form Table 6-4 shows, for SNM, the percentage loss of relative optimality that would have to be tolerated to bring 50% and 68% respectively of the successful test cases within the tolerance. The 50% and 68% points were chosen as approximations to the mean and mean plus standard deviation values (assuming approximately symmetric Gaussian distributions). The data in Table 6-4 are relative to the best value obtained by optimization for each test case (which was most often the DE result for d_v = 8), whereas the optimality results in Chapter 5 are relative to v_{max}: hence the values in Table 6-4 should be less than the values in Table 5-3 and Table 5-4 of Section 5.3.3.2 by approximately the Table 5-3 and Table 5-4 d_v = 8 values. It can be seen that this is the case.

SNM:	**Loss of optimality for *p*% of successful test cases**	
d_v	p = 50%	p = 68%
3	28.6	42.0
4	4.0	5.5
5	2.6	3.8
6	1.0	2.0
7	0.4	0.9
8	0	0.2

Table 6-4. SNM Loss of Optimality vs d_v

The corresponding data for the most robust algorithm, DE, are shown in Table 6-5 and are consistent with the data in Table 6-4..

DE:	Loss of optimality for p% of successful test cases	
d_v	p = 50%	p = 68%
3	27.5	42.5
4	3.9	5.5
5	2.9	4.1
6	0.9	1.9
7	0.4	1.1
8	0	0.2

Table 6-5. DE Loss of Optimality vs d_v

These results therefore support the results of Chapter 5 on the dependence of optimality on the degree of airspeed parameterization.

6.3.3 Comparative Optimality of the NLP Algorithms

Figure 6-6 shows α_0, the number of test cases for which each algorithm produced the lowest t_f, for each d_v, measured over all test cases.

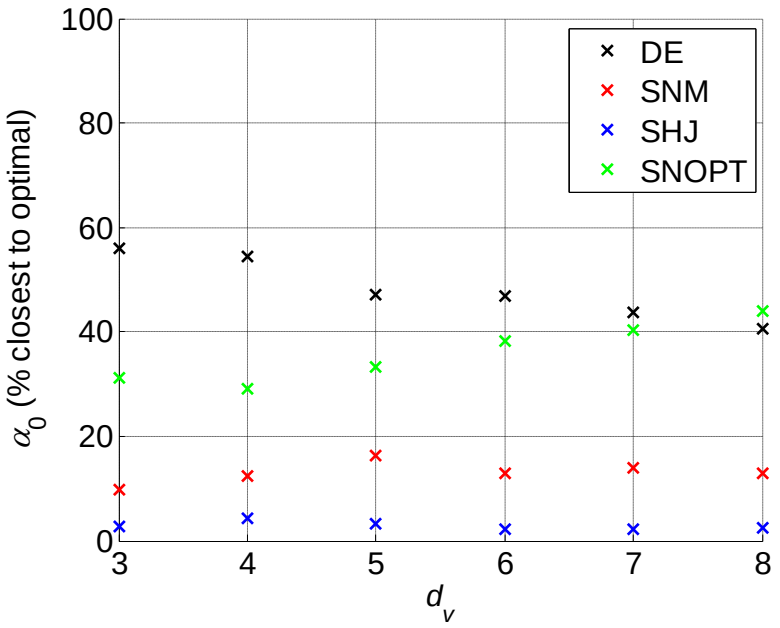

Figure 6-6. Percentage Each NLP was Closest to Optimal vs d_v

SNOPT, as the only one of the four NLP algorithms to explicitly use the KKT conditions, was expected to produce results closer to optimal than the other algorithms, and therefore to have the highest α_0 score. However, DE achieved the highest α_0 score: 48% versus 36% for SNOPT, although SNOPT achieved an α_0 score 2% higher than DE for $d_v = 8$. SNM achieved an α_0 score of 10-16%, and SHJ only 2-4%, all measured over all test cases.

Figures 6-7 and 6-8 show the α_2 and α_5 scores over all test cases. In these figures SNOPT results reduced in optimality more steeply than any of the other three algorithms: SNOPT dropped from second place to DE for α_0 (Figure 6-6) to third place for α_2 (Figure 6-7) and to last place for α_5 (Figure 6-8): this was due to SNOPT's lack of robustness which bounded SNOPT's optimality scores.

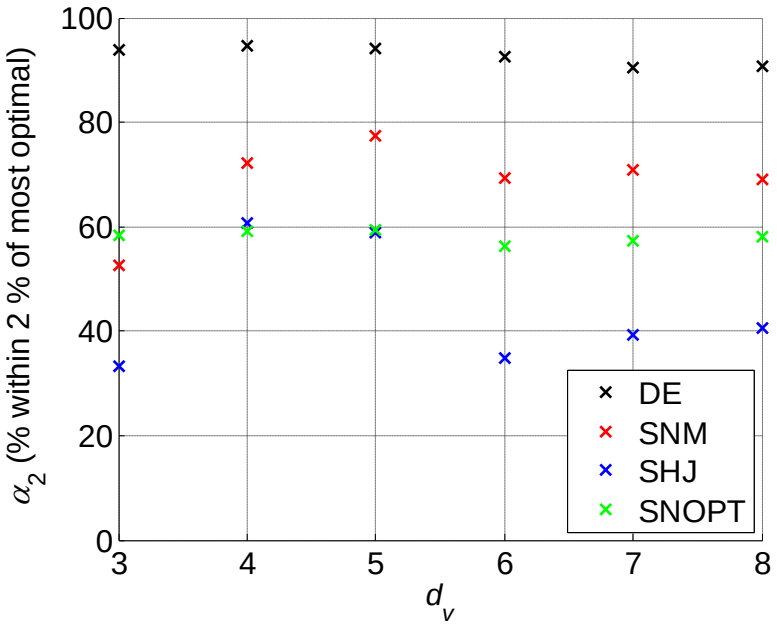

Figure 6-7. Percentage Each NLP was Within 2% of Optimal vs d_v

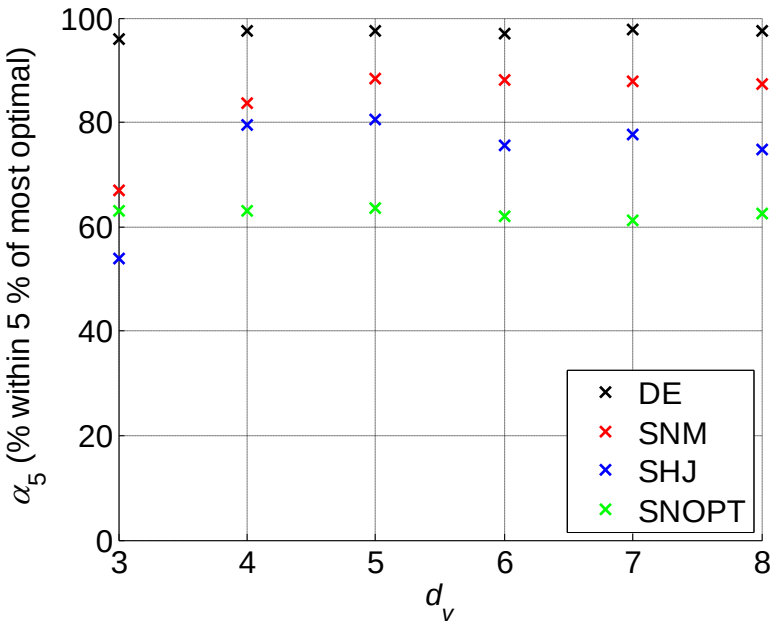

Figure 6-8. Percentage Each NLP was Within 5% of Optimal vs d_v

To remove the limiting effects of robustness from the analysis the optimality of each algorithm was analyzed over only successful tests, and the data was divided by β. Figure 6-9 shows the resulting performance profiles.

These graphs show that when SNOPT was successful it consistently produced better values of t_f than the two derivative-free algorithms. However, the figures also confirm that DE was not only the most robust but also the most consistently optimal. Although over successful cases SNOPT achieved the lowest t_f more often than DE for $d_v \geq 5$, (shown by the intersection of the curves with the y-axis), the crossover points, at which more DE results were within a tolerance than SNOPT results, were at tolerances of 0.02%, 0.2%, 1.4% and 1% for $d_v \in \{5,...8\}$ respectively. This shows that DE optimality was more consistently close to optimal than was SNOPT, even when SNOPT was successful.

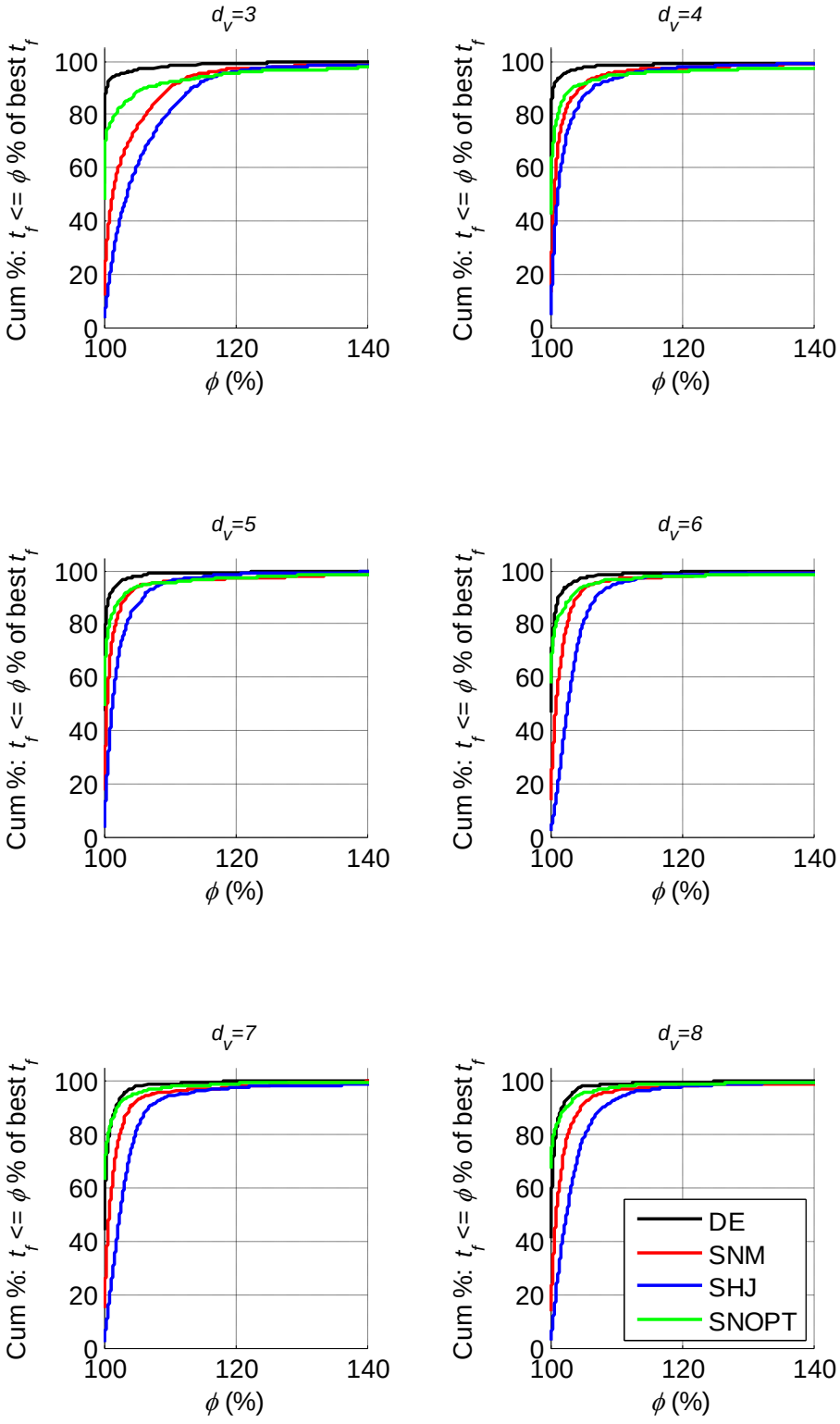

Figure 6-9. Scaled Optimality Profiles for Each d_v, vs NLP

When the effects of robustness are removed, Figure 6-9 shows that for $d_v \geq 5$ all of the NLP algorithms produced solutions within 10% of the lowest t_f in 95% of cases. However, when airspeed was excluded from the optimization only DE achieved more than 95% (98%) of successful results within 10% of the lowest t_f (SNOPT achieved 92%, SNM achieved 90% and SHJ achieved 82%).

SNM consistently outperformed SHJ for optimality but both achieved low optimality scores: α_0 = 10-16% for SNM and α_0 = 2-4% for SHJ.

6.3.4 Computational Speed

The means and standard deviations of σ for all test cases are shown in Tables 6-6 and 6-7 and in Figures 6-10 and 6-11.

NLP	**Degree of Airspeed Parameterization**					
	3	**4**	**5**	**6**	**7**	**8**
DE	12101	11088	12248	14422	16371	18266
SNM	10553	5672	4553	4520	4579	4671
SHJ	1573	1400	1333	1424	1610	1780
SNOPT	1428	1390	1483	1494	1601	1702

Table 6-6. Mean Trajectory Evaluations, All Test Cases

NLP	**Degree of Airspeed Parameterization**					
	3	**4**	**5**	**6**	**7**	**8**
DE	9414	5013	4073	4181	4627	4838
SNM	19055	13031	10795	10363	10079	9759
SHJ	2960	1609	1326	1495	1618	1825
SNOPT	1101	1184	1001	1001	1074	1063

Table 6-7. Standard Deviation of Trajectory Evaluations, All Test Cases

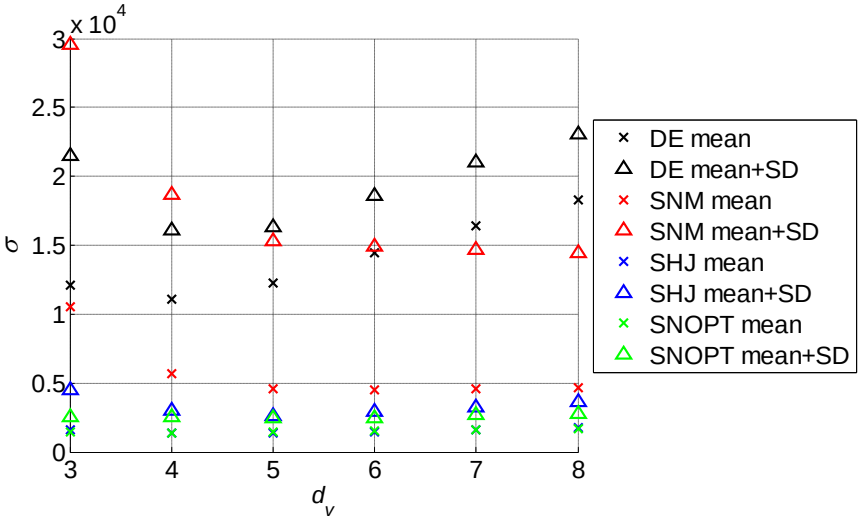

Figure 6-10. Trajectory Evaluations, All Test Cases

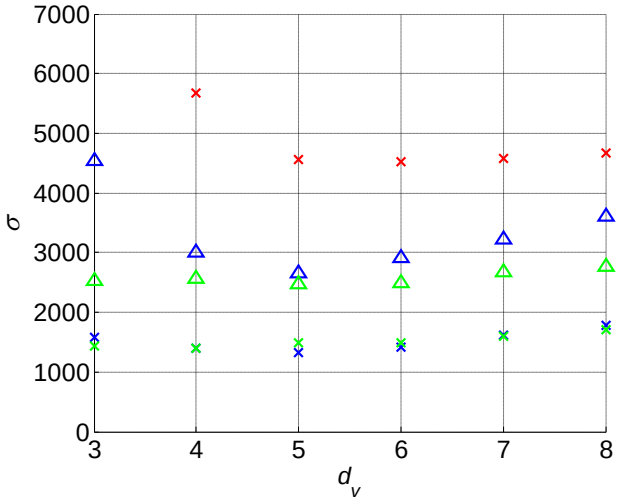

Figure 6-11. Expansion of Figure 6-10

A two-factor ANOVA was performed: at a significance level of 0.01, the effects of the NLP algorithm, of d_v, and of the interaction between NLP and d_v, were statistically significant.

DE was the slowest of the four algorithms: measured by mean σ over all test cases it required 8-12 times as many trajectory evaluations as did the two fastest algorithms SHJ and SNOPT, and up to 4 times as many as SNM. Mean σ for DE had a minimum at $d_v = 4$ and the standard deviation of σ was close to a minimum for $d_v \in \{4,5\}$.

Figure 6-11 shows the data of Figure 6-10 against an expanded *y*-axis, to present the data for SHJ and SNOPT more clearly: the two algorithms required similar numbers of trajectory evaluations but SHJ showed more sensitivity to d_v.

The means and standard deviations of σ for SNM were also high: for $d_v = 3$ their sum was 50% higher than that of DE. The SNM mean σ reduced as d_v increased to 6, but then showed a small rate of increase up to $d_v = 8$. The SNM standard deviation reduced steeply as d_v increased to 5, then continued to decrease, but at a lower rate, as d_v increased to $d_v = 8$. SNM showed a monotonic decrease in mean plus standard deviation of σ as d_v increased, i.e. the increase in degrees of freedom increased the rate of convergence faster than the NLP dimension reduced it, up to the highest d_v tested ($d_v = 8$).

Figure 6-12 shows the σ performance profiles for all test cases including failures to converge successfully: SHJ was most often the fastest for every value of d_v, but for $d_v \geq 4$ the SNOPT curves cross over the SHJ curves: this means that over a sufficiently large tolerance SNOPT would produce more results within the tolerance than SHJ.

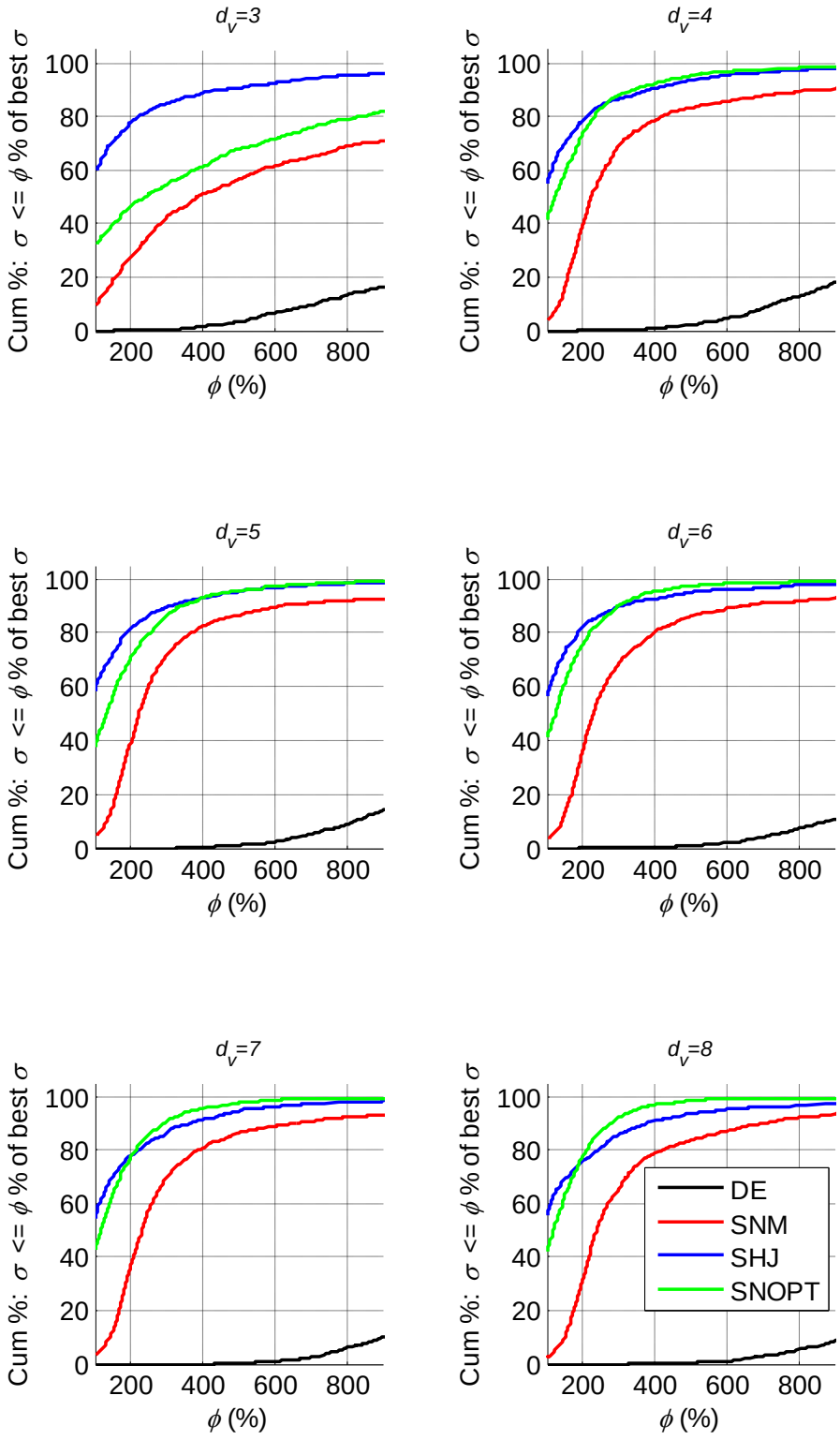

Figure 6-12. σ Profiles for Each d_v, vs NLP, All Test Cases

These results were due in part to the different behaviours of DE and SNM compared to SHJ and SNOPT for unsuccessful test cases: DE and SNM continued until the trajectory evaluation limit σ_{max} was reached, whereas SHJ and SNOPT terminated quickly. Tables 6-8 and 6-9 show the data when only successful test cases are included.

NLP	**Degree of Airspeed Parameterization**					
	3	**4**	**5**	**6**	**7**	**8**
DE	9964	10777	12149	14322	16340	18212
SNM	2112	1895	1963	2160	2316	2659
SHJ	830	1146	1115	1245	1400	1560
SNOPT	1543	1592	1648	1724	1886	2029

Table 6-8. Mean Trajectory Evaluations, Successful Test Cases

NLP	**Degree of Airspeed Parameterization**					
	3	**4**	**5**	**6**	**7**	**8**
DE	4407	3825	3660	3761	4523	4692
SNM	3769	1934	1288	1177	970	1379
SHJ	1655	1141	848	1118	1047	1230
SNOPT	890	1288	962	998	1044	1005

Table 6-9. Standard Deviation of Trajectory Evaluations, Successful Test Cases

The 90% confidence intervals for Table 6-8 were all non-overlapping, except for the following pairs: SNM $d_v \in \{4,5\}$; SHJ $d_v \in \{4,5\}$; and SNOPT $d_v \in \{3,4\}$, $d_v \in \{4,5\}$, and $d_v \in \{5,6\}$.

The same data is shown graphically in Figures 6-13 and 6-14.

Considering only successful test cases, on mean σ DE was the slowest algorithm by a factor of 5-12 compared to the other three algorithms. For $d_v = 3$ it was faster over successful cases than over all test cases, confirming that the apparent minimum in the DE "all test cases" data was due to the unsuccessful test cases.

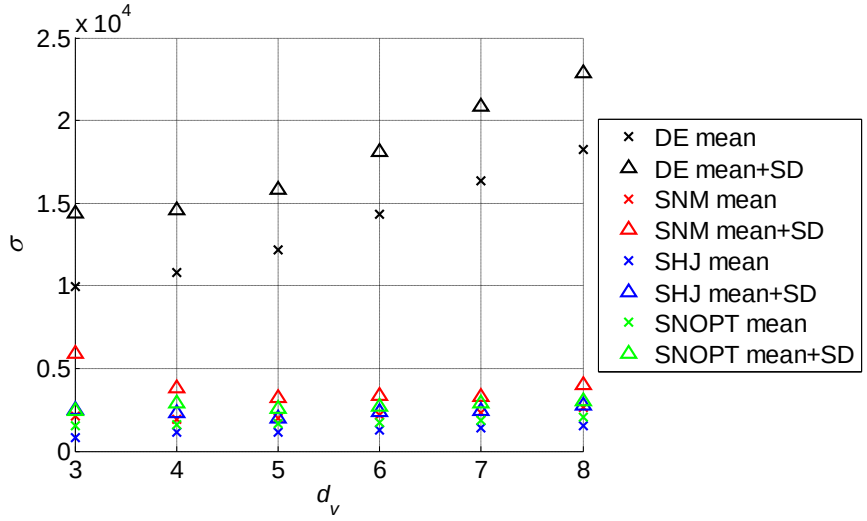

Figure 6-13 Trajectory Evaluations, Successful Test Cases

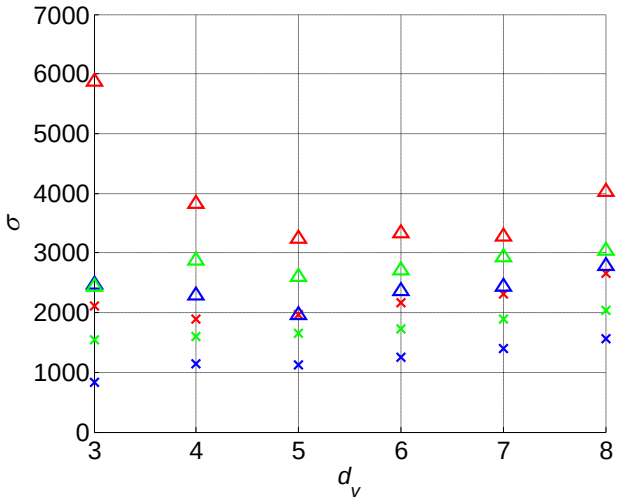

Figure 6-14. Expansion of Figure 6-13

Since for $d_v \geq 4$ DE was over 99% successful, excluding unsuccessful test cases only changed DE mean σ by less than 3% for $d_v = 4$ and less than 1% for $d_v \geq 5$. The computational speed of DE for $d_v \geq 4$ ($D \geq 8$) was approximately linear in the NLP dimension. Over only successful test cases, Table 6-8 shows that on mean σ SHJ was 26% to 46% faster than SNOPT.

The performance profiles of Figure 6-15, which include only successful test cases and scale the data by a factor of β to remove the effects of relative robustness, show that if each algorithm was 100% robust, SHJ would be the fastest for all d_v, and SNOPT, although scoring higher than SNM at $\phi = 100\%$, would be overtaken by SNM for a sufficiently large tolerance (of the order of a factor of 3). DE would remain the slowest.

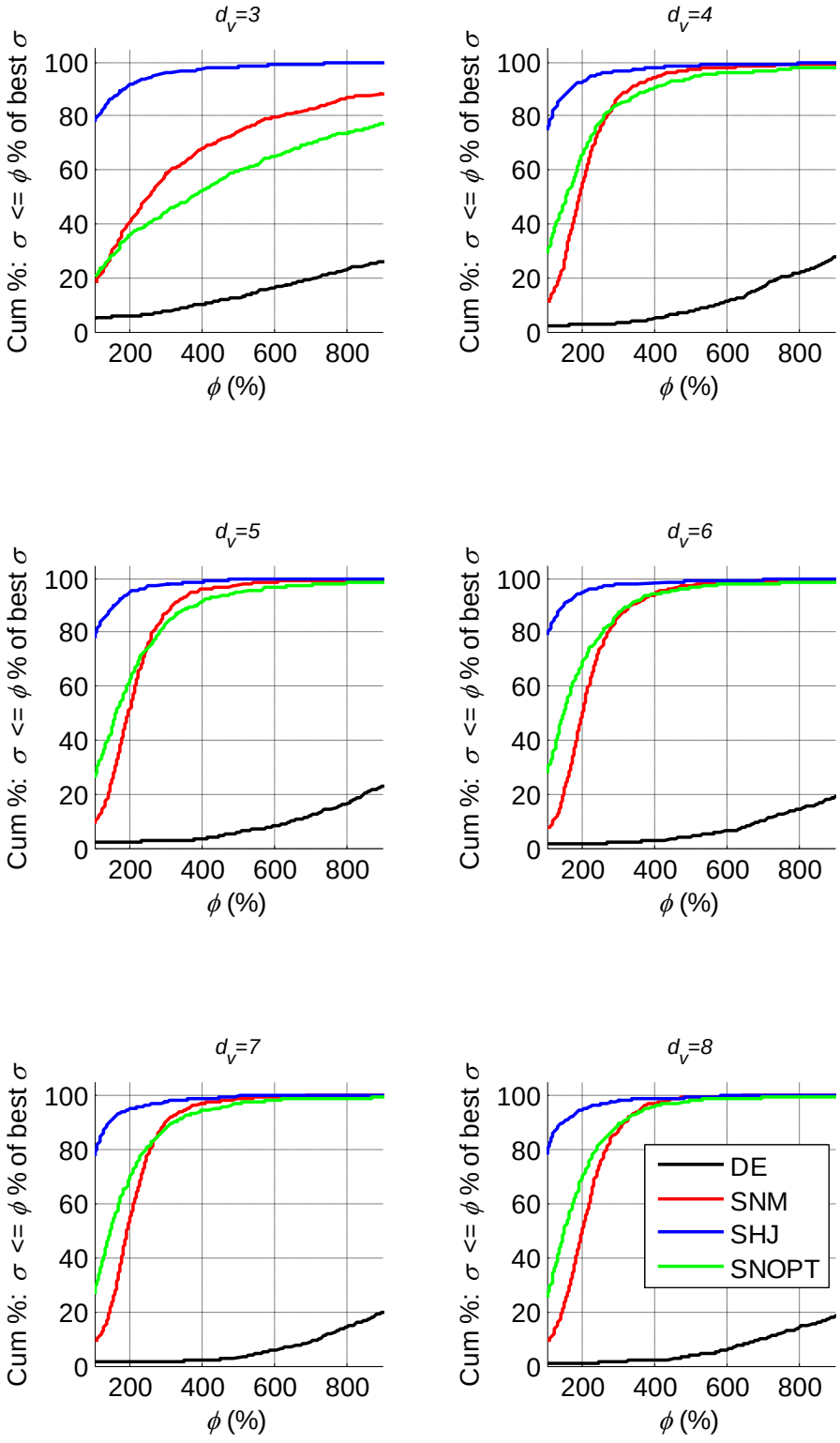

Figure 6-15. Scaled σ Profiles for Each d_v, vs NLP, Successful Tests

6.3.4.1 Computation Times

Table 6-10 shows mean computation times in seconds averaged over all test cases. The times excluded reading stored data such as boundary conditions from disc, initializing variables and arrays that were not dependent on the boundary conditions, and writing output data to disc. The times included transforming the boundary conditions to the virtual domain (Section 3.2.4), creating the corresponding initial guess if required (Section 6.2.5), invoking the NLP algorithm, and assigning the results to arrays.

NLP	**Degree of Airspeed Parameterization**					
	3	**4**	**5**	**6**	**7**	**8**
DE	303	279	307	361	416	468
SNM	268	146	116	116	115	118
SHJ	40	35	35	36	41	48
SNOPT	37	36	38	38	41	44

Table 6-10. Mean Elapsed Computation Times, All Test Cases (s)

Table 6-11 shows the percentage differences between computation times per trajectory node for each NLP algorithm relative to DE, because DE was the most efficient algorithm being fastest (per node) for 14 of the 18 comparisons in the table. Although it was the most complicated of the algorithms, SNOPT was comparable in speed with SNM and SHJ: this is attributed to the use of compiled *mexw64* code counterbalancing the additional complexity.

NLP	**Degree of Airspeed Parameterization**					
	3	**4**	**5**	**6**	**7**	**8**
DE	0.0	0.0	0.0	0.0	0.0	0.0
SNM	1.4	2.6	1.5	2.3	-0.8	-1.1
SHJ	1.8	0.8	4.6	1.5	-0.6	6.1
SNOPT	2.8	1.8	1.0	2.1	0.3	-0.2

Table 6-11. Percentage Differences in Computation Times per Node

The mean time taken to evaluate one trajectory node (N = 210), averaged over all test cases, all NLP algorithms and all d_v values, was 1.22×10^{-4} s, and the standard deviation of this value over all the algorithms and d_v values was less than 2% of the mean.

The mean computation times for DE were approximately 0.5-4% of the mean pseudo-optimal t_f times for $d_v \in \{3,...8\}$ (higher with higher d_v).

6.4 Discussion

Taking each of the three measures in isolation, on robustness DE was the most robust and SNOPT the least; on optimality DE was the most optimal and SHJ the least; on computational speed SHJ was the fastest and DE the slowest.

6.4.1 Objective 1: The Effects of d_v on Optimality

The results for all four NLP algorithms support the findings of Chapter 5 that achievable optimality depends on, and increases with, d_v. The results also show that this remains valid when the spatial path is optimized (using global degree 7 power series polynomials as spatial parameterization functions).

In all the results, the optimality achieved when airspeed was excluded from the optimization was lower than that achieved when airspeed was included in the optimization. Figure 6-9 shows that for $d_v \geq 5$ all of the NLP algorithms, when they converged successfully, produced solutions within 10% of the lowest t_f in 95% of cases. However, when airspeed was excluded from the optimization only DE achieved more than 95% of successful results within 10% of the lowest t_f. Not only peak optimality but the dispersion of optimality was better (i.e. more results were within a given tolerance of the lowest t_f) when airspeed was included in the optimization.

6.4.2 Objective 2: Optimality

DE was found to produce the most pseudo-optimal solutions: it achieved α_2 = 94% averaged over all d_v and all test cases. Since DE had no constraint violation tolerance whereas SNOPT, SNM, and SHJ each used a tolerance of 0.1, DE's high optimality and robustness are more favourable to DE than the basic arithmetic implies.

SNM achieved α_2 = 69% over all test cases, equivalent to 74% over successful cases. These results were lower than those of DE and SNOPT. SHJ's optimality results were the worst of the four algorithms: only α_2 = 45% over all test cases, equivalent to 49% over successful test cases.

When it converged successfully SNOPT was found to produce a high percentage of results close to optimal, better than SNM and SHJ but not as good as DE. It was expected to score highest for optimality due to its use of the KKT optimality conditions, but it achieved only α_2 = 58% over all test cases, equivalent to 88% over successful test cases. It only achieved an overall success rate of 66% which is attributed to the low frequency with which the initial guess was in a basin of attraction to feasible local minima because of the multimodality of the problem, and that SNOPT is more sensitive to the accuracy of the initial guess than SNM or SHJ.

Further, for test cases in which DE achieved a solution closer to optimal than any of the three "local" algorithms SNOPT, SNM, and SHJ, there are two causes to consider: either the local algorithms all failed to converge to a global minimum despite a good initial guess, or the initial guesses were not close enough to a feasible global minimum. Given DE's high α_0, α_2, and α_5 scores, investigation of improved initial guesses is a worthwhile area for future research.

No analytic method has yet appeared that is guaranteed to find an initial guess that lies in a basin of attraction to a feasible global minimum (if such a method does exist then it reduces the global optimization problem to a local optimization problem). One alternative is to use solutions of relaxed or similar problems, such as finding feasible solutions without minimizing an objective: Yakimenko reported[133] good results using this latter approach in 2000. However, even if this approach were to raise SNOPT's robustness to the same as DE, SNOPT's optimality would still be lower than that of DE.

6.4.3 Objective 3: Robustness and Computational Speed

DE was found to be the most robust of the four algorithms: it generated a successful solution in more than 99% of all test cases for $d_v \geq 4$, and in 94% of all test cases when

airspeed optimization was excluded ($d_v = 3$). SNOPT achieved an overall success rate of 66%.

The two derivative-free algorithms SNM and SHJ achieved higher robustness than SNOPT, but lower than DE. In particular when higher values of d_v were used ($d_v \in \{6,7,8\}$), the additional degrees of freedom helped both SNM and SHJ to achieve success rates of 93-96%, thus exceeding the 66% achieved by SNOPT and supporting (for these test data) Yakimenko's comments[133] that the Nelder-Mead and Hooke-Jeeves algorithms are more robust than gradient-based algorithms.

Airspeed optimization enabled DE, SNM, and SHJ to converge successfully for a wider range of boundary conditions than when airspeed was excluded from the optimization. When airspeed was included in the optimization the robustness of DE, SNM, and SHJ improved compared to robustness without airspeed optimization: from 94% to >99% for DE and from 81-82% to > 91% for SNM and SHJ. For $d_v = 6$ the success rates of DE, SNM, and SHJ were 99.7%, 95.1% and 95.4% respectively. SNOPT showed only a 4% variation in robustness over the tested range of d_v, including $d_v = 3$.

Except for DE results over only successful test cases, computational speed was faster when airspeed was included in the optimization provided that d_v was low. For DE results over only successful cases, excluding airspeed from the optimization was faster but by only 1.6%, for a 25-38% loss of optimality.

When both successful and unsuccessful test cases were included, it was found that SHJ was most often the fastest, that SNOPT was the fastest over a sufficiently large tolerance, that SNM was a factor of 3-7 times slower than SNOPT/SHJ, and that DE was the slowest by a factor of 8-12. When only successful test cases were included, DE was the slowest by a factor of 5-12, and SHJ was the fastest but at the expense of optimality.

The combination of the accelerating effect of increasing degrees of freedom and the decelerating effect of increasing NLP dimension produced minima in some measures of computational speed as a function of d_v. The corresponding values of d_v were

dependent on the choice of NLP algorithm and which measures of computational speed were used.

6.4.4 Objective 4: Computation Times

The mean computation times of 35-50 s that were achieved by SNOPT and SHJ with N = 210 were only approximately 0.05-0.4% of the corresponding t_f times. Yakimenko et al.[5, 135] reported a computation time of 3 s and a compute:flight time ratio of approximately 10%, but based on only a single problem, and with no data on the number of trajectory evaluations invoked.

6.4.5 Objective 5: Optimization Approaches

The main weaknesses of each algorithm were:

- DE. DE was 8-12 times slower overall than SNOPT/SHJ.
- SNOPT. The robustness of SNOPT was 66%, compared to DE with over 99% and SNM/SHJ with ~95%.
- SNM. SNM was 3-7 times slower than SNOPT/SHJ, and less optimal than DE and SNOPT for successful test cases.
- SHJ. The optimality of SHJ, measured by α_2 but including only successful test cases, was only 49% compared to 88% for SNOPT and 94% for DE.

DE is well suited to parallel implementation, because each member of a generation can be processed independently of the processing of each other member of the population, and each process always involves one trajectory evaluation. Therefore N_p processors used in parallel to process each generation, with a further processor for synchronization and data management, would increase computational speed by a factor approaching N_p. For N_p = 15 as used in this work, this would reduce the elapsed computation times per test case to approximately 18-31 s (assuming N = 210 and no net change due to processor clock speeds or changing from a Matlab to compiled code environment) which is only 51-71% of the shortest times in Table 6-10.

Two obvious initial guess approaches are:

- When the boundary conditions are in the same neighbourhood as the previous boundary conditions, use the previous pseudo-optimal solution as the new initial guess.
- Otherwise use a global algorithm to generate initial guesses for a deterministic local algorithm, e.g. use parallel DE to generate initial guesses for SNOPT.

The computation times were achieved using $N = 210$, and are approximately linear in N. Hence reducing N would improve the computational speed, albeit at the cost of accuracy. A possible two-stage optimization approach would be to use a low N value to obtain an initial guess then use higher N for the solution, an approach that was used by Yakimenko et al.[5, 135] to improve the performance of the Gauss and Legendre pseudospectral methods.

Therefore three promising paths to achieve the computational speed needed for on-board near-real-time trajectory generation with the inverse dynamics method are: improving the computational speed of DE through parallel processing; improving the robustness of SNOPT through better initial guesses; a combination of the two approaches to exploit the advantages of both DE and SNOPT. A fourth possibility is to seek to improve the optimality and robustness of SHJ through better initial guesses and/or termination criteria. These areas are suggested for future research.

7 BANK ANGLE ILL-CONDITIONING AND NEGATIVE-G TRAJECTORIES

7.1 Introduction

For surveillance and reconnaissance missions it is critical that the UAV provides a stable platform for its on-board sensors and communications. However, Assumption 2 (Section 3.2) restricts the aircraft model, and therefore the trajectories, to positive-g orientations and this restriction leads to ill-conditioning in orientation when $l_z \to 0$: small changes in l_z, for example due to round-off errors, can cause repeated large changes in bank angle. Such abrupt manoeuvres are difficult for the flight control system to track accurately and are likely to cause control actuator saturation. Further, even if the flight control system response is fast and accurate, the bank angle changes would demand high-bandwidth sensor dynamics, or very wide sensor fields of view, to maintain target tracking. Although this problem is most acute in sustained near-vertical flight it may occur at any orientation, such as during low level terrain following or during direction changes to avoid obstacles in an urban environment.The ill-conditioning can be removed by aligning the lift vector with either the positive or the negative direction of the load factor, whichever is nearest to the orientation at the previous node. This aim of the work described in this chapter is to reduce the probability of control actuator saturation or loss of sensor tracking due to the ill-conditioning by admitting negative-g trajectories and meeting the following objectives:

- Minimize rotation, subject to maintaining coordinated flight ($\beta = 0$) except during inversion.
- Implement inversions over time domains that minimize a defined objective function.
- Smoothly interpolate orientations during inversions.

In this chapter, "invert" and "inversion" refer to a rotation from positive to negative alignment of the lift vector with the normal load factor, or vice-versa.

The Euler-angle model has singularities at $\gamma = \pm\pi/2$, is sequence-dependent, and cannot in general be smoothly interpolated. Although mathematically correct, the

controls can be difficult for a flight control system to track: for example when transitioning through the vertical in a loop, both heading ξ and bank μ have discontinuities of approximately 180° (Figure 3-1). The quaternion-based model of Chapter 4 does not have orientation singularities, the magnitude of a change in orientation can be readily evaluated as a scalar, and orientations can be smoothly interpolated. These latter two properties are exploited to enable the negative-g extensions described in this chapter.

7.2 Algorithmic Extensions

The method may be extended to select the negative-g orientation $\mathbf{e}^-$ when $\mathbf{e}^-$ requires a smaller rotation from $\mathbf{e}_{j-1}$ than the positive-g orientation $\mathbf{e}^+$, subject to a negative-g load factor limit l^-, by replacing Assumption 2 by

$$\widehat{\mathbf{z}}_{W\,j} = \begin{cases} -d_j \frac{\mathbf{l}_{z\,j}}{\|\mathbf{l}_{z\,j}\|}, & \text{if } \|\mathbf{l}_{z\,j}\| \neq 0 \\ \widehat{\mathbf{z}}_{W\,j-1}, & \text{otherwise} \end{cases} \tag{7.1}$$

The node-based load factor is used in Eq. (7.1) to align the bank angle with the load factor at each node.

The variable $d \in \{-1,+1\}$ represents the alignment of the bank angle where $d = +1$ corresponds to positive-g orientation, and initial and final values of d, (d_0 and d_f), are included in the boundary conditions.

Three steps are required at each node $j \in [2, N]$:

1. Determine the orientation **e** nearest to $\mathbf{e}_{j-1}$ that satisfies Eq. (7.1), and set a flag χ if the orientation has to be inverted.
2. Update the inversion domain $[j_s, j_m]$ over which a cost function Λ has a minimal value.
3. If χ is true, interpolate the orientation at each node in $[j_s+1, j_m+1]$, re-evaluate the controls over $[j_s, j-1]$ and reset the orientations over $[j_m, j]$ to match.

At the final node ($j = N$) step 1 is modified so that χ is set to true if $d \neq d_f$.

Although the algorithm is described with the implicit assumption that the positive-g load factor limit is higher than the negative-g load factor limit because this is the usual case, the algorithm may be applied to the opposite case with only minor adaptations.

7.2.1 Determining Orientation

The orientation nearest to $\mathbf{e}_{j-1}$ that satisfies Eq. (7.1) may be evaluated as (omitting the suffix *j* from Eqs. (7.2)-(7.6) for clarity)

$$\mathbf{e}^{+} = \text{Shepperd}\left(\widehat{\mathbf{x}}_W, \frac{-\mathbf{l}_z}{\|\mathbf{l}_z\|} \times \widehat{\mathbf{x}}_W, \frac{-\mathbf{l}_z}{\|\mathbf{l}_z\|} \right) \tag{7.2}$$

$$\mathbf{e}^{+} \leftarrow \begin{cases} \mathbf{e}^{+}, & \text{if } \mathbf{e}^{+} \cdot \mathbf{e}_{j-1} \geq 0 \\ -\mathbf{e}^{+}, & \text{otherwise} \end{cases} \tag{7.3}$$

$$\mathbf{h} = \left(0, \widehat{\mathbf{x}}_W^T \right)^T \tag{7.4}$$

$$\mathbf{e}^{-} = \mathbf{h} * \mathbf{e}^{+} \tag{7.5}$$

$$\mathbf{e}^{-} \leftarrow \begin{cases} \mathbf{e}^{-}, & \text{if } \mathbf{e}^{-} \cdot \mathbf{e}_{j-1} \geq 0 \\ -\mathbf{e}^{-}, & \text{otherwise} \end{cases} \tag{7.6}$$

$$\mathbf{e}_j = \arg \max_{\mathbf{e} \in \{\mathbf{e}^{+}, \mathbf{e}^{-}\}} \left(\mathbf{e} \cdot \mathbf{e}_{j-1} \right) \tag{7.7}$$

$$d_j = \begin{cases} -1 & \text{if } \mathbf{e}_j = \mathbf{e}^{-} \\ 1 & \text{if } \mathbf{e}_j = \mathbf{e}^{+} \end{cases} \tag{7.8}$$

Eqs. (7.3) and (7.6) ensure that $\mathbf{e}^{+}$ and $\mathbf{e}^{-}$ are the nearest quaternions to $\mathbf{e}_{j-1}$ on S^3 that satisfy Eqs. (7.2) and (7.5), and are required because the unit quaternion group double covers $SO(3)$, i.e. $\mathbf{e}$ and $-\mathbf{e}$ represent the same rotation.

The resulting orientation is rotation-minimizing subject to maintaining zero sideslip. The “invert” flag is given by

$$\chi = \begin{cases} \text{true}, & \text{if } \left(\mathbf{e}_j = \mathbf{e}^-\right) \cap \left(\left\| \mathbf{l}_{z\ j}^{seg} \right\| > l^- \right) \\ \text{false}, & \text{otherwise} \end{cases} \tag{7.9}$$

An additional flag, η, denotes whether the load factor is within the negative load factor limit

$$\eta_j = \begin{cases} \text{true}, & \text{if } \left\| \mathbf{l}_{z\ j}^{seg} \right\| \le l^- \\ \text{false}, & \text{otherwise} \end{cases} \tag{7.10}$$

The segment-based load factor (Eq.(4.66)) is used in Eqs. (7.9) and (7.10) because load factor magnitude is required.

7.2.2 Updating the Inversion Domain

When a requirement for inversion is detected at a node *j* by Eq. (7.9) evaluating to true, the inversion can be distributed over any contiguous sequence of segments starting from the nearest preceding node *k* for which η_k = false:

$$k = \max\left(i \mid \eta_i = \text{false}, i \ge 2, i \le j\right) \tag{7.11}$$

and terminating at the current node *j*, providing that the path constraints remain satisfied. The inversion domain [j_s, j_m] is defined as the domain that minimizes a cost function Λ

$$[j_s, j_m] = \arg\min_{\Psi} \Lambda \tag{7.12}$$

$$\text{s.t.} \quad t(j_m) - t(j_s) \ge t_I \tag{7.13}$$

where

$$\Psi = \left\{ [a,b] \middle| a \ge k, b > a, b \le j \right\}, \quad a,b \in \mathbb{Z}^+ \tag{7.14}$$

and

$$\Lambda(a,b) = \max_{i \in [a,b]} \left(\left\| \mathbf{l}_{z\,i}^{seg} \right\| \right) \tag{7.15}$$

No additional iteration is required to solve Eq. (7.12), only a comparison of the current inversion domain with any new candidate domains as each node is processed.

Eq. (7.13) expresses the constraint that the time interval corresponding to $[j_s, j_m]$ is sufficiently long to enable the generated angular velocities to satisfy their constraints. If $t(j)$-$t(k) < t_I$ then Eq. (7.13) will be violated: this may be handled in the same way as other constraint violations. Eq. (7.15) is the cost function for the inversion; as expressed here it ensures that inversion takes place when the normal load factor is minimized, but other functions could be chosen.

7.2.3 Inverting the Orientation

The required orientation at the end of an inversion is given by

$$\mathbf{e}_m = \begin{cases} \mathbf{e}^+ & \text{if } d_m = -1 \text{ and } \mathbf{e}_s \cdot \mathbf{e}_m \geq 0 \\ -\mathbf{e}^+ & \text{if } d_m = -1 \text{ and } \mathbf{e}_s \cdot \mathbf{e}_m < 0 \\ \mathbf{e}^- & \text{if } d_m = 1 \text{ and } \mathbf{e}_s \cdot \mathbf{e}_m \geq 0 \\ -\mathbf{e}^- & \text{otherwise} \end{cases} \tag{7.16}$$

Quaternion interpolation may be applied to evaluate the orientation at each node during the inversion

$$\mathbf{e}_i = \text{slerp}\left(t_i, \mathbf{e}_s, \mathbf{e}_m\right), \quad t_i = \left\{ \frac{t(i)}{t(j_m) - t(j_s)}, \forall i \in \left[j_s + 1, j_m - 1 \right] \right\} \tag{7.17}$$

The angular velocity at each node is evaluated by applying Eq. (4.64) (or Eq. (4.47)) to the orientations produced by Eq. (7.17). Hence two stages of interpolation are applied during inversion: one stage to evaluate the orientations and a second stage to evaluate the angular velocities.

Between the end of the inversion and the current node, the orientations are inverted simply by exchanging $\mathbf{e}^+$ and $\mathbf{e}^-$, and the controls are inverted by setting $q = -q$ and $r = -r$.

7.3 Numerical Examples

Figure 7-1 shows results for a trajectory representing an aircraft flying at constant airspeed over a small hill which, if the aircraft remained erect, would cause a negative load factor around the peak of the trajectory. Figure 7-1a shows the vertical profile and Figure 7-1b shows the load factor for an erect aircraft. After 7.4 s as the aircraft pitched down the load factor changed from positive to negative: with the positive-g restriction active this caused a demand for a 180° step change in bank angle which can be seen as the positive red spike in Figure 7-1c. After 8.4 s the load factor changed again as the aircraft started to pitch up (relative to the erect vertical) causing a second spike in bank rate as shown in Figure 7-1c, from which it is clear that the demanded bank rates were infeasible and were likely to cause control actuator saturation. Figures 7-1c and 7-1d show that when negative-g orientations were allowed the bank rates were feasible (zero in this example) and the aircraft remained erect.

Figure 7-2 shows how the algorithm operates for a trajectory in which the load factor exceeds its negative limit. (Note that Figure 7-2 is plotted against node values for clarity.) Figure 7-2a shows the vertical profile: from straight and level flight the aircraft initiated a steep (80°) descent followed by a short climb then level-off into level flight. The aircraft maintained constant airspeed throughout. The boundary conditions included $d_0 = -1$ and $d_f = 1$ so that the aircraft started inverted and finished erect. Figure 7-2b shows what the load factor would be if negative-g were enabled but inversions were disabled: from an initial value of -1g the load factor would have increased to approximately +1g then decreased to a small negative value, with a local minimum of the load factor at $j = 58$, until the aircraft started to climb (still inverted). It can be seen that at $j = 125$ $\left\| \mathbf{l}_z^{seg} \right\| > l^-$ and $d = -1$, hence an inversion should be demanded over the inversion domain $[j_s, j_m]$ that minimizes Λ (inversions were disabled for Figure 7-2b to show the event at $j = 125$). Figure 7-2c shows that $j_s = 57$ at $j = 125$ so that the algorithm back-stepped to node 57 to start the inversion; this can be seen in Figure 7-2d which shows the load factor for the negative-g algorithm with inversions enabled. Figure 7-2d also shows that the orientation satisfied the boundary conditions on d.

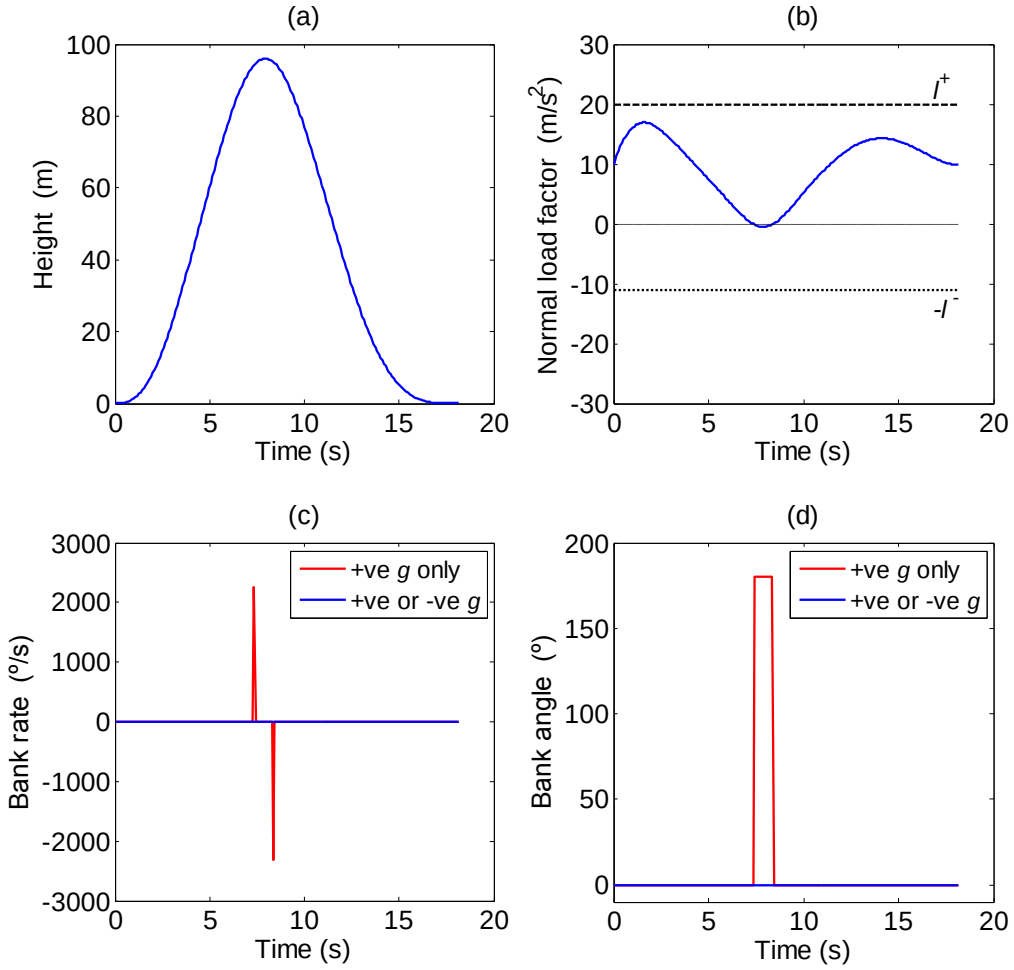

Figure 7-1. Climbing and Descending Over a Small Hill
a) the vertical profile, b) the load factor for an erect aircraft,
c) the bank rate controls, d) the bank angle demand

Figures 7-1 and 7-2 demonstrate that the algorithm removed the ill-conditioning of bank angle when $\|\mathbf{l}_z\| \approx 0$ by selecting the coordinated flight orientation that satisfied the minimum rotation criterion. Figure 7-2 also demonstrates that, when an inversion was necessary to satisfy the negative load factor limit, the algorithm implemented the inversion over a time domain that minimized the defined cost function. Figure 7-3 shows the bank rate and demanded bank angle for the trajectory of Figure 7-2, with inversions enabled, plotted against time. At t = 28 s (j = 57), a feasible bank rate was generated and maintained until the required inversion was complete.

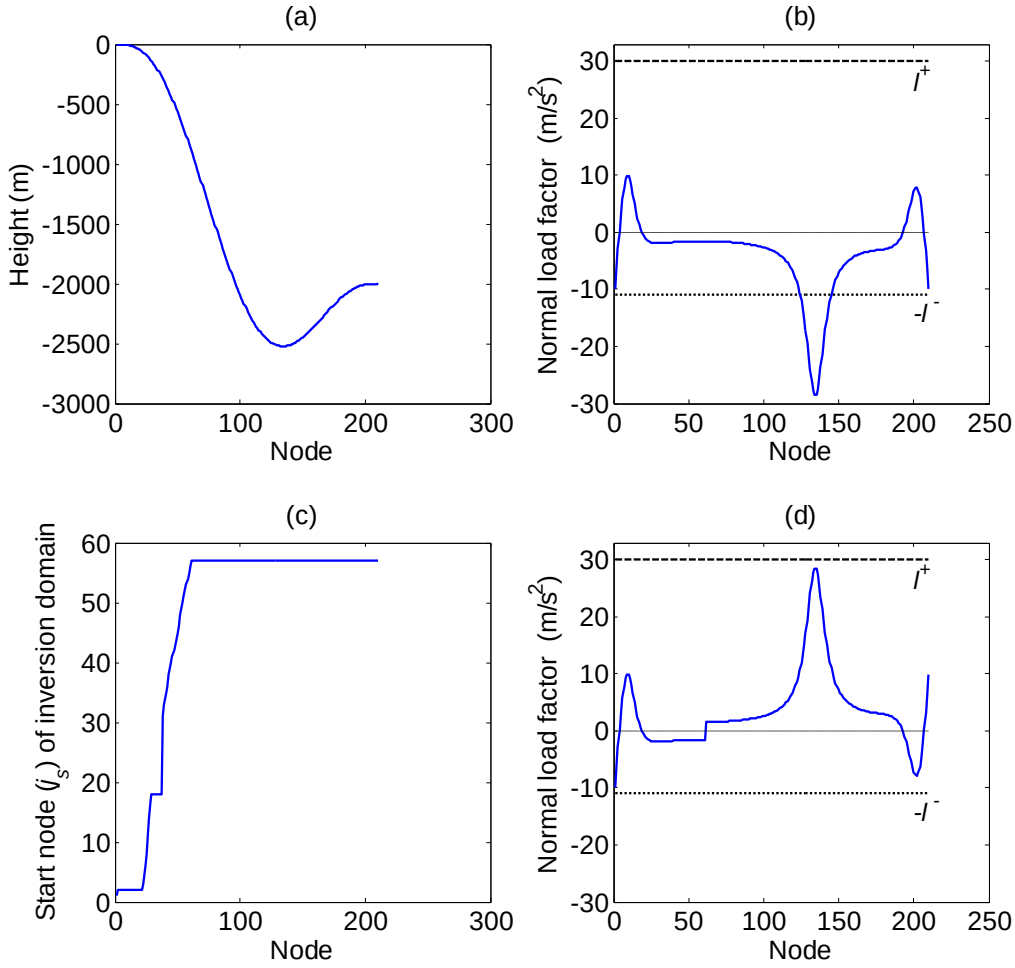

Figure 7-2. Steep Descent and Pull-Up
a) the vertical profile, b) the load factor when inversion is disabled,
c) the start node of the inversion domain,
d) the load factor when inversion is enabled

Figure 7-3 shows that the orientation was smoothly interpolated and a feasible bank rate was generated during the inversion. The final position error for the trajectory of Figures 7-2 and 7-3, using *ode45* to integrate the state equations, was < 0.3 m over a total path length of 3.3 km, confirming the accuracy of the interpolated controls.

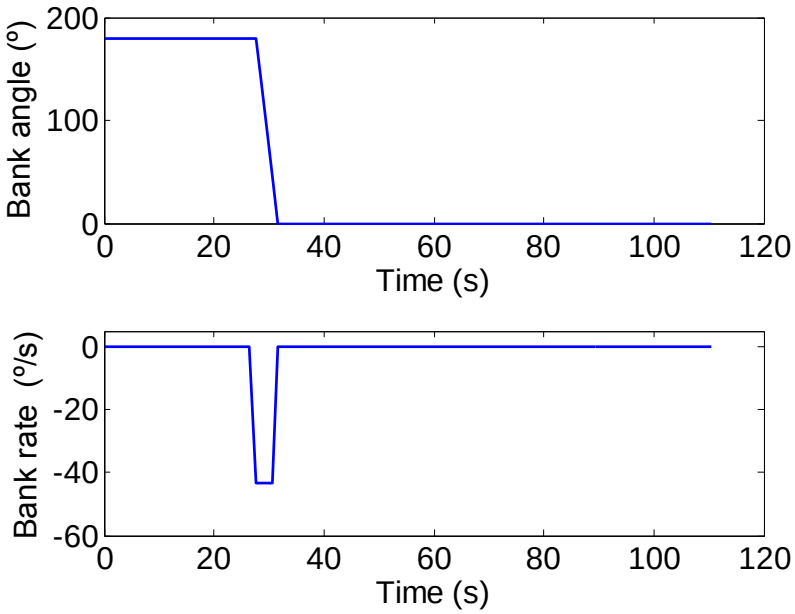

Figure 7-3. Bank Angle and Bank Rate for the Trajectory of Figure 7-2

8 CONCLUSION

The inverse dynamics method, applied to the minimization of flight time of a conventional fixed wing aeroplane, is algorithmically simple, does not require large data storage, and guarantees satisfaction of spatial, airspeed, acceleration and orientation boundary conditions. As a direct method it is computationally cheaper than indirect methods of the calculus of variations, has a larger radius of convergence, and does not require good initial guesses of constrained arcs or of non-intuitive costate variables. It has an NLP dimension typically in the range 4-15, automatically restricts the search space to regions in which the boundary conditions and state equations are satisfied, and does not require integration of the state equations. The method uses simple wind-axes point-mass aircraft models for computational speed and requires parameterization of the state vector: global low-degree polynomials have been used in most implementations to date.

The inverse dynamics method has a number of limitations that have been investigated in this work. Singularities arise from zeros of the spatial parametric speed, airspeed, and normal load factor: computational techniques to handle these singularities have been introduced. It has been shown that combining analytic and finite-difference expressions improves the accuracy of evaluation of algorithmic variables, constraints and controls. In particular it has been found that it is necessary for feasibility to use segment-based expressions for some path constraints: it is not sufficient to rely on analytic derivatives at the discretization nodes to evaluate the constraints and controls. Local quadratic interpolation of constraints has been found to improve computational efficiency by 35-40%. It has been found that in general the method produces multimodal optimization problems, with nonsmooth and potentially discontinuous constraints.

The singularities inherent in Euler-angle based orientation representation have been overcome by introducing a new wind-axes point-mass inverse dynamics model using unit quaternions to represent orientation. The model is approximately as computationally efficient as the Euler-angle model, permits smooth orientation parameterization and interpolation, and is linear in the controls. Five variants of control expressions have been derived and compared, and it was found that the maximum

position errors generated using the most accurate controls were within ~0.005% of path length using 257 nodes (in the absence of disturbances such as wind or noise).

The method has previously been limited to positive-g aircraft orientation, which leads to ill-conditioning of bank angle when the normal load factor transitions through zero. In turn this may cause control actuator saturation and/or exceed the ability of the sensor dynamics to maintain sensor tracking. An algorithmic extension has been introduced that admits negative-g orientations and exploits the new quaternion-based model by evaluating the magnitude of a change in orientation as a scalar, and smoothly interpolating orientation. This removes the ill-conditioning, thus improving platform stability around zero normal load factor, reducing the probability of control actuator saturation and the demands on sensor dynamics.

The optimality of a solution to a minimum-time aircraft trajectory generation problem depends on the closeness of the generated airspeed to the maximum airspeed that satisfies all path and boundary constraints. Airspeed is typically determined by optimizing the coefficients of low-degree airspeed polynomial parameterization. A new computational approach has been introduced to estimate maximum feasible airspeed without airspeed parameterization or optimization. Results obtained with this approach were used to measure the effects of the degree and form of polynomial airspeed parameterization on the robustness, optimality and computational speed of optimization. The effects of using Chebyshev, Bernstein, barycentric Lagrange and power series polynomial basis functions were compared. It was found that the form of the parameterization did not significantly affect the optimality of the solutions (except for the power series form): if a tolerance of ~3% mean loss of optimality due to airspeed parameterization was allowed, then there would be no need to use polynomials of degree higher than 8 for airspeed parameterization. However, the form of the airspeed parameterization did affect the robustness and rate of convergence: the power series form was found to perform worse than the other forms on optimality, robustness, and computational speed. The Chebyshev or Bernstein forms performed better.

Overall performance depends on the combined effects of spatial parameterization, airspeed parameterization, the fidelity of the aircraft dynamical model, initial guesses, and the chosen NLP algorithm. The performance of the quasi-Newton algorithm

SNOPT, sequential implementations of the derivative-free algorithms Nelder-Mead (SNM) and Hooke-Jeeves (SHJ), and the evolutionary algorithm Differential Evolution (DE), were compared. Using the measures defined in Chapter 6, DE was the most robust (achieving up to 99.9% success) and SNOPT the least (66%); DE was the most optimal (94%) and SHJ the least (45%); on computational speed SHJ was the fastest and DE the slowest (by an order of magnitude).

Mean computation times of 35-50 s were achieved by SNOPT and SHJ with $N = 210$. Whether the times are fast enough for on-board near-real-time trajectory generation depends on the architecture within which the inverse dynamics method is applied and the required trajectory update rate. However, the times justify the assertion that the method is potentially viable for such use.

Key areas for future research are: an optimization approach, including the choice of NLP algorithm (or algorithms) and initial guess expressions; hardware and software implementation; overall system analysis and design incorporating trajectory following, trajectory updates, the stability of the overall system, and the effects of disturbances such as wind; and reversionary modes required for flight safety.

The thesis, stated in Chapter 1, that the inverse dynamics method is a potentially viable method of on-board near-real-time trajectory generation for unmanned aircraft, is therefore maintained, but for this potential to be realized in practice further improvements in computational speed are desirable.

REFERENCES

1. Anderson, E. P., Beard, R. W., and McLain, T. W., "Real-Time Dynamic Trajectory Smoothing for Unmanned Air Vehicles", *IEEE Transactions on Control Systems Technology*, Vol. 13, No. 3, 2005, pp. 471-477.

2. Ariff, O., Żbikowski, R., Tsourdos, A., and White, B. A., "Differential Geometric Guidance Based on the Involute of the Target's Trajectory", *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 5, 2005, pp. 990-996.

3. Åström, K. J., and Murray, R. M., *Feedback Systems : an Introduction for Scientists and Engineers*, Princeton, 2008.

4. Baruh, H., *Analytical Dynamics*, McGraw-Hill, Boston, 1999.

5. Basset, G., Xu, Y., and Yakimenko, O. A., "Computing Short-Time Aircraft Maneuvers Using Direct Methods", *Journal of Computer and Systems Sciences International*, Vol. 49, No. 3, 2010, pp. 481-513.

6. Battles, Z., and Trefethen, L. N., "An Extension of Matlab to Continuous Functions and Operators", *Journal of Scientific Computing*, Vol. 25, 2004, pp. 1743-1770.

7. Bazaraa, M. S., Sherali, H. D., and Shetty, C. M., *Nonlinear Programming Theory and Algorithms*, 3rd ed., Wiley, Hoboken, 2006.

8. Bellingham, J., Richards, A., and How, J. P., "Receding Horizon Control of Autonomous Aerial Vehicles", *Proceedings of the American Control Conference*, Evanston Illinois, 2002, pp. 3741-3746.

9. Bellman, R. E., *Dynamic Programming*, Princeton University Press, Princeton, 1957.

10. Benson, D. A., Huntington, G. T., Thorvaldsen, T. P., and Rao, A. V., "Direct Trajectory Optimization and Costate Estimation via an Orthogonal Collocation Method", *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 6, 2006, pp. 1435-1440.

11. Berrut, J.-P., and Trefethen, L. N., "Barycentric Lagrange Interpolation", *SIAM Review*, Vol. 46, No. 3, 2004, pp. 501-517.

12. Betts, J. T., "Survey of Numerical Methods for Trajectory Optimization", *Journal of Guidance, Control, and Dynamics*, Vol. 21, No. 2, 1998, pp. 193-207.

13. Betts, J. T., *Practical Methods for Optimal Control Using Nonlinear Programming*, SIAM, Philadelphia, 2001.

14. Boas, M. L., *Mathematical Methods in the Physical Sciences*, Wiley, New York, 1966.

15. Bortoff, S. A., "Path Planning for UAVs", *Proceedings of the American Control Conference*, Chicago, 2000, pp. 364-368.

16. Boukari, D., and Fiacco, A. V., "Survey of Penalty, Exact-Penalty and Multiplier Methods from 1968 to 1993", *Optimization*, Vol. 32, 1995, pp. 301-334.

17. Boyarko, G. A., Romano, M., and Yakimenko, O. A., "Time-Optimal Reorientation of a Spacecraft Using a Direct Optimization Method Based on Inverse Dynamics", *Proceedings of the IEEE Aerospace Conference*, Big Sky, 2010.

18. Boyd, J. P., *Chebyshev and Fourier Spectral Methods*, 2nd ed., Dover, New York, 2001.

19. Boyd, J. P., "Computing Real Roots of a Polynomial in Chebyshev Series Form Through Subdivision with Linear Testing and Cubic Solves", *Applied Mathematics and Computation*, Vol. 174, 2006, pp. 1642-1658.

20. Boyd, J. P., "Computing the Zeros, Maxima, and Inflection Points of Chebyshev, Legendre and Fourier Series: Solving Transcendental Equations by Spectral Interpolation and Polynomial Rootfinding", *Journal of Engineering Mathematics*, Vol. 56, 2006, pp. 203-219.

21. Brent, R. P., *Minimization Without Derivatives*, Englewood Cliffs, Prentice-Hall, New Jersey, 1973.

22. Bryson, A. E., and Ho, Y. C., *Applied Optimal Control, Optimization, Estimation, and Control*, Revised Printing 1988, Taylor and Francis, New York, 1975.

23. Bulteel, C., "MQ-1 Predator and MQ-9 Reaper Operations", *Proceedings of the 25th Bristol International UAV Systems Conference*, Bristol, 2010.

24. Bunday, B. D., *Basic Optimisation Methods*, Edward Arnold, London, 1984.

25. Canuto, C. G., Hussaini, M. Y., Quarteroni, A., and Zang, T. A., *Spectral Methods: Fundamentals in Single Domains*, Springer-Verlag, Berlin, 2006.

26. Conway, B. A., and Larson, K. M., "Collocation vs Differential Inclusion in Direct Optimization", *Journal of Guidance, Control, and Dynamics*, Vol. 21, No. 5, 1998, pp. 780-785.

27. Dolan, E. D., and Moré, J. J., "Benchmarking Optimization Software with Performance Profiles", *Mathematical Programming*, Vol. 91, No. 2, 2002, pp. 201-213.

28. Dolan, E. D., Moré, J. J., and Munson, T. S., "Optimality Measures for Performance Profiles", *SIAM Journal on Optimization*, Vol. 16, No. 3, 2006, pp. 891-909.

29. Drury, R. G., Tsourdos, A., and Cooke, A. K., "Negative-g Trajectory Generation Using Quaternion-Based Inverse Dynamics", *Proceedings of the AIAA Atmospheric Flight Mechanics Conference*, Toronto, 2010.

30. Drury, R. G., Tsourdos, A., and Cooke, A. K., "Real-Time Trajectory Generation: Improving the Optimality and Speed of an Inverse Dynamics Method", *Proceedings of the IEEE Aerospace Conference*, Big Sky, 2010.

31. Drury, R. G., Tsourdos, A., and Cooke, A. K., "Negative-g Trajectory Generation Using Quaternion-Based Inverse Dynamics", *Journal of Guidance, Control, and Dynamics*, Vol. 34, No. 1, 2011, (to appear).

32. Drury, R. G., and Whidborne, J. F., "Quaternion-Based Inverse Dynamics Model for Evaluating Aerobatic Aircraft Trajectories", *Journal of Guidance, Control, and Dynamics*, Vol. 32, No. 4, 2009, pp. 1388-1391.

33. Drury, R. G., and Whidborne, J. F., "A Quaternion-Based Inverse Dynamics Model for Real-Time Trajectory Generation", *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Chicago, 2009.

34. Dubins, L. E., "On Curves of Minimal Length With a Constraint on Average Curvature and With Prescribed Initial and Terminal Positions and Tangent", *American Journal of Mathematics*, Vol. 79, 1957, pp. 497-516.

35. Eele, A., and Richards, A., "Path-Planning with Avoidance using Nonlinear Branch-and-Bound Optimization", *Journal of Guidance, Control, and Dynamics*, Vol. 32, No. 2, 2007, pp. 384-394.

36. Elnagar, J., and Kazemi, M. A., "Pseudospectral Chebyshev Optimal Control of Constrained Nonlinear Dynamical Systems", *Computational Optimization and Applications*, Vol. 11, 1998, pp. 195-217.

37. Elnagar, J., Kazemi, M. A., and Razzaghi, M., "The Pseudospectral Legendre Method for Discretizing Optimal Control Problems", *IEEE Transactions on Automatic Control*, Vol. 40, No. 10, 1995, pp. 1793-1796.

38. Enright, P. J., and Conway, B. A., "Discrete Approximations to Optimal Trajectories Using Direct Transcription and Nonlinear Programming", *Journal of Guidance, Control, and Dynamics*, Vol. 15, No. 4, 1992, pp. 994-1001.

39. Fahroo, F., and Ross, I. M., "Trajectory Optimization by Indirect Spectral Collocation Methods", *Proceedings of the AIAA/AAS Astrodynamics Specialist Conference*, Denver, 2000, pp. 123-129.

40. Fahroo, F., and Ross, I. M., "Costate Estimation by a Legendre Pseudospectral Method", *Journal of Guidance, Control, and Dynamics*, Vol. 24, No. 2, 2001, pp. 270-277.

41. Fahroo, F., and Ross, I. M., "Second Look at Approximating Differential Inclusions", *Journal of Guidance, Control, and Dynamics*, Vol. 24, No. 1, 2001, pp. 131-133.

42. Fahroo, F., and Ross, I. M., "Direct Trajectory Optimization by a Chebyshev Pseudospectral Method", *Journal of Guidance, Control, and Dynamics*, Vol. 25, No. 1, 2002, pp. 160-166.

43. Fahroo, F., and Ross, I. M., "Pseudospectral Methods for Infinite Horizon Nonlinear Optimal Control Problems", *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, San Francisco, 2005.

44. Fahroo, F., and Ross, I. M., "Advances in Pseudospectral Methods for Optimal Control", *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Honolulu, 2008.

45. Farouki, R. T., and Goodman, T. N. T., "On the Optimal Stability of the Bernstein Basis", *Mathematics of Computation*, Vol. 65, No. 216, 1996, pp. 1553-1566.

46. Farouki, R. T., and Rajan, V. T., "Algorithms for Polynomials in Bernstein Form", *Computer Aided Geometric Design*, Vol. 5, 1988, pp. 1-26.

47. Fletcher, R., *Practical Methods of Optimization*, 2nd ed., Wiley, Chichester, 1987.

48. Fliess, M., Levine, J., Martin, P., and Rouchon, P., "On Differentially Flat Nonlinear Systems", *Proceedings of the IFAC Symposium NOLCOS'92*, Bordeaux, France, 1992, pp. 408-412.

49. Fliess, M., Levine, J., Martin, P., and Rouchon, P., "Flatness and Defect of Nonlinear Systems: Introductory Theory and Examples", *International Journal of Control*, Vol. 61, No. 6, 1995, pp. 1327-1361.

50. Fornberg, B., *A Practical Guide to Pseudospectral Methods*, Cambridge University Press, Cambridge, 1996.

51. Frazzoli, E., Dahleh, M. A., and Feron, E., "Real-Time Motion Planning for Agile Autonomous Vehicles", *Journal of Guidance, Control, and Dynamics*, Vol. 25, No. 1, 2002, pp. 116-129.

52. Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: an SQP Algorithm for Large-Scale Constrained Optimization ", *SIAM Journal on Optimization*, Vol. 12, 2002, pp. 979-1006.

53. Gill, P. E., Murray, W., and Saunders, M. A., "User's Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming", Department of Mathematics, University of California, San Diego, 2007.

54. Gill, P. E., Murray, W., and Wright, M. H., *Practical Optimization*, Academic Press, London, 1981.

55. Gong, Q., Ross, I. M., Kang, W., and Fahroo, F., "Connections Between the Covector Mapping Theorem and Convergence of Pseudospectral Methods for Optimal Control", *Computational Optimization and Applications*, Vol. 41, No. 3, 2008, pp. 307-335.

56. Griffin, J. D., and Kolda, T. G., "Nonlinearly-Constrained Optimization Using Asynchronous Parallel Generating Set Search", Report SAND2007-3257, Sandia National Laboratories, Albuquerque, 2007.

57. Grubin, C., "Derivation of the Quaternion Scheme via the Euler Axis and Angle", *Journal of Spacecraft and Rockets*, Vol. 7, No. 10, 1970, pp. 1261-1263.

58. Grubin, C., "Quaternion Singularity Revisited", *Journal of Guidance, Control, and Dynamics*, Vol. 2, No. 3, 1979, pp. 255-256.

59. Gu, D.-W., Postlethwaite, I., and Kim, Y., "A Comprehensive Study on Flight Path Selection Algorithms", *Proceedings of the IEE Seminar on Target Tracking: Algorithms and Applications*, Birmingham, 2005, pp. 77-90.

60. Hargraves, C. R., and Paris, S. W., "Direct Trajectory Optimization using Nonlinear Programming and Collocation", *Journal of Guidance, Control, and Dynamics*, Vol. 10, No. 4, 1987, pp. 338-342.

61. Hess, R. A., Gao, C., and Wang, S. H., "Generalized Technique for Inverse Simulation Applied to Aircraft Maneuvers", *Journal of Guidance, Control, and Dynamics*, Vol. 14, No. 5, 1991, pp. 920-926.

62. Higham, D. J., and Higham, N. J., *Matlab Guide*, 2nd ed., SIAM, Philadelphia, 2005.

63. Ho, Y. C., Bryson, A. E., and Baron, S., "Differential Games and Optimal Pursuit Evasion Strategies", *IEEE Transactions on Automatic Control*, Vol. 10, 1965, pp. 385-389.

64. Hooke, R., and Jeeves, T. A., "Direct Search Solution of Numerical and Statistical Problems", *Journal of the Association for Computing Machinery*, Vol. 8, No. 2, 1961, pp. 212-229.

65. Hull, D. G., "Conversion of Optimal Control Problems into Parameter Optimization Problems", *Journal of Guidance, Control, and Dynamics*, Vol. 20, No. 1, 1997, pp. 57-61.

66. Huntington, G. T., "Advancement and Analysis of a Gauss Pseudospectral Transcription for Optimal Control Problems", PhD Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, 2007.

67. Ingber, A. L., "Simulated Annealing: Practice vs Theory", *Mathematical and Computer Modelling*, Vol. 18, No. 11, 1993, pp. 29-57.

68. Jaddu, H., and Shimemura, E., "Computation of Optimal Control Trajectories Using Chebyshev Polynomials, Parameterization, and Quadratic Programming", *Optimal Control Applications and Methods*, Vol. 20, 1999, pp. 21-42.

69. Jaddu, H., and Shimemura, E., "Computational Method Based on State Parameterization for Solving Constrained Nonlinear Optimal Control Problems", *International Journal of Systems Science*, Vol. 30, No. 3, 1999, pp. 275-282.

70. Kamal, W. A., Gu, D.-W., and Postlethwaite, I., "MILP and its Application in Flight Planning", *Proceedings of the 16th IFAC World Congress*, Prague, 2005.

71. Kaminer, I. I., Yakimenko, O. A., Dobrokhodov, V. N., Lizarraga, M. I., and Pascoal, A. M., "Cooperative Control of Small UAVs for Naval Applications", *Proceedings of the IEEE Conference on Decision and Control*, Atlantis, Bahamas, 2004, pp. 626-631.

72. Kaminer, I. I., Yakimenko, O. A., and Pascoal, A. M., "Coordinated Control of Multiple UAVs for Time-Critical Applications", *Proceedings of the IEEE Aerospace Conference*, Big Sky, 2006.

73. Kaminer, I. I., Yakimenko, O. A., Pascoal, A. M., and Ghabcheloo, R., "Path Generation, Path Following and Coordinated Control for Time Critical Missions of Multiple UAVs", *Proceedings of the American Control Conference*, Minneapolis, 2006, pp. 4906-4913.

74. Kato, O., and Sugiura, I., "An Interpretation of Airplane General Motion and Control as Inverse Problem", *Journal of Guidance, Control, and Dynamics*, Vol. 9, No. 2, 1986, pp. 198-204.

75. Kelley, C. T., *Iterative Methods for Optimization*, SIAM, Philadelphia, 1999.

76. Kim, M.-J., Kim, M.-S., and Shin, S. Y., "A Compact Differential Formula for the First Derivative of a Unit Quaternion Curve", *Journal of Visualization and Computer Animation*, Vol. 7, No. 1, 1996, pp. 43-57.

77. Kim, M.-S., and Nam, K.-W., "Interpolating Solid Orientations with Circular Blending Quaternion Curves", *Computer-Aided Design*, Vol. 27, No. 5, 1995, pp. 385-398.

78. Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., "Optimization by Simulated Annealing", *Science*, Vol. 220, No. 4598, 1983, pp. 671-680.

79. Knoebel, N. B., Osborne, S. R., Snyder, D. O., McLain, T. W., Beard, R. W., and Eldredge, A. M., "Preliminary Modeling, Control, and Trajectory Design for Minature Autonomous Tailsitters", *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Keystone, 2006.

80. Kolda, T. G., "Revisiting Asynchronous Parallel Pattern Search for Nonlinear Optimization", *SIAM Journal on Optimization*, Vol. 16, No. 2, 2005, pp. 563-586.

81. Kolda, T. G., Lewis, R. M., and Torczon, V., "Optimization by Direct Search: New Perspectives on Some Classical and Modern Methods", *SIAM Review*, Vol. 45, No. 3, 2003, pp. 385-482.

82. Krozel, J., and Andrisani, D., "Navigation Path Planning for Autonomous Aircraft: Voronoi Diagram Approach", *Journal of Guidance, Control, and Dynamics*, Vol. 13, No. 6, 1990, pp. 1152-1154.

83. Kumar, R., and Seywald, H., "Should Controls be Eliminated While Solving Optimal Control Problems via Direct Methods?", *Journal of Guidance, Control, and Dynamics*, Vol. 19, No. 2, 1996, pp. 418-423.

84. Lagarias, J. C., Reeds, J. A., Wright, M. H., and Wright, P. E., "Convergence Properties of the Nelder-Mead Simplex Algorithm in Low Dimensions", Report 96-4-07, Computing Sciences Research Center, Bell Laboratories, New Jersey, 1997.

85. Lampinen, J. A., "A Constraint Handling Approach for the Differential Evolution Algorithm", *Proceedings of the Congress on Evolutionary Computation*, Honolulu, 2002, pp. 1468-1473.

86. Lanczos, C., "Trigonometric Interpolation of Empirical and Analytical Functions", *Journal of Mathematics and Physics*, Vol. 17, 1938, pp. 123-199.

87. Lane, S. H., and Stengel, R. F., "Flight Control Design Using Nonlinear Inverse Dynamics", *Journal of Guidance, Control, and Dynamics*, Vol. 24, No. 4, 1988, pp. 471-483.

88. Lewis, F. L., and Syrmos, V. L., *Optimal Control*, 2nd ed., Wiley, New York, 1995.

89. Lin, K.-C., "Comment on Generalized Technique for Inverse Simulation Applied to Aircraft Maneuvers", *Journal of Guidance, Control, and Dynamics*, Vol. 16, No. 6, 1993, pp. 1196-1199.

90. Lou, K. Y., and Bryson, A. E., "Inverse and Optimal Control for Precision Aerobatic Maneuvers", *Journal of Guidance, Control, and Dynamics*, Vol. 19, No. 2, 1996, pp. 483-488.

91. Lu, P., "Inverse Dynamics Approach to Trajectory Optimization for an Aerospace Plane", *Journal of Guidance, Control, and Dynamics*, Vol. 16, No. 4, 1993, pp. 726-732.

92. Lu, P., and Khan, M. A., "Nonsmooth Trajectory Optimization: an Approach Using Continuous Simulated Annealing", *Journal of Guidance, Control, and Dynamics*, Vol. 17, No. 4, 1993, pp. 685-691.

93. Martin, P., Murray, R. M., and Rouchon, P., "Flat Systems, Equivalence and Trajectory Generation", Report CDS 2003-008, California Institute of Technology, Pasadena, 2003.

94. Menon, P. K. A., "Short-Range Nonlinear Feedback Strategies for Aircraft Pursuit-Evasion", *Journal of Guidance, Control, and Dynamics*, Vol. 12, No. 1, 1989, pp. 27-32.

95. Miele, A., *Flight Mechanics*, Addison-Wesley, Massachusetts, 1962.

96. Milam, M. B., "Real-Time Optimal Trajectory Generation for Constrained Dynamical Systems", PhD Thesis, California Institute of Technology, 2003.

97. Miller, N., "Watchkeeper in the UK - Update", *Proceedings of the 25th Bristol International UAV Systems Conference*, Bristol, 2010.

98. Nelder, J. A., and Mead, R., "A Simplex Method for Function Minimization", *The Computer Journal*, Vol. 8, No. 7, 1965, pp. 308-313.

99. Nocedal, J., and Wright, S. J., *Numerical Optimization*, 2nd ed., Springer, New York, 2006.

100. Oberle, H. J., and Grimm, W., "BNDSCO - a Program for the Numerical Solution of Optimal Control Problems", Report DLR IB 515-89/22, Institut für Dynamik der FlugSysteme, DLR, Oberpfaffenhofen, 1989.

101. Orszag, S. A., "Comparison of Pseudospectral and Spectral Approximations", *Studies in Applied Mathematics*, Vol. 51, No. 3, 1972, pp. 253-259.

102. Pesch, H. J., "A Practical Guide to the Solution of Real-Life Optimal Control Problems", *Control and Cybernetics*, Vol. 23, 1994, pp. 7-60.

103. Phillips, W. F., Hailey, C. E., and Gebert, G. A., "Review of Attitude Representations Used for Aircraft Kinematics", *Journal of Aircraft*, Vol. 38, No. 4, 2001, pp. 718-737.

104. Polak, E., *Optimization - Algorithms and Consistent Approximations*, Springer-Verlag, New York, 1998.

105. Pontryagin, L. S., Boltjanskiy, V. G., Gamkrelidze, R. V., and Mishenko, E. F., *The Mathematical Theory of Optimal Processes*, Interscience, New York, 1962.

106. Powell, M. J. D., "Problems Related to Unconstrained Optimization", *in Numerical Methods for Unconstrained Optimization*, Murray, W. (Ed.), Academic Press, London, 1972.

107. Powell, M. J. D., "A View of Algorithms for Optimization Without Derivatives", *Mathematics Today*, Vol. 43, No. 5, 2007, pp. 170-174.

108. Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P., *Numerical Recipes: The Art of Scientific Computing*, 3rd ed., Cambridge University Press, New York, 2007.

109. Price, C. J., Coope, I. D., and Byatt, D., "A Convergent Variant of the Nelder-Mead Algorithm", *Journal of Optimization Theory and Applications*, Vol. 113, No. 1, 2002, pp. 5-19.

110. Price, K. V., Storn, R. M., and Lampinen, J. A., *Differential Evolution: a Practical Approach to Global Optimization*, Springer-Verlag, Berlin, 2005.

111. Reynolds, R. G., "Quaternion Parameterization and a Simple Algorithm for Global Attitude Estimation", *Journal of Guidance, Control, and Dynamics*, Vol. 21, No. 4, 1998, pp. 669-671.

112. Ross, I. M., and Fahroo, F., "Pseudospectral Methods for Optimal Motion Planning of Differentially Flat Systems", *IEEE Transactions on Automatic Control*, Vol. 49, No. 8, 2004, pp. 1410-1413.

113. Ross, I. M., Gong, Q., and Sekhavat, P., "Low-Thrust, High Accuracy Trajectory Optimization", *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 4, 2007, pp. 921-933.

114. Rutherford, S., and Thomson, D. G., "Improved Methodology for Inverse Simulation", *The Aeronautical Journal*, Vol. 100, No. 993, 1996, pp. 79-86.

115. Sentoh, E., and Bryson, A. E., "Inverse and Optimal Control for Desired Outputs", *Journal of Guidance, Control, and Dynamics*, Vol. 15, No. 3, 1992, pp. 687-691.

116. Seywald, H., "Trajectory Optimization Based on Differential Inclusions", *Journal of Guidance, Control, and Dynamics*, Vol. 17, No. 3, 1994, pp. 480-487.

117. Shanmugavel, M., Tsourdos, A., Zbikowski, R., and White, B. A., "Path Planning of Multiple UAVs Using Dubins Sets", *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, San Francisco, 2005.

118. Shanmugavel, M., Tsourdos, A., Zbikowski, R., and White, B. A., "3D Dubins Sets Based Coordinated Path Planning for Swarm of UAVs", *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Keystone, 2006.

119. Shepperd, S. W., "Quaternion from Rotation Matrix", *Journal of Guidance and Control*, Vol. 1, No. 3, 1978, pp. 223-224.

120. Shoemake, K., "Animating Rotation with Quaternion Curves", *Computer Graphics*, Vol. 19, No. 3, 1985, pp. 245-254.

121. Shuster, M. D., "A Survey of Attitude Representations", *Journal of the Astronautical Sciences*, Vol. 41, No. 4, 1993, pp. 439-517.

122. Stengel, R. F., *Flight Dynamics*, Princeton University Press, Princeton, 2004.

123. Stevens, B. L., and Lewis, F. L., *Aircraft Control and Simulation*, 2nd ed., Wiley, Hoboken, 2003.

124. Stoer, J., and Bulirsch, R., *Introduction to Numerical Analysis*, 3rd ed., Springer-Verlag, New York, 2002.

125. Subchan, S., and Żbikowski, R., *Computational Optimal Control: Tools and Practice*, Wiley, Chichester, 2009.

126. Taranenko, V. T., "Experience of Employment of Ritz's, Poincaré's and Lyapunov's Methods for Solving the Problems of Flight Dynamics", Soviet Air Force Engineering Academy, Moscow, 1968, (in Russian).

127. Taranenko, V. T., and Momdzhi, V. G., *Direct Method of Calculus of Variations in Boundary Problems of Flight Dynamics*, Mashinostroenie Press, Moscow, 1986, (in Russian).

128. Torczon, V., "On the Convergence of Pattern Search Algorithms", *SIAM Journal on Optimization*, Vol. 7, No. 1, 1997, pp. 1-25.

129. Trefethen, L. N., *Spectral Methods in Matlab*, SIAM, Philadelphia, 2000.

130. von Stryk, O., and Bulirsch, R., "Direct and Indirect Methods for Trajectory Optimization", *Annals of Operational Research*, Vol. 37, 1992, pp. 357-373.

131. White, B. A., Zbikowski, R., and Tsourdos, A., "Aim Point Guidance: an Extension of Proportional Navigation to the Control of Terminal Guidance", *Proceedings of the American Control Conference*, Denver, 2003, pp. 384-389.

132. White, B. A., Zbikowski, R., and Tsourdos, A., "Direct Intercept Guidance using Differential Geometry Concepts", *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, San Francisco, 2005.

133. Yakimenko, O. A., "Direct Method for Rapid Prototyping of Near-Optimal Aircraft Trajectories", *Journal of Guidance, Control, and Dynamics*, Vol. 23, No. 5, 2000, pp. 865-875.

134. Yakimenko, O. A., and Kaminer, I. I., "Near-optimal trajectory generation for autonomous aircraft landing", *Proceedings of the IEEE International Symposium on Computer Aided Control System Design*, Honolulu, 1999, pp. 445-450.

135. Yakimenko, O. A., Xu, Y., and Basset, G., "Computing Short-Time Aircraft Maneuvers Using Direct Methods", *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Honolulu, 2008.

136. Yang, H. I., and Zhao, Y. J., "Trajectory Planning for Autonomous Aerospace Vehicles amid Known Obstacles and Conflicts", *Journal of Guidance, Control, and Dynamics*, Vol. 27, No. 6, 2004, pp. 997-1008.